Course Roadmap

Informatics 2A: Lecture 2

John Longley

School of Informatics
University of Edinburgh
jrl@inf.ed.ac.uk

18-20 September, 2012

Formal and Natural Languages

This course is about methods for describing, specifying and processing languages of various kinds:

- Formal (computer) languages, e.g. Java, Haskell, HTML, SQL, Postscript, ...
- Natural (human) languages, e.g. English, Greek, Japanese.
- 'Languages' that represent the behaviour of some machine or system. E.g. think about 'communicating' with a vending machine via coin insertions and button presses:

```
insert50p . pressButton1 . deliverMarsBar
```

A common theoretical core

We'll be focusing on certain theoretical concepts that can be applied to each of the above three domains:

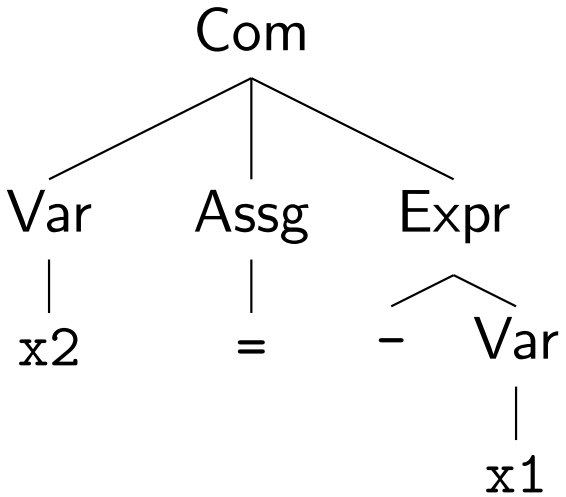
- regular languages
- finite state machines
- context-free languages, syntax trees
- types, compositional semantics

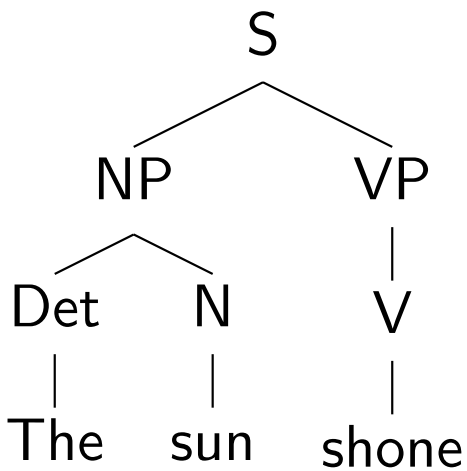
The fact that the same underlying theory can be applied in such diverse contexts suggests that the theory is somehow fundamental, and worth learning about!

Mostly, we'll be looking at various aspects of formal languages (mainly AS) and natural languages (mainly JL). As we'll see, there are some important similarities between formal and natural languages — and some important differences.

Syntax trees: a central concept

In both FLs and NLs, phrases have structure that can be represented via syntax trees.





Determining the structure of a phrase is an important first step towards doing other things with it. Much of this course will be about describing and computing syntax trees for phrases of some given language.

The language processing 'pipeline' (FL version)

Think about the phases in which a Java program is processed:

Raw source text (e.g. `x2=-x1`)

$\Downarrow$ lexing

Stream of tokens (e.g. `x2, =, -, x1`)

$\Downarrow$ parsing

Syntax tree (as on previous slide)

$\Downarrow$ typechecking etc.

Annotated syntax tree

$\Downarrow$ compiling

Java bytecode

$\Downarrow$ linking

JVM state

$\Downarrow$ running

Program behaviour

Language processing for programming languages

In the case of programming languages, the pipeline typically works in a very 'pure' way: each phase depends only on the output from the previous phase.

- In this course, we'll be concentrating mainly on the first half of this pipeline: lexing, parsing, typechecking. (Especially parsing).
- We'll be looking both at the theoretical concepts involved (e.g. what is a syntax tree?)
- And at algorithms for the various phases (e.g. how do we construct the syntax tree for a given program)?
- We won't say much about techniques for compilation etc.
- However, we'll briefly touched on how the intended runtime behaviour of programs (i.e. their semantics) may be specified.

The language processing 'pipeline' (NL version)

A broadly similar pipeline may be considered e.g. for English:

Raw soundwaves
$\Downarrow$ phonetics
Phones (e.g. [t^h]–toucan, [t]–starfish)
$\Downarrow$ phonology
Phonemes (e.g. /t/, /d/)
$\Downarrow$ segmentation, tagging
Words, morphemes
$\Downarrow$ parsing
Parse tree
$\Downarrow$ agreement checking etc.
Annotated parse tree
$\Downarrow$ semantics
Logical form or 'meaning'
$\Downarrow$ ...

Comparison between FLs and NLs

There are close relationships between these two pipelines. However, there are also important differences:

- FLs can be pinned down by a precise definition. NLs are fluid, fuzzy at the edges, and constantly evolving.
- NLs are riddled with ambiguity at all levels. This is normally avoidable in FLs.
- For FLs the pipeline is typically 'pure'. In NLs, information from later stages is sometimes used to resolve ambiguities at earlier stages, e.g.

  Time flies like an arrow.
  Fruit flies like a banana.

Kinds of ambiguity in NL

- Phonological ambiguity: e.g. 'an ice lolly' vs. 'a nice lolly'.
- Lexical ambiguity: e.g. 'fast' has many senses (as noun, verb, adjective, adverb).
- Syntactic ambiguity: e.g. two possible syntax trees for 'complaints about referees multiplying'.
- Semantic ambiguity: e.g. 'Please use all available doors when boarding the train'.

More on the NL pipeline

In the case of natural languages, one could in principle think of the pipeline . . .

- either as a model for how an artificial speech processing system might be structured,
- or as a proposed (crude) model for what naturally goes on in human minds.

In this course, we mostly emphasize the former perspective.

Also, in the NL setting, it's equally sensible to think of running the pipeline backwards: starting with a logical form or 'meaning' and generating a speech utterance to express it. But we won't say much about this in this course.

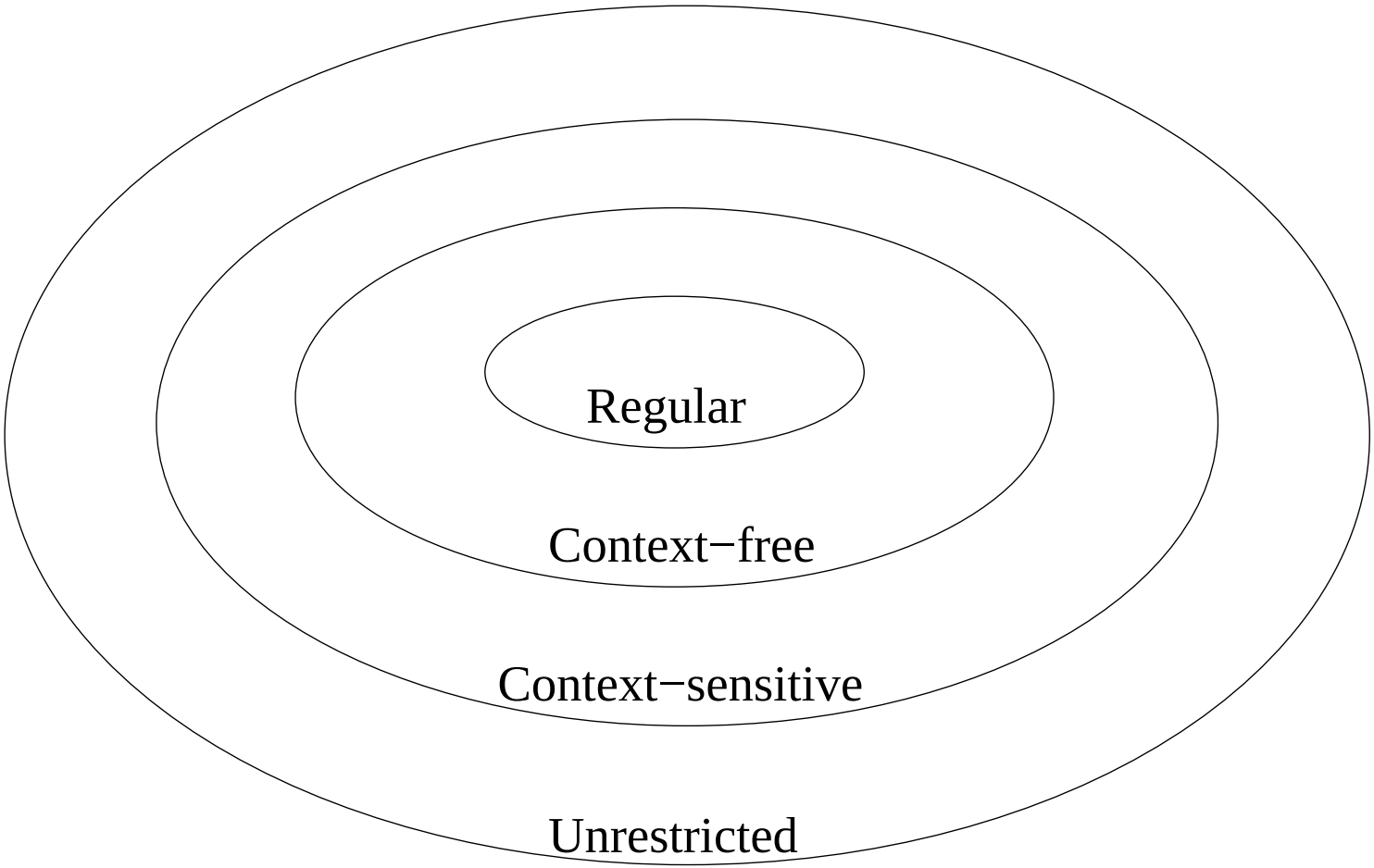
Levels of language complexity

Some languages / language features are 'more complex' (harder to describe, harder to process) than others. In fact, we can classify languages on a scale of complexity (the Chomsky hierarchy):

- Regular languages: those whose phrases can be 'recognized' by a finite state machine (cf. Informatics 1).
- Context-free languages. The basic structure of most programming languages, and many aspects of natural languages, can be described at this level.
- Context-sensitive languages. Some NLs involve features of this level of complexity.
- Unrestricted languages: *all* languages that can in principle be defined via mechanical rules.

Roughly speaking, we'll start with regular languages and work our way up the hierarchy. Context-free languages get most attention.

The Chomsky Hierarchy (picture)



Formal Language component: overview

Regular languages:

- Definition using finite state machines (as in Inf1A).
- Equivalence of deterministic FSMs, non-deterministic FSMs, regular expressions.
- Applications: pattern matching, lexing, morphology.
- The pumping lemma: proving a given language *isn't* regular.

Context-free languages:

- Context-free grammars, syntax trees.
- The corresponding machines: pushdown automata.
- Parsing: constructing the syntax tree for a given phrase.
- A parsing algorithm for LL(1) languages, in detail.

Formal Language component: overview (continued)

After a break to cover some NL material, we'll glance briefly at some concepts from further down the pipeline: e.g. typechecking and semantics for programming languages.

Then we continue up the Chomsky hierarchy:

Context-sensitive languages:

- Definition, examples.
- Relationship to linear bounded automata.

Unrestricted languages:

- Turing machines; theoretical limits of what's 'computable in principle'.
- Undecidable problems.

Natural Language component: overview

We'll look at various parts of the NL processing pipeline, concentrating especially on part-of-speech tagging and parsing, with a little bit on agreement checking and semantics.
Our main focus is on how to get computers to perform these tasks, for applications such as

- speech synthesis
- machine translation
- text summarization and simplification
- (simple) NL dialogue systems.

But there'll also be a couple of lectures on scientific studies of how we as humans perform them.

Natural Language component: overview (continued)

Some specific topics:

- Complexity of human languages: E.g. whereabouts do human languages sit in the Chomsky hierarchy?
- Parsing algorithms: Because NLs differ from FLs in various ways, it turns out that different kinds of parsing algorithms are suitable.
- Probabilistic versions of FL concepts: In NL, because of ambiguity, we're typically looking for the most likely way of analysing a phrase. For this purpose, probabilistic analogues of e.g. finite state machines or context-free grammars are useful.
- Use of text corpora: Rather than building in all the relevant knowledge of the language by hand, we sometimes get a NLP system to 'learn' it for itself from some large sample of pre-existing text.

Natural language semantics

Consider the sentence:

Every student has access to a computer.

The 'meaning' of this can be expressed by a logical formula:

$$\forall x.\,(\mathrm{student}(x) \;\Rightarrow\; \exists y.\,(\mathrm{computer}(y) \wedge \mathrm{hasAccessTo}(x, y)))$$

Or perhaps:

$$\exists y.\,(\mathrm{computer}(y) \wedge \forall x.\,(\mathrm{student}(x) \;\Rightarrow\; \mathrm{hasAccessTo}(x, y)))$$

Problem: how can (either of) these formulae be mechanically generated from a syntax tree for the original sentence? This is what semantics is all about.

The Python programming language

- Invented by Guido van Rossum (pictured)
- Object-oriented programming language (like Java): has classes and objects.
- Dynamic typing (unlike Java). More flexibility but more chance of run-time errors.
- Clear and powerful syntax – very succinct (unlike Java). Especially convenient for string processing.
- Typically driven interactively via a console session (like Haskell).
- Interfaces to many system calls, libraries, window systems, and other programming languages.

Natural language processing with Python

NLTK: Natural Language Toolkit

Developed by Steven Bird, Ewan Klein and Edward Loper; mainly addresses education and research; the book is online:
`http://www.nltk.org`

The NLTK provides support for many parts of the NL processing pipeline, e.g.

- Part-of-speech tagging
- Parsing
- Meaning extraction (semantics)

Lab sessions will introduce you to both Python and NLTK.

In Assignment 2, we'll show how one can fit these together to construct a (very simple) natural language dialogue system.

Summary

- What is Inf2a about?
- We will learn about formal and natural languages.
- We will discuss their similarities and differences.
- We will cover finite state machines, context-free grammars, syntax trees, parsing, pos-tagging, ambiguity.
- We will use Python for natural language processing.
- We will have lots of fun!

Next lecture: Finite state machines (revision)
Reading: Kozen chapter 1, 2; J&M[2nd Ed] chapter 1