Natural Language Processing

2004, 8 Lectures
Ann Copestake (aac@cl.cam.ac.uk)
http://www.cl.cam.ac.uk/users/aac/

Lecture Synopsis

Aims

This course aims to introduce the fundamental techniques of natural language processing and to develop an understanding of the limits of those techniques. It aims to introduce some current research issues, and to evaluate some current and potential applications.

- **Introduction.** Brief history of NLP research, current applications, generic NLP system architecture, knowledge-based *versus* probabilistic approaches.
- **Finite-state techniques.** Inflectional and derivational morphology, finite-state automata in NLP, finite-state transducers.
- **Prediction and part-of-speech tagging.** Corpora, simple N-grams, word prediction, stochastic tagging, evaluating system performance.
- **Parsing and generation.** Generative grammar, context-free grammars, parsing and generation with context-free grammars, weights and probabilities.
- **Parsing with constraint-based grammars.** Constraint-based grammar, unification.
- **Compositional and lexical semantics.** Simple compositional semantics in constraint-based grammar. Semantic relations, WordNet, word senses, word sense disambiguation.
- **Discourse and dialogue.** Anaphora resolution, discourse relations.
- **Applications.** Machine translation, email response, spoken dialogue systems.

Objectives

At the end of the course students should

- be able to describe the architecture of and basic design for a generic NLP system "shell"
- be able to discuss the current and likely future performance of several NLP applications, such as machine translation and email response
- be able to describe briefly a fundamental technique for processing language for several subtasks, such as morphological analysis, parsing, word sense disambiguation etc.
- understand how these techniques draw on and relate to other areas of (theoretical) computer science, such as formal language theory, formal semantics of programming languages, or theorem proving

Overview

NLP is a large and multidisciplinary field, so this course can only provide a very general introduction. The first lecture is designed to give an overview of the main subareas and a very brief idea of the main applications and the methodologies which have been employed. The history of NLP is briefly discussed as a way of putting this into perspective. The next six lectures describe some of the main subareas in more detail. The organisation is roughly based on increased 'depth' of processing, starting with relatively surface-oriented techniques and progressing to considering meaning of sentences and meaning of utterances in context. Most lectures will start off by considering the subarea as a whole and then go on to describe one or more sample algorithms which tackle particular problems. The algorithms have been chosen because they are relatively straightforward to describe and because they illustrate a specific technique which has been shown to be useful, but the idea is to exemplify an approach, not to give a detailed survey (which would be impossible in the time available). (Lecture 5 is a bit different in that it concentrates on a data structure instead of an algorithm.) The final lecture brings the preceding material together in order to describe the state of the art in three sample applications.

There are various themes running throughout the lectures. One theme is the connection to linguistics and the tension that sometimes exists between the predominant view in theoretical linguistics and the approaches adopted within NLP. A somewhat related theme is the distinction between knowledge-based and probabilistic approaches. Evaluation will be discussed in the context of the different algorithms.

Because NLP is such a large area, there are many topics that aren't touched on at all in these lectures. Speech recognition and speech synthesis is almost totally ignored. Information retrieval and information extraction are the topic of a separate course given by Simone Teufel, for which this course is a prerequisite.

Feedback on the handout, lists of typos etc, would be greatly appreciated.

Recommended Reading

Recommended Book:

Jurafsky, Daniel and James Martin, *Speech and Language Processing*, Prentice-Hall, 2000 (referenced as J&M throughout this handout).

Background:

These books are about linguistics rather that NLP/computational linguistics. They are not necessary to understand the course, but should give readers an idea about some of the properties of human languages that make NLP interesting and challenging, without being technical.

Pinker, S., *The Language Instinct*, Penguin, 1994.

This is a thought-provoking and sometimes controversial 'popular' introduction to linguistics.

Matthews, Peter, *Linguistics: a very short introduction*, OUP, 2003.

The title is accurate ...

Background/reference:

The Internet Grammar of English, http://www.ucl.ac.uk/internet-grammar/home.htm

Syntactic concepts and terminology.

Study and Supervision Guide

The handouts and lectures should contain enough information to enable students to adequately answer the exam questions, but the handout is not intended to substitute for a textbook. In most cases, J&M go into a considerable amount of further detail: rather than put lots of suggestions for further reading in the handout, in general I have assumed that students will look at J&M, and then follow up the references in there if they are interested. The notes at the end of each lecture give details of the sections of J&M that are relevant and details of any discrepancies with these notes.

Supervisors ought to familiarise themselves with the relevant parts of Jurafsky and Martin (see notes at the end of each lecture). However, good students should find it quite easy to come up with questions that the supervisors (and the

lecturer) can't answer! Language is like that ...

Generally I'm taking a rather informal/example-based approach to concepts such as finite-state automata, context-free grammars etc. Part II students should have already got the formal background that enables them to understand the application to NLP. Diploma and Part II (General) students may not have covered all these concepts before, but the expectation is that the examples are straightforward enough so that this won't matter too much.

This course inevitably assumes some very basic linguistic knowledge, such as the distinction between the major parts of speech. It introduces some linguistic concepts that won't be familiar to all students: since I'll have to go through these quickly, reading the first few chapters of an introductory linguistics textbook may help students understand the material. The idea is to introduce just enough linguistics to motivate the approaches used within NLP rather than to teach the linguistics for its own sake. At the end of this handout, there are some mini-exercises to help students understand the concepts: it would be very useful if these were attempted before the lectures as indicated. There are also some suggested post-lecture exercises.

Exam questions won't rely on students remembering the details of any specific linguistic phenomenon. As far as possible, exam questions will be suitable for people who speak English as a second language. For instance, if a question relied on knowledge of the ambiguity of a particular English word, a gloss of the relevant senses would be given.

Of course, I'll be happy to try and answer questions about the course or more general NLP questions, preferably by email.

1 Lecture 1: Introduction to NLP

The aim of this lecture is to give students some idea of the objectives of NLP. The main subareas of NLP will be introduced, especially those which will be discussed in more detail in the rest of the course. There will be a preliminary discussion of the main problems involved in language processing by means of examples taken from NLP applications. This lecture also introduces some methodological distinctions and puts the applications and methodology into some historical context.

1.1 What is NLP?

Natural language processing (NLP) can be defined as the automatic (or semi-automatic) processing of human language. The term 'NLP' is sometimes used rather more narrowly than that, often excluding information retrieval and sometimes even excluding machine translation. NLP is sometimes contrasted with 'computational linguistics', with NLP being thought of as more applied. Nowadays, alternative terms are often preferred, like 'Language Technology' or 'Language Engineering'. Language is often used in contrast with speech (e.g., Speech and Language Technology). But I'm going to simply refer to NLP and use the term broadly.

NLP is essentially multidisciplinary: it is closely related to linguistics (although the extent to which NLP overtly draws on linguistic theory varies considerably). It also has links to research in cognitive science, psychology, philosophy and maths (especially logic). Within CS, it relates to formal language theory, compiler techniques, theorem proving, machine learning and human-computer interaction. Of course it is also related to AI, though nowadays it's not generally thought of as part of AI.

1.2 Some linguistic terminology

The course is organised so that there are six lectures corresponding to different NLP subareas, moving from relatively 'shallow' processing to areas which involve meaning and connections with the real world. These subareas loosely correspond to some of the standard subdivisions of linguistics:

1. Morphology: the structure of words. For instance, *unusually* can be thought of as composed of a prefix *un-*, a stem *usual*, and an affix *-ly*. *composed* is *compose* plus the inflectional affix *-ed*: a spelling rule means we end up with *composed* rather than *composeed*. Morphology will be discussed in lecture 2.

2. Syntax: the way words are used to form phrases. e.g., it is part of English syntax that a determiner such as *the* will come before a noun, and also that determiners are obligatory with certain singular nouns. Formal and computational aspects of syntax will be discussed in lectures 3, 4 and 5.

3. Semantics. Compositional semantics is the construction of meaning (generally expressed as logic) based on syntax. This is contrasted to lexical semantics, i.e., the meaning of individual words. Compositional and lexical semantics is discussed in lecture 6.

4. Pragmatics: meaning in context. This will come into lecture 7, although linguistics and NLP generally have very different perspectives here.

1.3 Why is language processing difficult?

Consider trying to build a system that would answer email sent by customers to a retailer selling laptops and accessories via the Internet. This might be expected to handle queries such as the following:

- Has my order number 4291 been shipped yet?
- Is FD5 compatible with a 505G?
- What is the speed of the 505G?

Assume the query is to be evaluated against a database containing product and order information, with relations such as the following:

ORDER

Order number	Date ordered	Date shipped
4290	2/2/02	2/2/02
4291	2/2/02	2/2/02
4292	2/2/02	

USER: Has my order number 4291 been shipped yet?
DB QUERY: order(number=4291,date_shipped=?)
RESPONSE TO USER: Order number 4291 was shipped on 2/2/02

It might look quite easy to write patterns for these queries, but very similar strings can mean very different things, while very different strings can mean much the same thing. 1 and 2 below look very similar but mean something completely different, while 2 and 3 look very different but mean much the same thing.

1. How fast is the 505G?
2. How fast will my 505G arrive?
3. Please tell me when I can expect the 505G I ordered.

While some tasks in NLP can be done adequately without having any sort of account of meaning, others require that we can construct detailed representations which will reflect the underlying meaning rather than the superficial string.

In fact, in natural languages (as opposed to programming languages), ambiguity is ubiquitous, so exactly the same string might mean different things. For instance in the query:

> Do you sell Sony laptops and disk drives?

the user may or may not be asking about Sony disk drives. This particular ambiguity may be represented by different bracketings:

> Do you sell (Sony laptops) and (disk drives)?
> Do you sell (Sony (laptops and disk drives))?

We'll see lots of examples of different types of ambiguity in these lectures.

Often humans have knowledge of the world which resolves a possible ambiguity, probably without the speaker or hearer even being aware that there is a potential ambiguity.[1] But hand-coding such knowledge in NLP applications has turned out to be impossibly hard to do for more than very limited domains: the term *AI-complete* is sometimes used (by analogy to NP-complete), meaning that we'd have to solve the entire problem of representing the world and acquiring world knowledge.[2] The term AI-complete is intended jokingly, but conveys what's probably the most important guiding principle in current NLP: we're looking for applications which don't require AI-complete solutions: i.e., ones where we can work with very limited domains or approximate full world knowledge by relatively simple techniques.

1.4 Some NLP applications

The following list is not complete, but useful systems have been built for:

[1] I'll use *hearer* generally to mean the person who is on the receiving end, regardless of the modality of the language transmission: i.e., regardless of whether it's spoken, signed or written. Similarly, I'll use *speaker* for the person generating the speech, text etc and *utterance* to mean the speech or text itself. This is the standard linguistic terminology, which recognises that spoken language is primary and text is a later development.

[2] In this course, I will use *domain* to mean some circumscribed body of knowledge: for instance, information about laptop orders constitutes a limited domain.

- spelling and grammar checking
- optical character recognition (OCR)
- screen readers for blind and partially sighted users
- augmentative and alternative communication (i.e., systems to aid people who have difficulty communicating because of disability)
- machine aided translation (i.e., systems which help a human translator, e.g., by storing translations of phrases and providing online dictionaries integrated with word processors, etc)
- lexicographers' tools
- information retrieval
- document classification (filtering, routing)
- document clustering
- information extraction
- question answering
- summarization
- text segmentation
- exam marking
- report generation (possibly multilingual)
- machine translation
- natural language interfaces to databases
- email understanding
- dialogue systems

Several of these applications are discussed briefly below. Roughly speaking, they are ordered according to the complexity of the language technology required. The applications towards the top of the list can be seen simply as aids to human users, while those at the bottom are perceived as agents in their own right. Perfect performance on any of these applications would be AI-complete, but perfection isn't necessary for utility: in many cases, useful versions of these applications had been built by the late 70s. Commercial success has often been harder to achieve, however.

1.5 Spelling and grammar checking

All spelling checkers can flag words which aren't in a dictionary.

(1) * The neccesary steps are obvious.

(2) The necessary steps are obvious.

If the user can expand the dictionary, or if the language has complex productive morphology (see §2.1), then a simple list of words isn't enough to do this and some morphological processing is needed.[3]

More subtle cases involve words which are correct in isolation, but not in context. Syntax could sort some of these cases out. For instance, possessive *its* generally has to be immediately followed by a noun or by one or more adjectives which are immediately in front of a noun:

[3]Note the use of * ('star') above: this notation is used in linguistics to indicate a sentence which is judged (by the author, at least) to be incorrect. ? is generally used for a sentence which is questionable, or at least doesn't have the intended interpretation. # is used for a pragmatically anomalous sentence.

(3) * Its a fair exchange.

(4) It's a fair exchange.

(5) * The dog came into the room, it's tail wagging.

(6) The dog came into the room, its tail wagging.

But, it sometimes isn't locally clear what the context is: e.g. *fair* is ambiguous between a noun and an adjective.

(7) * 'Its fair', was all Kim said.

(8) 'It's fair', was all Kim said.

(9) * Every village has an annual fair, except Kimbolton: it's fair is held twice a year.

(10) Every village has an annual fair, except Kimbolton: its fair is held twice a year.

The most elaborate spelling/grammar checkers can get some of these cases right, but none are anywhere near perfect.

Spelling correction can require a form of *word sense disambiguation*:

(11) # The tree's bows were heavy with snow.

(12) The tree's boughs were heavy with snow.

Getting this right requires an association between *tree* and *bough*. In the past, attempts might have been made to hand-code this in terms of general knowledge of trees and their parts. However this sort of hand-coding is not suitable for applications that work on unbounded domains. These days machine learning techniques are generally used to derive word associations from corpora:[4] this can be seen as a substitute for the fully detailed world knowledge, but may actually be a more realistic model of how humans do word sense disambiguation. However, commercial systems don't (yet) do this systematically.

Simple subject verb agreement can be checked automatically:[5]

(13) My friends like pizza.

(14) My friend likes pizza.

(15) * My friends likes pizza.

(16) * My friend like pizza.

(17) My friends were unhappy.

(18) * My friend were unhappy.

But this isn't as straightforward as it may seem:

(19) A number of my friends were unhappy.

(20) The number of my friends who were unhappy was amazing.

(21) My family were unhappy.

Whether the last example is grammatical or not depends on your dialect of English: it is grammatical for most British English speakers, but not for many Americans.

Checking punctuation can be hard (even AI-complete):
BBC News Online, 3 October, 2001

[4] A *corpus* is a body of text that has been collected for some purpose, see §3.1.

[5] In English, the subject of a sentence is generally a noun phrase which comes before the verb, in contrast to the object, which follows the verb.

> Students at Cambridge University, who come from less affluent backgrounds, are being offered up to 1,000 a year under a bursary scheme.

This sentence contains a *non-restrictive relative clause*: *who come from less affluent backgrounds*. This is a form of parenthetical comment. The sentence implies that most/all students at Cambridge come from less affluent backgrounds. What the reporter probably meant was a restrictive relative, which should not have commas round it:

> Students at Cambridge University who come from less affluent backgrounds are being offered up to 1,000 a year under a bursary scheme.

A restrictive relative is a type of *modifier*: that is, it further specifies the entities under discussion. Thus it refers to a subset of the students.

1.6 Information retrieval, information extraction and question answering

Information retrieval involves returning a set of documents in response to a user query: Internet search engines are a form of IR. However, one change from classical IR is that Internet search now uses techniques that rank documents according to how many links there are to them (e.g., Google's PageRank) as well as the presence of search terms.

Information extraction involves trying to discover specific information from a set of documents. The information required can be described as a template. For instance, for company joint ventures, the template might have slots for the companies, the dates, the products, the amount of money involved. The slot fillers are generally strings.

Question answering attempts to find a specific answer to a specific question from a set of documents, or at least a short piece of text that contains the answer.

(22) What is the capital of France?
Paris has been the French capital for many centuries.

There are some question-answering systems on the Web, but most use very basic techniques. For instance, Ask Jeeves relies on a fairly large staff of people who search the web to find pages which are answers to potential questions. The system performs very limited manipulation on the input to map to a known question. The same basic technique is used in many online help systems.

1.7 Machine translation

MT work started in the US in the early fifties, concentrating on Russian to English. A prototype system was publicly demonstrated in 1954 (remember that the first electronic computer had only been built a few years before that). MT funding got drastically cut in the US in the mid-60s and ceased to be academically respectable in some places, but Systran was providing useful translations by the late 60s. Systran is still going (updating it over the years is an amazing feat of software engineering): Systran now powers AltaVista's BabelFish

`http://world.altavista.com/`

and many other translation services on the web.

Until the 80s, the utility of general purpose MT systems was severely limited by the fact that text was not available in electronic form: Systran used teams of skilled typists to input Russian documents.

Systran and similar systems are not a substitute for human translation: they are useful because they allow people to get an idea of what a document is about, and maybe decide whether it is interesting enough to get translated properly. This is much more relevant now that documents etc are available on the Web. Bad translation is also, apparently, good enough for chatrooms.

Spoken language translation is viable for limited domains: research systems include Verbmobil, SLT and CSTAR.

1.8 Natural language interfaces and dialogue systems

Natural language interfaces were the 'classic' NLP problem in the 70s and 80s. LUNAR is the classic example of a natural language interface to a database (NLID): its database concerned lunar rock samples brought back from the Apollo missions. LUNAR is described by Woods (1978) (but note most of the work was done several years earlier): it was capable of translating elaborate natural language expressions into database queries.

SHRDLU (Winograd, 1973) was a system capable of participating in a dialogue about a microworld (the blocks world) and manipulating this world according to commands issued in English by the user. SHRDLU had a big impact on the perception of NLP at the time since it seemed to show that computers could actually 'understand' language: the impossibility of scaling up from the microworld was not realised.

LUNAR and SHRDLU both exploited the limitations of one particular domain to make the natural language understanding problem tractable, particularly with respect to ambiguity. To take a trivial example, if you know your database is about lunar rock, you don't need to consider the music or movement senses of *rock* when you're analysing a query.

There have been many advances in NLP since these systems were built: systems have become much easier to build, and somewhat easier to use, but they still haven't become ubiquitous. Natural Language interfaces to databases were commercially available in the late 1970s, but largely died out by the 1990s: porting to new databases and especially to new domains requires very specialist skills and is essentially too expensive (automatic porting was attempted but never successfully developed). Users generally preferred graphical interfaces when these became available. Speech input would make natural language interfaces much more useful: unfortunately, speaker-independent speech recognition still isn't good enough for even 1970s scale NLP to work well. Techniques for dealing with misrecognised data have proved hard to develop. In many ways, current commercially-deployed spoken dialogue systems are using pre-SHRDLU technology.

1.9 Some more history

Before the 1970s, most NLP researchers were concentrating on MT as an application (see above). NLP was a very early application of CS and started about the same time as Chomsky was publishing his first major works in formal linguistics (Chomskyan linguistics quickly became dominant, especially in the US). In the 1950s and early 1960s, ideas about formal grammar were being worked out in linguistics and algorithms for parsing natural language were being developed at the same time as algorithms for parsing programming languages. However, most linguists were uninterested in NLP and the approach that Chomsky developed turned out to be only somewhat indirectly useful for NLP.

NLP in the 1970s and first half of the 1980s was predominantly based on a paradigm where extensive linguistic and real-world knowledge was hand-coded. There was controversy about how much linguistic knowledge was necessary for processing, with some researchers downplaying syntax, in particular, in favour of world knowledge. NLP researchers were very much part of the AI community (especially in the US and the UK), and the debate that went on in AI about the use of logic vs other meaning representations ('neat' vs 'scruffy') also affected NLP. By the 1980s, several linguistic formalisms had appeared which were fully formally grounded and reasonably computationally tractable, and the linguistic/logical paradigm in NLP was firmly established. Unfortunately, this didn't lead to many useful systems, partly because many of the difficult problems (disambiguation etc) were seen as somebody else's job (and mainstream AI was not developing adequate knowledge representation techniques) and partly because most researchers were concentrating on the 'agent-like' applications and neglecting the user aids. Although the symbolic, linguistically-based systems sometimes worked quite well as NLIDs, they proved to be of little use when it came to processing less restricted text, for applications such as IE. It also became apparent that lexical acquisition was a serious bottleneck for serious development of such systems.

Statistical NLP became the most common paradigm in the 1990s, at least in the research community. Speech recognition had demonstrated that simple statistical techniques worked, given enough training data. NLP systems were built which required very limited hand-coded knowledge, apart from initial training material. Most applications were much shallower than the earlier NLIDs, but the switch to statistical NLP coincided with a change in US funding, which started to emphasise speech-based interfaces and IE. There was also a general realization of the importance of serious evaluation and of reporting results in a way that could be reproduced by other researchers. US funding emphasised competitions with specific tasks and supplied test material, which encouraged this, although there was a downside in that some of the techniques developed were very task-specific. It should be emphasised that there had

been computational work on corpora for many years (much of it by linguists): it became much easier to do corpus work by the late 1980s as disk space became cheap and machine-readable text became ubiquitous. Despite the shift in research emphasis to statistical approaches, most commercial systems remained primarily based on hand-coded linguistic information.

More recently the symbolic/statistical split has become less pronounced, since most researchers are interested in both.[6] There is considerable emphasis on machine learning in general, including machine learning for symbolic processing. Linguistically-based NLP has made something of a comeback, with increasing availability of open source resources, and the realisation that at least some of the classic statistical techniques seem to be reaching limits on performance, especially because of difficulties in adapting to new types of text. However, the modern linguistically-based approaches are making use of machine learning and statistical processing. The dotcom boom and bust has considerably affected NLP, but it's too early to say what the long-term implications are. The ubiquity of the Internet has certainly changed the space of interesting NLP applications, and the vast amount of text available can potentially be exploited, especially for statistical techniques.

1.10 Generic 'deep' NLP application architecture

Many NLP applications can be adequately implemented with relatively shallow processing. For instance, spelling checking only requires a word list and simple morphology to be useful. I'll use the term 'deep' NLP for systems that build a meaning representation (or an elaborate syntactic representation), which is generally agreed to be required for applications such as NLIDs, email question answering and good MT.

The most important principle in building a successful NLP system is modularity. NLP systems are often big software engineering projects — success requires that systems can be improved incrementally.

The input to an NLP system could be speech or text. It could also be gesture (multimodal input or perhaps a Sign Language). The output might be non-linguistic, but most systems need to give some sort of feedback to the user, even if they are simply performing some action (issuing a ticket, paying a bill, etc). However, often the feedback can be very formulaic.

There's general agreement that the following system components can be described semi-independently, although assumptions about the detailed nature of the interfaces between them differ. Not all systems have all of these components:

- input preprocessing: speech recogniser or text preprocessor (non-trivial in languages like Chinese or for highly structured text for any language) or gesture recogniser. Such system might themselves be very complex, but I won't discuss them in this course — we'll assume that the input to the main NLP component is segmented text.
- morphological analysis: this is relatively well-understood for the most common languages that NLP has considered, but is complicated for many languages (e.g., Turkish, Basque).
- part of speech tagging: not an essential part of most deep processing systems, but sometimes used as a way of cutting down parser search space.
- parsing: this includes syntax and compositional semantics, which are sometimes treated as separate components
- disambiguation: this can be done as part of parsing, or (partially) left to a later phase
- context module: this maintains information about the context, for anaphora resolution, for instance.
- text planning: the part of language generation that's concerned with deciding what meaning to convey (I won't discuss this in this course)
- tactical generation: converts meaning representations to strings. This may use the same grammar and lexicon[7] as the parser.
- morphological generation: as with morphological analysis, this is relatively straightforward for English.

[6] At least, there are only a few researchers who avoid statistical techniques as a matter of principle and all statistical systems have a symbolic component!

[7] The term *lexicon* is generally used for the part of the NLP system that contains dictionary-like information — i.e. information about individual words.

- output processing: text-to-speech, text formatter, etc. As with input processing, this may be complex, but for now we'll assume that we're outputting simple text.

Application specific components, for instance:

1. For NL interfaces, email answering and so on, we need an interface between semantic representation (expressed as some form of logic, for instance) and the underlying knowledge base.
2. For MT based on *transfer*, we need a component that maps between semantic representations.

It is also very important to distinguish between the knowledge sources and the programs that use them. For instance, a morphological analyser has access to a lexicon and a set of morphological rules: the morphological generator might share these knowledge sources. The lexicon for the morphology system may be the same as the lexicon for the parser and generator.

Other things might be required in order to construct the standard components and knowledge sources:

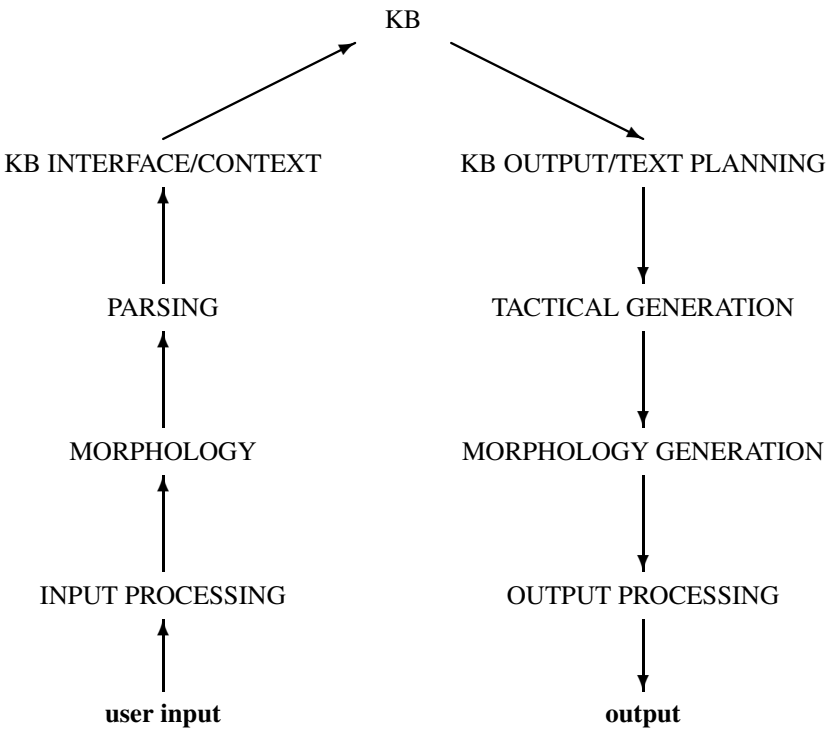
- lexicon acquisition
- grammar acquisition
- acquisition of statistical information

For a component to be a true module, it obviously needs a well-defined set of interfaces. What's less obvious is that it needs its own evaluation strategy and test suites: developers need to be able to work somewhat independently.

In principle, at least, components are *reusable* in various ways: for instance, a parser could be used with multiple grammars, the same grammar can be processed by different parsers and generators, a parser/grammar combination could be used in MT or in a natural language interface. However, for a variety of reasons, it is not easy to reuse components like this, and generally a lot of work is required for each new application, even if it's based on an existing grammar or the grammar is automatically acquired.

We can draw schematic diagrams for applications showing how the modules fit together.

1.11 Natural language interface to a knowledge base



In such systems, the context module generally gets included as part of the KB interface because the discourse state is quite simple, and contextual resolution is domain specific. Similarly, there's often no elaborate text planning requirement, though this depends very much on the KB and type of queries involved.

In lectures 2–7, various algorithms will be discussed which could be parts of modules in this generic architecture, although most are also useful in less elaborate contexts. Lecture 8 will discuss the architecture and requirements of a few applications in a bit more detail.

1.12 General comments

- Even 'simple' NLP applications, such as spelling checkers, need complex knowledge sources for some problems.
- Applications cannot be 100% perfect, because full real world knowledge is not possible.
- Applications that are less than 100% perfect can be useful (humans aren't 100% perfect anyway).
- Applications that aid humans are much easier to construct than applications which replace humans. It is difficult to make the limitations of systems which accept speech or language obvious to naive human users.
- NLP interfaces are nearly always competing with a non-language based approach.
- Currently nearly all applications either do relatively shallow processing on arbitrary input or deep processing on narrow domains. MT can be domain-specific to varying extents: MT on arbitrary text isn't very good, but has some applications.
- Limited domain systems require extensive and expensive expertise to port. Research that relies on extensive hand-coding of knowledge for small domains is now generally regarded as a dead-end, though reusable hand-coding is a different matter.
- The development of NLP has mainly been driven by hardware and software advances, and societal and infrastructure changes, not by great new ideas. Improvements in NLP techniques are generally incremental rather than revolutionary.

2 Lecture 2: Morphology and finite-state techniques

This lecture starts with a brief discussion of morphology, concentrating mainly on English morphology. The concept of a lexicon in an NLP system is discussed with respect to morphological processing. Spelling rules are introduced and the use of finite state transducers to implement spelling rules is explained. The lecture concludes with a brief overview of some other uses of finite state techniques in NLP.

2.1 A very brief and simplified introduction to morphology

Morphology concerns the structure of words. Words are assumed to be made up of *morpheme*s, which are the minimal information carrying unit. Morphemes which can only occur in conjunction with other morphemes are *affix*es: words are made up of a stem (more than one in the case of compounds) and zero or more affixes. For instance, *dog* is a stem which may occur with the plural suffix *+s* i.e., *dogs*. English only has suffixes (affixes which come after a stem) and prefixes (which come before the stem — in English these are limited to derivational morphology), but other languages have *infixes* (affixes which occur inside the stem) and circumfixes (affixes which go around a stem). For instance, Arabic has stems (root forms) such as *k_t_b*, which are combined with infixes to form words (e.g., *kataba*, he wrote; *kotob*, books). Some English irregular verbs show a relic of inflection by infixation (e.g. *sing*, *sang*, *sung*) but this process is no longer *productive* (i.e., it won't apply to any new words, such as *ping*).[8]

2.2 Inflectional vs derivational morphology

Inflectional and derivational morphology can be distinguished, although the dividing line isn't always sharp. The distinction is of some importance in NLP, since it means different representation techniques may be appropriate. Inflectional morphology can be thought of as setting values of slots in some *paradigm*. Inflectional morphology concerns properties such as tense, aspect, number, person, gender, and case, although not all languages code all of these: English, for instance, has very little morphological marking of case and gender. Derivational affixes, such as *un-*, *re-*, *anti-* etc, have a broader range of semantic possibilities and don't fit into neat paradigms. Inflectional affixes may be combined (though not in English). However, there are always obvious limits to this, since once all the possible slot values are 'set', nothing else can happen. In contrast, there are no obvious limitations on the number of derivational affixes (*antidisestablishmentarianism*, *antidisestablishmentarianismization*) and they may even be applied recursively (*antiantimissile*). In some languages, such as Inuit, derivational morphology is often used where English would use adjectival modification or other syntactic means. This leads to very long 'words' occurring naturally and is presumably responsible for the claim that 'Eskimo' has hundreds of words for snow.

Inflectional morphology is generally close to fully productive, in the sense that a word of a particular class will generally show all the possible inflections although the actual affix used may vary. For instance, an English verb will have a present tense form, a 3rd person singular present tense form, a past participle and a passive participle (the latter two being the same for regular verbs). This will also apply to any new words which enter the language: e.g., *text* as a verb — *texts*, *texted*. Derivational morphology is less productive and the classes of words to which an affix applies is less clearcut. For instance, the suffix *-ee* is relatively productive (*textee* sounds plausible, meaning the recipient of a text message, for instance), but doesn't apply to all verbs (?*snoree*, ?*jogee*, ?*dropee*). Derivational affixes may change the part of speech of a word (e.g., *-ise*/*-ize* converts nouns into verbs: *plural*, *pluralise*). However, there are also examples of what is sometimes called *zero derivation*, where a similar effect is observed without an affix: e.g. *tango*, *waltz* etc are words which are basically nouns but can be used as verbs.

Stems and affixes can be individually ambiguous. There is also potential for ambiguity in how a word form is split into morphemes. For instance, *unionised* could be *union -ise -ed* or (in chemistry) *un- ion -ise -ed*. This sort of structural ambiguity isn't nearly as common in English morphology as in syntax, however. Note that *un- ion* is not a possible form (because *un-* can't attach to a noun). Furthermore, although there is a prefix *un-* that can attach to verbs, it nearly always denotes a reversal of a process (e.g., *untie*), whereas the *un-* that attaches to adjectives means 'not', which is the meaning in the case of *un- ion -ise -ed*. Hence the internal structure of *un- ion -ise -ed* has to be *(un- ((ion -ise) -ed))*.

[8] Arguably, though, spoken English has one productive infixation process, exemplified by *absobloodylutely*.

2.3 Spelling rules

English morphology is essentially concatenative: i.e., we can think of words as a sequence of prefixes, stems and suffixes. Some words have irregular morphology and their inflectional forms simply have to be listed. However, in other cases, there are regular phonological or spelling changes associated with affixation. For instance, the suffix *-s* is pronounced differently when it is added to a stem which ends in *s*, *x* or *z* and the spelling reflects this with the addition of an *e* (*boxes* etc). For the purposes of this course, we'll just talk about spelling effects rather than phonological effects: these effects can be captured by *spelling rules* (also known as *orthographic rules*).

English spelling rules can be described independently of the particular stems and affixes involved, simply in terms of the affix boundary. The 'e-insertion' rule can be described as follows:

$$\varepsilon \rightarrow \mathrm{e} / \left\{ \begin{array}{c} \mathrm{s} \\ \mathrm{x} \\ \mathrm{z} \end{array} \right\} \hat{} \_ s$$

In such rules, the mapping is always given from the 'underlying' form to the surface form, the mapping is shown to the left of the slash and the context to the right, with the _ indicating the position in question. ε is used for the empty string and ^ for the affix boundary. This particular rule is read as saying that the empty string maps to 'e' in the context where it is preceded by an s,x, or z and an affix boundary and followed by an s. For instance, this maps *box^s* to *boxes*. This rule might look as though it is written in a context sensitive grammar formalism, but actually we'll see in §2.7 that it corresponds to a finite state transducer. Because the rule is independent of the particular affix, it applies equally to the plural form of nouns and the 3rd person singular present form of verbs. Other spelling rules in English include consonant doubling (e.g., *rat*, *ratted*, though note, not **auditted*) and y/ie conversion (*party*, *parties*).

2.4 Applications of morphological processing

It is possible to use a *full-form lexicon* for English NLP: i.e., to list all the inflected forms and to treat derivational morphology as non-productive. However, when a new word has to be treated (generally because the application is expanded but in principle because a new word has entered the language) it is redundant to have to specify (or learn) the inflected forms as well as the stem, since the vast majority of words in English have regular morphology. So a full-form lexicon is best regarded as a form of compilation. Many other languages have many more inflectional forms, which increases the need to do morphological analysis rather than full-form listing.

IR systems use *stemming* rather than full morphological analysis. For IR, what is required is to relate forms, not to analyse them compositionally, and this can most easily be achieved by reducing all morphologically complex forms to a canonical form. Although this is referred to as stemming, the canonical form may not be the linguistic stem. The most commonly used algorithm is the *Porter stemmer*, which uses a series of simple rules to strip endings (see J&M, section 3.4) without the need for a lexicon. However, stemming does not necessarily help IR. Search engines sometimes do inflectional morphology, but this can be dangerous. For instance, one search engine searches for *corpus* as well as *corpora* when given the latter as input, resulting in a large number of spurious results involving *Corpus Christi* and similar terms.

In most NLP applications, however, morphological analysis is a precursor to some form of parsing. In this case, the requirement is to analyse the form into a stem and affixes so that the necessary syntactic (and possibly semantic) information can be associated with it. Morphological analysis is often called *lemmatization*. For instance, for the part of speech tagging application which we will discuss in the next lecture, *mugged* would be assigned a part of speech tag which indicates it is a verb, though *mug* is ambiguous between verb and noun. For full parsing, as discussed in lectures 4 and 5, we'll need more detailed syntactic and semantic information. Morphological generation takes a stem and some syntactic information and returns the correct form. For some applications, there is a requirement that morphological processing is *bidirectional*: that is, can be used for analysis and generation. The finite state transducers we will look at below have this property.

2.5 Lexical requirements for morphological processing

There are three sorts of lexical information that are needed for full, high precision morphological processing:

- affixes, plus the associated information conveyed by the affix
- irregular forms, with associated information similar to that for affixes
- stems with syntactic categories (plus more detailed information if derivational morphology is to be treated as productive)

One approach to an affix lexicon is for it to consist of a pairing of affix and some encoding of the syntactic/semantic effect of the affix.[9] For instance, consider the following fragment of a suffix lexicon (we can assume there is a separate lexicon for prefixes):

```
ed PAST_VERB
ed PSP_VERB
s PLURAL_NOUN
```

Here `PAST_VERB`, `PSP_VERB` and `PLURAL_NOUN` are abbreviations for some bundle of syntactic/semantic information: we'll discuss this briefly in §5.7.

A lexicon of irregular forms is also needed. One approach is for this to just be a triple consisting of inflected form, 'affix information' and stem, where 'affix information' corresponds to whatever encoding is used for the regular affix. For instance:

```
began PAST_VERB begin
begun PSP_VERB begin
```

Note that this information can be used for generation as well as analysis, as can the affix lexicon.

In most cases, English irregular forms are the same for all senses of a word. For instance, *ran* is the past of *run* whether we are talking about athletes, politicians or noses. This argues for associating irregularity with particular word forms rather than particular senses, especially since compounds also tend to follow the irregular spelling, even non-productively formed ones (e.g., the plural of *dormouse* is *dormice*). However, there are exceptions: e.g., *The washing was hung/*hanged out to dry* vs *the murderer was hanged.*

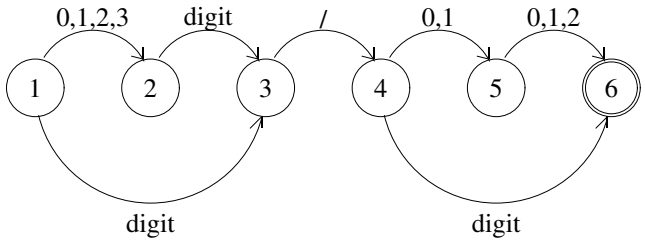
Morphological analysers also generally have access to a lexicon of regular stems. This is needed for high precision: e.g. to avoid analysing *corpus* as *corpu -s* we need to know that there isn't a word *corpu*. There are also cases where historically a word was derived, but where the base form is no longer found in the language: we can avoid analysing *unkempt* as *un- kempt*, for instance, simply by not having *kempt* in the stem lexicon. Ideally this lexicon should have syntactic information: for instance, *feed* could be *fee -ed*, but since *fee* is a noun rather than a verb, this isn't a possible analysis. However, in the approach we'll assume, the morphological analyser is split into two stages. The first of these only concerns morpheme forms and returns both *fee -ed* and *feed* given the input *feed*. A second stage which is closely coupled to the syntactic analysis then rules out *fee -ed* because the affix and stem syntactic information are not compatible (see §5.7 for one approach to this).

If morphology was purely concatenative, it would be very simple to write an algorithm to split off affixes. Spelling rules complicate this somewhat: in fact, it's still possible to do a reasonable job for English with ad hoc code, but a cleaner and more general approach is to use finite state techniques.

2.6 Finite state automata for recognition

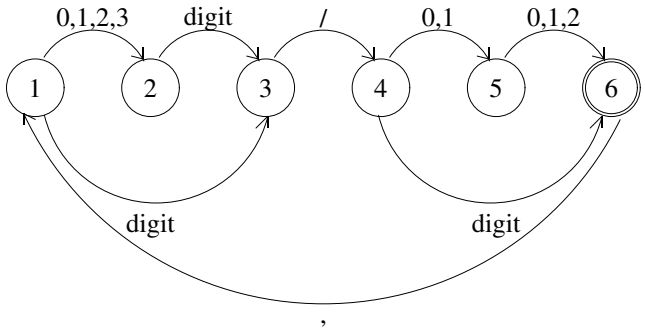
The approach to spelling rules that we'll describe involves the use of finite state transducers (FSTs). Rather than jumping straight into this, we'll briefly consider the simpler finite state automata and how they can be used in a simple recogniser. Suppose we want to recognise dates (just day and month pairs) written in the format day/month. The day and the month may be expressed as one or two digits (e.g. 11/2, 1/12 etc). This format corresponds to the following simple FSA, where each character corresponds to one transition:

[9] J&M describe an alternative approach which is to make the syntactic information correspond to a level in a finite state transducer. However, at least for English, this considerably complicates the transducers.



Accept states are shown with a double circle. This is a non-deterministic FSA: for instance, an input starting with the digit 3 will move the FSA to both state 2 and state 3. This corresponds to a *local ambiguity*: i.e., one that will be resolved by subsequent context. By convention, there must be no 'left over' characters when the system is in the final state.

To make this a bit more interesting, suppose we want to recognise a comma-separated list of such dates. The FSA, shown below, now has a cycle and can accept a sequence of indefinite length (note that this is iteration and not full recursion, however).



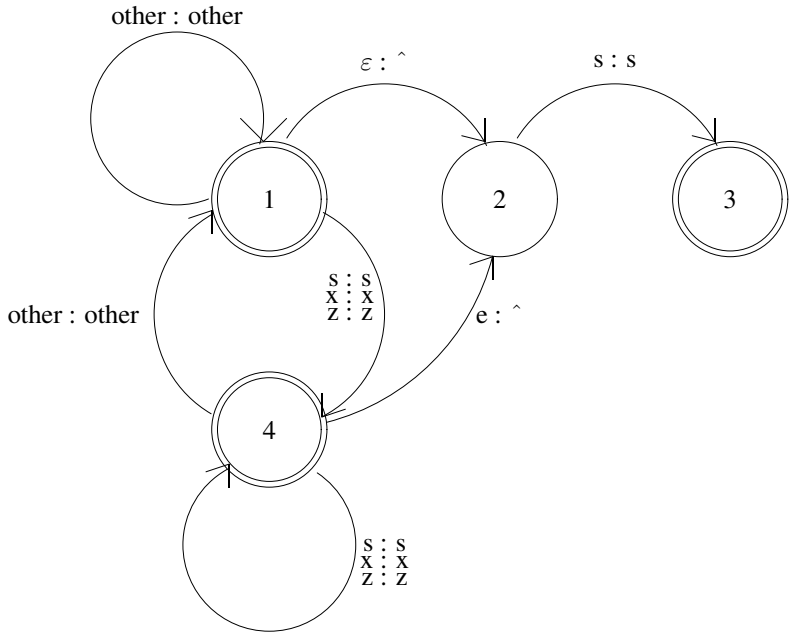
Both these FSAs will accept sequences which are not valid dates, such as 37/00. Conversely, if we use them to generate (random) dates, we will get some invalid output. In general, a system which generates output which is invalid is said to *overgenerate*. In fact, in many language applications, some amount of overgeneration can be tolerated, especially if we are only concerned with analysis.

2.7 Finite state transducers

FSAs can be used to recognise particular patterns, but don't, by themselves, allow for any analysis of word forms. Hence for morphology, we use finite state transducers (FSTs) which allow the surface structure to be mapped into the list of morphemes. FSTs are useful for both analysis and generation, since the mapping is bidirectional. This approach is known as *two-level morphology*.

To illustrate two-level morphology, consider the following FST, which recognises the affix *-s* allowing for environments corresponding to the e-insertion spelling rule shown in §2.3.[10]

[10]Actually, I've simplified this slightly so the correspondence to the spelling rule is not exact: J&M give a more complex transducer which is an accurate reflection of the spelling rule.



Transducers map between two representations, so each transition corresponds to a pair of characters. As with the spelling rule, we use the special character 'ε' to correspond to the empty character and '^' to correspond to an affix boundary. The abbreviation 'other : other' means that any character not mentioned specifically in the FST maps to itself. As with the FSA example, we assume that the FST only accepts an input if the end of the input corresponds to an accept state (i.e., no 'left-over' characters are allowed).

For instance, with this FST, 'd o g s' maps to 'd o g ^ s', 'f o x e s' maps to 'f o x ^ s' and 'b u z z e s' maps to 'b u z z ^ s'. When the transducer is run in analysis mode, this means the system can detect an affix boundary (and hence look up the stem and the affix in the appropriate lexicons). In generation mode, it can construct the correct string. This FST is non-deterministic.

Similar FSTs can be written for the other spelling rules for English (although to do consonant doubling correctly, information about stress and syllable boundaries is required and there are also differences between British and American spelling conventions which complicate matters). Morphology systems are usually implemented so that there is one FST per spelling rule and these operate in parallel.

One issue with this use of FSTs is that they do not allow for any internal structure of the word form. For instance, we can produce a set of FSTs which will result in *unionised* being mapped into *un^ion^ise^ed*, but as we've seen, the affixes actually have to be applied in the right order and this isn't modelled by the FSTs.

2.8 Some other uses of finite state techniques in NLP

- Grammars for simple spoken dialogue systems. Finite state techniques are not adequate to model grammars of natural languages: we'll discuss this a little in §4.12. However, for very simple spoken dialogue systems, a finite-state grammar may be adequate. More complex grammars can be written as CFGs and compiled into finite state approximations.

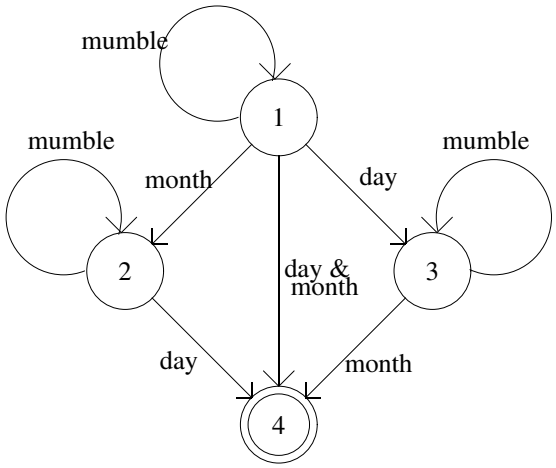
- Partial grammars for named entity recognition (briefly discussed in §4.12).

- Dialogue models for spoken dialogue systems (SDS). SDS use dialogue models for a variety of purposes: including controlling the way that the information acquired from the user is instantiated (e.g., the slots that are filled in an underlying database) and limiting the vocabulary to achieve higher recognition rates. FSAs can be used to record possible transitions between states in a simple dialogue. For instance, consider the problem of

obtaining a date expressed as a day and a month from a user. There are four possible states, corresponding to the user input recognised so far:

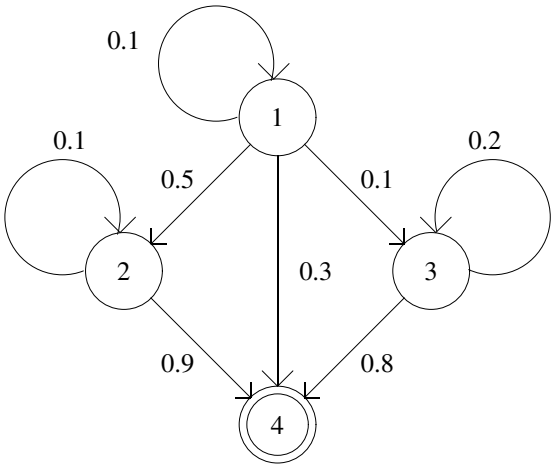
1. No information. System prompts for month and day.
2. Month only is known. System prompts for day.
3. Day only is known. System prompts for month.
4. Month and day known.

The FSA is shown below. The loops that stay in a single state correspond to user responses that aren't recognised as containing the required information (*mumble* is the term generally used for an unrecognised input).



2.9 Probabilistic FSAs

In many cases, it is useful to augment the FSA with information about transition probabilities. For instance, in the SDS system described above, it is more likely that a user will specify a month alone than a day alone. A probabilistic FSA for the SDS is shown below. Note that the probabilities on the outgoing arcs from each state must sum to 1.



2.10 Further reading

Chapters 2 and 3 of J&M. Much of Chapter 2 should be familiar from other courses in the CST (at least to Part II students). Chapter 3 uses more elaborate transducers than I've discussed.

3 Lecture 3: Prediction and part-of-speech tagging

This lecture introduces some simple statistical techniques and illustrates their use in NLP for prediction of words and part-of-speech categories. It starts with a discussion of corpora, then introduces word prediction. Word prediction can be seen as a way of (crudely) modelling some syntactic information (i.e., word order). Similar statistical techniques can also be used to discover parts of speech for uses of words in a corpus. The lecture concludes with some discussion of evaluation.

3.1 Corpora

A *corpus* (corpora is the plural) is simply a body of text that has been collected for some purpose. A *balanced corpus* contains texts which represent different genres (newspapers, fiction, textbooks, parliamentary reports, cooking recipes, scientific papers etc etc): early examples were the Brown corpus (US English) and the Lancaster-Oslo-Bergen (LOB) corpus (British English) which are each about 1 million words: the more recent British National Corpus (BNC) contains approx 100 million words and includes 20 million words of spoken English. Corpora are important for many types of linguistic research, although mainstream linguists have tended to dismiss their use in favour of reliance on intuitive judgements. Corpora are essential for much modern NLP research, though NLP researchers have often used newspaper text (particularly the Wall Street Journal) rather than balanced corpora.

Distributed corpora are often annotated in some way: the most important type of annotation for NLP is part-of-speech tagging (POS tagging), which we'll discuss further below.

Corpora may also be collected for a specific task. For instance, when implementing an email answering application, it is essential to collect samples of representative emails. For interface applications in particular, collecting a corpus requires a simulation of the actual application: generally this is done by a *Wizard of Oz* experiment, where a human pretends to be a computer.

Corpora are needed in NLP for two reasons. Firstly, we have to evaluate algorithms on real language: corpora are required for this purpose for any style of NLP. Secondly, corpora provide the data source for many machine-learning approaches.

3.2 Prediction

The essential idea of prediction is that, given a sequence of words, we want to determine what's most likely to come next. There are a number of reasons to want to do this: the most important is as a form of *language modelling* for automatic speech recognition. Speech recognisers cannot accurately determine a word from the sound signal for that word alone, and they cannot reliably tell where each word starts and finishes.[11] So the most probable word is chosen on the basis of the language model, which predicts the most likely word, given the prior context. The language models which are currently most effective work on the basis of *N-grams* (a type of *Markov chain*), where the sequence of the prior $n-1$ words is used to predict the next. Trigram models use the preceding 2 words, bigram models the preceding word and unigram models use no context at all, but simply work on the basis of individual word probabilities. Bigrams are discussed below, though I won't go into details of exactly how they are used in speech recognition.

Word prediction is also useful in communication aids: i.e., systems for people who can't speak because of some form of disability. People who use text-to-speech systems to talk because of a non-linguistic disability usually have some form of general motor impairment which also restricts their ability to type at normal rates (stroke, ALS, cerebral palsy etc). Often they use alternative input devices, such as adapted keyboards, puffer switches, mouth sticks or eye trackers. Generally such users can only construct text at a few words a minute, which is too slow for anything like normal communication to be possible (normal speech is around 150 words per minute). As a partial aid, a word prediction system is sometimes helpful: this gives a list of candidate words that changes as the initial letters are entered by the user. The user chooses the desired word from a menu when it appears. The main difficulty with using statistical prediction models in such applications is in finding enough data: to be useful, the model really has to be trained on an individual speaker's output, but of course very little of this is likely to be available.

[11] In fact, although humans are better at doing this than speech recognisers, we also need context to recognise words, especially words like *the* and *a*.

Prediction is important in estimation of entropy, including estimations of the entropy of English. The notion of entropy is important in language modelling because it gives a metric for the difficulty of the prediction problem. For instance, speech recognition is much easier in situations where the speaker is only saying two easily distinguishable words than when the vocabulary is unlimited: measurements of entropy can quantify this, but won't be discussed further in this course.

Other applications for prediction include optical character recognition (OCR), spelling correction and text segmentation for languages such as Chinese, which are conventionally written without explicit word boundaries. Some approaches to word sense disambiguation, to be discussed in lecture 6, can also be treated as a form of prediction.

3.3 bigrams

A bigram model assigns a probability to a word based on the previous word: i.e. $P(w_n|w_{n-1})$ where w_n is the nth word in some string. The probability of some string of words $P(W_1^n)$ is thus approximated by the product of these conditional probabilities:

$$P(W_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

For example, suppose we have the following tiny corpus of utterances:

good morning
good afternoon
good afternoon
it is very good
it is good

We'll use the symbol ⟨s⟩ to indicate the start of an utterance, so the corpus really looks like:

⟨s⟩ good morning ⟨s⟩ good afternoon ⟨s⟩ good afternoon ⟨s⟩ it is very good ⟨s⟩ it is good ⟨s⟩

The bigram probabilities are given as

$$\frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

i.e. the count of a particular bigram, normalised by dividing by the total number of bigrams starting with the same word (which is equivalent to the total number of occurrences of that word, except in the case of the last token, a complication which can be ignored for a reasonable size of corpus).

sequence	count	bigram probability
<s>	5	
<s> good	3	.6
<s> it	2	.4
good	5	
good morning	1	.2
good afternoon	2	.4
good <s>	2	.4
morning	1	
morning <s>	1	1
afternoon	2	
afternoon <s>	2	1
it is	2	1
is very	1	.5
is good	1	.5
very good	1	1

This yields a probability of 0.24 for the string '⟨s⟩ good ⟨s⟩' which is the highest probability utterance that we can construct on the basis of the bigrams from this corpus, if we impose the constraint that an utterance must begin with ⟨s⟩ and end with ⟨s⟩.

For application to communication aids, we are simply concerned with predicting the next word: once the user has made their choice, the word can't be changed. For speech recognition, the N-gram approach is applied to maximise the likelihood of a sequence of words, hence we're looking to find the most likely sequence overall. Notice that we can regard bigrams as comprising a simple deterministic weighted FSA. The *Viterbi algorithm*, an efficient method of applying N-grams in speech recognition and other applications, is usually described in terms of an FSA.

The probability of '⟨s⟩ very good' based on this corpus is 0, since the conditional probability of 'very' given '⟨s⟩' is 0 since we haven't found any examples of this in the training data. In general, this is problematic because we will never have enough data to ensure that we will see all possible events and so we don't want to rule out unseen events entirely. To allow for *sparse data* we have to use *smoothing*, which simply means that we make some assumption about the 'real' probability of unseen or very infrequently seen events and distribute that probability appropriately. A common approach is simply to add one to all counts: this is *add-one smoothing* which is not sound theoretically, but is simple to implement. A better approach in the case of bigrams is to *backoff* to the unigram probabilities: i.e., to distribute the unseen probability mass so that it is proportional to the unigram probabilities. This sort of estimation is extremely important to get good results from N-gram techniques, but we won't discuss the details in this course.

3.4 Part of speech tagging

Prediction techniques can be used for word classes, rather than just individual words. One important application is to part-of-speech tagging (POS tagging), where the words in a corpus are associated with a tag indicating some syntactic information that applies to that particular use of the word. For instance, consider the example sentence below:

> They can fish.

This has two readings: one (the most likely) about ability to fish and other about putting fish in cans. *fish* is ambiguous between a singular noun, plural noun and a verb, while *can* is ambiguous between singular noun, verb (the 'put in cans' use) and modal verb. However, *they* is unambiguously a pronoun. (I am ignoring some less likely possibilities, such as proper names.) These distinctions can be indicated by POS tags:

```
they PNP
can VM0 VVB VVI NN1
fish NN1 NN2 VVB VVI
```

There are several standard tagsets used in corpora and in POS tagging experiments. The one I'm using for the examples in this lecture is CLAWS 5 (C5) which is given in full in appendix C in J&M. The meaning of the tags above is:

```
NN1 singular noun
NN2 plural noun
PNP personal pronoun
VM0 modal auxiliary verb
VVB base form of verb (except infinitive)
VVI infinitive form of verb (i.e. occurs with 'to')
```

A POS tagger resolves the lexical ambiguities to give the most likely set of tags for the sentence. In this case, the right tagging is likely to be:

> They_PNP can_VM0 fish_VVB ._PUN

Note the tag for the full stop: punctuation is treated as unambiguous. POS tagging can be regarded as a form of very basic word sense disambiguation.

The other syntactically possible reading is:

> They_PNP can_VVB fish_NN2 ._PUN

However, POS taggers (unlike full parsers) don't attempt to produce globally coherent analyses. Thus a POS tagger might return:

> They_PNP can_VM0 fish_NN2 ._PUN

despite the fact that this doesn't correspond to a possible reading of the sentence.

POS tagging is useful as a way of annotating a corpus because it makes it easier to extract some types of information (for linguistic research or NLP experiments). It also acts as a basis for more complex forms of annotation. Named entity recognisers (discussed in lecture 4) are generally run on POS-tagged data. POS taggers are sometimes run as preprocessors to full parsing, since this can cut down the search space to be considered by the parser.

3.5 Stochastic POS tagging

One form of POS tagging applies the N-gram technique that we saw above, but in this case it applies to the POS tags rather than the individual words. The most common approaches depend on a small amount of manually tagged *training data* from which POS N-grams can be extracted.[12] I'll illustrate this with respect to another trivial corpus:

> They used to can fish in those towns. But now few people fish there.

This might be tagged as follows:

```
They_PNP used_VVD to_TO0 can_VVI fish_NN2 in_PRP those_DT0 towns_NN2 ._PUN
But_CJC now_AV0 few_DT0 people_NN2 fish_VVB in_PRP these_DT0 areas_NN2 ._PUN
```

This yields the following counts and probabilities:

```
sequence        count      bigram probability

AV0               1
AV0 DT0           1                1

CJC               1
CJC AV0           1                1

DT0               3
DT0 NN2           3                1

NN2               4
NN2 PRP           1                0.25
NN2 PUN           2                0.5
NN2 VVB           1                0.25

PNP               1
PNP VVD           1                1

PRP               1
PRP DT0           2                1

PUN               1
PUN CJC           1                1

TO0               1
```

[12] It is possible to build POS taggers that work without a hand-tagged corpus, but they don't perform as well as a system trained on even a 1,000 word corpus which can be tagged in a few hours. Furthermore, these algorithms still require a lexicon which associates possible tags with words.

```
TO0 VVI          1                1

VVB              1
VVB PRP          1                1

VVD              1
VVD TO0          1                1

VVI              1
VVI NN2          1                1
```

We can also obtain a lexicon from the tagged data:

```
word   tag  count

they   PNP    1
used   VVD    1
to     TO0    1
can    VVI    1
fish   NN2    1
       VVB    1
in     PRP    2
those  DT0    1
towns  NN2    1
.      PUN    1
but    CJC    1
now    AV0    1
few    DT0    1
people NN2    1
these  DT0    1
areas  NN2    1
```

The idea of stochastic POS tagging is that the tag can be assigned based on consideration of the lexical probability (how likely it is that the word has that tag), plus the sequence of prior tags. For a bigram model, we only look at a single previous tag. This is slightly more complicated than the word prediction case because we have to take into account both words and tags.

We are trying to estimate the probability of a sequence of tags given a sequence of words: $P(T|W)$. By Bayes theorem:

$$P(T|W) = \frac{P(T)P(W|T)}{P(W)}$$

Since we're looking at assigning tags to a particular sequence of words, $P(W)$ is constant, so for a relative measure of probability we can use:

$$P(T|W) = P(T)P(W|T)$$

We now have to estimate $P(T)$ and $P(W|T)$. If we make the bigram assumption, $P(T)$ is approximated by $P(t_i|t_{i-1})$ — i.e., the probability of some tag given the immediately preceding tag. We approximate $P(W|T)$ as $P(w_i|t_i)$. These values can be estimated from the corpus frequencies.

Note that we end up multiplying $P(t_i|t_{i-1})$ with $P(w_i|t_i)$ (the probability of the word given the tag) rather than $P(t_i|w_i)$ (the probability of the tag given the word). For instance, if we're trying to choose between the tags NN2 and VVB for *fish* in the sentence *they fish*, we calculate $P(\text{NN2}_i|\text{PNP}_{i-1})$, $P(\text{fish}_i|\text{NN2}_i)$, $P(\text{VVB}_i|\text{PNP}_{i-1})$ and $P(\text{fish}_i|\text{VVB}_i)$.

In fact, POS taggers generally use trigrams rather than bigrams — the relevant equations are given in J&M, page 306. As with word prediction, backoff and smoothing are crucial for reasonable performance.

When a POS tagger sees a word which was not in its training data, we need some way of assigning possible tags to the word. One approach is simply to use all possible *open class* tags, with probabilities based on the unigram probabilities

of those tags. Open class words are ones for which we can never give a complete list for a living language, since words are always being added: i.e., verbs, nouns, adjectives and adverbs. The rest are considered closed class. A better approach is to use a morphological analyser to restrict this set: e.g., words ending in *-ed* are likely to be VVD (simple past) or VVN (past participle), but can't be VVG (-ing form).

3.6 Evaluation of POS tagging

POS tagging algorithms are evaluated in terms of percentage of correct tags. The standard assumption is that every word should be tagged with exactly one tag, which is scored as correct or incorrect: there are no marks for near misses. Generally there are some words which can be tagged in only one way, so are automatically counted as correct. Punctuation is generally given an unambiguous tag. Therefore the success rates of over 95% which are generally quoted for POS tagging are a little misleading: the baseline of choosing the most common tag based on the training set often gives 90% accuracy. Some POS taggers returns multiple tags in cases where more than one tag has a similar probability.

It is worth noting that increasing the size of the tagset does not necessarily result in decreased performance: this depends on whether the tags that are added can generally be assigned unambiguously or not. Potentially, adding more fine-grained tags could increase performance. For instance, suppose we wanted to distinguish between present tense verbs according to whether they were 1st, 2nd or 3rd person. With the C5 tagset, and the stochastic tagger described, this would be impossible to do with high accuracy, because all pronouns are tagged PRP, hence they provide no discriminating power. On the other hand, if we tagged *I* and *we* as PRP1, *you* as PRP2 and so on, the N-gram approach would allow some discrimination. In general, predicting on the basis of classes means we have less of a sparse data problem than when predicting on the basis of words, but we also lose discriminating power. There is also something of a tradeoff between the utility of a set of tags and their usefulness in POS tagging. For instance, C5 assigns separate tags for the different forms of *be*, which is redundant for many purposes, but helps make distinctions between other tags in tagging models where the context is given by a tag sequence alone (i.e., rather than considering words prior to the current one).

POS tagging exemplifies some general issues in NLP evaluation:

Training data and test data The assumption in NLP is always that a system should work on novel data, therefore test data must be kept unseen.

For machine learning approaches, such as stochastic POS tagging, the usual technique is to spilt a data set into 90% training and 10% test data. Care needs to be taken that the test data is representative.

For an approach that relies on significant hand-coding, the test data should be literally unseen by the researchers. Development cycles involve looking at some initial data, developing the algorithm, testing on unseen data, revising the algorithm and testing on a new batch of data. The seen data is kept for regression testing.

Baselines Evaluation should be reported with respect to a baseline, which is normally what could be achieved with a very basic approach, given the same training data. For instance, the baseline for POS tagging with training data is to choose the most common tag for a particular word on the basis of the training data (and to simply choose the most frequent tag of all for unseen words).

Ceiling It is often useful to try and compute some sort of ceiling for the performance of an application. This is usually taken to be human performance on that task, where the ceiling is the percentage agreement found between two annotators (*interannotator agreement*). For POS tagging, this has been reported as 96% (which makes existing POS taggers look impressive). However this raises lots of questions: relatively untrained human annotators working independently often have quite low agreement, but trained annotators discussing results can achieve much higher performance (approaching 100% for POS tagging). Human performance varies considerably between individuals. In any case, human performance may not be a realistic ceiling on relatively unnatural tasks, such as POS tagging.

Error analysis The error rate on a particular problem will be distributed very unevenly. For instance, a POS tagger will never confuse the tag PUN with the tag VVN (past participle), but might confuse VVN with AJ0 (adjective) because there's a systematic ambiguity for many forms (e.g., *given*). For a particular application, some errors

may be more important than others. For instance, if one is looking for relatively low frequency cases of denominal verbs (that is verbs derived from nouns — e.g., *canoe*, *tango*, *fork* used as verbs), then POS tagging is not directly useful in general, because a verbal use without a characteristic affix is likely to be mistagged. This makes POS-tagging less useful for lexicographers, who are often specifically interested in finding examples of unusual word uses. Similarly, in text categorisation, some errors are more important than others: e.g. treating an incoming order for an expensive product as junk email is a much worse error than the converse.

Reproducibility If at all possible, evaluation should be done on a generally available corpus so that other researchers can replicate the experiments.

3.7 Further reading

This lecture has skimmed over material that is covered in several chapters of J&M. See 5.9 for the Viterbi algorithm, Chapter 6 for N-grams (especially 6.3, 6.4 and 6.7), 7.1-7.3 for speech recognition and Chapter 8 on POS tagging.

4 Lecture 4: Parsing and generation

In this lecture, we'll discuss syntax in a way which is much closer to the standard notions in formal linguistics than POS-tagging is. To start with, we'll briefly motivate the idea of a generative grammar in linguistics, review the notion of a context-free grammar and then show a context-free grammar for a tiny fragment of English. We'll then show how context free grammars can be used to implement generators and parsers, and discuss chart parsing, which allows efficient processing of strings containing a high degree of ambiguity. Finally we'll briefly touch on probabilistic context-free approaches.

4.1 Generative grammar

Since Chomsky's work in the 1950s, much work in formal linguistics has been concerned with the notion of a *generative grammar* — i.e., a formally specified grammar that can generate all and only the acceptable sentences of a natural language. It's important to realise that nobody has actually written such a grammar for any natural language or even come close to doing so: what most linguists are really interested in is the principles that underly such grammars, especially to the extent that they apply to all natural languages. NLP researchers, on the other hand, are at least sometimes interested in actually building and using large-scale detailed grammars.

The formalisms which are of interest to us for modelling syntax assign internal structure to the strings of a language, which can be represented by bracketing. We already saw some evidence of this in derivational morphology (the unionised example), but here we are concerned with the structure of phrases. For instance, the sentence:

> the dog slept

can be bracketed

> ((the (big dog)) slept)

The phrase, *big dog*, is an example of a *constituent* (i.e. something that is enclosed in a pair of brackets): *the big dog* is also a constituent, but *the big* is not. Constituent structure is generally justified by arguments about substitution which I won't go into here: J&M discuss this briefly, but see an introductory syntax book for a full discussion. In this course, I will simply give bracketed structures and hope that the constituents make sense intuitively, rather than trying to justify them.

Two grammars are said to be *weakly-equivalent* if they generate the same strings. Two grammars are *strongly-equivalent* if they assign the same bracketings to all strings they generate.

In most, but not all, approaches, the internal structures are given labels. For instance, *the big dog* is a *noun phrase* (abbreviated NP), *slept*, *slept in the park* and *licked Sandy* are *verb phrase*s (VPs). The labels such as NP and VP correspond to non-terminal symbols in a grammar. In this lecture, we'll discuss the use of simple context-free grammars for language description, moving onto a more expressive formalism in lecture 5.

4.2 Context free grammars

The idea of a context-free grammar (CFG) should be familiar from formal language theory. A CFG has four components, described here as they apply to grammars of natural languages:

1. a set of non-terminal symbols (e.g., S, VP), conventionally written in uppercase;
2. a set of terminal symbols (i.e., the words), conventionally written in lowercase;
3. a set of rules (productions), where the left hand side (the mother) is a single non-terminal and the right hand side is a sequence of one or more non-terminal or terminal symbols (the daughters);
4. a start symbol, conventionally S, which is a member of the set of non-terminal symbols.

The formal description of a CFG generally allows productions with an empty righthandside (e.g., Det $\rightarrow \varepsilon$). It is convenient to exclude these however, since they complicate parsing algorithms, and a weakly-equivalent grammar can always be constructed that disallows such *empty productions*.

A grammar in which all nonterminal daughters are the leftmost daughter in a rule (i.e., where all rules are of the form $X \rightarrow Ya*$), is said to be *left-associative*. A grammar where all the nonterminals are rightmost is *right-associative*. Such grammars are weakly-equivalent to regular grammars (i.e., grammars that can be implemented by FSAs), but natural languages seem to require more expressive power than this (see §4.12).

4.3 A simple CFG for a fragment of English

The following tiny fragment is intended to illustrate some of the properties of CFGs so that we can discuss parsing and generation. It has some serious deficiencies as a representation of even this fragment, which we'll ignore for now, though we'll discuss some of them in lecture 5.

```
S -> NP VP
VP -> VP PP
VP -> V
VP -> V NP
VP -> V VP
NP -> NP PP
PP -> P NP
;;; lexicon
V -> can
V -> fish
NP -> fish
NP -> rivers
NP -> pools
NP -> December
NP -> Scotland
NP -> it
NP -> they
P -> in
```

The rules with terminal symbols on the RHS correspond to the lexicon. Here and below, comments are preceded by `;;;`

Here are some strings which this grammar generates, along with their bracketings:

they fish
(S (NP they) (VP (V fish)))

they can fish
(S (NP they) (VP (V can) (VP (V fish))))
;;; the modal verb 'are able to' reading
(S (NP they) (VP (V can) (NP fish)))
;;; the less plausible, put fish in cans, reading

they fish in rivers
(S (NP they) (VP (VP (V fish)) (PP (P in) (NP rivers))))

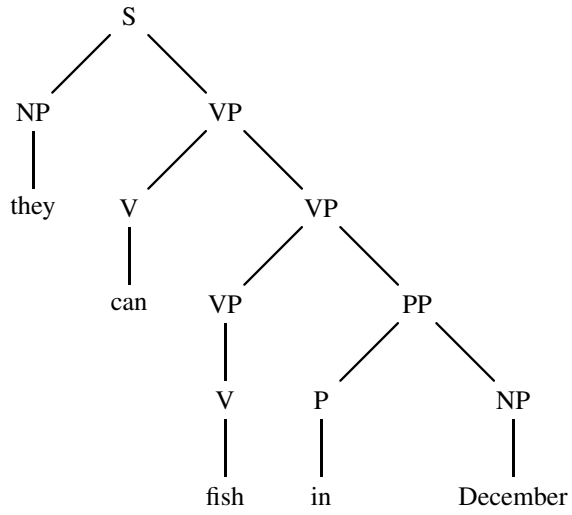
they fish in rivers in December
(S (NP they) (VP (VP (V fish)) (PP (P in) (NP (NP rivers) (PP (P in) (NP December))))))
;;; i.e. the implausible reading where the rivers are in December
;;; (cf rivers in Scotland)
(S (NP they) (VP (VP (VP (V fish)) (PP (P in) (NP (NP rivers)))) (PP (P in) (NP December))))
;;; i.e. the fishing is done in December

One important thing to notice about these examples is that there's lots of potential for ambiguity. In the *they can fish* example, this is due to *lexical ambiguity* (it arises from the dual lexical entries of *can* and *fish*), but the last example demonstrates purely *structural ambiguity*. In this case, the ambiguity arises from the two possible *attachments* of the prepositional phrase (PP) *in December*: it can attach to the NP (*rivers*) or to the VP. These attachments correspond to different semantics, as indicated by the glosses. PP attachment ambiguities are a major headache in parsing, since sequences of four or more PPs are common in real texts and the number of readings increases as the Catalan series, which is exponential. Other phenomena have similar properties: for instance, compound nouns (e.g. *long-stay car park shuttle bus*).

Notice that *fish* could have been entered in the lexicon directly as a VP, but that this would cause problems if we were doing derivational morphology, because we want to say that suffixes like *-ed* apply to Vs. Making *rivers* etc NPs rather than nouns is a simplification I've adopted here to keep the grammar smaller.

4.4 Parse trees

Parse trees are equivalent to bracketed structures, but are easier to read for complex cases. A parse tree and bracketed structure for one reading of *they can fish in December* is shown below. The correspondence should be obvious.



(S (NP they) (VP (V can) (VP (VP (V fish)) (PP (P in) (NP December)))))

4.5 Using a grammar as a random generator

The following simple algorithm illustrates how a grammar can be used to generate sentences.

Expand cat *category sentence-record*:
Let *possibilities* be a set containing all lexical items which match *category* and all rules with left-hand side *category*
If *possibilities* is empty,
then fail
else
Randomly select a possibility *chosen* from *possibilities*
If *chosen* is lexical,
then append it to *sentence-record*
else **expand cat** on each rhs category in *chosen* (left to right) with the updated *sentence-record*
return *sentence-record*

For instance:

Expand cat S ()

possibilities = `S -> NP VP`
chosen = `S -> NP VP`

Expand cat NP ()
possibilities = it, they, fish
chosen = fish
sentence-record = (fish)
Expand cat VP (fish)
possibilities = `VP -> V, VP -> V VP, VP -> V NP`
chosen = `VP -> V`

Expand cat V (fish)
possibilities = fish, can
chosen = fish
sentence-record = (fish fish)

Obviously, the strings generated could be arbitrarily long. If in this naive generation algorithm, we explored all the search space rather than randomly selecting a possible expansion, the algorithm wouldn't terminate.

Real generation operates from semantic representations, which aren't encoded in this grammar, so in what follows we'll concentrate on describing parsing algorithms instead. However, it's important to realise that CFGs are, in principle, bidirectional.

4.6 Chart parsing

In order to parse with reasonable efficiency, we need to keep a record of the rules that we have applied so that we don't have to backtrack and redo work that we've done before. This works for parsing with CFGs because the rules are independent of their context: a VP can always expand as a V and an NP regardless of whether or not it was preceded by an NP or a V, for instance. (In some cases we may be able to apply techniques that look at the context to cut down the search space, because we can tell that a particular rule application is never going to be part of a sentence, but this is strictly a filter: we're never going to get incorrect results by reusing partial structures.) This record keeping strategy is an application of dynamic programming which is used in processing formal languages too. In NLP the data structure used for recording partial results is generally known as a *chart* and algorithms for parsing using such structures are referred to as *chart parsers*.

A chart is a list of *edges*. In the simplest version of chart parsing, each edge records a rule application and has the following structure:

[*id*,*left_vertex*, *right_vertex*,*mother_category*, *daughters*]

A vertex is an integer representing a point in the input string, as illustrated below:

```
. they . can . fish .
0      1     2      3
```

mother_category refers to the rule that has been applied to create the edge. *daughters* is a list of the edges that acted as the daughters for this particular rule application: it is there purely for record keeping so that the output of parsing can be a labelled bracketing.

For instance, the following edges would be among those found on the chart after a complete parse of *they can fish* according to the grammar given above (id numbering is arbitrary):

```
id  left   right  mother   daughters
3    1       2      V        (can)
4    2       3      NP       (fish)
5    2       3      V        (fish)
6    2       3      VP       (5)
7    1       3      VP       (3 5)
8    1       3      VP       (3 4)
```

The daughters for the terminal rule applications are simply the input word strings. Note that local ambiguities correspond to situations where a particular span has more than one associated edge. We'll see below that we can *pack* structures so that we never have two edges with the same category and the same span, but we'll ignore this for the moment (see §4.9). Also, in this chart we're only recording complete rule applications: this is *passive* chart parsing. The more efficient *active* chart is discussed below, in §4.10.

4.7 A bottom-up passive chart parser

The following pseudo-code sketch is for a very simple chart parser. The main function is **Add new edge** which is called for each word in the input going left to right. **Add new edge** recursively scans backwards looking for other daughters.

Parse:
  Initialise the chart (i.e., clear previous results)
  For each word *word* in the input sentence, let *from* be the left vertex, *to* be the right vertex and *daughters* be (*word*)
    For each category *category* that is lexically associated with *word*
      Add new edge *from*, *to*, *category*, *daughters*
  Output results for all spanning edges
  (i.e., ones that cover the entire input and which have a mother corresponding to the root category)

Add new edge *from*, *to*, *category*, *daughters*:
  Put edge in chart: [*id*,*from*,*to*, *category*,*daughters*]
  For each *rule* in the grammar of form *lhs* –> $cat_1 \ldots cat_{n-1}$,*category*
    Find set of lists of contiguous edges [id_1,$from_1$,to_1, cat_1,$daughters_1$] … [id_{n-1},$from_{n-1}$,*from*, cat_{n-1},$daughters_{n-1}$]
    (such that $to_1 = from_2$ etc)
    For each list of edges, **Add new edge** $from_1$, *to*, *lhs*, ($id_1 \ldots id$)

Notice that this means that the grammar rules are indexed by their rightmost category, and that the edges in the chart must be indexed by their *to* vertex (because we scan backward from the rightmost category). Consider:

```
. they . can . fish .
0      1     2      3
```

The following diagram shows the chart edges as they are constructed in order (when there is a choice, taking rules in a priority order according to the order they appear in the grammar):

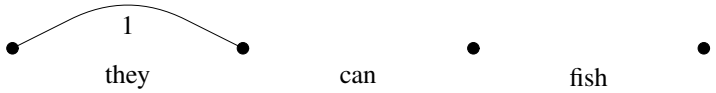
id	left	right	mother	daughters
1	0	1	NP	(they)
2	1	2	V	(can)
3	1	2	VP	(2)
4	0	2	S	(1 3)
5	2	3	V	(fish)
6	2	3	VP	(5)
7	1	3	VP	(2 6)
8	0	3	S	(1 7)
9	2	3	NP	(fish)
10	1	3	VP	(2 9)
11	0	3	S	(1 10)

The spanning edges are 11 and 8: the output routine to give bracketed parses simply outputs a left bracket, outputs the category, recurses through each of the daughters and then outputs a right bracket. So, for instance, the output from edge 11 is:

```
(S (NP they) (VP (V can) (NP fish)))
```

4.8 A detailed trace of the simple chart parser

Parse
word = they
categories = NP
Add new edge 0, 1, NP, (they)

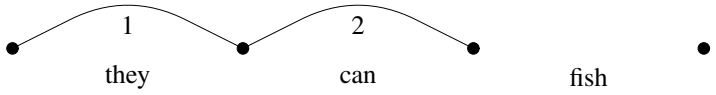


Matching grammar rules are:

```
VP -> V NP
PP -> P NP
```

No matching edges corresponding to V or P

word = can
categories = V
Add new edge 1, 2, V, (can)

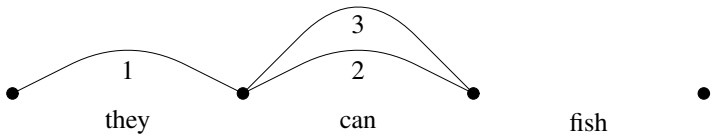


Matching grammar rules are:

```
VP -> V
```

set of edge lists = $\{(2)\}$

Add new edge 1, 2, VP, (2)

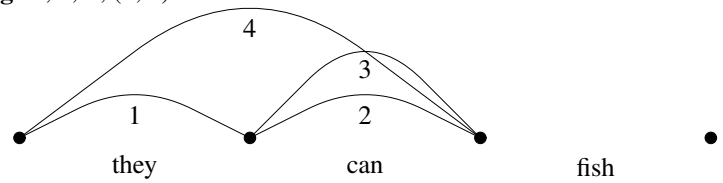


Matching grammar rules are:

```
S -> NP VP
VP -> V VP
```

set of edge lists corresponding to NP VP = $\{(1, 3)\}$

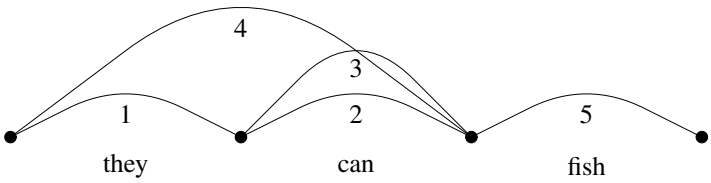
Add new edge 0, 2, S, (1, 3)



No matching grammar rules for S

No edges matching V VP

word = fish
categories = V, NP
Add new edge 2, 3, V, (fish)

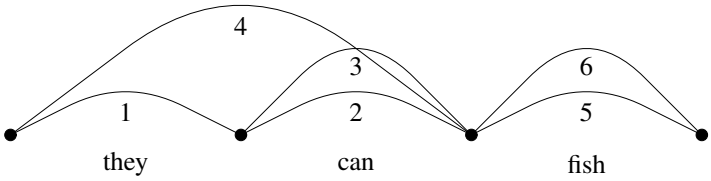


Matching grammar rules are:

```
VP -> V
```

set of edge lists = $\{(5)\}$

Add new edge 2, 3, VP, (5)

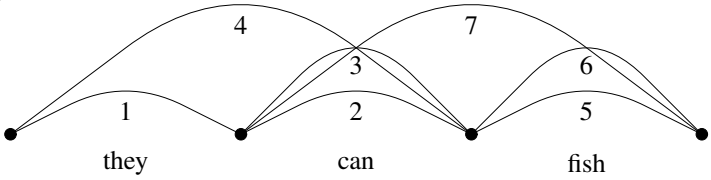


Matching grammar rules are:

```
S -> NP VP
VP -> V VP
```

No edges match NP
set of edge lists for V VP = $\{(2, 6)\}$

Add new edge 1, 3, VP, (2, 6)

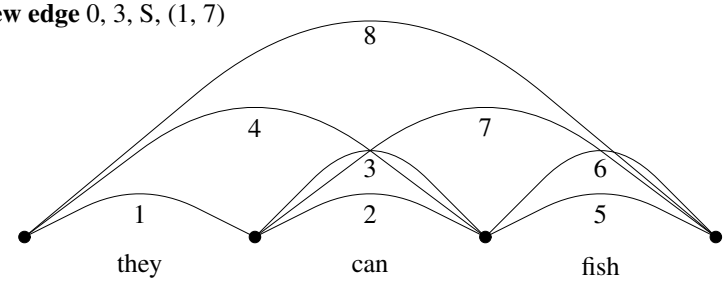


Matching grammar rules are:

```
S -> NP VP
VP -> V VP
```

set of edge lists for NP VP = $\{(1, 7)\}$

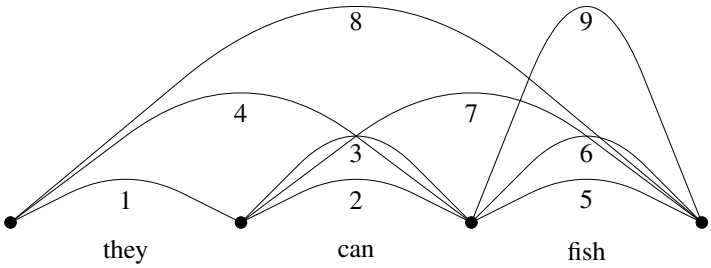
Add new edge 0, 3, S, (1, 7)



No matching grammar rules for S

No edges matching V

Add new edge 2, 3, NP, (fish)

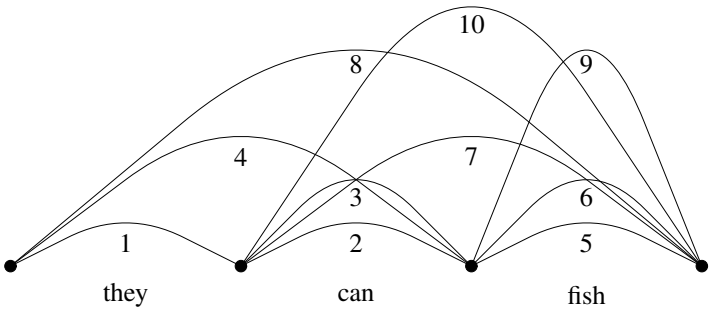


Matching grammar rules are:

```
VP -> V NP
PP -> P NP
```

set of edge lists corresponding to V NP $= \{(2, 9)\}$

Add new edge 1, 3, VP, (2, 9)

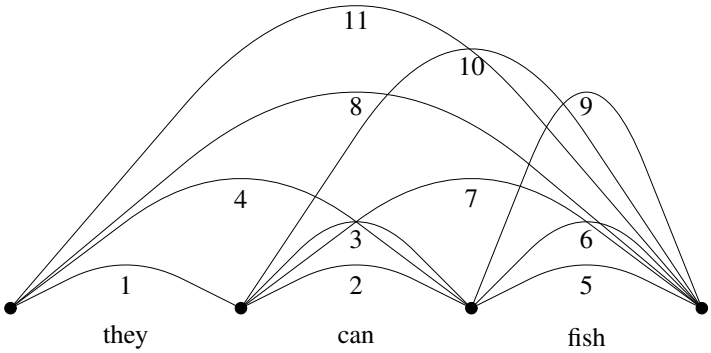


Matching grammar rules are:

```
S -> NP VP
VP -> V VP
```

set of edge lists corresponding to NP VP $= \{(1, 10)\}$

Add new edge 0, 3, S, (1, 10)



No matching grammar rules for S

No edges corresponding to V VP

No edges corresponding to P NP

No further words in input

Spanning edges are 8 and 11: Output results for 8

```
(S (NP they) (VP (V can) (VP (V fish))))
```

Output results for 11

```
(S (NP they) (VP (V can) (NP fish)))
```

4.9 Packing

The algorithm given above is exponential in the case where there are an exponential number of parses. The body of the algorithm can be modified so that it runs in cubic time, though producing the output is still exponential. The modification is simply to change the daughters value on an edge to be a set of lists of daughters and to make an equality check before adding an edge so we don't add one that's equivalent to an existing one. That is, if we are about to add an edge:

[*id*,*left_vertex*, *right_vertex*,*mother_category*, *daughters*]

and there is an existing edge:

[*id-old*,*left_vertex*, *right_vertex*,*mother_category*, *daughters-old*]

we simply modify the old edge to record the new daughters:

[*id-old*,*left_vertex*, *right_vertex*,*mother_category*, *daughters-old* $\sqcup$ *daughters*]

There is no need to recurse with this edge, because we couldn't get any new results.
For the example above, everything proceeds as before up to edge 9:

```
id  left   right  mother   daughters
1    0       1      NP       {(they)}
2    1       2      V        {(can)}
3    1       2      VP       {(2)}
4    0       2      S        {(1 3)}
5    2       3      V        {(fish)}
6    2       3      VP       {(5)}
7    1       3      VP       {(2 6)}
8    0       3      S        {(1 7)}
9    2       3      NP       {(fish)}
```

However, rather than add edge 10, which would be:

```
10   1       3      VP       (2 9)
```

we match this with edge 7, and simply add the new daughters to that.

```
7    1       3      VP       {(2 6), (2 9)}
```

The algorithm then terminates. We only have one spanning edge (edge 8) but the display routine is more complex because we have to consider the alternative sets of daughters for edge 7. (You should go through this to convince yourself that the same results are obtained as before.) Although in this case, the amount of processing saved is small, the effects are much more important with longer sentences (consider *he believes they can fish*, for instance).

4.10 Active chart parsing

A more minor efficiency improvement is obtained by storing the results of partial rule applications. This is *active* chart parsing, so called because the partial edges are considered to be active: i.e. they 'want' more input to make them complete. An active edge records the input it expects as well as the daughters it has already seen. For instance, with an active chart parser, we might have the following edges after seeing *they*:

```
id  left   right  mother    expected      daughters
1    0       1      NP                   {(they)}
2    0       1      S         VP         {(1 ?)}
```

The daughter marked as ? will be instantiated by the edge corresponding to the VP when it is found.

4.11 Ordering the search space

In the pseudo-code above, the order of addition of edges to the chart was determined by the recursion. In general, chart parsers make use of an *agenda* of edges, so that the next edges to be operated on are the ones that are first on the agenda. Different parsing algorithms can be implemented by making this agenda a stack or a queue, for instance.

So far, we've considered *bottom up* parsing: an alternative is *top down* parsing, where the initial edges are given by the rules whose mother corresponds to the start symbol.

Some efficiency improvements can be obtained by ordering the search space appropriately, though which version is most efficient depends on properties of the individual grammar. However, the most important reason to use an explicit agenda is when we are returning parses in some sort of priority order, corresponding to weights on different grammar rules or lexical entries.

Weights can be manually assigned to rules and lexical entries in a manually constructed grammar. However, in the last decade, a lot of work has been done on automatically acquiring probabilities from a corpus annotated with trees (a *treebank*), either as part of a general process of automatic grammar acquisition, or as automatically acquired additions to a manually constructed grammar. Probabilistic CFGs (PCFGs) can be defined quite straightforwardly, if the assumption is made that the probabilities of rules and lexical entries are independent of one another (of course this assumption is not correct, but the orderings given seem to work quite well in practice). The importance of this is that we rarely want to return all parses in a real application, but instead we want to return those which are top-ranked: i.e., the most likely parses. This is especially true when we consider that realistic grammars can easily return many thousands of parses for sentences of quite moderate length (20 words or so). If edges are prioritised by probability, very low priority edges can be completely excluded from consideration if there is a cut-off such that we can be reasonably certain that no edges with a lower priority than the cut-off will contribute to the highest-ranked parse. Limiting the number of analyses under consideration is known as *beam search* (the analogy is that we're looking within a beam of light, corresponding to the highest probability edges). Beam search is linear rather than exponential or cubic. Just as importantly, a good priority ordering from a parser reduces the amount of work that has to be done to filter the results by whatever system is processing the parser's output.

4.12 Why can't we use FSAs to model the syntax of natural languages?

In this lecture, we started using CFGs. This raises the question of why we need this more expressive (and hence computationally expensive) formalism, rather than modelling syntax with FSAs. One reason is that the syntax of natural languages cannot be described by an FSA, even in principle, due to the presence of *centre-embedding*, i.e. structures which map to:

$$A \rightarrow \alpha A \beta$$

and which generate grammars of the form $a^n b^n$. For instance:

the students the police arrested complained

has a centre-embedded structure. However, humans have difficulty processing more than two levels of embedding:

? the students the police the journalists criticised arrested complained

If the recursion is finite (no matter how deep), then the strings of the language can be generated by an FSA. So it's not entirely clear whether formally an FSA might not suffice.

There's a fairly extensive discussion of these issues in J&M , but there are two essential points for our purposes:

1. Grammars written using finite state techniques alone are very highly redundant, which makes them very difficult to build and maintain.

2. Without internal structure, we can't build up good semantic representations.

Hence the use of more powerful formalisms: in the next lecture, we'll discuss the inadequacies of simple CFGs from a similar perspective.

However, FSAs are very useful for partial grammars which don't require full recursion. In particular, for information extraction, we need to recognise *named entities*: e.g. Professor Smith, IBM, 101 Dalmatians, the White House, the Alps and so on. Although NPs are in general recursive (*the man who likes the dog which bites postmen*), relative clauses are not generally part of named entities. Also the internal structure of the names is unimportant for IE. Hence FSAs can be used, with sequences such as 'title surname', 'DT0 PNP' etc

CFGs can be automatically compiled into approximately equivalent FSAs by putting bounds on the recursion. This is particularly important in speech recognition engines.

4.13 Further reading

This lecture has covered material which J&M discuss in chapters 9 and 10, though we also touched on PCFGs (covered in their chapter 12) and issues of language complexity which they discuss in chapter 13. J&M's discussion covers the Earley algorithm, which can be thought of as a form of active top-down chart parsing. I chose to concentrate on bottom-up parsing in this lecture, mainly because I find it easier to describe, but also because it is easier to see how to extend this to PCFGs. Bottom-up parsing also seems to have better practical performance with the sort of grammars we'll look at in lecture 5.

There are a large number of introductory linguistics textbooks which cover elementary syntax and discuss concepts such as constituency. For instance, students could usefully look at the first five chapters of Tallerman (1998):

Tallerman, Maggie, *Understanding Syntax*, Arnold, London, 1998

An alternative would be the first two chapters of Sag and Wasow (1999) — copies should be in the Computer Laboratory library. This has a narrower focus than most other syntax books, but covers a much more detailed grammar fragment. The later chapters (particularly 3 and 4) are relevant for lecture 5.

Sag, Ivan A. and Thomas Wasow, *Syntactic Theory — a formal introduction*, CSLI Publications, Stanford, CA, USA, 1999

5 Lecture 5: Parsing with constraint-based grammars

The CFG approach which we've looked at so far has some serious deficiencies as a model of natural language. In this lecture, I'll discuss some of these and give an introduction to a more expressive formalism which is widely used in NLP, again with the help of a sample grammar. In the first part of the next lecture, I will also sketch how we can use this approach to do compositional semantics.

5.1 Deficiencies in atomic category CFGs

If we consider the grammar we saw in the last lecture, several problems are apparent. One is that there is no account of agreement, so, for instance, **it fish* is allowed by the grammar as well as *they fish*.[13]

We could, of course, allow for agreement by increasing the number of atomic symbols in the CFG, introducing NP-sg, NP-pl, VP-sg and VP-pl, for instance. But this approach would soon become very tedious:

```
S -> NP-sg VP-sg
S -> NP-pl VP-pl
VP-sg -> V-sg NP-sg
VP-sg -> V-sg NP-pl
VP-pl -> V-pl NP-sg
VP-pl -> V-pl NP-pl
```

Note that we have to expand out the symbols even when there's no constraint on agreement, since we have no way of saying that we don't care about the value of number for a category.

Another linguistic phenomenon that we are failing to deal with is *subcategorization*. This is the lexical property that tells us how many *argument*s a verb can have (among other things). A verb such as *adore*, for instance, is transitive: a sentence such as **Kim adored* is strange, while *Kim adored Sandy* is usual. A verb such as *give* is *ditransitive*: *Kim gave Sandy an apple* (or *Kim gave an apple to Sandy*). Without going into details of exactly how subcategorization is defined, or what an argument is, it should be intuitively obvious that we're not encoding this property with our CFG. The grammar allows the following, for instance:

> *they fish fish it*
> (S (NP they) (VP (V fish) (VP (V fish) (NP it))))

Again this could be dealt with by multiplying out symbols (V-intrans, V-ditrans etc), but the grammar becomes extremely cumbersome.

Finally, consider the phenomenon of *long-distance dependencies*, exemplified, for instance, by:

> which problem did you say you don't understand?
> who do you think Kim asked Sandy to hit?
> which kids did you say were making all that noise?

Traditionally, these sentences are said to contain 'gap's, corresponding to the place where the noun phrase would normally appear: the gaps are marked by underscores below:

> which problem did you say you don't understand _?
> who do you think Kim asked Sandy to hit?
> which kids did you say _ were making all that noise?

Notice that, in the third example, the verb *were* shows plural agreement.

Doing this in standard CFGs is possible, but extremely verbose, potentially leading to millions of rules. Instead of having simple atomic categories in the CFG, we want to allow for features on the categories, which can have values indicating things like plurality. As the long-distance dependency examples should indicate, the features need to be complex-valued. For instance,

[13] There was also no account of *case*: this is only reflected in a few places in modern English, but **they can they* is clearly ungrammatical (as opposed to *they can them*, which is grammatical with the transitive verb use of *can*).

* what kid did you say _ were making all that noise?

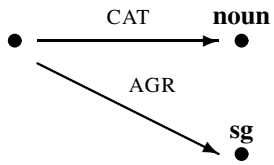
is not grammatical. The analysis needs to be able to represent the information that the gap corresponds to a plural noun phrase.

In what follows, I will illustrate a simple *constraint-based grammar* formalism, using *feature structures*. A constraint-based grammar describes a language using a set of independently stated constraints, without imposing any conditions on processing or processing order. A CFG can be taken as an example of a constraint-based grammar, but usually the term is reserved for richer formalisms. The simplest way to think of feature structures (FSs) is that we're replacing the atomic categories of a CFG with more complex data structures. I'll first illustrate this idea intuitively, using a grammar fragment like the one in lecture 4 but enforcing agreement. I'll then go through the feature structure formalism in more detail. This is followed by an example of a more complex grammar, which allows for subcategorization (I won't show how case and long-distance dependencies are dealt with).

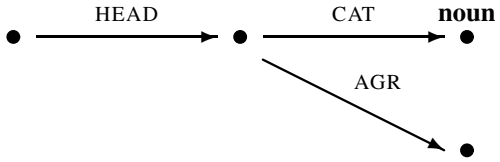
5.2 A very simple FS grammar encoding agreement

In a FS grammar, rules are described as relating FSs: i.e., lexical entries and phrases are FSs. In these formalisms, the term *sign* is often used to refer to lexical entries and phrases collectively. In fact, rules themselves can be treated as FSs. Feature structures are singly-rooted directed acyclic graphs, with arcs labelled by features and terminal nodes associated with values. A particular feature in a structure may be *atomic-valued*, meaning it points to a terminal node in the graph, or *complex-valued*, meaning it points to a non-terminal node. A sequence of features is known as a *path*.

For instance, in the structure below, there are two arcs, labelled with CAT and AGR, and three nodes, with the two terminal nodes having values **noun** and **sg**. Each of the features is thus atomic-valued.



In the graph below, the feature HEAD is complex-valued, and the value of AGR (i.e., the value of the path HEAD AGR) is unspecified:

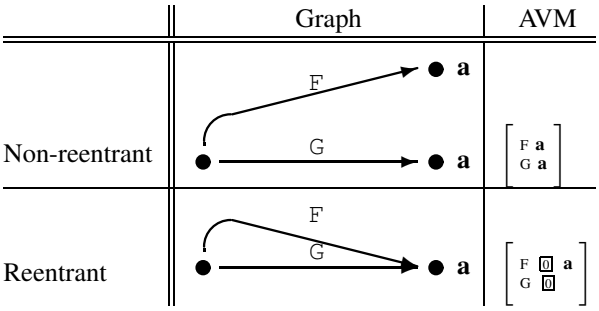


FSs are usually drawn as *attribute-value matrices* or AVMs. The AVMs corresponding to the two FSs above are as follows:

$$\begin{bmatrix} \text{CAT} & \textbf{noun} \\ \text{AGR} & \textbf{sg} \end{bmatrix}$$

$$\begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{CAT} & \textbf{noun} \\ \text{AGR} & [\ ] \end{bmatrix} \end{bmatrix}$$

Since FSs are graphs, rather than trees, a particular node may be accessed from the root by more than one path: this is known as *reentrancy*. In AVMs, reentrancy is conventionally indicated by boxed integers, with node identity indicated by integer identity. The actual integers used are arbitrary. This is illustrated with an abstract example using features F and G below:



When using FSs in grammars, structures are combined by *unification*. This means that all the information in the two structures is combined. The empty square brackets ([ ]) in an AVM indicate that a value is unspecified: i.e. this is a node which can be unified with a terminal node (i.e., an atomic value) or a complex value. More details of unification are given below.

When FSs are used in a particular grammar, all signs will have a similar set of features (although sometimes there are differences between lexical and phrasal signs). Feature structure grammars can be used to implement a variety of linguistic frameworks. For the first example of a FS grammar, we'll just consider how agreement could be encoded.

Suppose we are trying to model a grammar which is weakly equivalent to the CFG fragment below:

```
S -> NP-sg VP-sg
S -> NP-pl VP-pl
VP-sg -> V-sg NP-sg
VP-sg -> V-sg NP-pl
VP-pl -> V-pl NP-sg
VP-pl -> V-pl NP-pl
V-pl -> like
V-sg -> likes
NP-sg -> it
NP-pl -> they
NP-sg -> fish
NP-pl -> fish
```

The FS equivalent shown below splits up the categories so that the main category and the agreement values are distinct. In the grammar below, I have used the arrow notation for rules as an abbreviation: I will describe the actual FS encoding of rules shortly. The FS grammar just needs two rules. There is a single rule corresponding to the `S-> NP VP` rule, which enforces identity of agreement values between the NP and the VP by means of reentrancy (indicated by the tag [1]). The rule corresponding to `VP-> V NP` simply makes the agreement values of the V and the VP the same but ignores the agreement value on the NP.[14] The lexicon specifies agreement values for *it*, *they*, *like* and *likes*, but leaves the agreement value for *fish* uninstantiated (i.e., underspecified). Note that the grammar also has a root FS: a structure only counts as a valid parse if it is unifiable with the root.

FS grammar fragment encoding agreement

Grammar rules

Rule1 $\begin{bmatrix} \text{CAT } \mathbf{S} \\ \text{AGR } \boxed{1} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT } \mathbf{NP} \\ \text{AGR } \boxed{1} \end{bmatrix}, \begin{bmatrix} \text{CAT } \mathbf{VP} \\ \text{AGR } \boxed{1} \end{bmatrix}$

Rule2 $\begin{bmatrix} \text{CAT } \mathbf{VP} \\ \text{AGR } \boxed{1} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT } \mathbf{V} \\ \text{AGR } \boxed{1} \end{bmatrix}, \begin{bmatrix} \text{CAT } \mathbf{NP} \\ \text{AGR } [\ ] \end{bmatrix}$

Lexicon:

;;; noun phrases

they $\begin{bmatrix} \text{CAT } \mathbf{noun} \\ \text{AGR } \mathbf{pl} \end{bmatrix}$

[14] Note that the reentrancy indicators are local to each rule: the [1] in rule 1 is not the same structure as the [1] in rule 2.

fish $\left[\left[\begin{array}{l} \text{CAT } \mathbf{noun} \\ \text{AGR } [\ ] \end{array} \right] \right]$

it $\left[\begin{array}{l} \text{CAT } \mathbf{noun} \\ \text{AGR } \mathbf{sg} \end{array} \right]$

;;; verbs

like $\left[\begin{array}{l} \text{CAT } \mathbf{verb} \\ \text{AGR } \mathbf{pl} \end{array} \right]$

likes $\left[\begin{array}{l} \text{CAT } \mathbf{verb} \\ \text{AGR } \mathbf{sg} \end{array} \right]$

Root structure:

$\left[\text{CAT } \mathbf{S} \right]$

Consider parsing *they like it* with this grammar. The lexical structures for *like* and *it* are unified with the corresponding structure to the right hand side of rule 2. Both unifications succeed, and the structure corresponding to the mother of the rule is:

$$\left[\begin{array}{l} \text{CAT } \mathbf{VP} \\ \text{AGR } \mathbf{pl} \end{array} \right]$$

The agreement value is **pl** because of the coindexation with the agreement value of *like*. This structure can unify with the rightmost daughter of rule 1. The structure for *they* is unified with the leftmost daughter. Rule 1 says that both daughters have to have the same agreement value, which is the case in this example. Rule application therefore succeeds and since the result unifies with the rule structure, there is a valid parse.

To see what is going on a bit more precisely, we need to show the rules as FSs. There are several ways of encoding this, but for current purposes I will assume that rules have features MOTHER, DTR1, DTR2 ... DTRN. So Rule2, which I informally wrote as:

$$\left[\begin{array}{l} \text{CAT } \mathbf{VP} \\ \text{AGR } \boxed{1} \end{array} \right] \rightarrow \left[\begin{array}{l} \text{CAT } \mathbf{V} \\ \text{AGR } \boxed{1} \end{array} \right], \left[\begin{array}{l} \text{CAT } \mathbf{NP} \\ \text{AGR } [\ ] \end{array} \right]$$

is actually:

$$\left[\begin{array}{ll} \text{MOTHER} & \left[\begin{array}{l} \text{CAT } \mathbf{VP} \\ \text{AGR } \boxed{1} \end{array} \right] \\ \text{DTR1} & \left[\begin{array}{l} \text{CAT } \mathbf{V} \\ \text{AGR } \boxed{1} \end{array} \right] \\ \text{DTR2} & \left[\begin{array}{l} \text{CAT } \mathbf{NP} \\ \text{AGR } [\ ] \end{array} \right] \end{array} \right]$$

The structure for *like* can be unified with the value of DTR1 in the rule. Unification means all information is retained, so the result includes the agreement value from *like*:

$$\left[\begin{array}{ll} \text{MOTHER} & \left[\begin{array}{l} \text{CAT } \mathbf{VP} \\ \text{AGR } \boxed{1}\ \mathbf{pl} \end{array} \right] \\ \text{DTR1} & \left[\begin{array}{l} \text{CAT } \mathbf{V} \\ \text{AGR } \boxed{1} \end{array} \right] \\ \text{DTR2} & \left[\begin{array}{l} \text{CAT } \mathbf{NP} \\ \text{AGR } [\ ] \end{array} \right] \end{array} \right]$$

The structure for *it* is unified with the value for DTR2:

$$\begin{bmatrix} \text{MOTHER} & \begin{bmatrix} \text{CAT} & \textbf{VP} \\ \text{AGR} & \boxed{1}\ \textbf{pl} \end{bmatrix} \\ \text{DTR1} & \begin{bmatrix} \text{CAT} & \textbf{V} \\ \text{AGR} & \boxed{1} \end{bmatrix} \\ \text{DTR2} & \begin{bmatrix} \text{CAT} & \textbf{NP} \\ \text{AGR} & \textbf{sg} \end{bmatrix} \end{bmatrix}$$

The rule application thus succeeds. The MOTHER value acts as the DTR2 of Rule 1. That is:

$$\begin{bmatrix} \text{CAT} & \textbf{VP} \\ \text{AGR} & \textbf{pl} \end{bmatrix}$$

is unified with the DTR2 value of:

$$\begin{bmatrix} \text{MOTHER} & \begin{bmatrix} \text{CAT} & \textbf{S} \\ \text{AGR} & \boxed{1} \end{bmatrix} \\ \text{DTR1} & \begin{bmatrix} \text{CAT} & \textbf{NP} \\ \text{AGR} & \boxed{1} \end{bmatrix} \\ \text{DTR2} & \begin{bmatrix} \text{CAT} & \textbf{VP} \\ \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

This gives:

$$\begin{bmatrix} \text{MOTHER} & \begin{bmatrix} \text{CAT} & \textbf{S} \\ \text{AGR} & \boxed{1}\ \textbf{pl} \end{bmatrix} \\ \text{DTR1} & \begin{bmatrix} \text{CAT} & \textbf{NP} \\ \text{AGR} & \boxed{1} \end{bmatrix} \\ \text{DTR2} & \begin{bmatrix} \text{CAT} & \textbf{VP} \\ \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

The FS for *they* is:

$$\begin{bmatrix} \text{CAT} & \textbf{NP} \\ \text{AGR} & \textbf{pl} \end{bmatrix}$$

The unification of this with the value of DTR1 succeeds but adds no new information:

$$\begin{bmatrix} \text{MOTHER} & \begin{bmatrix} \text{CAT} & \textbf{S} \\ \text{AGR} & \boxed{1}\ \textbf{pl} \end{bmatrix} \\ \text{DTR1} & \begin{bmatrix} \text{CAT} & \textbf{NP} \\ \text{AGR} & \boxed{1} \end{bmatrix} \\ \text{DTR2} & \begin{bmatrix} \text{CAT} & \textbf{VP} \\ \text{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Similarly, this structure unifies with the root structure, so this is a valid parse.

Note however, that if we had tried to parse *it like it*, a unification failure would have occurred, since the AGR on the lexical entry for *it* has the value **sg** which clashes with the value **pl**.

I have described these unifications as occurring in a particular order, but it is very important to note that order is not significant and that the same overall result would have been obtained if another order had been used. This means that different parsing algorithms are guaranteed to give the same result. The one proviso is that with some FS grammars, just like CFGs, some algorithms may terminate while others do not.

5.3 Feature structures in detail

So far, I have been using a rather informal description of FSs. The following section gives more formal definitions.

FSs can be thought of as graphs which have labelled arcs connecting nodes (except for the case of the simplest FSs, which consist of a single node with no arcs) The labels on the arcs are the features. Arcs are regarded as having a direction, conventionally regarded as pointing into the structure, away from the single root node. The set of features and the set of atomic values are assumed to be finite.

Properties of FSs

Connectedness and unique root A FS must have a unique root node: apart from the root node, all nodes have one or more parent nodes.

Unique features Any node may have zero or more arcs leading out of it, but the label on each (that is, the feature) must be unique.

No cycles No node may have an arc that points back to the root node or to a node that intervenes between it and the root node. (Some variants of FS formalisms allow cycles.)

Values A node which does not have any arcs leading out of it may have an associated atomic value.

Finiteness An FS must have a finite number of nodes.

Sequences of features are known as *path*s.

Feature structures can be regarded as being ordered by information content — an FS is said to *subsume* another if the latter carries extra information. This is important because we define unification in terms of subsumption.

Properties of subsumption FS1 subsumes FS2 if and only if the following conditions hold:

Path values For every path P in FS1 there is a path P in FS2. If P has a value t in FS1, then P also has value t in FS2.

Path equivalences Every pair of paths P and Q which are reentrant in FS1 (i.e., which lead to the same node in the graph) are also reentrant in FS2.

Unification corresponds to conjunction of information, and thus can be defined in terms of subsumption, which is a relation of information containment. The unification of two FSs is defined to be the most general FS which contains all the information in both of the FSs. Unification will fail if the two FSs contain conflicting information. As we saw with the simple grammar above, this prevented *it like it* getting an analysis, because the AGR values conflicted.

Properties of unification The unification of two FSs, FS1 and FS2, is the most general FS which is subsumed by both FS1 and FS2, if it exists.

5.4 A grammar enforcing subcategorization

Although the grammar shown above improves on the simple CFG, it still doesn't encode subcategorization. The grammar shown overleaf does this. It moves further away from the CFG. In particular, in the previous grammar the cat feature encoded both the part-of-speech (i.e., noun or verb) and the distinction between the lexical sign and the phrase (i.e., N vs NP and V vs VP). In the grammar below, the CAT feature just encodes the major category (noun vs verb) and the phrasal distinction is encoded in terms of whether the subcategorization requirements have been satisfied. The CAT and AGR features are now inside another feature *head*. Signs have three features at the top-level: HEAD, COMP and SPR. This reflects a portion of a linguistic framework which is described in great detail in Sag and Wasow (1999).[15]

Briefly, HEAD contains information which is shared between the lexical entries and phrases of the same category: e.g., nouns share this information with the noun phrase which dominates them in the tree, while verbs share head information with verb phrases and sentences. So HEAD is used for agreement information and for category information (i.e. noun, verb etc). In contrast, COMP and SPR are about subcategorization: they contain information about what can combine with this sign. In the grammar below, the specifier is the subject of a verb, but in a larger grammar a determiner would be the specifier of a noun. For instance, an intransitive verb will have a SPR corresponding to its subject 'slot' and a value of **filled** for its COMP.[16]

[15] You aren't expected to know any details of the linguistic framework for the exam, and you do not have to remember the feature structure architecture described here. The point of giving this more complicated grammar is that it starts to demonstrate the power of the feature structure framework, in a way that the simple grammar using agreement does not.

[16] There's a more elegant way of doing this using lists, but since this complicates the grammar quite a lot, I won't show this here.

The grammar below has just two rules, one for combining a sign with its complement, another for combining a sign with its specifier. Rule 1 says that, when building the phrase, the COMP value of the first daughter is to be equated (unified) with the whole structure of the second daughter (indicated by [2]). The head of the mother is equated with the head of the first daughter ([1]). The SPR of the mother is also equated with the SPR of the first daughter ([3]). The COMP value of the mother is stipulated as being **filled**: this means the mother can't act as the first daughter in another application of the rule, since **filled** won't unify with a complex feature structure. The specifier rule is fairly similar, in that a SPR 'slot' is being instantiated, although in this case it's the second daughter that contains the slot and is sharing its head information with the mother. The rule also stipulates that the AGR values of the two daughters have to be unified and that the specifier daughter has to have a filled complement. These rules are controlled by the lexical entries in the sense that it's the lexical entries which determine the required complements and specifier of a word.

As an example, consider analysing *they fish*. The verb entry for *fish* can be unified with the second daughter position of rule 2, giving the following partially instantiated rule:

$$\begin{bmatrix} \text{HEAD } [1] \begin{bmatrix} \text{CAT } \mathbf{verb} \\ \text{AGR } [3]\ \mathbf{pl} \end{bmatrix} \\ \text{COMP } \mathbf{filled} \\ \text{SPR } \mathbf{filled} \end{bmatrix} \rightarrow [2] \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \mathbf{noun} \\ \text{AGR } [3] \end{bmatrix} \\ \text{COMP } \mathbf{filled} \\ \text{SPR } \mathbf{filled} \end{bmatrix}, \begin{bmatrix} \text{HEAD } [1] \\ \text{COMP } \mathbf{filled} \\ \text{SPR } [2] \end{bmatrix}$$

The first daughter of this result can be unified with the structure for *they*, which in this case returns the same structure, since it adds no new information. The result can be unified with the root structure, so this is a valid parse.

On the other hand, the lexical entry for the noun *fish* does not unify with the second daughter position of rule2. The entry for *they* does not unify with the first daughter position of rule1. Hence there is no other parse.

Simple FS grammar fragment encoding subcategorization

Rule1 ;;; comp filling

$$\begin{bmatrix} \text{HEAD } [1] \\ \text{COMP } \mathbf{filled} \\ \text{SPR } [3] \end{bmatrix} \rightarrow \begin{bmatrix} \text{HEAD } [1] \\ \text{COMP } [2] \\ \text{SPR } [3] \end{bmatrix}, [2] \begin{bmatrix} \text{COMP } \mathbf{filled} \end{bmatrix}$$

Rule2 ;;; spr filling:

$$\begin{bmatrix} \text{HEAD } [1] \\ \text{COMP } \mathbf{filled} \\ \text{SPR } \mathbf{filled} \end{bmatrix} \rightarrow [2] \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{AGR } [3] \end{bmatrix} \\ \text{COMP } \mathbf{filled} \\ \text{SPR } \mathbf{filled} \end{bmatrix}, \begin{bmatrix} \text{HEAD } [1] \begin{bmatrix} \text{AGR } [3] \end{bmatrix} \\ \text{COMP } \mathbf{filled} \\ \text{SPR } [2] \end{bmatrix}$$

Lexicon:

;;; noun phrases

they
$$\begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \mathbf{noun} \\ \text{AGR } \mathbf{pl} \end{bmatrix} \\ \text{COMP } \mathbf{filled} \\ \text{SPR } \mathbf{filled} \end{bmatrix}$$

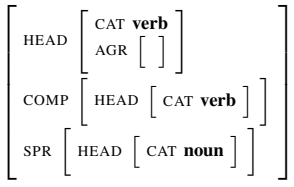
fish
$$\begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \mathbf{noun} \\ \text{AGR } [\ ] \end{bmatrix} \\ \text{COMP } \mathbf{filled} \\ \text{SPR } \mathbf{filled} \end{bmatrix}$$

it
$$\begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \mathbf{noun} \\ \text{AGR } \mathbf{sg} \end{bmatrix} \\ \text{COMP } \mathbf{filled} \\ \text{SPR } \mathbf{filled} \end{bmatrix}$$

;;; verbs

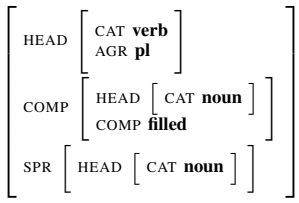
fish
$$\begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \mathbf{verb} \\ \text{AGR } \mathbf{pl} \end{bmatrix} \\ \text{COMP } \mathbf{filled} \\ \text{SPR } \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \mathbf{noun} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

can



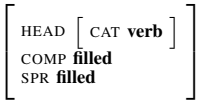
;;; auxiliary verb

can



;;; transitive verb

Root structure:



5.5 Parsing with feature structure grammars

Formally we can treat feature structure grammars in terms of subsumption. I won't give details here, but the intuition is that the rule FSs, the lexical entry FSs and the root FS all act as constraints on the parse, which have to be satisfied simultaneously. This means the system has to build a parse structure which is subsumed by all the applicable constraints. However, this description of what it means for something to be a valid parse doesn't give any hint of a sensible algorithm.

The standard approach to implementation is to use chart parsing, as described in the previous lecture, but the notion of a grammar rule matching an edge in the chart is more complex. In a naive implementation, when application of a grammar rule is checked, all the feature structures in the edges in the chart that correspond to the possible daughters have to be copied, and the grammar rule feature structure itself is also copied. The copied daughter structures are unified with the daughter positions in the copy of the rule, and if unification succeeds, the copied structure is associated with a new edge on the chart.

The need for copying is often discussed in terms of the destructive nature of the standard algorithm for unification (which I won't describe here), but this is perhaps a little misleading. Unification, however implemented, involves sharing information between structures. Assume, for instance, that the FS representing the lexical entry of the noun for *fish* is underspecified for number agreement. When we parse a sentence like:

> the fish swims

the part of the FS in the result that corresponds to the original lexical entry will have its AGR value instantiated. This means that the structure corresponding to a particular edge cannot be reused in another analysis, because it will contain 'extra' information. Consider, for instance, parsing:

> the fish in the lake which is near the town swim

A possible analysis of:

> fish in the lake which is near the town

is:

> (fish (in the lake) (which is near the town))

i.e., the fish (sg) is near the town. If we instantiate the AGR value in the FS for *fish* as sg while constructing this parse, and then try to reuse that same FS for *fish* in the other parses, analysis will fail. Hence the need for copying, so we can use a fresh structure each time. Copying is potentially extremely expensive, because realistic grammars involve FSs with many hundreds of nodes.

So, although unification is very near to linear in complexity, naive implementations of FS formalisms are very inefficient. Furthermore, packing is not straightforward, because two structures are rarely identical in real grammars (especially ones that encode semantics).

Reasonably efficient implementations of FS formalisms can nevertheless be developed. Copying can be greatly reduced:

1. by doing an efficient pretest before unification, so that copies are only made when unification is likely to succeed
2. by sharing parts of FSs that aren't changed
3. by taking advantage of *locality principles* in linguistic formalisms which limit the need to percolate information through structures

Packing can also be implemented: the test to see if a new edge can be packed involves subsumption rather than equality.

As with CFGs, for real efficiency we need to control the search space so we only get the most likely analyses. Defining probabilistic FS grammars in a way which is theoretically well-motivated is much more difficult than defining a PCFG. Practically it seems to turn out that treating a FS grammar much as though it were a CFG works fairly well, but this is an active research issue.

5.6 Templates

The lexicon outlined above has the potential to be very redundant. For instance, as well as the intransitive verb *fish*, a full lexicon would have entries for *sleep*, *snore* and so on, which would be essentially identical. We avoid this redundancy by associating names with particular feature structures and using those names in lexical entries. For instance:
fish INTRANS_VERB
sleep INTRANS_VERB
snore INTRANS_VERB
where the template is specified as:

INTRANS_VERB

$$\begin{bmatrix} \text{HEAD} \begin{bmatrix} \text{CAT} & \textbf{verb} \\ \text{AGR} & \textbf{pl} \end{bmatrix} \\ \text{COMP} \ \textbf{filled} \\ \text{SPR} \begin{bmatrix} \text{HEAD} \begin{bmatrix} \text{CAT} & \textbf{noun} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The lexical entry may have some specific information associated with it (e.g., semantic information, see next lecture) which will be expressed as a FS: in this case, the template and the lexical feature structure are combined by unification.

5.7 Interface to morphology

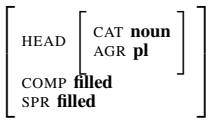
So far we have assumed a full-form lexicon, but we can now return to the approach to morphology that we saw in lecture 2, and show how this relates to feature structures. Recall that we have spelling rules which can be used to analyse a word form to return a stem and list of affixes and that each affix is associated with an encoding of the information it contributes. For instance, the affix *s* is associated with the template `PLURAL_NOUN`, which would correspond to the following information in our grammar fragment:

$$\begin{bmatrix} \text{HEAD} \begin{bmatrix} \text{CAT} & \textbf{noun} \\ \text{AGR} & \textbf{pl} \end{bmatrix} \end{bmatrix}$$

A stem for a noun is generally assumed to be uninstantiated for number (i.e., neutral between sg and pl). So the lexical entry for the noun *dog* in our fragment would be:

$$\begin{bmatrix} \text{HEAD} \begin{bmatrix} \text{CAT} & \textbf{noun} \\ \text{AGR} & [\ ] \end{bmatrix} \\ \text{COMP} \ \textbf{filled} \\ \text{SPR} \ \textbf{filled} \end{bmatrix}$$

One simple way of implementing inflectional morphology in FSs is simply to unify the contribution of the affix with that of the stem. If we unify the FS corresponding to the stem for *dog* to the FS for `PLURAL_NOUN`, we get:



This approach assumes that we also have a template `SINGULAR_NOUN`, where this is associated with a 'null' affix.

In the case of an example such as *feed* incorrectly analysed as *fee -ed*, discussed in §2.5, the affix information will fail to unify with the stem, ruling out that analysis.

There are other ways of encoding inflectional morphology with FS, which I won't discuss here. Note that this simple approach is not, in general, adequate for derivational morphology. For instance, the affix *-ize*, which combines with a noun to form a verb (e.g., *lemmatization*), cannot be represented simply by unification, because it has to change a nominal form into a verbal one. This can be implemented by some form of *lexical rule* (which are essentially grammar rules with single daughters), but I won't discuss this in this course. Note, however, that this reflects the distinction between inflectional and derivational morphology that we saw in §2.2:while inflectional morphology can be seen as simple addition of information, derivational morphology converts feature structures into new structures. However, derivational morphology is often not treated as productive, especially in limited domain systems.

5.8 Further reading

J&M describe feature structures as augmenting a CFG rather than replacing it, but most of their discussion applies equally to the FS formalism I've outlined here.

LinGO (Linguistic Grammars Online: `http://lingo.stanford.edu`) distributes Open Source FS grammars for a variety of languages. The LinGO English Resource Grammar (ERG) is probably the largest freely available bidirectional grammar.

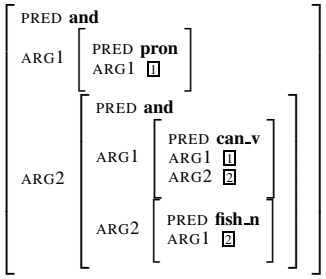
6 Lecture 6: Compositional and lexical semantics

This lecture will give a rather superficial account of semantics and some of its computational aspects:

1. Compositional semantics in feature structure grammars
2. Meaning postulates
3. Classical lexical relations: hyponymy, meronymy, synonymy, antonymy
4. Taxonomies and WordNet
5. Classes of polysemy: homonymy, regular polysemy, vagueness
6. Word sense disambiguation

6.1 Simple semantics in feature structures

The grammar fragment below is based on the one in the previous lecture. It is intended as a rough indication of how it is possible to build up semantic representations using feature structures. The lexical entries have been augmented with pieces of feature structure reflecting predicate-argument structure. With this grammar, the FS for *they can fish* will have a SEM value of:

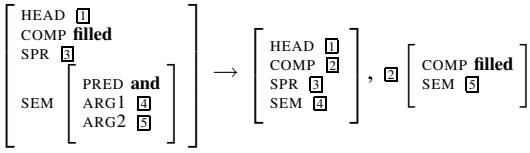


This can be taken to be equivalent to the logical expression $\text{pron}(x) \wedge (\text{can\_v}(x, y) \wedge \text{fish\_n}(y))$ by translating the reentrancy between argument positions into variable equivalence.

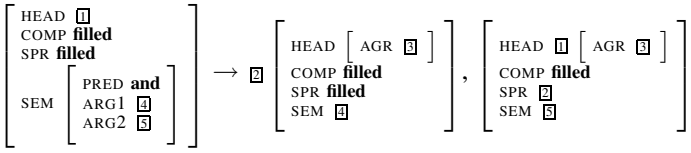
The most important thing to notice is how the syntactic argument positions in the lexical entries are linked to their semantic argument positions. This means, for instance, that for the transitive verb *can*, the syntactic subject will always correspond to the first argument position, while the syntactic object will correspond to the second position.

Simple FS grammar with crude semantic composition

Rule1 ;;; comp filling

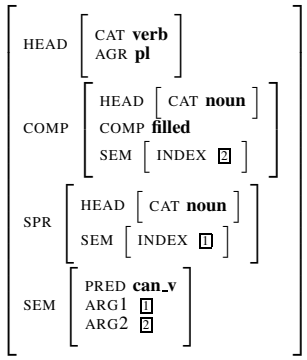


Rule2 ;;; spr filling:



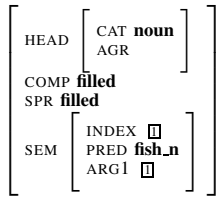
Lexicon:

can



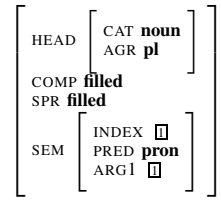
;;; transitive verb

fish



;;; noun phrase

they



;;; noun phrase

An alternative approach to encoding semantics is to write the semantic composition rules in a separate formalism such as *typed lambda calculus*. This corresponds more closely to the approach most commonly assumed in formal linguistics: variants of lambda calculus are sometimes used in NLP, but I won't discuss this further here.

In general, a semantic representation constructed for a sentence is called the *logical form* of the sentence. The semantics shown above can be taken to be equivalent to a form of predicate calculus without variables or quantifiers: i.e. the 'variables' in the representation actually correspond to constants. It turns out that this very impoverished form of semantic representation is adequate for many NLP applications: template representations, used in information extraction or simple dialogue systems can be thought of as equivalent to this. But for a fully adequate representation we need something richer — for instance, to do negation properly. Minimally we need full first-order predicate calculus (FOPC). FOPC logical forms can be passed to theorem-provers in order to do inference about the meaning of a sentence. However, although this approach has been extensively explored in research work, especially in the 1980s, it hasn't so far led to practical systems. There are many reasons for this, but perhaps the most important is the difficulty of acquiring detailed domain knowledge expressed in FOPC. There is also a theoretical AI problem, because we seem to need some form of probabilistic reasoning for many applications. So, although most researchers who are working in computational compositional semantics take support for inference as a desideratum, many systems actually use some form of shallow inference (e.g., semantic transfer in MT, mentioned in lecture 8).

FOPC also has the disadvantage that it forces quantifiers to be in a particular scopal relationship, and this information is not (generally) overt in NL sentences. One classic example is:

Every man loves a woman

which is ambiguous between:

$$\forall x[\text{man}'(x) \Rightarrow \exists y[\text{woman}'(y) \wedge \text{love}'(x, y)]]$$

and the less-likely, 'one specific woman' reading:

$$\exists y[\text{woman}'(y) \wedge \forall x[\text{man}'(x) \Rightarrow \text{love}'(x, y)]]$$

Most current systems construct an underspecified representation which is neutral between these readings, if they represent quantifier scope at all. There are several different alternative formalisms for underspecification.

6.2 Generation

We can generate from a semantic representation with a suitable FS grammar. Producing an output string given an input logical form is generally referred to as *tactical generation* or *realization*, as opposed to *strategic generation* or *text planning*, which concerns how you might build the logical form in the first place. Strategic generation is an open-ended problem: it depends very much on the application and I won't have much to say about it here. Tactical generation is more tractable, and is useful without a strategic component in some contexts, such as the semantic transfer approach to MT, which I'll briefly discuss in lecture 8.

Tactical generation can use similar techniques to parsing: for instance one approach is *chart generation* which uses many of the same techniques as chart parsing. There has been much less work on generation than on parsing in general, and building bidirectional grammars is hard: most grammars for parsing allow through many ungrammatical strings. Recently there has been some work on statistical generation, where n-grams are used to choose between realisations constructed by a grammar that overgenerates. But even relatively 'tight' bidirectional grammars may need to use statistical techniques in order to generate natural sounding utterances.

6.3 Meaning postulates

Inference rules can be used to relate open class predicates: i.e., predicates that correspond to open class words. This is the classic way of representing lexical meaning in formal semantics within linguistics:[17]

$$\forall x[bachelor(x) \leftrightarrow man(x) \wedge unmarried(x)]$$

Linguistically and philosophically, this gets pretty dubious. Is the current Pope a bachelor? Technically presumably yes, but *bachelor* seems to imply someone who could be married: it's a strange word to apply to the Pope under current assumptions about celibacy. Meaning postulates are also too unconstrained: I could construct a predicate 'bachelor-weds-thurs' to correspond to someone who was unmarried on Wednesday and married on Thursday, but this isn't going to correspond to a word in any natural language. In any case, very few words are as simple to define as *bachelor*: consider how you might start to define *table*, *tomato* or *thought*, for instance.[18]

For computational semantics, perhaps the best way of regarding meaning postulates is simply as one reasonable way of linking compositionally constructed semantic representations to a specific domain. In NLP, we're normally concerned with implication rather than definition and this is less problematic philosophically:

$$\forall x[bachelor(x) \rightarrow man(x) \wedge unmarried(x)]$$

However, the big computational problems with meaning postulates are their acquisition and the control of inference once they have been obtained. Building meaning postulates for anything other than a small, bounded domain is an AI-complete problem.

The more general, shallower, relationships that are classically discussed in lexical semantics are currently more useful in NLP, especially for broad-coverage processing.

6.4 Hyponymy: IS-A

Hyponymy is the classical IS-A relation: e.g. *dog* is a *hyponym* of *animal*. That is, the relevant sense of *dog* is the hyponym of *animal*: as nearly everything said in this lecture is about word senses rather than words, I will avoid explicitly qualifying all statements in this way, but this should be globally understood.

animal is the *hypernym* of *dog*. Hyponyms can be arranged into *taxonomies*: classically these are tree-structured: i.e., each term has only one *hypernym*.

Despite the fact that hyponymy is by far the most important meaning relationship assumed in NLP, many questions arise which don't currently have very good answers:

[17] Generally, linguists don't actually write meaning postulates for open-class words, but this is the standard assumption about how meaning would be represented if anyone could be bothered to do it!

[18] There has been a court case that hinged on the precise meaning of *table* and also one that depended on whether tomatoes were fruits or vegetables.

1. What classes of words can be categorised by hyponymy? Some nouns, classically biological taxonomies, but also human artifacts, professions etc work reasonably well. Abstract nouns, such as *truth*, don't really work very well (they are either not in hyponymic relationships at all, or very shallow ones). Some verbs can be treated as being hyponyms of one another — e.g. *murder* is a *hyponym* of *kill*, but this is not nearly as clear as it is for concrete nouns. Event-denoting nouns are similar to verbs in this respect. Hyponymy is essentially useless for adjectives.

2. Do differences in quantisation and individuation matter? For instance, is *chair* a hyponym of *furniture*? is *beer* a hyponym of *drink*? is *coin* a hyponym of *money*?

3. Is multiple inheritance allowed? Intuitively, multiple parents might be possible: e.g. *coin* might be *metal* (or *object*?) and also *money*. Artifacts in general can often be described either in terms of their form or their function.

4. What should the top of the hierarchy look like? The best answer seems to be to say that there is no single top but that there are a series of hierarchies.

6.5 Other lexical semantic relations

Meronymy i.e., PART-OF

The standard examples of meronymy apply to physical relationships: e.g., *arm* is part of a *body* (*arm* is a *meronym* of *body*); *steering wheel* is a meronym of *car*. Note the distinction between 'part' and 'piece': if I attack a car with a chainsaw, I get pieces rather than parts!

Synonymy i.e., two words with the same meaning (or nearly the same meaning)

True synonyms are relatively uncommon: most cases of true synonymy are correlated with dialect differences (e.g., *eggplant* / *aubergine*, *boot* / *trunk*). Often synonymy involves register distinctions, slang or jargons: e.g., *policeman*, *cop*, *rozzer* . . . Near-synonyms convey nuances of meaning: *thin*, *slim*, *slender*, *skinny*.

Antonymy i.e., opposite meaning

Antonymy is mostly discussed with respect to adjectives: e.g., *big*/*little*, though it's only relevant for some classes of adjectives.

6.6 WordNet

WordNet is the main resource for lexical semantics for English that is used in NLP — primarily because of its very large coverage and the fact that it's freely available. WordNets are under development for many other languages, though so far none are as extensive as the original.

The primary organisation of WordNet is into *synsets*: synonym sets (near-synonyms). To illustrate this, the following is part of what WordNet returns as an 'overview' of *red*:

```
wn red -over

Overview of adj red

The adj red has 6 senses (first 5 from tagged texts)

1. (43) red, reddish, ruddy, blood-red, carmine,
cerise, cherry, cherry-red, crimson, ruby, ruby-red,
scarlet -- (having any of numerous bright or strong
colors reminiscent of the color of blood or cherries
or tomatoes or rubies)
2. (8) red, reddish -- ((used of hair or fur) of a
reddish brown color; "red deer"; reddish hair")
```

Nouns in WordNet are organised by hyponymy, as illustrated by the fragment below:

```
Sense 6
big cat, cat
       => leopard, Panthera pardus
           => leopardess
           => panther
       => snow leopard, ounce, Panthera uncia
       => jaguar, panther, Panthera onca, Felis onca
       => lion, king of beasts, Panthera leo
           => lioness
           => lionet
       => tiger, Panthera tigris
           => Bengal tiger
           => tigress
       => liger
       => tiglon, tigon
       => cheetah, chetah, Acinonyx jubatus
       => saber-toothed tiger, sabertooth
           => Smiledon californicus
           => false saber-toothed tiger
```

The following is an overview of the information available in WordNet for the various POS classes:

- all classes
    1. synonyms (ordered by frequency)
    2. familiarity / polysemy count
    3. compound words (done by spelling)
- nouns
    1. hyponyms / hypernyms (also sisters)
    2. holonyms / meronyms
- adjectives
    1. antonyms
- verbs
    1. antonyms
    2. hyponyms / hypernyms (also sisters)
    3. syntax (very simple)
- adverbs

Taxonomies have also been automatically or semi-automatically extracted from machine-readable dictionaries, but these are not distributed. Microsoft's MindNet is the best known example (it has many more relationships than just hyponymy). There are other collections of terms, generally hierarchically ordered, especially medical ontologies. There have been a number of attempts to build an ontology for world knowledge: none of the more elaborate ones are generally available. There is an ongoing attempt at standardisation of ontologies. Ontology support is an important component of the semantic web.

6.7 Using lexical semantics

By far the most commonly used lexical relation is hyponymy. Hyponymy relations can be used in many ways:

- Semantic classification: e.g., for selectional restrictions (e.g., the object of *eat* has to be something edible) and for named entity recognition
- Shallow inference: 'X murdered Y' implies 'X killed Y' etc
- Back-off to semantic classes in some statistical approaches
- Word-sense disambiguation
- MT: if you can't translate a term, substitute a hypernym
- Query expansion for information retrieval: if a search doesn't return enough results, one option is to replace an over-specific term with a hypernym

Synonymy or near-synonymy is relevant for some of these reasons and also for generation. (However dialect and register haven't been investigated much in NLP, so the possible relevance of different classes of synonym for customising text hasn't really been looked at.)

6.8 Polysemy

Polysemy refers to the state of a word having more than one sense: the standard example is *bank* (river bank) vs *bank* (financial institution).

This is *homonymy* — the two senses are unrelated (not entirely true for *bank*, actually, but historical relatedness isn't actually important — it's whether ordinary speakers of the language feel there's a relationship). Homonymy is the most obvious case of polysemy, but is actually relatively infrequent compared to uses which have different but related meanings, such as *bank* (financial institution) vs *bank* (in a casino).

If polysemy were always homonymy, word senses would be discrete: two senses would be no more likely to share characteristics than would morphologically unrelated words. But most senses are actually related. Regular or systematic polysemy concerns related but distinct usages of words, often with associated syntactic effects. For instance, *strawberry, cherry* (fruit / plant), *rabbit, turkey, halibut* (meat / animal), *tango, waltz* (dance (noun) / dance (verb)).

There are a lot of complicated issues in deciding whether a word is polysemous or simply general/vague. For instance, *teacher* is intuitively general between male and female teachers rather than ambiguous, but giving good criteria as a basis of this distinction is difficult. Dictionaries are not much help, since their decisions as to whether to split a sense or to provide a general definition are very often contingent on external factors such as the size of the dictionary or the intended audience, and even when these factors are relatively constant, lexicographers often make different decisions about whether and how to split up senses.

6.9 Word sense disambiguation

Word sense disambiguation (WSD) is needed for most NL applications that involve semantics (explicitly or implicitly). In limited domains, WSD is not too big a problem, but for large coverage text processing it's a serious bottleneck.

WSD needs depend on the application — there is no objective notion of word sense (dictionaries differ extensively) and it's very hard to come up with good criteria to judge whether or not to distinguish senses. But in order to experiment with WSD as a standalone module, there has to be a standard: most commonly WordNet, because it is the only extensive modern resource for English with no problematic IPR issues. This is controversial, because WordNet has a very fine granularity of senses — it's also obvious that its senses often overlap. However, the only current alternative is a pre-1920 version of Webster's. Recently WSD 'competitions' have been organised: SENSEVAL and SENSEVAL 2.

WSD up to the early 1990s was mostly done by hand-constructed rules (still used in some MT systems). Dahlgren investigated WSD in a fairly broad domain in the 1980s. Reasonably broad-coverage WSD generally depends on:

- frequency
- collocations
- selectional restrictions/preferences

What's changed since the 1980s is that various statistical or machine-learning techniques have been used to avoid hand-crafting rules.

- supervised learning. Requires a sense-tagged corpus, which is extremely time-consuming to construct systematically (examples are the Semcor and SENSEVAL corpora, but both are really too small). Most experimentation has been done with a small set of words which can be sense-tagged by the experimenter (e.g., *plant*). Supervised learning techniques do not carry over well from one corpus to another.
- unsupervised learning (see below)
- Machine readable dictionaries (MRDs). Disambiguating dictionary definitions according to the internal data in dictionaries is necessary to build taxonomies from MRDs. MRDs have also been used as a source of selectional preference and collocation information for general WSD (quite successfully).

Until recently, most of the statistical or machine-learning techniques have been evaluated on homonyms: these are relatively easy to disambiguate. So 95% disambiguation in e.g., Yarowsky's experiments sounds good (see below), but doesn't translate into high precision on all words when target is WordNet senses (in SENSEVAL 2 the best system was around 70%).

There have also been some attempts at automatic *sense induction*, where an attempt is made to determine the clusters of usages in texts that correspond to senses. In principle, this is a very good idea, since the whole notion of a word sense is fuzzy: word senses can be argued to be artifacts of dictionary publishing. However, so far sense induction has not been much explored in monolingual contexts, though it could be considered as an inherent part of statistical approaches to MT.

6.10 Collocations

Informally, a collocation is a group of two or more words that occur together more often than would be expected by chance (there are other definitions — this is not really a precise notion). Collocations have always been the most useful source of information for WSD, even in Dahlgren's early experiments. For instance:

(23) Striped bass are common.

(24) Bass guitars are common.

striped is a good indication that we're talking about the fish (because it's a particular sort of bass), similarly with *guitar* and music. In both *bass guitar* and *striped bass*, we've arguably got a multiword expression (i.e., a conventional phrase that might be listed in a dictionary), but the principle holds for any sort of collocation. The best collocates for WSD tend to be syntactically related in the sentence to the word to be disambiguated, but many techniques simply use a window of words.

J&M make a useful (though non-standard) distinction between collocation and co-occurrence: co-occurrence refers to the appearance of another word in a larger window of text than a collocation. For instance, *trout* might co-occur with the fish sense of bass.

6.11 Yarowsky's unsupervised learning approach to WSD

Yarowsky (1995) describes a technique for unsupervised learning using collocates (collocates and co-occurrences in J&M's terms). A few seed collocates are chosen for each sense (manually or via an MRD), then these are used to accurately identify distinct senses. The sentences in which the disambiguated senses occur can then be used to learn other discriminating collocates automatically, producing a decision list. The process can then be iterated. The

algorithm allows bad collocates to be overridden. This works because of the general principle of 'one sense per collocation' (experimentally demonstrated by Yarowsky — it's not absolute, but there are very strong preferences).

In a bit more detail, using Yarowsky's example of disambiguating *plant* (which is homonymous between factory vs vegetation senses):

1. Identify all examples of the word to be disambiguated in the training corpus and store their contexts.

sense	training example
?	company said that the *plant* is still operating
?	although thousands of *plant* and animal species
?	zonal distribution of *plant* life
?	company manufacturing *plant* is in Orlando
	etc

2. Identify some seeds which reliably disambiguate a few of these uses. Tag the disambiguated senses and count the rest as residual. For instance, choosing 'plant life' as a seed for the vegetation sense of plant (sense A) and 'manufacturing plant' as the seed for the factory sense (sense B):

sense	training example
?	company said that the *plant* is still operating
?	although thousands of *plant* and animal species
A	zonal distribution of *plant* life
B	company manufacturing *plant* is in Orlando
	etc

This disambiguated 2% of uses in Yarowsky's corpus, leaving 98% residual.

3. Train a *decision list* classifier on the Sense A/Sense B examples. A decision list approach gives a list of criteria which are tried in order until an applicable test is found: this is then applied. The tests are each associated with a reliability metric. The original seeds are likely to be at the top of the initial decision list, followed by other discriminating terms. e.g. the decision list might include:

reliability	criterion	sense
8.10	*plant* life	A
7.58	manufacturing *plant*	B
6.27	*animal* within 10 words of *plant*	A
	etc	

4. Apply the decision list classifier to the training set and add all examples which are tagged with greater than a threshold reliability to the Sense A and Sense B sets.

sense	training example
?	company said that the *plant* is still operating
A	although thousands of *plant* and animal species
A	zonal distribution of *plant* life
B	company manufacturing *plant* is in Orlando
	etc

5. Iterate the previous steps 3 and 4 until convergence
6. Apply the classifier to the unseen test data

Yarowsky also demonstrated the principle of 'one sense per discourse' (again, a very strong, but not absolute effect). This can be used as an additional refinement for the algorithm above.

Yarowsky argues that decision lists work better than many other statistical frameworks because no attempt is made to combine probabilities. This would be complex, because the criteria are not independent of each other.

Yarowsky's experiments were nearly all on homonyms: these principles probably don't hold as well for sense extension.

6.12 Evaluation of WSD

The baseline for WSD is generally 'pick the most frequent' sense: this is hard to beat! However, in many applications, we don't know the frequency of senses.

SENSEVAL and SENSEVAL-2 evaluated WSD in multiple languages, with various criteria, but generally using WordNet senses for English. The human ceiling for this task varies considerably between words: probably partly because of inherent differences in semantic distance between groups of uses and partly because of WordNet itself, which sometimes makes very fine-grained distinctions. An interesting variant in SENSEVAL-2 was to do one experiment on WSD where the disambiguation was with respect to uses requiring different translations into Japanese. This has the advantage that it is useful and relatively objective, but sometimes this task requires splitting terms which aren't polysemous in English (e.g., *water* — hot vs cold). Performance of WSD on this task seems a bit better than the general WSD task.

6.13 Further reading

J&M go into quite a lot of detail about compositional semantics including underspecification.

WordNet is freely downloadable: the website has pointers to several papers which provide a good introduction.

For a lot more detail of WSD than provided by J&M, see Manning and Schütze who have a very detailed account of WSD and word-sense induction:

Manning, Christopher and Hinrich Schütze (1999), *Foundations of Statistical Natural Language Processing*, MIT Press

Yarowsky's paper is well-written and should be understandable:

Yarowsky, David (1995)
Unsupervised word sense disambiguation rivalling supervised methods,
Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95) MIT, 189–196

Like many other recent NLP papers, this can be downloaded via www.citeseer.com

7 Lecture 7: Discourse

Utterances are always understood in a particular context. Context-dependent situations include:

1. Referring expressions: pronouns, definite expressions etc.
2. Universe of discourse: *every dog barked*, doesn't mean every dog in the world but only every dog in some explicit or implicit contextual set.
3. Responses to questions, etc: only make sense in a context: *Who came to the party? Not Sandy.*
4. Implicit relationships between events: *Max fell. John pushed him* — the second sentence is (usually) understood as providing a causal explanation.

In the first part of this lecture, I give a brief overview of *rhetorical relations* which can be seen as structuring text at a level above the sentence. I'll then go on to talk about one particular case of context-dependent interpretation — anaphor resolution. I will describe an algorithm for anaphor resolution which uses a relatively broad-coverage shallow parser and then discuss a variant of it that relies on POS-tagging and regular expression matching rather than parsing.

7.1 Rhetorical relations and coherence

Consider the following discourse:

> Max fell. John pushed him.

This discourse can be interpreted in at least two ways:

1. Max fell because John pushed him.
2. Max fell and then John pushed him.

There seems to be an implicit relationship between the two original sentences: a *discourse relation* or *rhetorical relation*. (I will use the terms interchangeably here, though different theories use different terminology, and rhetorical relation tends to refer to a more surfacy concept than discourse relation.) In 1 the link is a form of explanation, but 2 is an example of narration. Theories of discourse/rhetorical relations reify link types such as *Explanation* and *Narration*. The relationship is made more explicit in 1 and 2 than it was in the original sentence: *because* and *and then* are said to be *cue phrases*.

7.2 Coherence

Discourses have to have connectivity to be coherent:

> Kim got into her car. Sandy likes apples.

Both of these sentences make perfect sense in isolation, but taken together they are incoherent. Adding context can restore coherence:

> Kim got into her car. Sandy likes apples, so Kim thought she'd go to the farm shop and see if she could get some.

The second sentence can be interpreted as an explanation of the first. In many cases, this will also work if the context is known, even if it isn't expressed.

Strategic generation requires a way of implementing coherence. For example, consider a system that reports share prices. This might generate:

> In trading yesterday: Dell was up 4.2%, Safeway was down 3.2%, Compaq was up 3.1%.

This is much less acceptable than a connected discourse:

> Computer manufacturers gained in trading yesterday: Dell was up 4.2% and Compaq was up 3.1%. But retail stocks suffered: Safeway was down 3.2%.

Here *but* indicates a Contrast. Not much actual information has been added (assuming we know what sort of company Dell, Compaq and Safeway are), but the discourse is easier to follow.

Discourse coherence assumptions can affect interpretation:

> John likes Bill. He gave him an expensive Christmas present.

If we interpret this as Explanation, then 'he' is most likely Bill. But if it is Justification (i.e., the speaker is justifying the first sentence), then 'he' is John.

7.3 Factors influencing discourse interpretation

1. Cue phrases. These are sometimes unambiguous, but not usually. e.g. *and* is a cue phrase when used in sentential or VP conjunction.
2. Punctuation (also prosody) and text structure. For instance, parenthetical information cannot be related to a main clause by Narration, but a list is often interpreted as Narration:

   > Max fell (John pushed him) and Kim laughed.
   > Max fell, John pushed him and Kim laughed.

   Similarly, enumerated lists can indicate a form of narration.
3. Real world content:

   > Max fell. John pushed him as he lay on the ground.

4. Tense and aspect.

   > Max fell. John had pushed him.
   > Max was falling. John pushed him.

It should be clear that it is potentially very hard to identify rhetorical relations. In fact, recent research that simply uses cue phrases and punctuation is proving quite promising. This can be done by hand-coding a series of finite-state patterns, or by a form of supervised learning.

7.4 Discourse structure and summarization

If we consider a discourse relation as a relationship between two phrases, we get a binary branching tree structure for the discourse. In many relationships, such as Explanation, one phrase depends on the other: e.g., the phrase being explained is the main one and the other is subsidiary. In fact we can get rid of the subsidiary phrases and still have a reasonably coherent discourse. (The main phrase is sometimes called the *nucleus* and the subsidiary one is the *satellite*.) This can be exploited in summarization.

For instance:

> We get a binary branching tree structure for the discourse. In many relationships one phrase depends on the other. In fact we can get rid of the subsidiary phrases and still have a reasonably coherent discourse.

Other relationships, such as Narration, give equal weight to both elements, so don't give any clues for summarization.

Rather than trying to find rhetorical relations for arbitrary text, genre-specific cues can be exploited, for instance for scientific texts. This allows more detailed summaries to be constructed.

7.5 Referring expressions

I'll now move on to talking about another form of discourse structure, specifically the link between referring expressions. The following example will be used to illustrate referring expressions and anaphora resolution:

> Niall Ferguson is prolific, well-paid and a snappy dresser. Stephen Moss hated him — at least until he spent an hour being charmed in the historian's Oxford study. (quote taken from the Guardian)

Some terminology:

referent a real world entity that some piece of text (or speech) refers to. e.g., the two people who are mentioned in this quote.

referring expressions bits of language used to perform reference by a speaker. In, the paragraph above, *Niall Ferguson*, *him* and *the historian* are all being used to refer to the same person (they *corefer*).

antecedent the text evoking a referent. *Niall Ferguson* is the antecedent of *him* and *the historian*

anaphora the phenomenon of referring to an antecedent: *him* and *the historian* are *anaphoric* because they refer to a previously introduced entity.

What about *a snappy dresser*? Traditionally, this would be described as predicative: that is, it is a predicate, like an adjective, rather than being a referring expression itself.

Generally, entities are introduced in a discourse (technically, *evoked*) by indefinite noun phrases or proper names. Demonstratives and pronouns are generally anaphoric. Definite noun phrases are often anaphoric (as above), but often used to bring a mutually known and uniquely identifiable entity into the current discourse. e.g., *the president of the US*.

Sometimes, pronouns appear before their referents are introduced: this is *cataphora*. E.g., at the start of a discourse:

> Although she couldn't see any dogs, Kim was sure she'd heard barking.

both cases of *she* refer to Kim - the first is a *cataphor*.

7.6 Pronoun agreement

Pronouns generally have to agree in number and gender with their antecedents. In cases where there's a choice of pronoun, such as *he*/*she* or *it* for an animal (or a baby, in some dialects), then the choice has to be consistent.

(25) A little girl is at the door — see what she wants, please?

(26) My dog has hurt his foot — he is in a lot of pain.

(27) * My dog has hurt his foot — it is in a lot of pain.

Complications include the gender neutral *they* (some dialects), use of *they* with *everybody*, group nouns, conjunctions and discontinuous sets:

(28) Somebody's at the door — see what they want, will you?

(29) I don't know who the new teacher will be, but I'm sure they'll make changes to the course.

(30) Everybody's coming to the party, aren't they?

(31) The team played really well, but now they are all very tired.

(32) Kim and Sandy are asleep: they are very tired.

(33) Kim is snoring and Sandy can't keep her eyes open: they are both exhausted.

7.7 Reflexives

(34) John$_i$ cut himself$_i$ shaving. (himself = John, subscript notation used to indicate this)

(35) # John$_i$ cut him$_j$ shaving. (i $\neq$ j — a very odd sentence)

The informal and not fully adequate generalisation is that reflexive pronouns must be co-referential with a preceding argument of the same verb (i.e., something it subcategorizes for), while non-reflexive pronouns cannot be. In linguistics, the study of inter-sentential anaphora is known as *binding theory*: I won't discuss this further, since the constraints on reference involved are quite different from those with intra-sentential anaphora.

7.8 Pleonastic pronouns

Pleonastic pronouns are semantically empty, and don't refer:

(36) It is snowing

(37) It is not easy to think of good examples.

(38) It is obvious that Kim snores.

(39) It bothers Sandy that Kim snores.

Note also:

(40) They are digging up the street again

This is an (informal) use of *they* which, though probably not technically pleonastic, doesn't apparently refer to a discourse referent in the standard way (they = 'the authorities'??).

7.9 Salience

There are a number of effects which cause particular pronoun referents to be preferred, after all the hard constraints discussed above are taken into consideration.

Recency More recent referents are preferred. Only relatively recently referred to entities are accessible.

(41) Kim has a fast car. Sandy has an even faster one. Lee likes to drive it.

it preferentially refers to Sandy's car, rather than Kim's.

Grammatical role Subjects $>$ objects $>$ everything else:

(42) Fred went to the Grafton Centre with Bill. He bought a CD.

he is more likely to be interpreted as Fred than as Bill.

Repeated mention Entities that have been mentioned more frequently are preferred:

(43) Fred was getting bored. He decided to go shopping. Bill went to the Grafton Centre with Fred. He bought a CD.

He=Fred (maybe) despite the general preference for subjects.

Parallelism Entities which share the same role as the pronoun in the same sort of sentence are preferred:

(44) Bill went with Fred to the Grafton Centre. Kim went with him to Lion Yard.

Him=Fred, because the parallel interpretation is preferred.

Coherence effects The pronoun resolution may depend on the rhetorical/discourse relation that is inferred.

(45) Bill likes Fred. He has a great sense of humour.

He = Fred preferentially, possibly because the second sentence is interpreted as an explanation of the first, and having a sense of humour is seen as a reason to like someone.

7.10 Algorithms for resolving anaphora

Most work has gone into the problem of resolving pronoun referents. As well as discourse understanding, this is often important in MT. For instance, English *it* has to be resolved to translate into German because German has grammatical gender (though note, if there are two possible antecedents, but both have the same gender, we probably do not need to resolve between the two for MT). I will describe one approach to anaphora resolution and a modification of it that requires fewer resources.

7.11 Lappin and Leass (1994)

The algorithm relies on parsed text (from a fairly shallow, very broad-coverage parser, which unfortunately isn't generally available). The text the system was developed and tested on was all from online computer manuals. The following description is a little simplified:

The discourse model consists of a set of referring NPs arranged into equivalence classes, each class having a global salience value.

For each sentence:

1. Divide by two the global salience factors for each existing equivalence class.
2. Identify referring NPs (i.e., exclude pleonastic *it* etc)
3. Calculate global salience factors for each NP (see below)
4. Update the discourse model with the referents and their global salience scores.
5. For each pronoun:
   (a) Collect potential referents (cut off is four sentences back).
   (b) Filter referents according to binding theory and agreement constraints.
   (c) Calculate the per pronoun adjustments for each referent (see below).
   (d) Select the referent with the highest salience value for its equivalence class plus its per-pronoun adjustment. In case of a tie, prefer the closest referent in the string.
   (e) Add the pronoun in to the equivalence class for that referent, and increment the salience factor by the non-duplicate salience factors pertaining to the pronoun.

The salience factors were determined experimentally. Global salience factors mostly take account of grammatical function — they encode the hierarchy mentioned previously. They give lowest weight to an NP in an adverbial position, such as inside an adjunct PP. This is achieved by giving every non-adverbial an extra positive score, because we want all global salience scores to be positive integers. Embedded NPs are also downweighted by giving a positive score to non-embedded NPs. Recency weights mean that intra-sentential binding is preferred.

Global salience factors.

recency	100
subject	80
objects of existential sentences	70
direct object	50
indirect object	40
oblique complement	40
non-embedded noun	80
other non-adverbial	50

'Existential objects' refers to NPs which are in syntactic object position in sentences such as:

There is a cat in the garden.

Here *a cat* is syntactically an object, but functions more like a subject, while *there*, which is syntactically the the subject, does not refer. An oblique complement is a complement other than a noun phrase, such as a PP.

The per-pronoun modifications have to be calculated each time a candidate pronoun is being evaluated. The modifications strongly disprefer cataphora and slightly prefer referents which are 'parallel', where parallel here just means having the same syntactic role.

Per pronoun salience factors:

cataphora	-175
same role	35

Applying this to the sample discourse:

> Niall Ferguson is prolific, well-paid and a snappy dresser.
> Stephen Moss hated him — at least until he spent an hour being charmed in the historian's Oxford study.

Assume we have processed up to '—' and are resolving *he*. Discourse referents:

N	*Niall Ferguson*, *him*	435
S	*Stephen Moss*	310

I am assuming that *a snappy dresser* is ignored, although it might actually be treated as another potential referent, depending on the parser.

N has score 155 + 280 ((subject + non-embedded + non-adverbial + recency)/2 + (direct object + head + non-adverbial + recency))

S has score 310 (subject + non-embedded + non-adverbial + recency) + same role per-pronoun 35

So in this case, the wrong candidate wins.

We now add *he* to the discourse referent equivalence class. The additional weight is only 80, for subject, because we don't add weights for a given factor more than once in a sentence.

N	*Niall Ferguson*, *him*, *he*	515

Note that the wrong result is quite plausible:

> Niall Ferguson is prolific, well-paid and a snappy dresser.
> Stephen Moss hated him — at least until he spent an afternoon being interviewed at very short notice.

The overall performance of the algorithm reported by Lappin and Leass was 86% but this was on computer manuals alone. Their results can't be directly replicated, due to their use of a proprietary parser, but other experiments suggest that the accuracy on other types of text could be lower.

7.12 Anaphora for everyone

It is potentially important to resolve anaphoric expressions, even for 'shallow' NLP tasks, such as Web search, where full parsing is impractical. An article which mentions the name 'Niall Ferguson' once, but then has multiple uses of 'he', 'the historian' etc referring to the same person is more relevant to a search for 'Niall Ferguson' than one which just mentions the name once. It is therefore interesting to see whether an algorithm can be developed which does not require parsed text. Kennedy and Boguraev (1996) describe a variant of Lappin and Leass which was developed for text which had just been tagged for part-of-speech.

The input text was tagged with the Lingsoft tagger (a very high precision and recall tagger that uses manually developed rules: see `http://www.lingsoft.fi/demos.html`). Besides POS tags, this gives some grammatical function information: e.g., it notates subjects (for English, this is quite easy to do on the basis of POS-tagged text with some simple regular expressions). The text was then run through a series of regular expression filters to identify NPs and mark expletive *it*. Heuristics defined as regular expressions are also used to identify the NPs grammatical role. Global salience factors are as in Lappin and Leass, but Kennedy and Boguraev add a factor for context (as determined by a text segmentation algorithm). They also use a distinct factor for possessive NPs.

Because this algorithm doesn't have access to a parser, the implementation of binding theory has to rely on heuristics based on the role relationships identified. Otherwise, the algorithm is much the same as for Lappin and Leass.

Overall accuracy is quoted as 75%, measured on a mixture of genres (so it isn't possible to directly compare with Lappin and Leass, since that was only tested on computer manual information). Few errors were caused by the lack of detailed syntactic information. 35% of errors were caused by failure to identify gender correctly, 14% were caused because quoted contexts weren't handled.

7.13 Another note on evaluation

The situation with respect to evaluation of anaphora resolution is less satisfactory than POS tagging or WSD. This is partly because of lack of evaluation materials such as independently marked-up corpora. Another factor is the difficulty in replication: e.g., Lappin and Leass's algorithm can't be fully replicated because of lack of availability of the parser. This can be partially circumvented by evaluating algorithms on treebanks, but existing treebanks are relatively limited in the sort of text they contain. Alternatively, different parsers can be compared according to the accuracy with which they supply the necessary information, but again this requires a suitable testing environment.

7.14 Further reading

J&M discuss the most popular approach to rhetorical relations, *rhetorical structure theory* or RST. I haven't discussed it in detail here, partly because I find the theory very unclear: attempts to annotate text using RST approaches tend not to yield good interannotator agreement (see comments on evaluation in lecture 3), although to be fair, this is a problem with all approaches to rhetorical relations. The discussion of the factors influencing anaphora resolution and the description of the Lappin and Leass algorithm that I've given here are partly based on J&M's account.

The references below are for completeness rather than suggested reading:

Lappin, Shalom and Herb Leass (1994)
An algorithm for pronominal anaphora resolution,
Computational Linguistics 20(4), 535–561

Kennedy, Christopher and Branimir Boguraev (1996)
Anaphora for everyone: pronominal anaphora resolution without a parser,
Proceedings of the 16th International Conference on Computational Linguistics (COLING 96), Copenhagen, Denmark, 113–118

8 Lecture 8: Applications

This lecture considers three applications of NLP: machine translation, spoken dialogue systems and email response. This isn't intended as a complete overview of these areas, but just as a way of describing how some of the techniques we've seen in the previous lectures are being used in current systems or how they might be used in the future.

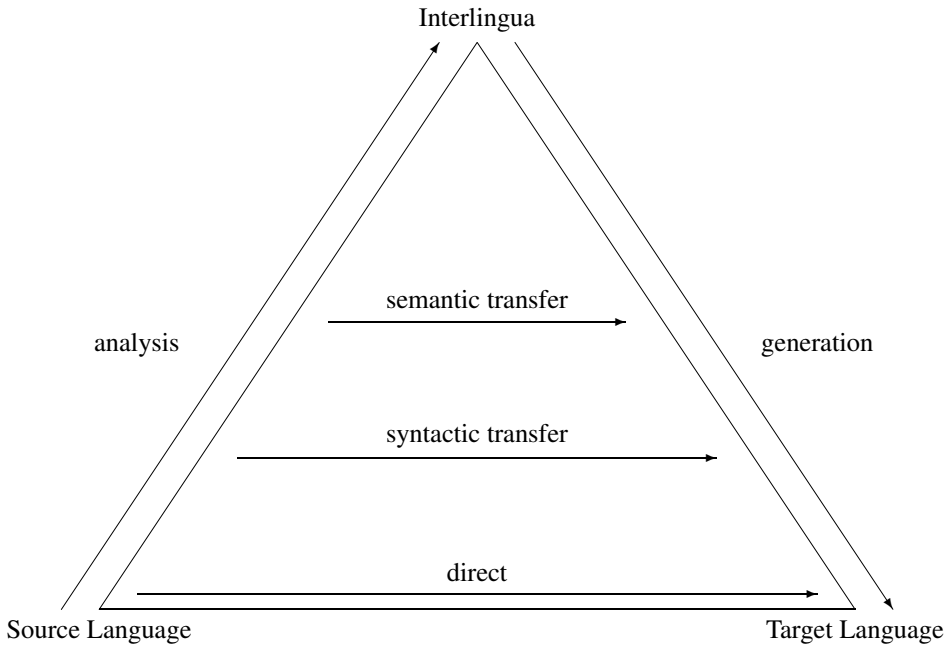
Machine translation

8.1 Methodology for MT

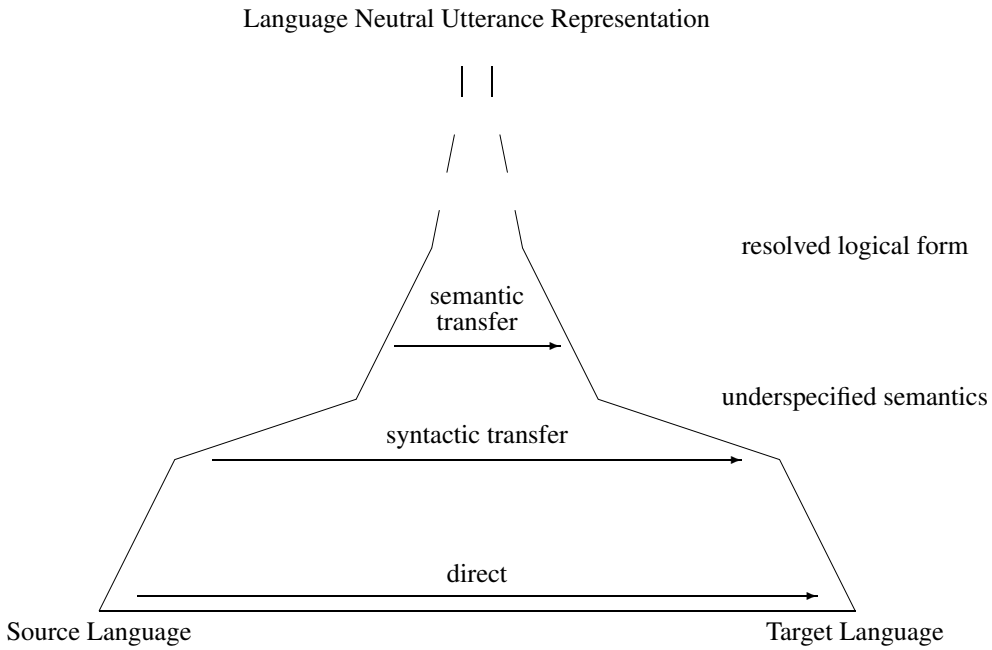
There are four main classical approaches to MT:

- Direct transfer: map between morphologically analysed structures.
- Syntactic transfer: map between syntactically analysed structures.
- Semantic transfer: map between semantics structures.
- Interlingua: construct a language-neutral representation from parsing and use this for generation.

The standard illustration of the different classical approaches to MT is the Vauquois triangle. This is supposed to illustrate the amount of effort required for analysis and generation as opposed to transfer in the different approaches. e.g., direct transfer requires very little effort for analysis or generation, since it simply involves morphological analysis, but it requires more effort on transfer than syntactic or semantic transfer do.



The Vauquois triangle is potentially misleading, because it suggests a simple trade-off in effort. It is at least as plausible that the correct geometry is as below (the Vauquois inverted funnel with very long spout):



This diagram is intended to indicate that the goal of producing a language-neutral representation may be extremely difficult!

Statistical MT involves learning translations from a *parallel corpus*: i.e. a corpus consisting of multiple versions of a single text in different languages. The classic work was done on the proceedings of the Canadian parliament (the Canadian Hansard). It is necessary to align the texts, so that sentences which are translations of each other are paired: this is non-trivial (the mapping may not be one-to-one). The original statistical MT approach can be thought of as involving direct transfer, with some more recent work being closer to syntactic (or even semantic) transfer.

Example-based MT involves using a database of existing translation pairs and trying to find the closest matching phrase. It is very useful as part of machine-aided translation.

8.2 MT using semantic transfer

Semantic transfer is an approach to MT which involves:

1. Parsing a *source language* string to produce a meaning representation
2. Transforming that representation to one appropriate for the *target language*
3. Generating from the transformed representation

Constraint-based grammars are potentially well suited to semantic transfer.

For instance:

Input:	Kim singt
Source LF:	named$(x,$ "Kim"$)$, singen(e,x)
Target LF:	named$(x,$ "Kim"$)$, sing(e,x)
Output:	Kim sings

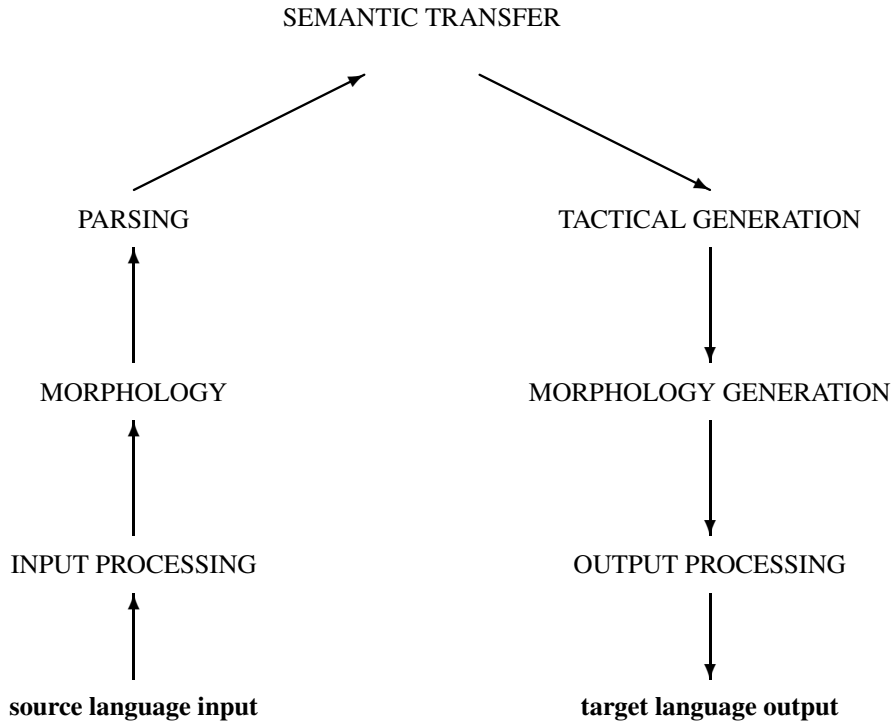
Transfer rules:

singen$(e,x) \leftrightarrow$ sing(e,x)

$\leftrightarrow$ indicates *transfer equivalence* or *translation equivalence*: the double arrow indicates reversibility:

Input:	Kim sings
Source LF:	named$(x,$ "Kim"$)$, sing(e,x)
Target LF:	named$(x,$ "Kim"$)$, singen(e,x)
Output:	Kim singt

'named' can be regarded as a language-neutral predicate, so no transfer is necessary. We also assume we don't change strings like "Kim".



Semantic transfer rules are a form of quasi-inference: they map between meaning representations. Obviously the example above was trivial: more generally some form of *mismatch* is likely to be involved, although the idea of semantic transfer is that there is less mismatch at the semantic level than at a syntactic level. Semantic transfer does not require that quantifier scope be resolved. Semantic transfer requires detailed bidirectional grammars for the languages involved, which currently makes it more suitable for high-precision, limited domain systems.

An anaphora resolution module is potentially needed when translating between languages like English and German, since English *it* can correspond to German *er*, *sie* or *es*, for instance. But the resolution should be done on an 'as-needed' basis, triggered by transfer, since it some contexts there is no ambiguity.

Some deployed MT systems use a form of semantic transfer, but syntactic transfer is more common. In these systems, generation is usually a form of text reconstruction, rather than 'proper' tactical generation. Direct transfer is used as a fallback if syntactic analysis fails. Systran uses a mixture of direct transfer and syntactic transfer: it works reasonably well because it has an enormous lexicon of phrases. Handling multiword expressions is a major problem in MT. Statistical MT is the commonest approach in the research community, followed by semantic transfer.

All MT systems require some form of WSD: potentially big improvements could be made in this area. One difficulty, however, is that MT systems often have to operate with rather small amounts of text, which limits the availability of cues.

Dialogue systems

8.3 Human dialogue basics

Turn-taking: generally there are points where a speaker invites someone else to take a turn (possibly choosing a specific person), explicitly (e.g., by asking a question) or otherwise.

Pauses: pauses between turns are generally very short (a few hundred milliseconds, but highly culture specific). Longer pauses are assumed to be meaningful: example from Levinson (1983: 300)

> A: Is there something bothering you or not? (1.0 sec pause)
> A: Yes or no? (1.5 sec pause)

A: Eh?
B: No.

Turn-taking disruption is very difficult to adjust to. This is evident in situations such as delays on phone lines and people using speech prostheses, as well as slow automatic systems.

Overlap: Utterances can overlap (the acceptability of this is dialect/culture specific but unfortunately humans tend to interrupt automated systems — this is known as *barge in*).

Backchannel: Utterances like *Uh-huh*, *OK* can occur during other speaker's utterance as a sign that the hearer is paying attention.

Attention: The speaker needs reassurance that the hearer is understanding/paying attention. Often eye contact is enough, but this is problematic with telephone conversations, dark sunglasses, etc. Dialogue systems should give explicit feedback.

Cooperativity: Because participants assume the others are cooperative, we get effects such as indirect answers to questions.

When do you want to leave?
My meeting starts at 3pm.

All of these phenomena mean that the problem of spoken dialogue understanding is very complex. This together with the unreliability of speech recognition means that spoken dialogue systems are currently only usable for very limited interactions.

8.4 Spoken dialogue systems

1. Single initiative systems (also known as system initiative systems): system controls what happens when.

   System: Which station do you want to leave from?
   User: King's Cross

   Generally very limited: for instance, in the example above the system won't accept anything that's not a station name. So it wouldn't accept *either King's Cross or Liverpool Street, depending on when the next train to Cambridge is*. Designing such systems tends to involve HCI issues (persuading the user not to complicate things), rather than language related ones.

2. Mixed initiative dialogue. Both participants can control the dialogue to some extent.

   System: Which station do you want to leave from?
   User: I don't know, tell me which station I need for Cambridge.

   The user has responded to a question with a question of their own, thereby taking control of the dialogue. Unfortunately, getting systems like this to work properly is very difficult and although research systems and a small number of practical systems have been built, performance is often better if you don't allow this sort of interaction. The term 'mixed-initiative' is often used (somewhat misleadingly) for systems which simply allow users to optionally specify more than one piece of information at once:

   System: Which day do you want to leave?
   User: the twenty-third
   OR
   User: the twenty-third of February

3. Dialogue tracking. Explicit dialogue models may improve performance in other tasks such as spoken language machine translation or summarising a human-to-human dialogue. Generally it's less critical to get everything right in such cases, which means broader domains are potentially realistic.

The use of FSAs in controlling dialogues was mentioned in lecture 2. Initial versions of simple SDSs can now be built in a few weeks using toolkits developed by Nuance and other companies: CFGs are generally hand-built for each dialogue state. This is time-consuming, but testing the SDS with real users and refining it to improve performance is probably a more serious bottleneck in deploying systems.

Email response using deep grammars

8.5 A large coverage grammar

The email response application that I mentioned in lecture 1 might be addressed using domain-specific grammars, but unlike in dialogue systems, it is much more difficult to make the limitations in the grammar obvious to the user (and if the coverage is very limited a menu-driven system might well work better). It is too expensive to manually build a new broad-coverage grammar for each new application and grammar induction is generally not feasible because the data that is available is too limited. The LinGO ERG constraint-based grammar mentioned in lecture 5 has been used for parsing in commercially-deployed email response systems. The grammar was slightly tailored for the different domains, but this mostly involved adding lexical entries. The ERG had previously been used on the Verbmobil spoken language MT task: the examples below are taken from this.

Indication of coverage of the ERG:

1. The week of the twenty second, I have two hour blocks available.
2. If you give me your name and your address we will send you the ticket.
3. Okay, actually I forgot to say that what we need is a two hour meeting.
4. The morning is good, but nine o'clock might be a little too late, as I have a seminar at ten o'clock.
5. Well, I am going on vacation for the next two weeks, so the first day that I would be able to meet would be the eighteenth
6. Did you say that you were free from three to five p.m. on Wednesday, the third, because if so that would be a perfect time for me.

Coverage was around 80% on Verbmobil.

Efficiency (with the PET system on an 850MHz CPU):

Item	Word Length	Lexical Entries	Readings	First Reading	All Readings	Passive Edges
1	12	33	15	150 ms	270 ms	1738
2	15	41	2	70 ms	110 ms	632
3	15	63	8	70 ms	140 ms	779
4	21	76	240	90 ms	910 ms	5387
5	26	87	300	1460 ms	8990 ms	41873
6	27	100	648	1080 ms	1450 ms	7850

The ERG and other similar systems have demonstrated that it is possible to use a general purpose grammar in multiple applications. However, it is crucial that there is a fallback strategy when a parse fails. For email response, the fallback is to send the email to a human. Reliability of the automated system is extremely important: sending an inappropriate response can be very costly.

A big difficulty for email response is connecting the semantics produced by the general purpose grammar to the underlying knowledge base or database. This is expensive in terms of manpower, but does not require much linguistic expertise. Hence, this sort of approach is potentially commercially viable for organisations that have to deal with a lot of fairly routine email. Although tailoring the grammar by adding lexical entries is not too hard, it is much more difficult to manually adjust the weights on grammar rules and lexical entries so that the best parse is preferred: automatic methods are definitely required here. Much less training data is required to tune a grammar than to induce one.

8.6 Further reading

J&M discuss MT and spoken dialogue systems.

A glossary/index of some of the terms used in the lectures

This is primarily intended to cover concepts which are mentioned in more than one lecture. The lecture where the term is explained in most detail is generally indicated. In some cases, I have just given a pointer to the section in the lectures where the term is defined. Note that IGE stands for *The Internet Grammar of English*, http://www.ucl.ac.uk/internet-grammar/home.htm There are a few cases where this uses a term in a slightly different way from these course notes: I have tried to indicate these.

active chart See §4.10.

adjective See IGE or notes for prelecture exercises in lecture 3.

adjunct See **argument** and also IGE.

adverb See IGE or notes for prelecture exercises in lecture 3.

affix A morpheme which can only occur in conjunction with other morphemes (lecture 2).

AI-complete A half-joking term, applied to problems that would require a solution to the problem of representing the world and acquiring world knowledge (lecture 1).

agreement The requirement for two phrases to have compatible values for grammatical features such as number and gender. For instance, in English, *dogs bark* is grammatical but *dog bark* and *dogs barks* are not. See IGE. (lecture 5)

ambiguity The same string (or sequence of sounds) meaning different things. Contrasted with **vagueness**.

anaphora The phenomenon of referring to something that was mentioned previously in a text. An anaphor is an expression which does this, such as a pronoun (see §7.5).

antonymy Opposite meaning: such as *clean* and *dirty* (§6.5).

argument In syntax, the phrases which are lexically required to be present by a particular word (prototypically a verb). This is as opposed to **adjunct**s, which modify a word or phrase but are not required. For instance, in:

> Kim saw Sandy on Tuesday

Sandy is an argument but *on Tuesday* is an adjunct. Arguments are specified by the **subcategorization** of a verb etc. Also see the IGE. (lecture 5)

aspect A term used to cover distinctions such as whether a verb suggests an event has been completed or not (as opposed to tense, which refers to the time of an event). For instance, *she was writing a book* vs *she wrote a book*.

backoff Usually used to refer to techniques for dealing with data sparseness in probabilistic systems: using a more general classification rather than a more specific one. For instance, using unigram probabilities instead of bigrams; using word classes instead of individual words (lecture 3).

baseline In evaluation, the performance produced by a simple system against which the experimental technique is compared (§3.6).

bidirectional Usable for both analysis and generation (lecture 2).

case Distinctions between nominals indicating their syntactic role in a sentence. In English, some pronouns show a distinction: e.g., *she* is used for subjects, while *her* is used for objects. e.g., *she likes her* vs **her likes she*. Languages such as German and Latin mark case much more extensively.

ceiling In evaluation, the performance produced by a 'perfect' system (such as human annotation) against which the experimental technique is compared (§3.6).

chart parsing See §4.6.

Chomsky Noam Chomsky, professor at MIT. The founder of modern theories of syntax in linguistics.

closed class Refers to parts of speech, such as conjunction, for which all the members could potentially be enumerated (lecture 3).

coherence See §7.2

collocation See §6.10

complement For the purposes of this course, an **argument** other than the subject.

compositionality The idea that the meaning of a phrase is a function of the meaning of its parts. **compositional semantics** is the study of how meaning can be built up by semantic rules which mirror syntactic structure (lecture 6).

constituent A sequence of words which is considered as a unit in a particular grammar (lecture 4).

constraint-based grammar A formalism which describes a language using a set of independently stated constraints, without imposing any conditions on processing or processing order (lecture 5).

context The situation in which an utterance occurs: includes prior utterances, the physical environment, background knowledge of the speaker and hearer(s), etc etc.

corpus A body of text used in experiments (plural *corpora*). See §3.1.

cue phrases Phrases which indicates particular **rhetorical relations**.

denominal Something derived from a noun: e.g., the verb *tango* is a denominal verb.

derivational morphology See §2.2

determiner See IGE or notes for prelecture exercises in lecture 3.

deverbal Something derived from a verb: e.g., the adjective *surprised*.

direct object See IGE. Contrast **indirect object**.

discourse In NLP, a piece of connected text.

discourse relations See **rhetorical relations**.

domain Not a precise term, but I use it to mean some restricted set of knowledge appropriate for an application.

error analysis In evaluation, working out what sort of errors are found for a given approach (§3.6).

feature structure See Lecture 5.

full-form lexicon A lexicon where all morphological variants are explicitly listed (lecture 2).

generation The process of constructing strings from some input representation. With bidirectional grammars using compositional semantics, generation can be split into **strategic generation**, which is the process of deciding on the **logical form** (also known as **text planning**), and **tactical generation** which is the process of going from the **logical form** to the string (also known as **realization**). §6.2.

generative grammar The family of approaches to linguistics where a natural language is treated as governed by rules which can produce all and only the well-formed utterances. Lecture 4.

grammar Formally, in the generative tradition, the set of rules and the lexicon. Lecture 4.

head In syntax, the most important element of a phrase.

hearer Anyone on the receiving end of an utterance (spoken, written or signed). §1.3.

homonymy Instances of **polysemy** where the two senses are unrelated (§6.8).

hyponymy An 'IS-A' relationship (§6.4) More general terms are **hypernym**s, more specific **hyponym**s.

indirect object The beneficiary in verb phrases like *give a present to Sandy* or *give Sandy a present*. In this case the indirect object is *Sandy* and the **direct object** is *a present*.

interannotator agreement The degree of agreement between the decisions of two or more humans with respect to some categorisation (§3.6).

language model A term generally used in speech recognition, for a statistical model of a natural language (lecture 3).

lemmatization Finding the stem and affixes for words (lecture 2).

lexical ambiguity Ambiguity caused because of multiple senses for a word.

lexicon The part of an NLP system that contains information about individual words (lecture 1).

linking Relating syntax and semantics in lexical entries (§6.1).

local ambiguity Ambiguity that arises during analysis etc, but which will be resolved when the utterance is completely processed.

logical form The semantic representation constructed for an utterance (§6.1).

meaning postulates Inference rules that capture some aspects of the meaning of a word.

meronymy The 'part-of' lexical semantic relation (§6.5).

morpheme Minimal information carrying units within a word (§2.1).

morphology See §1.2

MT Machine translation

multiword expression A conventional phrase that has something idiosyncratic about it and therefore might be listed in a dictionary.

mumble input Any unrecognised input in a spoken dialogue system (lecture 2).

n-gram A sequence of n words (§3.2).

named entity recognition Recognition and categorisation of person names, names of places, dates etc (lecture 4).

noun See IGE or notes for prelecture exercises in lecture 3.

noun phrase (NP) A phrase which has a noun as syntactic **head**. See IGE.

ontology In NLP and AI, a specification of the entities in a particular domain and (sometimes) the relationships between them. Often hierarchically structured.

open class Opposite of **closed class**.

orthographic rules **spelling rules** (§2.3)

overgenerate Of a grammar, to produce strings which are invalid, e.g., because they are not grammatical according to human judgements.

packing See §4.9

passive chart parsing See §4.7

parse tree See §4.4

part of speech The main syntactic categories: noun, verb, adjective, adverb, preposition, conjunction etc.

part of speech tagging Automatic assignment of syntactic categories to the words in a text. The set of categories used is actually generally more fine-grained than traditional parts of speech.

polysemy The phenomenon of words having different senses (§6.8).

pragmatics See §1.2

predicate In logic, something that takes zero or more arguments and returns a truth value. (Used in IGE for the verb phrase following the subject in a sentence, but I don't use that terminology.)

prefix An **affix** that precedes the **stem**.

probabilistic context free grammars (PCFGs) CFGs with probabilities associated with rules (lecture 4).

realization Another term for **tactical generation** — see **generation**.

referring expression See §7.5

relative clause See IGE.
A **restrictive relative clause** is one which limits the interpretation of a noun to a subset: e.g. *the students who sleep in lectures are obviously overworking* refers to a subset of students. Contrast **non-restrictive**, which is a form of parenthetical comment: e.g. *the students, who sleep in lectures, are obviously overworking* means all (or nearly all) are sleeping.

selectional restrictions Constraints on the semantic classes of arguments to verbs etc (e.g., the subject of *think* is restricted to being sentient). The term **selectional preference** is used for non-absolute restrictions.

semantics See §1.2

sign As used in lecture 5, the bundle of properties representing a word or phrase.

smoothing Redistributing observed probabilities to allow for **sparse data**, especially to give a non-zero probability to unseen events (lecture 2).

sparse data Especially in statistical techniques, data concerning rare events which isn't adequate to give good probability estimates (lecture 2).

speaker Someone who makes an **utterance** (§1.3).

spelling rules §2.3

stem A **morpheme** which is a central component of a word (contrast **affix**). §2.1.

stemming Stripping **affix**es (see §2.4).

strong equivalence Of grammars, accepting/rejecting exactly the same strings and assigning the same bracketings (contrast **weak equivalence**). Lecture 4.

structural ambiguity The situation where the same string corresponds to multiple bracketings.

subcategorization The lexical property that tells us how many *argument*s a verb etc can have.

suffix An **affix** that follows the **stem**.

summarization Producing a shorter piece of text (or speech) that captures the essential information in the original.

synonymy Having the same meaning (§6.5).

syntax See §1.2

taxonomy Traditionally, the scheme of classification of biological organisms. Extended in NLP to mean a hierarchical classification of word senses. The term **ontology** is sometimes used in a rather similar way, but ontologies tend to be classifications of domain-knowledge, without necessarily having a direct link to words, and may have a richer structure than a taxonomy.

template In feature structure grammars, see 5.6

tense Past, present, future etc.

text planning Another term for **strategic generation**: see **generation**.

training data Data used to train any sort of machine-learning system. Must be separated from test data which is kept unseen. Manually-constructed systems should ideally also use strictly unseen data for evaluation.

transfer In MT, the process of going from a representation appropriate to the original (source) language to one appropriate for the target language.

treebank a corpus annotated with trees (lecture 4).

unification See Lecture 5, especially §5.3.

weak equivalence Of grammars, accepting/rejecting exactly the same strings (contrast **strong equivalence**). Lecture 4.

Wizard of Oz experiment An experiment where data is collected, generally for a dialogue system, by asking users to interact with a mock-up of a real system, where some or all of the 'processing' is actually being done by a human rather than automatically.

WordNet See §6.6

word-sense disambiguation See §6.9

utterance A piece of speech or text (sentence or fragment) generated by a speaker in a particular context.

verb See IGE or notes for prelecture exercises in lecture 3.

verb phrase (VP) A phrase headed by a verb.

Exercises for NLP course, 2004

Notes on exercises

These exercises are organised by lecture. They are divided into two classes: prelecture and postlecture. The prelecture exercises are intended to review the basic concepts that you'll need to fully understand the lecture. Depending on your background, you may find these trivial or you may need to read the notes, but in either case they shouldn't take more than a few minutes. The first one or two examples generally come with answers, other answers are at the end (where appropriate).

Answers to the postlecture exercises are with the supervision notes (where appropriate). These are mostly intended as quick exercises to check understanding of the lecture, though some are more open-ended.

A Lecture 1

A.1 Postlecture exercises

If you use a word processor with a spelling and grammar checker, try looking at its treatment of agreement and some of the other phenomena discussed in the lecture. If possible, try switching settings between British English and American English.

B Lecture 2

B.1 Prelecture exercises

1. Split the following words into morphological units, labelling each as stem, suffix or prefix. If there is any ambiguity, give all possible splits.
   (a) dries
   answer: dry (stem), -s (suffix)
   (b) cartwheel
   answer: cart (stem), wheel (stem)
   (c) carries
   (d) running
   (e) uncaring
   (f) intruders
   (g) bookshelves
   (h) reattaches
   (i) anticipated
2. List the simple past and past/passive participle forms of the following verbs:
   (a) sing
   Answer: simple past *sang*, participle *sung*
   (b) carry
   (c) sleep
   (d) see

Note that the simple past is used by itself (e.g., *Kim sang well*) while the participle form is used with an auxiliary (e.g., *Kim had sung well*). The passive participle is always the same as the past participle in English: (e.g., *Kim began the lecture early*, *Kim had begun the lecture early*, *The lecture was begun early*).

B.2 Post-lecture exercises

1. For each of the following surface forms, give a list of the states that the FST given in the lecture notes for e-insertion passes through, and the corresponding underlying forms:

   (a) c a t s

   (b) c o r p u s

   (c) a s s e s

   (d) a s s e s s

   (e) a x e s

2. Modify the FSA for dates so that it only accepts valid months. Turn your revised FSA into a FST which maps between the numerical representation of months and their abbreviations (Jan ... Dec).

C Lecture 3

C.1 Pre-lecture

Label each of the words in the following sentences with their part of speech, distinguishing between nouns, proper nouns, verbs, adjectives, adverbs, determiners, prepositions, pronouns and others. (Traditional classifications often distinguish between a large number of additional parts of speech, but the finer distinctions won't be important here.) There are notes on part of speech distinctions below, if you have problems.

1. The brown fox could jump quickly over the dog, Rover. Answer: The/Det brown/Adj fox/Noun could/Verb(modal) jump/Verb quickly/Adverb over/Preposition the/Determiner dog/Noun, Rover/Proper noun.
2. The big cat chased the small dog into the barn.
3. Those barns have red roofs.
4. Dogs often bark loudly.
5. Further discussion seems useless.
6. Kim did not like him.
7. Time flies.

Notes on parts of speech. These notes are English-specific and are just intended to help with the lectures and the exercises: see a linguistics textbook for definitions! Some categories have fuzzy boundaries, but none of the complicated cases will be important for this course.

Noun prototypically, nouns refer to physical objects or substances: e.g., *aardvark*, *chainsaw*, *rice*. But they can also be abstract (e.g. *truth*, *beauty*) or refer to events, states or processes (e.g., *decision*). If you can say *the X* and have a sensible phrase, that's a good indication that X is a noun.

Pronoun something that can stand in for a noun: e.g., *him*, *his*

Proper noun / Proper name a name of a person, place etc: e.g., *Elizabeth*, *Paris*

Verb Verbs refer to events, processes or states but since nouns and adjectives can do this as well, the distinction between the categories is based on distribution, not semantics. For instance, nouns can occur with determiners like *the* (e.g., *the decision*) whereas verbs can't (e.g., * *the decide*). In English, verbs are often found with auxiliaries (*be*, *have* or *do*) indicating tense and aspect, and sometime occur with modals, like *can*, *could* etc. Auxiliaries and modals are themselves generally treated as subclasses of verbs.

Adjective a word that modifies a noun: e.g., *big*, *loud*. Most adjectives can also occur after the verb *be* and a few other verbs: e.g., *the students are unhappy*. Numbers are sometimes treated as a type of adjective by linguists but generally given their own category in traditional grammars. Past participle forms of verbs can also often be used as adjectives (e.g., *worried* in *the very worried man*). Sometimes it's impossible to tell whether something is a participle or an adjective (e.g., *the man was worried*).

Adverb a word that modifies a verb: e.g. *quickly*, *probably*.

Determiner these precede nouns e.g., *the*, *every*, *this*. It is not always clear whether a word is a determiner or some type of adjective.

Preposition e.g., *in*, *at*, *with*

Nouns, proper nouns, verbs, adjectives and adverbs are the *open classes*: new words can occur in any of these categories. Determiners, prepositions and pronouns are closed classes (as are auxiliary and modal verbs).

C.2 Post-lecture

Try out one or more of the following POS tagging sites:

`http://www.coli.uni-sb.de/~thorsten/tnt/`
`http://www.comp.lancs.ac.uk/computing/research/ucrel/claws/trial.html`
`http://l2r.cs.uiuc.edu/~cogcomp/eoh/posdemo.html`

Only the first site uses an approach comparable to that described in the lecture. Find two short pieces of naturally occurring English text, one of which you think should be relatively easy to tag correctly and one which you predict to be difficult. Look at the tagged output and estimate the percentage of correct tags in each case, concentrating on the open-class words. You might like to get another student to look at the same output and see if you agree on which tags are correct.

D Lecture 4

D.1 Pre-lecture

Put brackets round the noun phrases and the verb phrases in the following sentences (if there is ambiguity, give two bracketings):

1. The cat with white fur chased the small dog into the barn.
   Answer: ((The cat)$_{np}$ with (white fur)$_{np}$)$_{np}$ chased (the small dog)$_{np}$ into (the barn)$_{np}$
   The cat with white fur (chased the small dog into the barn)$_{vp}$

2. The big cat with black fur chased the dog which barked.

3. Three dogs barked at him.

4. Kim saw the birdwatcher with the binoculars.

Note that noun phrases consist of the noun, the determiner (if present) and any modifiers of the noun (adjective, prepositional phrase, relative clause). This means that noun phrases may be nested. Verb phrases include the verb and any auxiliaries, plus the object and indirect object etc (in general, the complements of the verb — discussed in lecture 5) and any adverbial modifiers. The verb phrase does not include the subject.

D.2 Post-lecture

Using the CFG given in the lecture notes (section 4.3):

1. show the edges generated when parsing *they fish in rivers in December* with the simple chart parser in 4.7
2. show the edges generated for this sentence if packing is used (as described in 4.9)
3. show the edges generated for *they fish in rivers* if an active chart parser is used (as in 4.10)

E Lecture 5

E.1 Pre-lecture

1. A very simple form of semantic representation corresponds to making verbs one-, two- or three- place logical predicates. Proper names are assumed to correspond to constants. The first argument should always correspond to the subject of the active sentence, the second to the object (if there is one) and the third to the indirect object (i.e., the beneficiary, if there is one). Give representations for the following examples:

   (a) Kim likes Sandy
   Answer: like(Kim, Sandy)

   (b) Kim sleeps

   (c) Sandy adores Kim

   (d) Kim is adored by Sandy (note, this is passive: the *by* should not be represented)

   (e) Kim gave Rover to Sandy (the *to* is not represented)

   (f) Kim gave Sandy Rover

2. List three verbs that are intransitive only, three which are simple transitive only, three which can be intransitive or transitive and three which are ditransitives.

   The distinction between intransitive, transitive and ditransitive verbs can be illustrated by examples such as:
   sleep — intransitive. No object is (generally) possible: * *Kim slept the evening.*
   adore — transitive. An object is obligatory: **Kim adored.*
   give —- ditransitive. These verbs have an object and an indirect object. *Kim gave Sandy an apple* (or *Kim gave an apple to Sandy*).

E.2 Post-lecture

1. Give the unification of the following feature structures:

   (a) $\begin{bmatrix} \text{CAT} & [\ ] \\ \text{AGR} & \textbf{pl} \end{bmatrix}$ unified with $\begin{bmatrix} \text{CAT} & \textbf{VP} \\ \text{AGR} & [\ ] \end{bmatrix}$

   (b) $\begin{bmatrix} \text{MOTHER} & \begin{bmatrix} \text{CAT} & \textbf{VP} \\ \text{AGR} & \boxed{1} \end{bmatrix} \\ \text{DTR1} & \begin{bmatrix} \text{CAT} & \textbf{V} \\ \text{AGR} & \boxed{1} \end{bmatrix} \\ \text{DTR2} & \begin{bmatrix} \text{CAT} & \textbf{NP} \\ \text{AGR} & [\ ] \end{bmatrix} \end{bmatrix}$ unified with $\begin{bmatrix} \text{DTR1} & \begin{bmatrix} \text{CAT} & \textbf{V} \\ \text{AGR} & \textbf{sg} \end{bmatrix} \end{bmatrix}$

   (c) $\begin{bmatrix} \text{F} & \boxed{1} \\ \text{G} & \boxed{1} \end{bmatrix}$ unified with $\begin{bmatrix} \text{F} & [\ \text{J} & \textbf{a}\ ] \\ \text{G} & \begin{bmatrix} \text{J} & [\ ] \\ \text{K} & \textbf{b} \end{bmatrix} \end{bmatrix}$

(d) $\begin{bmatrix} F & \boxed{1} & \mathbf{a} \\ G & \boxed{1} \end{bmatrix}$ unified with $\begin{bmatrix} G & \mathbf{b} \end{bmatrix}$

(e) $\begin{bmatrix} F & \boxed{1} \\ G & \boxed{1} \end{bmatrix}$ unified with $\begin{bmatrix} F & \begin{bmatrix} J & \mathbf{a} \end{bmatrix} \\ G & \begin{bmatrix} J & \mathbf{b} \\ K & \mathbf{b} \end{bmatrix} \end{bmatrix}$

(f) $\begin{bmatrix} F & \begin{bmatrix} G & \boxed{1} \end{bmatrix} \\ H & \boxed{1} \end{bmatrix}$ unified with $\begin{bmatrix} F & \boxed{1} \\ H & \boxed{1} \end{bmatrix}$

(g) $\begin{bmatrix} F & \boxed{1} \\ G & \boxed{1} \\ H & \boxed{2} \\ J & \boxed{2} \end{bmatrix}$ unified with $\begin{bmatrix} F & \boxed{1} \\ J & \boxed{1} \end{bmatrix}$

(h) $\begin{bmatrix} F & \begin{bmatrix} G & \boxed{1} \end{bmatrix} \\ H & \boxed{1} \end{bmatrix}$ unified with $\begin{bmatrix} F & \boxed{2} \\ H & \begin{bmatrix} J & \boxed{2} \end{bmatrix} \end{bmatrix}$

2. Add case to the initial FS grammar in order to prevent sentences such as *they can they* from parsing.
3. Work though parses of the following strings for the second FS grammar, deciding whether they parse or not:
   (a) fish fish
   (b) they can fish
   (c) it fish
   (d) they can
   (e) they fish it
4. Modify the second FS grammar to allow for verbs which take two complements. Also add a lexical entry for *give* (just do the variant which takes two noun phrases).

F Lecture 6

F.1 Pre-lecture

Without looking at a dictionary, write down brief definitions for as many senses as you can think of for the following words:

1. plant
2. shower
3. bass

If possible, compare your answers with another student's and with a dictionary.

F.2 Post-lecture

1. If you did the exercise associated with the previous lecture to add ditransitive verbs to the grammar, amend your modified grammar so that it produces semantic representations.
2. Give hypernyms and (if possible) hyponyms for the nominal senses of the following words:
   (a) horse
   (b) rice
   (c) curtain
3. List some possible seeds for Yarowsky's algorithm that would distinguish between the senses of *shower* and *bass* that you gave in the prelecture exercise.

G Lecture 7

G.1 Pre-lecture

No suggested exercises.

G.2 Post-lecture

Work through the Lappin and Leass algorithm with a short piece of naturally occurring text. If there are cases where the algorithm gets the results wrong, suggest the sorts of knowledge that would be needed to give the correct answer.

H Lecture 8

H.1 Exercises (pre- or post- lecture)

If you have never used a spoken dialogue system, try one out. One example is British Airways flight arrivals (0870 551 1155 — charged at standard national rate). Think of a realistic task before you phone: for instance, to find arrival times for a morning flight from Edinburgh to Heathrow. Another example is the Transco Meter Helpline (0870 608 1524 — standard national rate) which tells customers who their gas supplier is. This system uses a very simple dialogue model but allows for interruptions etc.

Use Systran (via `http://world.altavista.com/`) to translate some text and investigate whether the text it outputs is grammatical and whether it deals well with issues discussed in the course, such as lexical ambiguity and pronoun resolution. Ideally you would get the help of someone who speaks a language other than English for this if you're not fairly fluent in another language yourself: the language pairs that Systran deals with are listed on the site.

I Answers to some of the pre-lecture exercises

I.1 Lecture 2

1. (a) carries
   carry (stem) s (suffix)
   (b) running
   run (stem) ing (suffix)
   (c) uncaring
   un (prefix) care (stem) ing (suffix)
   (d) intruders
   intrude (stem) er (suffix) s (suffix)
   Note that in- is not a real prefix here
   (e) bookshelves
   book (stem) shelf (stem) s (suffix)
   (f) reattaches
   re (prefix) attach (stem) s (suffix)
   (g) anticipated
   anticipate (stem) ed (suffix)
2. (a) carry
   Answer: simple past *carried*, past participle *carried*
   (b) sleep
   Answer: simple past *slept*, past participle *slept*
   (c) see
   Answer: simple past *saw*, past participle *seen*

I.2 Lecture 3

1. The/Det big/Adj cat/Noun chased/Verb the/Det small/Adj dog/Noun into/Prep the/Det barn/Noun.
2. Those/Det barns/Noun have/Verb red/Adj roofs/Noun.
3. Dogs/Noun often/Adverb bark/Verb loudly/Adverb.
4. Further/Adj discussion/Noun seems/Verb useless/Adj.
5. Kim/Proper noun did/Verb(aux) not/Adverb(or Other) like/Verb him/Pronoun.
6. Time/Noun flies/Verb.
   Time/Verb flies/Noun. (the imperative!)

I.3 Lecture 4

1. The big cat with black fur chased the dog which barked.
   ((The big cat)$_{np}$ with (black fur)$_{np}$)$_{np}$ chased (the dog which barked)$_{np}$
   The big cat with black fur (chased the dog which barked)$_{vp}$
2. Three dogs barked at him. (Three dogs)$_{np}$ barked at (him)$_{np}$ Three dogs (barked at him)$_{vp}$
3. Kim saw the birdwatcher with the binoculars.
   Analysis 1 (the birdwatcher has the binoculars) (Kim)$_{np}$ saw ((the birdwatcher)$_{np}$ with (the binoculars)$_{np}$)$_{np}$
   Kim (saw the birdwatcher with the binoculars)$_{vp}$
   Analysis 2 (the seeing was with the binoculars) (Kim)$_{np}$ saw (the birdwatcher)$_{np}$ with (the binoculars)$_{np}$
   Kim (saw the birdwatcher with the binoculars)$_{vp}$

I.4 Lecture 5

1. Kim sleeps
   sleep(Kim)

2. Sandy adores Kim
   adore(Sandy, Kim)

3. Kim is adored by Sandy
   adore(Sandy, Kim)

4. Kim gave Rover to Sandy
   give(Kim, Rover, Sandy)

5. Kim gave Sandy Rover
   give(Kim, Rover, Sandy)

Some examples of different classes of verb (obviously you have almost certainly come up with different ones!)

sleep, snore, sneeze, cough — intransitive only
adore, comb, rub — simple transitive only eat, wash, shave, dust — transitive or intransitive
give, hand, lend — ditransitive