MAU22C00 Lecture 0

John Stalker

Trinity College Dublin

General module info

- I'm John Stalker (stalker@maths.tcd.ie) and I'm teaching the first semester.
- Andreea Nicoara (anicoara@maths.tcd.ie) teaches the second semester.
- There is a Blackboard page for the module and a publicly accessible webpage, at least for this semester, at https://www.maths.tcd.ie/~stalker/22C00/. I will try to post most things to both places, including notes!
- There is an exam at the end counting 60% and continuous assessment counts for 40%.
- There are weekly tutorials starting, hopefully, next week.

Module content

- Languages and grammars (regular, context free, ...)
- Logic (zeroeth order, a.k.a. propositional calculus, first order)
- Numbers
- Sets (finite, countable, uncountable)
- Trees and graphs
- Idealised machines (finite state automata, pushdown automata, Turing machines)
- Semigroups, monoids and groups

These things are all related!

How are these things related? An example

Before we consider these topics individually let's see how they're related.

Suppose you had to create a module enrollment system. You'll need to create

- organisational policies, e.g. who is responsible for doing what when?
- user interfaces for various roles, e.g. student, school administrator, etc.
- core functionality, i.e. checking whether students are allowed to take the set of modules they've selected.

The first two problems are interesting, but not closely related to this module, so let's concentrate on the last.

Core functionality

The core of the system has two inputs:

- Module selections, input by students or generated by them using the user interface
- Selection rules, input by staff or generated by them using the user interface

We need a way to check module selections against module rules and accept or reject them accordingly.

First, though, we need ways to describe them, i.e. languages.

These are *formal languages*, suited to automated processing, not the language you find in a course handbook.

A language for student module selections

The student language is (mostly) easy. You can just use a list of modules.

This isn't necessarily how the students interact with the system. The UI might convert tick boxes to such a list used by the core system, for example.

You might want to use module codes rather than names. Otherwise, is "Probability and Statistics" the name of one module or the names of two modules joined by "and"?

For our example system I'll use module names though.

Module selection rules and modal verbs

You could get by with a non-linguistic representation of student input but it would be harder to avoid a language for rules.

Course handbooks are full of "modal verbs", e.g. "You *must* take ...", "You *can* take either ... or ...", "if you want to ... then you *should* take ...".

Internet RFC's are full of them as well, e.g. "they MUST only be used where it is actually required for interoperation or to limit behavior which has potential for causing harm".

This is from RFC 2119, which specifies the meaning of the terms "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in RFCs, including in itself!

A language for module rules, continued

The meaning of modal verbs has been discussed by linguists, logicians and philosphers for centuries.

Luckily we can ignore all of their work. We're writing a checker, and checkers generally check declarative sentences.

The course handbook says "you *must* take Compilers" but the checker needs to check the validity of the sentence "this student *is* taking compilers.

The other thing we find a lot of in course handbooks is Boolean operators, e.g. "You must take Probability *and* Statistics *or* Algebra *and* Geometry".

Unfortunately humans use these in very ambiguous ways, but computers hate ambiguity.

How not to write rules

An EU applicant is a person:
1. who is ordinarily resident in the EU
AND
. who will have received full-time post
  primary education
  AND/OR
. who has worked full-time in the EU for three
  of the five years immediately preceding
  admission to Trinity
OR
2. who has
. official refugee status or has been granted
  humanitarian leave to remain in the State
  AND
. who has been ordinarily resident in the EU
  for three of the five years immediately
  preceding admission to Trinity.

Problems with Booleans

Humans abuse Boolean operators in the following ways:

- Confusing "and" and "or". "and/or" is a clear sign of confusion but people also write "and" where they mean "or" and vice versa.
- Failing to specify operator precedence, e.g. is "Probability and Statistics or Algebra and Geometry" to be read "Probability and Statistics *or* Algebra and Geometry" or as "Probability *and* Statistics or Algebra *and* Geometry"?
- Using "or" both inclusively and exclusively.
- Using "and" both inclusively and exclusively, e.g. does "you must take Algebra and Geometry" mean you must take only those modules or can you take others as well?

Disambiguating Booleans

In maths, logic, computer science etc. the conventions are that "and" has lower precedence than "not" and higher precedence than "or" and that both "and" and "or" are inclusive.

The prececedence rules enable use to parse complex sentences. The usual way to illustrate parsing is with trees, e.g.

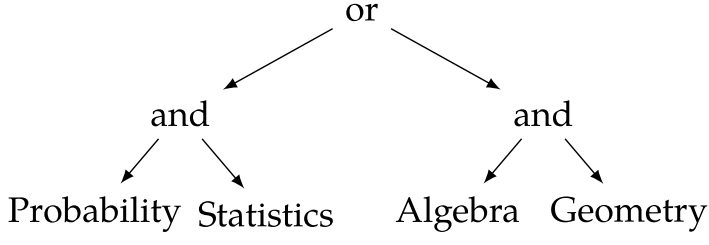


Figure 1: an abstract syntax tree

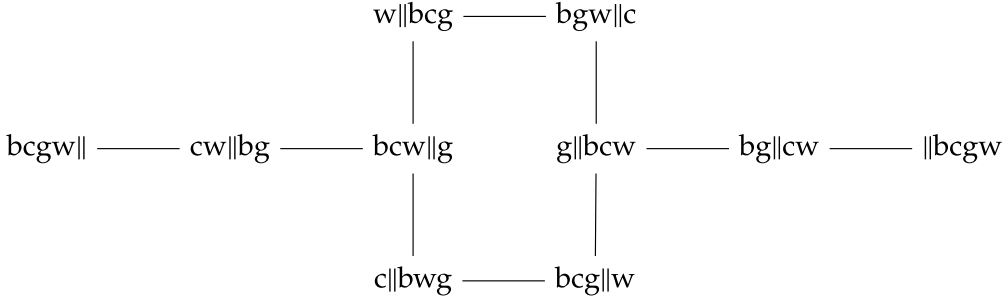
Trees and graphs

Trees are a special case of graphs.

These graphs are unrelated to graphs of functions. Here a graph is a set of *vertices*, connected by *edges*.

Graphs can be directed or undirected, depending on whether the edges have a preferred direction.

Graphs and trees appear everywhere.



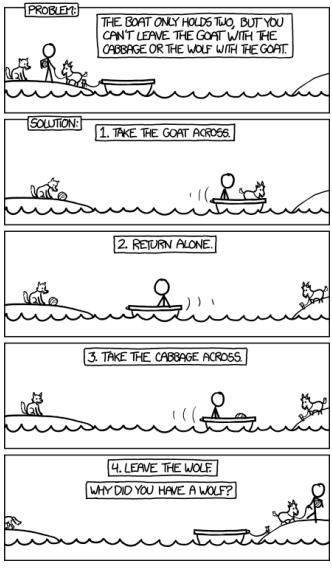


Figure 3: https://xkcd.com/1134

Rules and logic

Having reformulated our modal sentences as declarative and disambiguated our Booleans we've converted our checking problem to a problem in (zeroeth order) logic, i.e. does a particular assignment of truth values to the variables in a statement make it tree.

There are other interesting questions we can ask, e.g.

- Is the set of rules *satisfiable*? In other words, is there any way to make it true?
- Are any rules redundant, i.e. *logical consequences* of other rules?
- Is any rule *tautological*, i.e. always satisfied?

Because our situation is equivalent to zeroeth order logic there are techniques we can borrow to answer these questions.

Idealised machines

Some problems are simpler than others, and some can't be solved at all.

One way to describe this is with a hierarchy of idealised machines. A problem is characterised by how sophisticated a machine you need to solve it, or it may not be solvable by any of them.

At the high end we have Turing machines, which define computability. At an intermediate level we have pushdown automata, which can parse our module rules language. At the lowest level we have finite state automata, which can recognise valid module selections.

A finite state automaton

Remember I said graphs are everywhere? Finite state automata are described by graphs, e.g.