Natural Language Processing

2010, 8 Lectures, Michaelmas Term
September 13, 2010
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.
- **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.** Combination of components into applications.

Objectives

At the end of the course students should

- be able to discuss the current and likely future performance of several NLP applications;
- be able to describe briefly a fundamental technique for processing language for several subtasks, such as morphological processing, parsing, word sense disambiguation etc.;
- understand how these techniques draw on and relate to other areas of computer science.

Overview

NLP is a large and multidisciplinary field, so this course can only provide a very general introduction. The idea is that this is a 'taster' course that gives an idea of the different subfields and shows a few of the huge range of computational techniques that are used. The first lecture is designed to give an overview including 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 subdisciplines 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 is intended to give further context: it will include demos of some NLP systems. The material in Lecture 8 will not be directly examined. Slides for Lecture 8 will be made available via the course webpage after the lecture.

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 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, 2008 (second edition): referenced as J&M throughout this handout. In most cases, the first edition is still suitable, but the second edition has a much clearer description of the material covered in lecture 3 and some of the material in lecture 7 is only in the second edition. Section references given in these notes are to the second edition.

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 (or for thought). 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. The assumption is that students will have already covered this material in other contexts and that this

course will illustrate some NLP applications.

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.

Model answers to past examination questions are available to supervisors via student admin in the usual way.

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

Changes to the course since previous years.

The changes for 2010-11 are minor. I have moved some of the material from the beginning of lecture 5 to the end of lecture 4.

One significant change to the course in 2009-10 compared to previous years is that the Lappin and Leass algorithm, previously described in Lecture 7, has been replaced by a description of pronoun resolution using a classifier. This means that 2004 Paper 9 Question 14 is no longer applicable.

Most of the other changes to previous versions of these notes involve putting in more examples rather than any real changes. In 2005/2006 there was one change in terminology to make the notes easier to follow: current notes use subject (SUBJ) and object (OBJ) for syntactic roles in lecture 5 and 6 rather than specifier (SPR) and complement (COMP) as in the versions prior to 2005/2006. This means that an exam question for 2005 needs minor modification to be usable with this version of the notes: this should be obvious, but let me know of problems.

URLs

Nearly all the URLs given in these notes should be linked from:
`http://www.cl.cam.ac.uk/~aac10/stuff.html`
(apart from this one of course . . .). If any links break, I will put corrected versions there, if available.

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 computational modelling 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'. The term '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). Like NLP, formal linguistics deals with the development of models of human languages, but the currently dominant approaches in linguistics reject the validity of statistical techniques, which are seen as an essential part of computational linguistics. NLP 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 (a word 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 are 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	2/2/09	2/2/09
4291	2/2/09	2/2/09
4292	2/2/09	

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 2/2/09

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 essentially mean the same in this context.

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

Natural language has properties which are essential to communication which are not found in formal languages, such as predicate calculus, computer programming languages, semantic web languages and so on. Natural language is incredibly flexible. It is learnable, but compact. Natural languages are emergent, evolving systems. Ambiguity and synonymy are inherent to flexibility and learnability. Despite ambiguity, natural language can be indefinitely precise: ambiguity is largely local[1] (at least for humans) and natural languages accommodate (semi-)formal additions.

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.[2] 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.[3] 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 either work with very limited domains or approximate full world knowledge by relatively simple techniques.

[1] i.e., immediate context resolves the ambiguity: examples of this will be discussed in later lectures.

[2] 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.

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

1.4 Some NLP applications

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

- 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 Sentiment classification

Politicians want to know what people think about them. Companies want to know what users think about their products. Extracting this sort of information from the Web is a huge and lucrative business but much of the work is still done by humans who have to read through the relevant documents and classify them by hand, although automation is increasingly playing a role. The full problem involves finding all the references to an entity from some document set (e.g., all newspaper articles appearing in September 2010), and then classifying them as positive, negative or neutral. Customers want to see summaries of the data (e.g., to see whether popularity is going up or down), but may also want to see actual examples (text snippets). Companies may want a fine-grained classification of aspects of their product (e.g., laptop batteries, MP3 player screens).

The full problem involves retrieving relevant text, recognition of *named entities* (e.g., *Sony 505G, Hilary Clinton, 2,4-dinitrotoluene*) and of parts of the text that refer to them. But academic researchers have looked at a simpler version of sentiment classification by starting from a set of documents which are already known to be opinions about a particular topic or entity (e.g., reviews) and where the problem is just to work out whether the author is expressing positive or negative opinions. This still turns out to be hard for computers, though generally easy for humans, especially if neutral reviews are excluded from the data set (as is often done). Much of the work has been done on movie reviews. The rating associated with each review is known (5 stars, 1 star or whatever), so there is an objective standard as to whether the review is positive or negative. The research problem is to guess this automatically over the entire corpus.[4]

The most basic technique is to look at the words in the review in isolation of each other, and to classify the document on the basis of whether those words generally indicate positive or negative reviews. This is a *bag of words* technique: we model the document as an unordered collection of words (*bag* rather than set because there will be repetition). A document with more positive words than negative ones should be a positive review. In principle, this could be done by using human judgements of positive/negative words, but using machine learning techniques works better[5] (humans don't consider many words that turn out to be useful indicators). However, the accuracy of the classification is only around 80% (for a problem where there is a 50% chance success rate).[6] One source of errors is negation: (e.g., *Ridley Scott has never directed a bad film* is a positive statement). Another problem is that the machine learning technique may match the data too closely: e.g., if the machine learner is trained on reviews which include a lot of films from before 2005, it may decide that *Ridley* is a strong positive indicator but then tend to misclassify reviews for 'Kingdom of Heaven'. More subtle problems arise from not tracking the contrasts in the discourse:

> This film should be brilliant. It sounds like a great plot, the actors are first grade, and the supporting cast is good as well, and Stallone is attempting to deliver a good performance. However, it can't hold up.

Another example:

> AN AMERICAN WEREWOLF IN PARIS is a failed attempt ... Julie Delpy is far too good for this movie. She imbues Serafine with spirit, spunk, and humanity. This isnt necessarily a good thing, since it prevents us from relaxing and enjoying AN AMERICAN WEREWOLF IN PARIS as a completely mindless, campy entertainment experience. Delpys injection of class into an otherwise classless production raises the specter of what this film could have been with a better script and a better cast ... She was radiant, charismatic, and effective ...

Both examples are from Pang et al (2002).

Unfortunately, although in principle NLP techniques can deal with syntax, semantics and discourse and thus address these sort of problems, doing this in a way that can significantly improve performance over the simple system turns out to be (very) hard. To understand whether a statement is positive or negative is ultimately AI-complete: the real question is whether automatic methods are good enough on the easy cases to be useful.

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.

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

[4] A *corpus* (plural *corpora*) is the technical term for a body of text that has been collected for some purpose, see §3.1.

[5] Classifiers are discussed in more detail in lecture 7.

[6] Pang, Lee and Vaithyanatha (2002), *Thumbs up? Sentiment Classification using Machine Learning Techniques* In Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP).

There are some question-answering systems on the Web, but most use very basic techniques. One common approach involves employing a large staff of people who search the web to find pages which are answers to potential questions. The question-answering system performs very limited manipulation on the actual 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 powers BabelFish `http://babelfish.yahoo.com/` and many other translation services on the web. Systran was used for most of the language pairs available from Google until about 2007/2008, but Google now uses a *statistical MT* system which was developed in-house, exploiting Google's access to the huge amount of *parallel text* available on the web (i.e., source documents which are available alongside translations).

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 originally used teams of skilled typists to input Russian documents.

None of these systems are 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 important now that documents are available on the Web.

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: natural language interface 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 some 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 computer science 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 approaches 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. By this point, there was a huge divide between mainstream linguists and the NLP community. Chomsky had declared:

> But it must be recognized that the notion 'probability of a sentence' is an entirely useless one, under any known interpretation of this term. (Chomsky 1969)

Certain linguistics journals would not even review theoretical linguistics papers which had a quantitative component. But speech and NLP researchers wanted results:

> Whenever I fire a linguist our system performance improves. (Fred Jelinek, said at a workshop in 1988 (probably), various forms of the quotation have been attested. He has said he never actually fired anyone.)

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 recognition 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.[7] 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 of acquiring training data and in adapting to new types of text. However, modern linguistically-based NLP approaches are making use of machine learning and statistical processing.

The dotcom boom and bust at the turn of the millenium considerably affected NLP in industry but interest increased again more recently. The ubiquity of the Internet has completely changed the space of interesting NLP applications since the early 1990s, and the vast amount of text available can potentially be exploited, especially for statistical techniques.

[7] 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!

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 and email question answering.

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 systems 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[8] as the parser.
- morphological generation: as with morphological analysis, this is relatively straightforward for English.
- 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 NL interfaces, email answering and so on, we need an interface between the semantic representation output by the parser (or accepted by the generator) and the underlying knowledge base. Other types of application have different requirements.

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

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

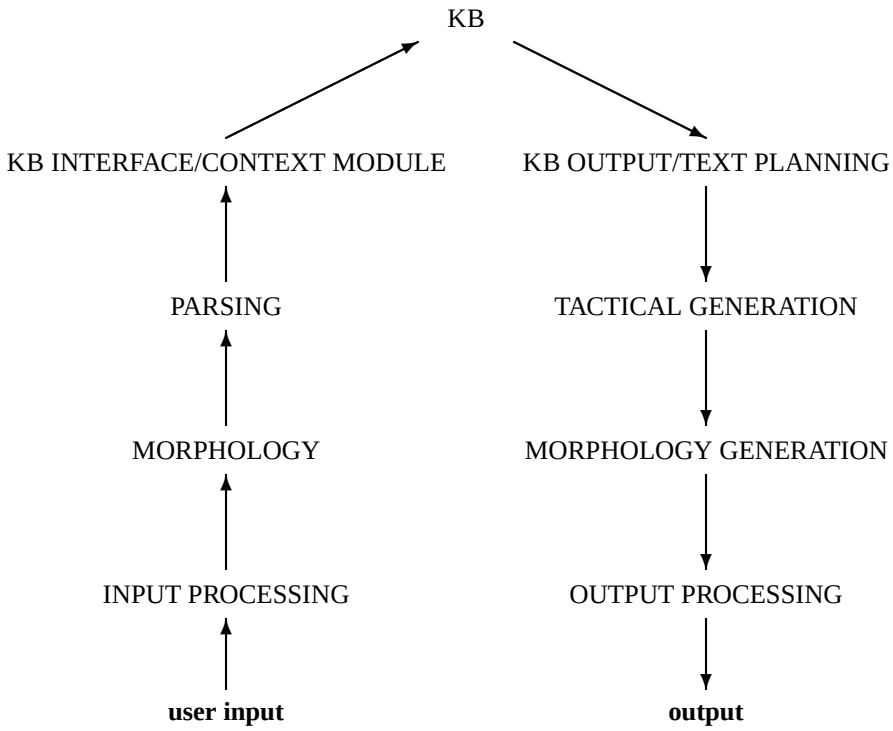
- acquisition of statistical information

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 a few applications in some more detail.

1.12 General comments

- Even 'simple' NLP applications 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 still isn't very good, but can be useful.
- 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 been driven as much by hardware and software advances, and societal and infrastructure changes as 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*. The compound *bookshop* has two stems (*book* and *shop*): most English compounds are spelled with a space, however. 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, such as the *ge-t* in German *gekauft*). 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*).[9]

Note the requirement that a morpheme can be regarded as a unit. There are cases where there seems to be a similarity in meaning between some clusters of words with similar spellings: e.g., *slink*, *slide*, *slither*, *slip*. But such examples cannot be decomposed (i.e., there is no *sl-* morpheme) because the rest of the word does not stand as a unit.

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* (i.e., there is a fixed set of slots which can be thought of as being filled with simple values). 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 (there seems no principled limit on what they can mean) 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 (mistaken) 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

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

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))*.

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, I'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*).[10]

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 3.8) without the need for a lexicon. However, stemming does not necessarily help IR. Search engines now generally do inflectional morphology, but this can be dangerous. For instance, searching for *corpus* as well as *corpora* when given the latter as input (as some search engines sometimes do) results 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 I 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

[10] Note the use of * ('star') above: this notation is used in linguistics to indicate a word or 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.

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.[11] 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 and form the interface between morphology and the syntax/semantics: I'll discuss this briefly in §5.6.

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*.

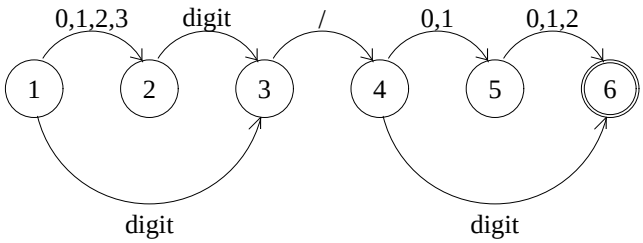
Morphological analysers also generally have access to a lexicon of regular stems. This is needed for high precision: e.g. to avoid analysing *corpus* as *corpu -s*, we need to know that there isn't a word *corpu*. There are also cases where historically a word was derived, but where the base form is 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 I'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 §5.6 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.

[11] 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.

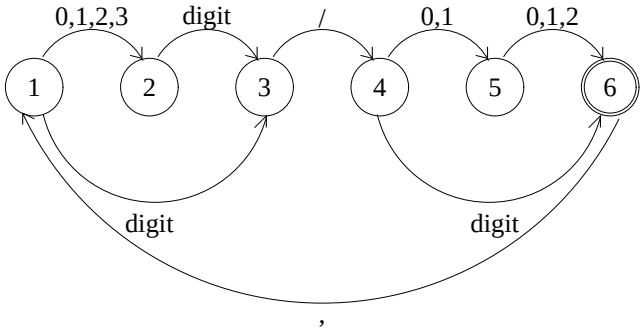
2.6 Finite state automata for recognition

The approach to spelling rules that I'll describe involves the use of finite state transducers (FSTs). Rather than jumping straight into this, I'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:



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.

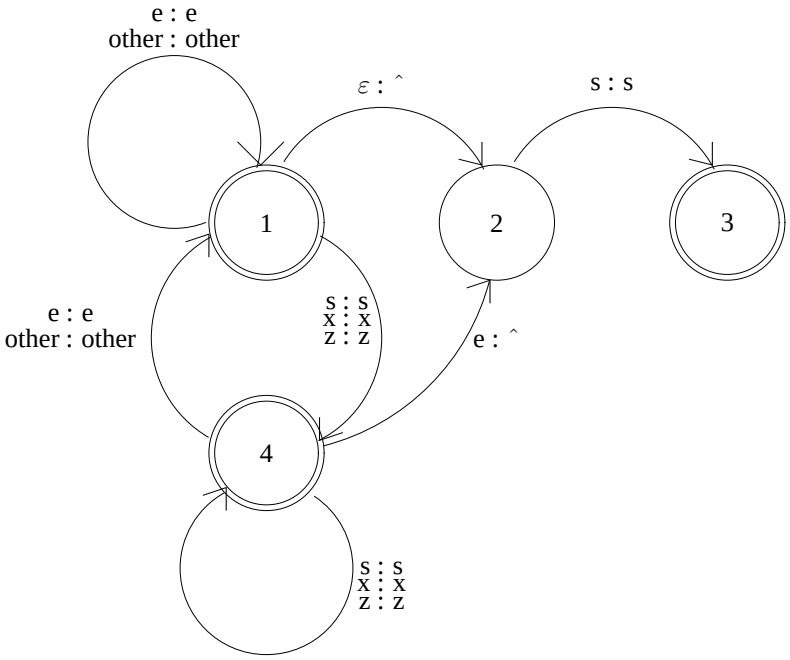
2.7 Finite state transducers

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 and repeated below.[12]

[12] Actually, I've simplified this slightly so the FST works correctly but 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. They also use an explicit terminating character while I prefer to rely on the 'use all the input' convention, which results in simpler rules.

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



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.[13] 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, the surface form *cakes* would start from 1 and go through the transitions/states (c:c) 1, (a:a) 1, (k:k) 1, (e:e) 1, (ε:^) 2, (s:s) 3 (accept, underlying *cake^s*) and also (c:c) 1, (a:a) 1, (k:k) 1, (e:e) 1, (s:s) 4 (accept, underlying *cakes*). 'd o g s' maps to 'd o g ^ s', 'f o x e s' maps to 'f o x ^ s' and to 'f o x e ^ s', and 'b u z z e s' maps to 'b u z z ^ s' and 'b u z z e ^ s'.[14] 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: I'll discuss this a little in §4.12. However, for very simple spoken dialogue systems, a finite-

[13] The solution notes for the 2003 FST question are slightly wrong in that they should have y : y as well as other : other on one transition.

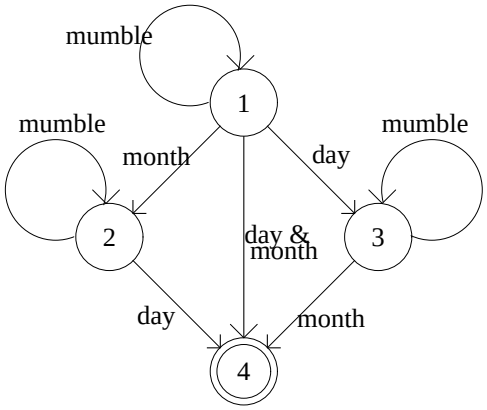
[14] In all cases they also map to themselves: e.g., 'b u z z e s' maps to 'b u z z e s' without the affix marker: this is necessary because words ending in 's' and 'es' are not always inflected forms. e.g., *Moses*

state grammar may be adequate. More complex grammars can be written as context free grammars (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:

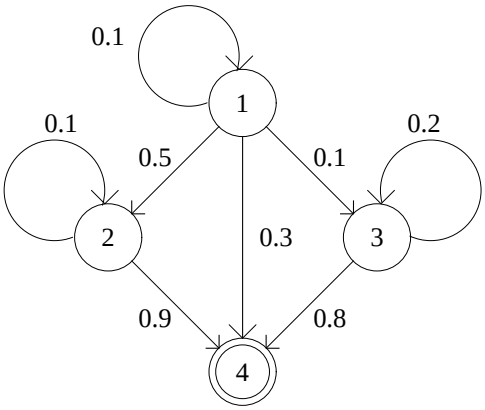
  1. No information. System prompts for month and day.
  2. Month only is known. System prompts for day.
  3. Day only is known. System prompts for month.
  4. Month and day known.

  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. 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 in the past tended to dismiss their use in favour of reliance on intuitive judgements about whether or not an utterance is grammatical. A corpus can only (directly) provide positive evidence about grammaticality. Many linguists are gradually coming round to their use. Corpora are essential for most 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 I'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.[15] 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

[15] 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*. If a recording is made of normal, fluently spoken, speech and the segments corresponding to *the* and *a* are presented to a subject in isolation, it's generally not possible to tell the difference.

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. Training a conversational aid on newspaper text can be worse than using a unigram model from the user's own data.

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 vastly easier in situations where the speaker is only saying two easily distinguishable words (e.g., when a dialogue system prompts by saying *answer 'yes' or 'no'*) 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 alone: i.e., $P(w_n|w_{n-1})$ (the probability of w_n conditional on w_{n-1}) where w_n is the nth word in some string. 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. However, for speech recognition and similar applications, we require the probability of some string of words $P(w_1^n)$ which is approximated by the product of the bigram probabilities:

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})$$

We acquire these probabilities from a corpus. For example, suppose we have the following tiny corpus of utterances:

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

I'll use the symbol ⟨s⟩ to indicate the beginning of the sentence and ⟨/s⟩ to indicate the end, so the corpus really looks like:

⟨s⟩ good morning ⟨/s⟩ ⟨s⟩ good afternoon ⟨/s⟩ ⟨s⟩ good afternoon ⟨/s⟩ ⟨s⟩ it is very good ⟨/s⟩ ⟨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               2
it is            2       1
is               2
is very          1       .5
is good          1       .5
very             1
very good        1       1
</s>             5
</s><s>          4       1
```

This yields a probability of 0.24 for the string '⟨s⟩ good ⟨/s⟩' and also for '⟨s⟩ good afternoon ⟨/s⟩'.

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 dynamic programming technique for efficiently applying n-grams in speech recognition and other applications to find the highest probability sequence (or sequences), is usually described in terms of an FSA.

The probability of '⟨s⟩ very good ⟨/s⟩' 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 I won't discuss the details in this course.

3.4 Part of speech tagging

Sometimes we are interested in a form of prediction that involves assigning classes to items in a sequence rather than predicting the next item. 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 Figure 5.9 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' and in similar contexts)
```

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_VVI ._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. They can also be used as part of a method for dealing with words which are not in the parser's lexicon (unknown words).

3.5 Stochastic POS tagging using Hidden Markov Models

One form of POS tagging uses a technique known as *Hidden Markov Modelling* (HMM). It involves an n-gram technique, but in this case the n-grams are sequences of POS tags rather than of words. The most common approaches depend on a small amount of manually tagged *training data* from which POS n-grams can be extracted.[16] I'll illustrate this with respect to another trivial corpus:

They used to can fish in those towns. But now few people fish in these areas.

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
```

[16] 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.

```
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
TO0 VVI         1               1

VVB             1
VVB PRP         1               1

VVD             1
VVD TO0         1               1

VVI             1
VVI NN2         1               1
```

I have used the correct PUN CJC probability, allowing for the final PUN. 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 more complicated than the word prediction case because we have to take into account both words and tags.

We wish to produce a sequence of tags which have the maximum probability given a sequence of words. I will follow J&M's notation: the hat, $\hat{}$, means "estimate of", so $\hat{t}_1^n$ means "estimate of the sequence of n tags", and $\underset{x}{\operatorname{argmax}} f(x)$

means “the x such that f(x) is maximized”. Hence:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}}\, P(t_1^n|w_1^n)$$

We can’t estimate this directly (mini-exercise: explain why not). By Bayes theorem:

$$P(t_1^n|w_1^n) = \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

Since we’re looking at assigning tags to a particular sequence of words, $P(w_1^n)$ is constant, so for a relative measure of probability we can use:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}}\, P(w_1^n|t_1^n)P(t_1^n)$$

We now have to estimate $P(t_1^n)$ and $P(w_1^n|t_1^n)$. If we make the bigram assumption, then the probability of a tag depends on the previous tag, hence the tag sequence is estimated as a product of the probabilities:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1})$$

We will also assume that the probability of the word is independent of the words and tags around it and depends only on its own tag:

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i)$$

These values can be estimated from the corpus frequencies. So our final equation for the HMM POS tagger using bigrams is:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

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 $P(\text{NN2}|\text{PNP})$, $P(\text{fish}|\text{NN2})$, $P(\text{VVB}|\text{PNP})$ and $P(\text{fish}|\text{VVB})$ (assuming PNP is the only possible tag for ***they***).

As the equation above indicates, in order to POS tag a sentence, we maximise the overall tag sequence probability (again, this can be implemented efficiently using the Viterbi algorithm). So a tag which has high probability considering its individual bigram estimate will not be chosen if it does not form part of the highest probability path. For example:
they_PNP can_VVB fish_NN2
they_PNP can_VM0 fish_VVI
The product of $P(\text{VVI}|\text{VM0})$ and $P(\text{fish}|\text{VVI})$ may be lower than that of $P(\text{NN2}|\text{VVB})$ and $P(\text{fish}|\text{NN2})$ but the overall probability depends also on $P(\text{can}|\text{VVB})$ versus $P(\text{can}|\text{VM0})$ and the latter (modal) use has much higher frequency in a balanced corpus.

In fact, POS taggers generally use trigrams rather than bigrams — the relevant equations are given in J&M, 5.5.4. As with word prediction, backoff (to bigrams) and smoothing are crucial for reasonable performance because of sparse data.

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 (without a lexicon) 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 return 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 such as the one described here 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 since some perform at higher accuracy). 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. Fatigue can cause errors, even with very experienced annotators. 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

N-grams are described in Chapter 4 of J&M, POS tagging in Chapter 5. The description in the second edition is considerably clearly than that in the first edition.

4 Lecture 4: Parsing and generation

In this lecture, I'll discuss syntax in a way which is much closer to the standard notions in formal linguistics than POS-tagging is. To start with, I'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 see 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 a complete grammar of this type 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 big 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, I'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 I'll ignore for now, though we'll see some of them at the end of the lecture. Notice that for this fragment there is no distinction between main verb *can* and the modal verb *can*.

```
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 right hand side 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))))

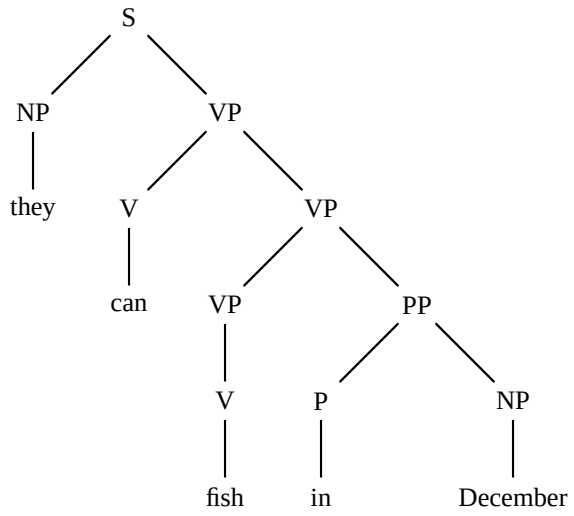
they fish in rivers in December
(S (NP they) (VP (VP (V fish)) (PP (P in) (NP (NP rivers) (PP (P in) (NP December))))))
;;; i.e. the implausible reading where the rivers are in December
;;; (cf rivers in Scotland)
(S (NP they) (VP (VP (VP (V fish)) (PP (P in) (NP rivers))) (PP (P in) (NP December))))
;;; 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*). Humans disambiguate such attachments as they hear a sentence, but they're relying on the meaning in context to do this, in a way we cannot currently emulate, except when the sentences are restricted to a very limited domain.

Notice that *fish* could have been entered in the lexicon directly as a VP, but that this would cause problems if we were doing inflectional 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 random 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 I'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/memoization 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*.[17] Chart parsing strategies are designed to be *complete*: that is, if there is a valid analysis according to a grammar, the chart parser will find it.

A chart is a collection of *edges*, usually implemented as a vector of edges, indexed by edge identifiers. 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.

[17] Natural languages have vastly higher degrees of ambiguity than programming languages: chart parsing is well-suited to this.

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 6)
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. Informally, it proceeds by adding the next word (in left to right order), and adding each lexical category possible for that word, doing everything it can immediately after each lexical category is added. 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)
    (i.e., find all edges that match a rule)
    For each list of edges, **Add new edge** $from_1$, *to, lhs,* ($id_1 \ldots id$)
    (i.e., apply the rule to the edges)

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)
```

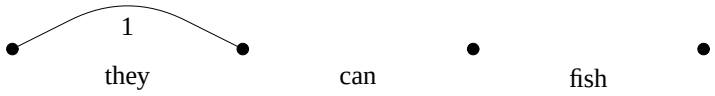
The spanning edges are 11 and 8: the output routine to give bracketed parses simply outputs a left bracket, outputs the category, recurses through each of the daughters and then outputs a right bracket. So, for instance, the output from edge 11 is:

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

This chart parsing algorithm is *complete*: it returns all possible analyses, except in the case where it does not terminate because there is a recursively applicable rule.

4.8 A detailed trace of the simple chart parser

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

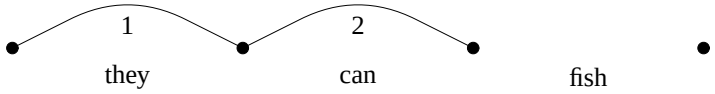


Matching grammar rules are:

```
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)

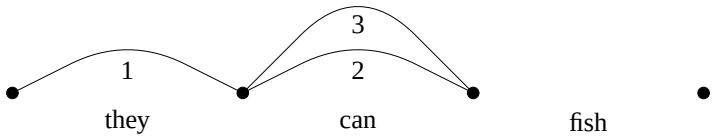


Matching grammar rules are:

```
VP -> V
```

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

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

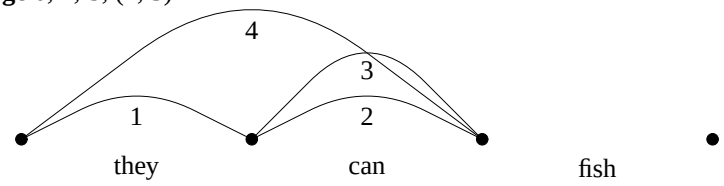


Matching grammar rules are:

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

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

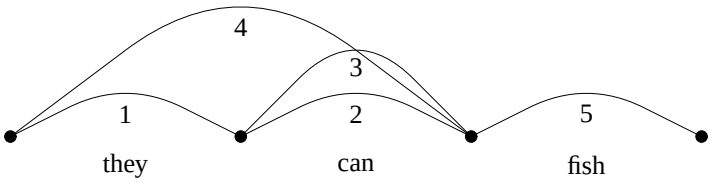
Add new edge 0, 2, S, (1, 3)



No matching grammar rules for S

No edges matching V VP

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

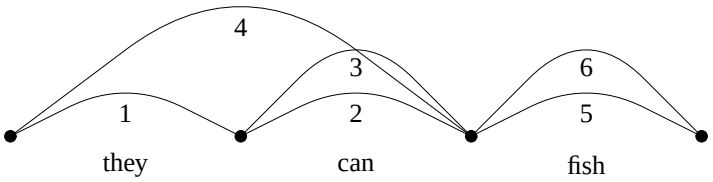


Matching grammar rules are:

```
VP -> V
```

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

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

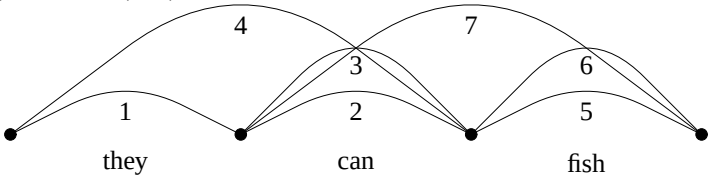


Matching grammar rules are:

```
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)

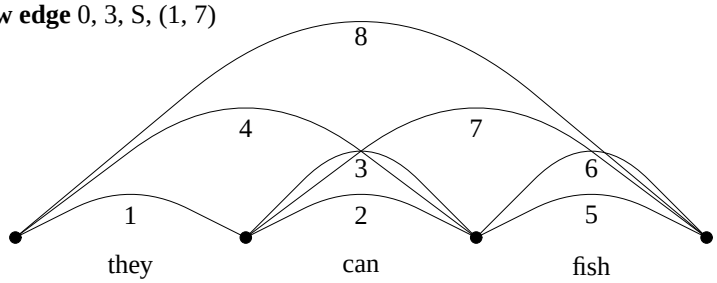


Matching grammar rules are:

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

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

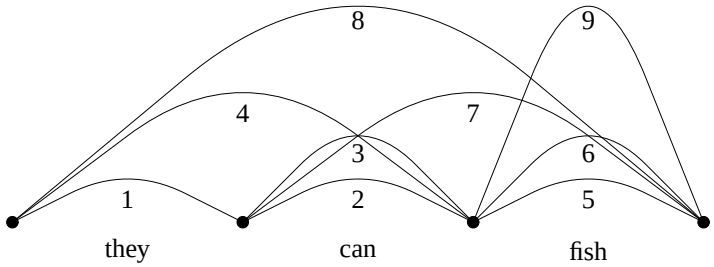
Add new edge 0, 3, S, (1, 7)



No matching grammar rules for S

No edges matching V

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

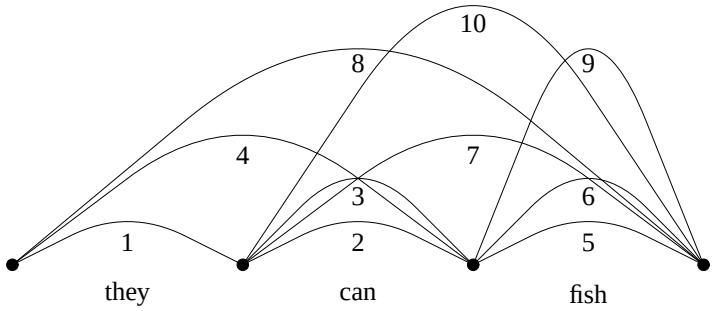


Matching grammar rules are:

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

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

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

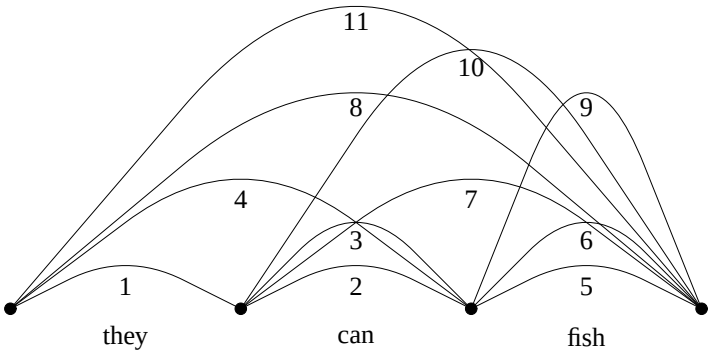


Matching grammar rules are:

```
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* ⊔ *daughters*]

There is no need to recurse with this edge, because we couldn't get any new results: once we've found we can pack an edge, we always stop that part of the search. Thus packing saves computation and in fact leads to cubic time operation, though I won't go through the proof of this.

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. Active edges are stored on the chart as well as passive edges. For instance, with an active chart parser, we might have the following edges when parsing a sentence starting *they fish*:

```
id  left   right  mother    expected      daughters
1    0       1      NP                     (they)
2    0       1      S         VP           (1 ?)
3    0       1      NP        PP           (1,?)
4    1       2      V                      (fish)
5    1       2      VP                     (4)
6    0       2      S                      (2,5)
7    1       2      VP        NP           (4,?)
8    1       2      VP        VP           (4,?)
9    1       2      VP        PP           (5,?)
```

Edge 1 is complete (a passive edge). Edge 2 is active: the daughter marked as ? will be instantiated by the edge corresponding to the VP when it is found (e.g., edge 5 instantiates the active part of edge 2 to give edge 6).

Each word gives rise to a passive edge. Each passive edge of category C gives rise to active edges corresponding to rules with leftmost daughter C (although there are various possible pruning strategies than can be used to cut down on spurious active edges). Every time a passive edge is added, the active edges are searched to see if the new passive edge can complete an active edge.

I will not give full details of the active chart parser here: there are several possible variants. The main thing to note is that active edges may be used to create more than one passive edge. For instance, if we have the string *they fish in Scotland*, edge 2 will be completed by *fish* and also by *fish in Scotland*. Whether this leads to a practical improvement in efficiency depends on whether the saving in time that results because the NP is only combined with the S rule once outweighs the overhead of storing the edge. Active edges may be packed. Active chart parsing is generally more efficient than passive parsing for feature structure grammars (explained in the next lecture) because there is some cost associated with combining a daughter with a rule.

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, since the beginning of the 1990s, a lot of work has been done on automatically acquiring probabilities from a corpus annotated with syntactic 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 tens of 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 section, I'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 Deficiencies in atomic category CFGs

If we consider the sample grammar in §4.3, several problems are apparent. One is that there is no account of subject-verb agreement, so, for instance, **it fish* is allowed by the grammar as well as *they fish*.[18]

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
NP-sg -> he
NP-sg -> fish
NP-pl -> fish
```

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 (e.g., past tense verbs).

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). Subcategorization tends to mirror semantics, although there are many complications. A verb such as *adore*, for instance, relates two entities and 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 in lecture 4 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, each of these sentences is said to contain a *gap*, 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 trillions of rules. Instead of having simple atomic categories in the CFG, we want to allow for features on the categories, which can have values

[18] 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. The subject and the verb must (usually) either both have singular morphology or both have plural mophology: i.e., they must *agree*. 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*).

indicating things like plurality. As the long-distance dependency examples should indicate, the features need to be complex-valued. For instance,

> * 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 the next lecture, I will illustrate a simple *constraint-based grammar* formalism, using *feature structures* which allows us to encode these phenomena.

4.14 Further reading

This lecture has described material which J&M discuss in chapters 12 and 13, though we also touched on PCFGs (covered in their chapter 14) and issues of language complexity which they discuss in chapter 16. I chose to concentrate on bottom-up chart parsing in this lecture, mainly because I find it easier to describe than the Earley algorithm and the full version of chart parsing given in J&M, 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

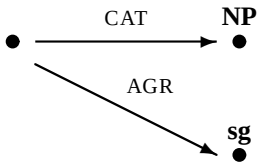
As discussed at the end of the last lecture, the simple CFG approach which we've looked at so far has some serious deficiencies as a model of natural language. In this lecture, I'll give an introduction to a more expressive formalism which is widely used in NLP, again with the help of a sample grammar. As I outlined in the last lecture, instead of simple atomic categories in the CFG, we will allow for features on the categories, which can have values indicating things like plurality. To allow for long-distance dependency examples, the features need to be complex-valued. In the first part of lecture 6, I will go on to sketch how we can use this approach to do compositional semantics.

The formalism we will look at is a type of constraint-based grammar: a *feature structure* grammar. 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).

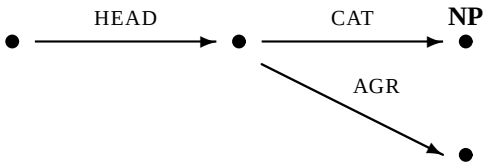
5.1 A very simple FS grammar encoding agreement

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 **NP** 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:

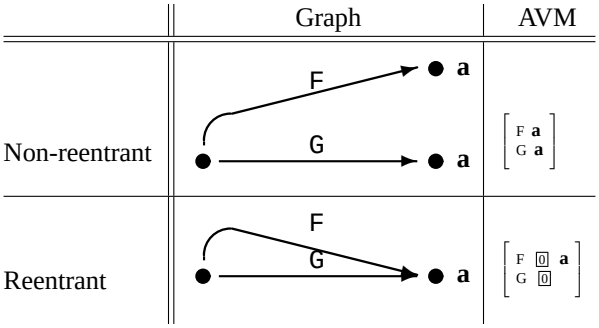


FSs are usually drawn as *attribute-value matrices* or AVMs. The AVMs corresponding to the two FSs above are as follows:

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

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

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 replaces the atomic categories with FSs, splitting 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.[19] 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

Subject-verb rule $\begin{bmatrix} \text{CAT } \mathbf{S} \\ \text{AGR } \boxed{1} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT } \mathbf{NP} \\ \text{AGR } \boxed{1} \end{bmatrix}, \begin{bmatrix} \text{CAT } \mathbf{VP} \\ \text{AGR } \boxed{1} \end{bmatrix}$

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

Lexicon:

;;; noun phrases

[19] Note that the reentrancy indicators are local to each rule: the [1] in the subject-verb rule is not the same structure as the [1] in the verb-object rule.

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

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

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

;;; verbs

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

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

Root structure:

$\begin{bmatrix} \text{CAT} & \textbf{S} \end{bmatrix}$

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 the verb-object rule. Both unifications succeed, and the structure corresponding to the mother of the rule is:

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

The agreement value is **pl** because of the reentrancy with the agreement value of *like*. This structure can unify with the rightmost daughter of the subject-verb rule. The structure for *they* is unified with the leftmost daughter. The subject-verb rule says that both daughters have to have the same agreement value, which is true in this example. Rule application therefore succeeds and since the result unifies with the root 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 the verb-object rule, which I informally wrote as:

$$\begin{bmatrix} \text{CAT} & \textbf{VP} \\ \text{AGR} & \boxed{1} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} & \textbf{V} \\ \text{AGR} & \boxed{1} \end{bmatrix}, \begin{bmatrix} \text{CAT} & \textbf{NP} \\ \text{AGR} & \begin{bmatrix} \end{bmatrix} \end{bmatrix}$$

is actually:

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

Thus the rules in the CFG correspond to FSs in this formalism and we can formalise rule application by unification. For instance, a rule application in bottom-up parsing involves unifying each of the DTR slots in the rule with the feature structures for the phrases already in the chart.

Consider parsing *they like it* again.
STEP1: parsing *like it* with the rule above.
Step 1a
The structure for *like* can be unified with the value of DTR1 in the rule.

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

Unification means all information is retained, so the result includes the agreement value from *like*:

$$\begin{bmatrix} \text{MOTHER} \begin{bmatrix} \text{CAT } \mathbf{VP} \\ \text{AGR } \boxed{1}\ \mathbf{pl} \end{bmatrix} \\ \text{DTR1} \begin{bmatrix} \text{CAT } \mathbf{V} \\ \text{AGR } \boxed{1} \end{bmatrix} \\ \text{DTR2} \begin{bmatrix} \text{CAT } \mathbf{NP} \\ \text{AGR } [\ ] \end{bmatrix} \end{bmatrix}$$

Step 1b
The structure for *it* is unified with the value for DTR2 in the result of Step 1a:

$$\begin{bmatrix} \text{MOTHER} \begin{bmatrix} \text{CAT } \mathbf{VP} \\ \text{AGR } \boxed{1}\ \mathbf{pl} \end{bmatrix} \\ \text{DTR1} \begin{bmatrix} \text{CAT } \mathbf{V} \\ \text{AGR } \boxed{1} \end{bmatrix} \\ \text{DTR2} \begin{bmatrix} \text{CAT } \mathbf{NP} \\ \text{AGR } [\ ] \end{bmatrix} \end{bmatrix} \sqcap \begin{bmatrix} \text{DTR2} \begin{bmatrix} \text{CAT } \mathbf{NP} \\ \text{AGR } \mathbf{sg} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \text{MOTHER} \begin{bmatrix} \text{CAT } \mathbf{VP} \\ \text{AGR } \boxed{1}\ \mathbf{pl} \end{bmatrix} \\ \text{DTR1} \begin{bmatrix} \text{CAT } \mathbf{V} \\ \text{AGR } \boxed{1} \end{bmatrix} \\ \text{DTR2} \begin{bmatrix} \text{CAT } \mathbf{NP} \\ \text{AGR } \mathbf{sg} \end{bmatrix} \end{bmatrix}$$

The rule application thus succeeds.

Step 2: application of the subject verb rule.
Step 2a.
The MOTHER value acts as the DTR2 of the subject-verb rule. That is:

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

is unified with the DTR2 value of:

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

This gives:

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

Step 2b
The FS for *they* is:

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

The unification of this with the value of DTR1 from Step 2a succeeds but adds no new information:

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

Step 3:
Finally, the MOTHER of 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 at Step 2b, 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.

5.2 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. (Although 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 an atomic 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.

5.3 A grammar enforcing subcategorization

Although the grammar shown above improves on the simple CFG, it still doesn't encode subcategorization (e.g., the difference between transitive and intransitive). The grammar shown below 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, OBJ and SUBJ.

Simple FS grammar fragment encoding subcategorization

Subject-verb rule

$$\begin{bmatrix} \text{HEAD } \boxed{1} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \textbf{filled} \end{bmatrix} \rightarrow \boxed{2} \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{AGR } \boxed{3} \end{bmatrix} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \textbf{filled} \end{bmatrix}, \begin{bmatrix} \text{HEAD } \boxed{1} \begin{bmatrix} \text{AGR } \boxed{3} \end{bmatrix} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \boxed{2} \end{bmatrix}$$

Verb-object rule

$$\begin{bmatrix} \text{HEAD } \boxed{1} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \text{HEAD } \boxed{1} \\ \text{OBJ } \boxed{2} \\ \text{SUBJ } \boxed{3} \end{bmatrix}, \boxed{2} \begin{bmatrix} \text{OBJ } \textbf{filled} \end{bmatrix}$$

Lexicon:

;;; noun phrases

they
$$\begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{noun} \\ \text{AGR } \textbf{pl} \end{bmatrix} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \textbf{filled} \end{bmatrix}$$

fish
$$\begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{noun} \\ \text{AGR } [\,] \end{bmatrix} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \textbf{filled} \end{bmatrix}$$

it
$$\begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{noun} \\ \text{AGR } \textbf{sg} \end{bmatrix} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \textbf{filled} \end{bmatrix}$$

;;; verbs

fish
$$\begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{verb} \\ \text{AGR } \textbf{pl} \end{bmatrix} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{noun} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

can
$$\begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{verb} \\ \text{AGR } [\,] \end{bmatrix} \\ \text{OBJ } \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{verb} \end{bmatrix} \end{bmatrix} \\ \text{SUBJ } \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{noun} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$
;;; auxiliary verb

can
$$\begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{verb} \\ \text{AGR } \textbf{pl} \end{bmatrix} \\ \text{OBJ } \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{noun} \end{bmatrix} \\ \text{OBJ } \textbf{filled} \end{bmatrix} \\ \text{SUBJ } \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{noun} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$
;;; transitive verb

Root structure:

$$\begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{verb} \end{bmatrix} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \textbf{filled} \end{bmatrix}$$

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, OBJ and SUBJ are about subcategorization: they contain information about what can

combine with this sign. For instance, an intransitive verb will have a SUBJ corresponding to its subject 'slot' and a value of **filled** for its OBJ.[20] You do not have to memorize the precise details of the feature structure architecture described here for the exam (questions that assume knowledge of details will give an example). 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.

The grammar has just two rules, one for combining a verb with its subject and another for combining a verb with its object.

- The subject rule says that, when building the phrase, the SUBJ value of the second daughter is to be equated (unified) with the whole structure of the first daughter (indicated by $\boxed{2}$). The head of the mother is equated with the head of the second daughter ($\boxed{1}$). The rule also stipulates that the AGR values of the two daughters have to be unified and that the subject has to have a filled object slot.
- The verb-object rule says that, when building the phrase, the OBJ value of the first daughter is to be equated (unified) with the whole structure of the second daughter (indicated by $\boxed{2}$). The head of the mother is equated with the head of the first daughter ($\boxed{1}$). The SUBJ of the mother is also equated with the SUBJ of the first daughter ($\boxed{3}$): this ensures that any information about the subject that was specified on the lexical entry for the verb is preserved. The OBJ 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. This is what we want in order to prevent an ordinary transitive verb taking two objects.

These rules are controlled by the lexical entries in the sense that it's the lexical entries which determine the required subject and object of a word.

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

$$\begin{bmatrix} \text{HEAD } \boxed{1} \begin{bmatrix} \text{CAT } \textbf{verb} \\ \text{AGR } \boxed{3}\ \textbf{pl} \end{bmatrix} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \textbf{filled} \end{bmatrix} \rightarrow \boxed{2} \begin{bmatrix} \text{HEAD } \begin{bmatrix} \text{CAT } \textbf{noun} \\ \text{AGR } \boxed{3} \end{bmatrix} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \textbf{filled} \end{bmatrix}, \begin{bmatrix} \text{HEAD } \boxed{1} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \boxed{2} \end{bmatrix}$$

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 the subject-verb rule. The entry for *they* does not unify with the first daughter position of the verb-object rule. Hence there is no other parse.

The rules in this grammar are *binary*: i.e., they have exactly two daughters. The formalism allows for *unary* rules (one daughter) and also for *ternary* rules (three daughters) *quaternary* rules and so on. Grammars can be defined using only unary and binary rules which are weakly equivalent to grammars which use rules of higher arity: some approaches avoid the use of rules with arity of more than 2.

5.4 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.

[20] There are more elegant ways of doing this using lists, but these are more complicated. The subcategorisation idea applies to other parts of speech as well as verbs: e.g., in *Kim was happy to see her*, *happy* subcategorises for the infinitival VP.

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.

5.5 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
$$\begin{bmatrix} \text{HEAD} \begin{bmatrix} \text{CAT} & \textbf{verb} \\ \text{AGR} & \textbf{pl} \end{bmatrix} \\ \text{OBJ} \ \textbf{filled} \\ \text{SUBJ} \begin{bmatrix} \text{HEAD} \begin{bmatrix} \text{CAT} & \textbf{noun} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

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.

5.6 Interface to morphology

So far we have assumed a full-form lexicon (i.e., one that has entries for all the inflected forms), 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 the structure for the stem:

$$\begin{bmatrix} \text{HEAD} \begin{bmatrix} \text{CAT} & \textbf{noun} \\ \text{AGR} & \begin{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{OBJ} \ \textbf{filled} \\ \text{SUBJ} \ \textbf{filled} \end{bmatrix}$$

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:

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

This approach assumes that we also have a template `SINGULAR_NOUN`, where this is associated with a 'null' affix. Notice how this is an implementation of the idea of a morphological paradigm, mentioned in §2.2.

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.

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 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.

5.7 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.

DELPH-IN (`http://www.delph-in.net/`) distributes Open Source FS grammars for a variety of languages. The 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 like fish* will have a SEM value of:

$$\begin{bmatrix} \text{PRED } \textbf{and} \\ \text{ARG1} \begin{bmatrix} \text{PRED } \textbf{pron} \\ \text{ARG1 } \boxed{1} \end{bmatrix} \\ \text{ARG2} \begin{bmatrix} \text{PRED } \textbf{and} \\ \text{ARG1} \begin{bmatrix} \text{PRED } \textbf{like\_v} \\ \text{ARG1 } \boxed{1} \\ \text{ARG2 } \boxed{2} \end{bmatrix} \\ \text{ARG2} \begin{bmatrix} \text{PRED } \textbf{fish\_n} \\ \text{ARG1 } \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

This can be taken to be equivalent to the logical expression $\text{pron}(x) \wedge (\text{like\_v}(x, y) \wedge \text{fish\_n}(y))$ 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 *like*, 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

Subject-verb rule

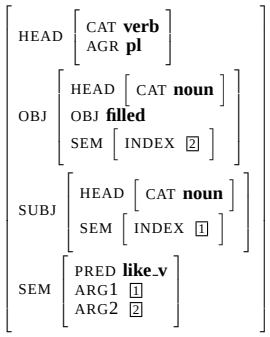
$$\begin{bmatrix} \text{HEAD } \boxed{1} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \textbf{filled} \\ \text{SEM} \begin{bmatrix} \text{PRED } \textbf{and} \\ \text{ARG1 } \boxed{4} \\ \text{ARG2 } \boxed{5} \end{bmatrix} \end{bmatrix} \rightarrow \boxed{2} \begin{bmatrix} \text{HEAD} \begin{bmatrix} \text{AGR } \boxed{3} \end{bmatrix} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \textbf{filled} \\ \text{SEM } \boxed{4} \end{bmatrix}, \begin{bmatrix} \text{HEAD } \boxed{1} \begin{bmatrix} \text{AGR } \boxed{3} \end{bmatrix} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \boxed{2} \\ \text{SEM } \boxed{5} \end{bmatrix}$$

Verb-object rule

$$\begin{bmatrix} \text{HEAD } \boxed{1} \\ \text{OBJ } \textbf{filled} \\ \text{SUBJ } \boxed{3} \\ \text{SEM} \begin{bmatrix} \text{PRED } \textbf{and} \\ \text{ARG1 } \boxed{4} \\ \text{ARG2 } \boxed{5} \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{HEAD } \boxed{1} \\ \text{OBJ } \boxed{2} \\ \text{SUBJ } \boxed{3} \\ \text{SEM } \boxed{4} \end{bmatrix}, \boxed{2} \begin{bmatrix} \text{OBJ } \textbf{filled} \\ \text{SEM } \boxed{5} \end{bmatrix}$$

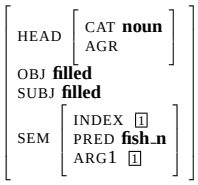
Lexicon:

like



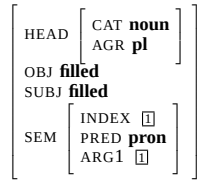
;;; transitive verb

fish



;;; noun phrase

they



;;; noun phrase

Notice the use of the 'and' predicate to relate different parts of the logical form. With very very simple examples as covered by this grammar, it might seem preferable to use an approach where the nouns are embedded in the semantics for the verb e.g., like_v(fish_n, fish_n) for *fish like fish*. But this sort of representation does not extend to more complex sentences.

In these simple examples, syntax and semantics are very closely related. But this is often not the case. For instance, in *it rains*, the *it* does not refer to a real entity (see §7.8), so the semantics should simply be rain_v. More complex examples include verbs like *seem*: for instance *Kim seems to sleep* means much the same thing as *it seems that Kim sleeps* (contrast this with the behaviour of *believe*). There are many examples of this sort that make the syntax/semantics interface much more complex than it first appears: we cannot simply read the compositional semantics off a syntax tree.

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.

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:

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

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

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

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.

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:[21]

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

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.[22]

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.

[21] 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!

[22] 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.

6.4 Hyponymy: IS-A

Hyponymy is the classical IS-A relation: e.g. *dog* is a *hyponym* of *animal*. To be more precise, the relevant sense of *dog* is the hyponym of *animal* (*dog* can also be a verb or used in a metaphorical and derogatory way to refer to a human). 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*.

Often the term hyponymy is used quite informally, but it clearly relates to ideas which have been formalised in description logics and used in ontologies and the semantic web.

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:

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
```

Taxonomies have also been extracted from machine-readable dictionaries: Microsoft's MindNet is the best known example. There has been considerable work on extracting taxonomic relationships from corpora, including some aimed at automatically extending WordNet.

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 (for instance, WordNet classes can be used in document classification).
- Word-sense disambiguation
- 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 important — it's whether ordinary speakers of the language feel there's a relationship). Homonymy is the most obvious case of polysemy, but is 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 (zero derivation, as mentioned in §2.2) 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, but in order to experiment with WSD as a standalone module, there has to be a standard: most commonly WordNet, because it was the only extensive modern resource for English that was freely available. This is controversial, because WordNet has a very fine granularity of senses and the senses often overlap, but there's no clear alternative. Various WSD 'competitions' have been organised (SENSEVAL).

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). Often experiments have been done with a small set of words which can be sense-tagged by the experimenter. 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:

(2) Striped bass are common.

(3) 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.

The term collocation is sometimes restricted to the situation where there is a syntactic relationship between the words. J&M (second edition) define collocation as a position-specific relationship (in contrast to *bag-of-words*, where position is ignored) but this is not a standard definition.

6.11 Yarowsky's unsupervised learning approach to WSD

Yarowsky (1995) describes a technique for unsupervised learning using collocates. A few seed collocates (possibly position-specific) 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 automatically 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 decision list classifier takes a set of already classified examples and returns criteria which distinguish them (e.g., word before / after / within window). The tests are each associated with a reliability metric. The original seeds are likely to be at the top of the decision list that is returned, 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	

Here '*animal* within 10 words of *plant*' is a new criterion, learned by the classifier.

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.

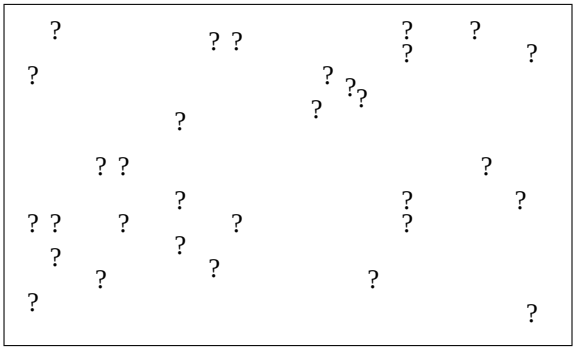
sense	training example
?	company said that the *plant* is still operating
A	although thousands of *plant* and animal species
A	zonal distribution of *plant* life
B	company manufacturing *plant* is in Orlando
	etc

5. Iterate the previous steps 3 and 4 until convergence

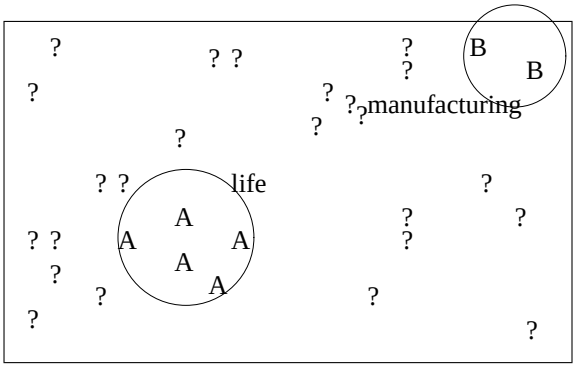
6. Apply the classifier to the unseen test data

The following schematic diagrams may help:

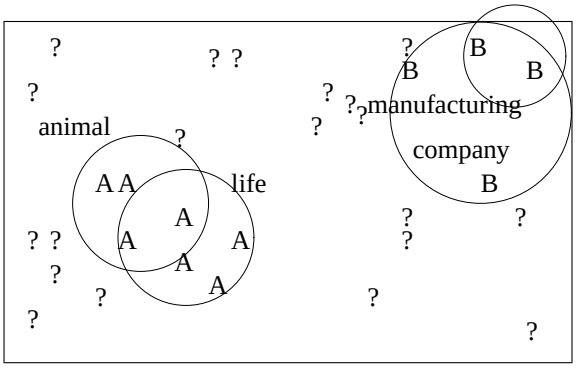
Initial state:



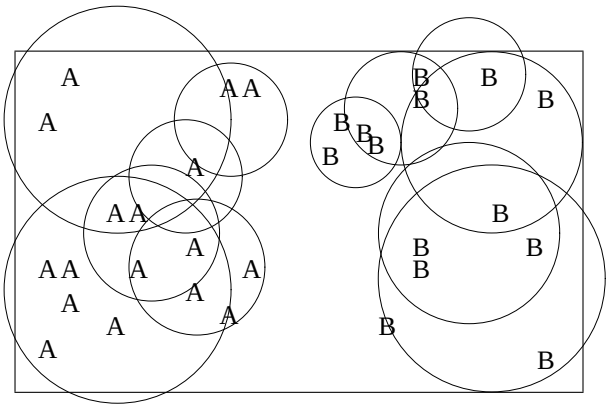
Seeds



Iterating:



Final:



Yarowsky also demonstrated the principle of ‘one sense per discourse’. For instance, if *plant* is used in the botanical sense in a particular text, then subsequent instances of *plant* in the same tense will also tend to be used in the botanical sense. Again, this is a very strong, but not absolute effect. This can be used as an additional refinement for the algorithm above, assuming we have a way of detecting the boundaries between distinct texts in the corpus.

Decision list classifiers can be thought of as automatically trained case statements. The experimenter decides on the classes of test (e.g., word next to word to be disambiguated; word within window 10). The system automatically generates and orders the specific tests based on the training data.

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. More details of this approach are in J&M (section 20.5).

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.

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 NLP papers, this can be downloaded via the ACL Anthology `http://aclweb.org/anthology-new/`.

7 Lecture 7: Discourse

The techniques we have seen in lectures 2–6 relate to the interpretation of words and individual sentences, but 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.

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.

This is yet another form of ambiguity: there are two different interpretations but there is no syntactic or semantic ambiguity in the interpretation of the individual sentences. 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%, HP 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 HP 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, HP 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 providing evidence to justify 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 (or the way the sentence is said — intonation etc) and text structure. For instance, parenthetical information cannot be related to a main clause by Narration (it is generally Explanation), 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 quite promising. This can be done by hand-coding a series of finite-state patterns, or by 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, suppose we remove the satellites in the first three sentences of this subsection:

> 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 initially 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 property of some entity (similar to adjectival behaviour) rather than being a referring expression itself.

Generally, entities are introduced in a discourse (technically, *evoked*) by indefinite noun phrases or proper names. Demonstratives (e.g., *this*) 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 by a proper name or definite description: 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.

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

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

(6) * 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:

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

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

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

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

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

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

7.7 Reflexives

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

(14) # 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*:

7.8 Pleonastic pronouns

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

(15) It is snowing

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

(17) It is obvious that Kim snores.

(18) It bothers Sandy that Kim snores.

Note also:

(19) They are digging up the street again

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

7.9 Salience

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

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

(20) Kim has a big car. Sandy has a small one. Lee likes to drive it.

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

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

(21) 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:

(22) 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:

(23) 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.

(24) 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 Lexical semantics and world knowledge effects

The made-up examples above were chosen so that the meaning of the utterance did not determine the way the pronoun was resolved. In real examples, world knowledge may override salience effects. For instance (from Radio 5):

(25) Andrew Strauss again blamed the batting after England lost to Australia last night. They now lead the series three-nil.

Here *they* has to refer to Australia, despite the general preference for subjects as antecendents. The analysis required to work this out is actually non-trivial: you might like to try writing down some plausible meaning postulates which would block the inference that *they* refers to England. (Note also the plural pronoun with singular antecedent, which is normal for sports teams.)

Note, however, that violation of salience effects can easily lead to an odd discourse:

(26) The England football team won last night. Scotland lost. ? They have qualified for the World Cup with a 100% record.

Systems which output natural language disourses, such as summarization systems, have to keep track of anaphora to avoid such problems.

7.11 Algorithms for resolving anaphora

NLP researchers are interested in all types of coreference, but most work has gone into the problem of finding antecedents for pronouns. 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 (although if all the candidate antecedents have the same gender, we don't need to do any further resolution). I will outline an approach to anaphora resolution using a statistical classifier, but there are many other approaches.

We can formulate pronoun resolution as a classification problem, which can be implemented using one of the standard machine learning approaches to supervised classification (examples of approaches include Naive Bayes, perceptron, k-nearest neighbour), assuming that we have a suitable set of training data. For each pairing of a (non-pleonastic) pronoun and a candidate antecedent, the classifier has to make a binary decision as to whether the candidate is an actual antecedent, based on some features associated with the pairing. For simplicity, we can assume that the candidate antecedents for a pronoun are all the noun phrases within a window of the surrounding text consisting of the current sentence and the preceeding 5 sentences (excluding pleonastic pronouns). For example:

> 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.
>
> Pronoun *he*, candidate antecedents: *Niall Ferguson*, *a snappy dresser*, *Stephen Moss*, *him*, *an hour*, *the historian*, *the historian's Oxford study*.

Notice that this simple approach leads to *a snappy dresser* being included as a candidate antecendent and that a choice had to be made as to how to treat the possessive. I've included the possibility of cataphors, although these are sufficiently rare that they are often excluded.

For each such pairing, we build a *feature vector*[23] using features corresponding to some of the factors discussed in the previous sections. For instance (using t/f rather than 1/0 for binary features for readability):

Cataphoric Binary: t if the pronoun occurs before the candidate antecedent.

Number agreement Binary: t if the pronoun agrees in number with the candidate antecedent.

Gender agreement Binary: t if the pronoun agrees in gender with the candidate antecedent.

Same verb Binary: t if the pronoun and the candidate antecedent are arguments of the same verb (for binding theory).

[23] The term 'instance' is sometimes used in AI, but I prefer 'feature vector', because we're mainly interested in the nature of the features.

Sentence distance Discrete: { 0, 1, 2 . . . } The number of sentences between pronoun and candidate.

Grammatical role Discrete: { subject, object, other } The role of the potential antecedent.

Parallel Binary: t if the potential antecedent and the pronoun share the same grammatical role.

Linguistic form Discrete: { proper, definite, indefinite, pronoun } This indicates something about the syntax of the potential antecedent noun phrase.

Taking some pairings from the example above:

pronoun	antecedent	cataphoric	num	gen	same	distance	role	parallel	form
him	*Niall Ferguson*	f	t	t	f	1	subj	f	prop
him	*Stephen Moss*	f	t	t	t	0	subj	f	prop
him	*he*	t	t	t	f	0	subj	f	pron
he	*Niall Ferguson*	f	t	t	f	1	subj	t	prop
he	*Stephen Moss*	f	t	t	f	0	subj	t	prop
he	*him*	f	t	t	f	0	obj	f	pron

Notice that with this set of features, we cannot model the "repeated mention" effect mentioned in §7.9. It would be possible to model it with a classifier-based system, but it requires that we keep track of the coreferences that have been assigned and thus that we maintain a model of the discourse as individual pronouns are resolved. I will return to the issue of discourse models below. Coherence effects are very complex to model and world knowledge effects are indefinitely difficult (AI-complete in the limit), so both of these are excluded from this simple feature set. Realistic systems use many more features and values than shown here and can approximate some partial world knowledge via classification of named entities, for instance.

To implement the classifier, we require some knowledge of syntactic structure, but not necessarily full parsing. We could approximately determine noun phrases and grammatical role by means of a series of regular expressions over POS-tagged data instead of using a full parser. Even if a full syntactic parser is available, it may be necessary to augment it with special purpose rules to detect pleonastic pronouns.

The training data for this task is produced from a corpus which is marked up by humans with pairings between pronouns and antecedent phrases. The classifier uses the marked-up pairings as positive examples (class TRUE), and all other possible pairings between the pronoun and candidate antecendant as negative examples (class FALSE). For instance, if the pairings above were used as training data, we would have:

class	cataphoric	num	gen	same	distance	role	parallel	form
TRUE	f	t	t	f	1	subj	f	prop
FALSE	f	t	t	t	0	subj	f	prop
FALSE	t	t	t	f	0	subj	f	pron
FALSE	f	t	t	f	1	subj	t	prop
TRUE	f	t	t	f	0	subj	t	prop
FALSE	f	t	t	f	0	obj	f	pron

Note the prelecture exercise which suggests that you participate in an online experiment to collect training data. If you do this, you will discover a number of complexities that I have ignored in this account.

In very general terms, a supervised classifier uses the training data to determine an appropriate mapping (i.e., *hypothesis* in the terminology used in the Part 1B AI course) from feature vectors to classes. This mapping is then used when classifying the test data. To make this more concrete, if we are using a probabilistic approach, we want to choose the class c out of the set of classes C ({ TRUE, FALSE } here) which is most probable given a feature vector $\vec{f}$:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|\vec{f})$$

(See §3.5 for the explanation of argmax and $\hat{c}$.) As with the POS tagging problem, for a realistic feature space, we will be unable to model this directly. The Naive Bayes classifier is based on the assumption that we rewrite this formula

using Bayes Theorem and then treat the features as conditionally independent (the independence assumption is the "naive" part). That is:

$$P(c|\vec{f}) = \frac{P(\vec{f}|c)P(c)}{P(\vec{f})}$$

As with the models discussed in Lecture 3, we can ignore the denominator because it is constant, hence:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(\vec{f}|c)P(c)$$

Treating the features as independent means taking the product of the probabilities of the individual features in $\vec{f}$ for the class:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{i=1}^{n} P(f_i|c)$$

In practice, the Naive Bayes model is often found to perform well even with a set of features that are clearly not independent.

There are fundamental limitations on performance caused by treating the problem as classification of individual pronoun-antecedent pairs rather than as building a discourse model including all the coreferences. Inability to implement 'repeated mention' is one such limitation, another is the inability to use information gained from one linkage in resolving further pronouns. Consider yet another 'team' example:

(27) Sturt think they can perform better in Twenty20 cricket. It requires additional skills compared with older forms of the limited over game.

A classifier which treats each pronoun entirely separately might well end up resolving the *it* at the start of the second sentence to *Sturt* rather than the correct *Twenty20 cricket*. However, if we already know that *they* corefers with *Sturt*, coreference with *it* will be dispreferred because number agreement does not match (recall from §7.6 that pronoun agreement has to be consistent). This type of effect is especially relevant when general coreference resolution is considered. One approach is to run a simple classifier initially to acquire probabilities of links and to use those results as the input to a second system which clusters the entities to find an optimal solution. I will not discuss this further here, however.

7.12 Evaluation of pronoun resolution

At first sight it seems that we could require that every (non-pleonastic) pronoun is linked to an antecedent, and just measure the accuracy of the links found compared to the test data. One issue which complicates this concerns the identification of the pronouns (some may be pleonastic, others may refer to concepts which aren't expressed in the text as noun phrases) and also identification of the target noun phrases, with embedded noun phrases being a particular issue. We could treat this as a separate problem and assume we're given data with the non-pleonastic pronouns and the candidate antecedents identified, but this isn't fully realistic.

A further range of problems arise essentially because we are using the identification of some piece of text as an antecedent for the pronoun as a surrogate for the real problem, which is identification of a coreference to a real world entity. For instance, suppose that, in the example below, our algorithm links *him* to *Andrew* and also links *he* to *Andrew*, but the training data has linked *him* to *Andrew* and *he* to *him*.

> Sally met Andrew in town and took him to the new restaurant. He was impressed.

Our algorithm has successfully linked the coreferring expressions, but if we consider the evaluation approach of comparing the individual links to the test material, it will be penalised. Of course it is trivial to take the transitive closure of the links, but it is not easy to develop an evaluation metric that correctly allows for this and does not, for example, unfairly reward algorithms that link all the pronouns together into one cluster. As a consequence of this sort of issue, it has been difficult to develop agreed metrics for evaluation.

7.13 Statistical classification in language processing

Many problems in natural language can be treated as classification problems: besides pronoun resolution, we have seen sentiment classification and word sense disambiguation, which are straightforward examples of classification. POS-tagging is also a form of classification, but there we take the tag sequence of highest probability rather than considering each tag separately. As we have seen above, we actually need to consider relationships between coreferences to model some discourse effects.

Pronoun resolution has a more complex feature set than the previous examples of classification that we've seen and determination of some of the features requires considerable processing, which is itself error prone. A statistical classifier is somewhat robust to this, assuming that the training data features have been assigned by the same mechanism as used in the test system. For example, if the grammatical role assignment is unreliable, the weight assigned to that feature might be less than if it were perfect.

One serious disadvantage of supervised classification is reliance on training data, which is often expensive and difficult to obtain and may not generalise across domains. Research on unsupervised methods is therefore popular.

There are no hard and fast rules for choosing which statistical approach to classification to use on a given task. Many NLP researchers are only interested in classifiers as tools for investigating problems: they may either simply use the same classifier that previous researchers have tried or experiment with a range of classifiers using a toolkit such as WEKA.[24]

Performance considerations may involve speed as well as accuracy: if a lot of training data is available, then a classifier with faster performance in the training phase may enable one to use more of the available data. The research issues in developing a classifier-based algorithm for an NLP problem generally center around specification of the problem, development of the labelling scheme and determination of the feature set to be used.

7.14 Further reading

J&M discuss the most popular approach to rhetorical relations, *rhetorical structure theory* or RST (section 21.2.1). 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 classifier approach that I've given here are partly based on J&M's account in Chapter 21: they discuss a log-linear classifier there, but Naive Bayes is decribed in 20.2.2 and I have followed that description.

[24] `http://www.cs.waikato.ac.nz/ml/weka/` Ian H. Witten and Eibe Frank (2005) "Data Mining: Practical machine learning tools and techniques", 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

8 Lecture 8: Applications

No notes: copies of slides will be made available after the lecture.

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*.

attribute-value matrix A way of drawing **feature structure**s: see §5.1.

AVM Attribute-value matrix.

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).

bag of words Unordered collection of words in some text.

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).

CFG context-free grammar.

chart parsing See §4.6.

Chomsky Noam Chomsky, professor at MIT. His work underlies most modern approaches to syntax in linguistics. Not so hot on probability theory.

classifier A system which assigns classes to items, usually using a machine learning approach.

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. Nothing to do with context-free grammar.

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).

expletive pronoun Another term for **pleonastic pronoun**: see §7.8.

feature Either: a labelled arc in a **feature structure**
Or: a characteristic property used in machine learning.

feature structure See Lecture 5.

FS Feature structure.

FSA Finite state automaton

FST Finite state transducer

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.

genre Type of text: e.g., newspaper, novel, textbook, lecture notes, scientific paper. Note the difference to **domain** (which is about the type of knowledge): it's possible to have texts in different genre discussing the same domain (e.g., discussion of human genome in newspaper vs textbook vs paper).

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.

Hidden Markov Model See §3.5

HMM Hidden Markov Model

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).

long-distance dependency See §4.13

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

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

modifier Something that further specifies a particular entity or event: e.g., *big house*, *shout loudly*.

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).

NL Natural language.

NLID Natural language interface to a database.

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 Same as **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.

pleonastic Non-referring (esp. of pronouns): see §7.8

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

POS Part of speech (in the context of POS tagging).

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 5.5

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 also use strictly unseen data for evaluation.

treebank a corpus annotated with trees (lecture 4).

unification See Lecture 5, especially §5.2.

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

WSD Word-sense disambiguation

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

vagueness Of word meanings, contrasted with **ambiguity** : see §6.8.

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

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

Exercises for NLP course, 2010

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 available to supervisors 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

Without looking at any film reviews beforehand, write down 10 words which you think would be good indications of a positive review (when taken in isolation) and 10 words which you think would be negative. Then go through a review of a film and see whether you find there are more of your positive words than the negative ones. Are there words in the review which you think you should have added to your initial lists?

Have a look at `http://www.cl.cam.ac.uk/~aac10/stuff.html` for pointers to sentiment analysis data used in experiments.

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/Det 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://alias-i.com/lingpipe/web/demos.html`
`http://www.lingsoft.fi/demos.html`
`http://ucrel.lancs.ac.uk/claws/trial.html`
`http://l2r.cs.uiuc.edu/~cogcomp/pos_demo.php`
The Lingpipe tagger uses an HMM approach as described in the lecture, the others use different techniques. Lingsoft give considerably more information than the POS tag: their system uses hand-written rules.

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) and any adverbial modifiers.[25] 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 $\begin{bmatrix} \text{DTR1} & \begin{bmatrix} \text{CAT} & \textbf{V} \\ \text{AGR} & \textbf{sg} \end{bmatrix} \end{bmatrix}$

[25] A *modifier* is something that further specifies a particular entity or event: e.g., *big house, shout loudly*.

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

(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 an indirect object as well as an object. 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

There is an online experiment to collect training data for anaphor resolution at `http://anawiki.essex.ac.uk/phrasedetectives/`. Spending a few minutes on this will give you an idea of the issues that arise in anaphora resolution: there are a series of tasks which are intended to train new participants which take you through progressively more complex cases. Note that you have to register but that you don't have to give an email address unless you want to be eligible for a prize.

G.2 Post-lecture

Take a few sentences of real text and work out the values you would obtain for the features discussed in the lecture. See if you can identify some other easy-to-implement features that might help resolution.

Try out the Lingpipe coreference system at `http://alias-i.com/lingpipe/web/demos.html`

H Lecture 8

H.1 Exercises (post- lecture)

Use Systran (via `http://babelfish.yahoo.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. Try and compare a Systran translation to one given by Google.

Open ended: Suppose you were supervising undergraduates: what would it take to build a system that responded to email requests for supervisions and automatically updated an online diary? To think about this, ideally you should collect a corpus of emails.

I Answers to some of the exercises

I.1 Lecture 1 (post-lecture)

Something like this experiment was tried by Pang et al (2002) to provide a baseline for their machine learning system. The table below shows the accuracy they obtained on movie reviews by counting the positive and negative terms in the document. The third set was obtained with the help of preliminary frequency data: note the inclusion of '?' and '!'.

	Terms	Accuracy
Human 1	positive: *dazzling, brilliant, phenomenal, excellent, fantastic* negative: *suck, terrible, awful, unwatchable, hideous*	58%
Human 2	positive: *gripping, mesmerizing, riveting, spectacular, cool, awesome, thrilling, badass, excellent, moving, exciting* negative: *bad, cliched, sucks, boring, stupid, slow*	64%
Human 3 (with stats)	positive: *love, wonderful, best, great, superb, still, beautiful* negative: *bad, worst, stupid, waste, boring, ?, !*	69%

I.2 Lecture 2 (pre-lecture)

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.3 Lecture 3 (pre-lecture)

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.4 Lecture 4 (pre-lecture)

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.5 Lecture 5 (pre-lecture)

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