General information

Paul W. Goldberg

1Department of Computer Science
University of Liverpool, U.K.

2012-13

General information

2 class tests; let me know if you need any special provision to take the tests.
unassessed exercises from time to time; bring paper
Some handout notes, available later
today – general introduction, motivation

Languages

Spoken languages and programming languages: in both cases we like to know whether a sequence of symbols belongs to the language.

Formal Languages have the property that there is a precise rule that governs what strings belong to the language.

Formal languages include programming languages, database query languages, various file formats. (so, in the world of computers, they are everywhere...) By contrast, English, French etc. are not formal languages, although you can still try to write down rules that work most of the time.

Brief Description

This module aims to introduce formal concepts of automata, grammars and languages; to introduce ideas of computability and decidability, and to illustrate the importance of automata, formal language theory and general models of computation in Computer Science and Artificial Intelligence.
“valid” English sentences?

letters: a,b,c,...,z — no problem

words: dog, house, therefore, fine,... — mostly we can agree on what words are real; there are not too many to write down in a list

sentences? (e.g. “Colourless green ideas sleep furiously.” composed by Noam Chomsky as example of grammatical but nonsensical sentence)

But: “Furiously sleep ideas green colorless.” is not a sentence! (see wikipedia)

Can we write down a specification of a large collection of valid sentences...?

How we usually describe or teach syntax

Informal descriptions:

"An arithmetic expression is constructed from variables and numbers, and infix operators +, -, *, /, and subexpressions may possibly be enclosed in parentheses, in which case every ( must have a corresponding ) ...”

"A comment begins with /* and ends with */

"your password should have 4-8 characters and contain a non-alphabetic character"

By examples:

Let $E$ denote the set of arithmetic expressions

$$E = \{ x, 1 + (2 - x), x \ast y + ((z)), ... \}$$

a42 is a valid variable name; 42a is not, because a variable name can’t start with a number.

(The variable-name description is a combination of English and examples.) We need some notation to express these descriptions more precisely!

An analogy

COMP109: propositional logic. Express facts about the world in formal logic. Why? So we can follow computational procedures to draw inferences from them.

COMP218: notations for representing formal languages. These give us

- ways to define them precisely
- ways to build compilers that recognise the languages
- ways to check whether
  - a string of symbols belongs to a language
  - whether two alternative descriptions of languages are actually the same language
What COMP218 is about

Tools to analyse languages
These tool originate in analysis of natural (spoken) language as well as programming languages.
Natural language: want to recognise valid sentence
Programming language: want to recognise valid program
No length limit on sentences, so you can’t list them. A list is infeasible even you limit the length to some “reasonable” amount (e.g. 50 words), also not very enlightening.

Some notations/methods for describing a language (other than explicitly listing it) include
- java programs
- finite automaton (example on board)
- regular expressions
- Backus-Naur form
- context-free grammar (example on next slide)

We will see that some of the above can describe more languages than others.

1 but there are problems if we impose no restrictions on the programs...

Grammars

Sets of rules for generating syntactically correct programs/sentences.
A grammar for generating some English sentences:

\[
\begin{align*}
\langle S \rangle & \rightarrow \langle S \rangle \text{ and } \langle S \rangle \\
\langle S \rangle & \rightarrow \langle \text{subject phrase} \rangle \langle \text{Verb} \rangle \langle \text{object phrase} \rangle \\
\langle \text{subject phrase} \rangle & \rightarrow \langle \text{subject pronoun} \rangle \\
\langle \text{subject phrase} \rangle & \rightarrow \langle \text{Article} \rangle \langle \text{Noun} \rangle \\
\langle \text{object phrase} \rangle & \rightarrow \langle \text{object pronoun} \rangle \\
\langle \text{object phrase} \rangle & \rightarrow \langle \text{Article} \rangle \langle \text{Noun} \rangle \\
\langle \text{subject pronoun} \rangle & \rightarrow \text{he} | \text{she} \\
\langle \text{object pronoun} \rangle & \rightarrow \text{him} | \text{her} | \text{me} \\
\langle \text{Article} \rangle & \rightarrow \text{a} | \text{the} \\
\langle \text{Noun} \rangle & \rightarrow \text{dog} | \text{cat} | \text{mouse} | \text{house} \\
\langle \text{Verb} \rangle & \rightarrow \text{sees} | \text{likes} | \text{finds} | \text{leaves}
\end{align*}
\]

Observations

- The grammar can generate sentences like “the dog sees the cat and the mouse leaves the house”, which may be nonsense e.g. “the house sees me”.
- Arbitrarily long sentences can be generated (which you can’t do by enumeration!)
- Semantics can be given with reference to grammar, e.g. logical conjunction of subsentences formed by word “and”
- The grammar could be extended to handle subordinate clauses, adverbs etc.
- You can’t define natural language sentences completely this way, but you can for programming languages
Stages of compilation:

- **lexical analysis**: divide sequence of characters into tokens, such as variable names, operators, labels. In a natural language tokens are strings of consecutive letters (easy to recognise!)
- **parsing**: identify relationships between tokens
- **code generation**: generate object code
- **code optimisation**

```
pay = salary + (overtimerate * overtime);
```

Break into tokens as follows:
```
pay = salary + (overtimerate * overtime);
```
Definitions and notation

- An *alphabet* is a finite set of symbols.
- A *word* over alphabet $A$ is a string of symbols belonging to $A$.
- The *empty word* will be denoted $\epsilon$.
- $A^*$ denotes the set of all words over $A$.
- $A^+$ denotes the set of all non-empty words over $A$.

Languages

For $w \in A^*$, the *reverse* of $w$ is denoted $w^R$ and consists of $w$’s letters in reverse order.

A *palindrome* is a word $w$ satisfying $w = w^R$.

If $u, v, w$ are words and $w = uv$ then $u$ is a *prefix* of $w$ and $v$ is a *suffix* of $w$. A *proper prefix* of $w$ is a prefix that is not equal to $\epsilon$ or $w$. (Similarly for proper suffix)

The *concatenation* (a.k.a. "product") of two words is obtained by appending them together to form one long word.

Concatenation of words $w_1$ and $w_2$ can be written $w_1w_2$.

For any word $w$, note that $w\epsilon = \epsilon w = w$.

Concatenation is associative.

$w^n$ denotes the concatenation of $n$ copies of $w$.

$|w|$ denotes the length (number of letters) of $w$.

$|w_1w_2| = |w_1| + |w_2|$