Computability

Announcements:

- PS6 due Monday 12/6 at 11:59PM
  - I will push the course staff about office hours and the newsgroup
- Final exam on Thursday 12/16, 2:00-4:30
  - PS6 tournament and review session M 12/13 and Tu 12/14
  - Which evenings do you prefer?

- What have we covered in CS3110?
- Tools for solving difficult computational problems
  - Abstraction, specification, design
  - Functional programming
  - Concurrency
  - Reasoning about programs
  - Data structures and algorithms

- My personal view of computer scientists versus computer programmers
  - Note that there are 100x as many programmers
- At any time there are some existing programs
- And some programs that don’t exist but clearly could
  - Example: problem set (before anyone solves it)
  - Ukrainian spellchecker for Android

- Computer programmers write such programs
- This can be hard work, and well paid
- Always clear that such a program exists,
  - but not necessarily trivial to write it within resource constraints
    (programmer time, running time/space)

- Computer scientists expand the set of programs we know how to write
• Write programs whose existence is not at all clear
  o Can we make a car that drives itself?
  o Distinguish pictures of cats from dogs?
  o Find broken bones in x-ray images?
  o Create synthetic pictures that look as good as real ones?

• Sometimes we fail
  o Quite often, in fact
  o “If you aren’t occasionally failing, then you are working on problems that are too easy.”

• Sometimes we discover that a problem is fundamentally hard
  o It wasn’t just that the person who tried it wasn’t smart enough

• This is the topic of our final lecture
• Boolean-valued functions (true/false) are generally pretty easy to write.

• Consider the following question: does a function of one argument terminate or run forever, given this input?
  o halts(f,a) will be true or false depending on if f(a) halts
  o **Boolean-valued** function

• Note that we aren’t going to write in OCaml because types get in the way

• Now consider a new **Boolean-valued** function safely(g)
  o First we check if halts(g,g), and if so we return not(g(g))
  o Otherwise we just return false
  o In pseudocode (NOT in ML) we have

  ```
  safely(g) = if halts(g,g) then not(g,g) else false
  ```

  o Ignoring type checking you can do things like:

  ```
  safely(fun(f)->f(24) != 42)
  ```

• OK, now what is the value of safely(safely) ?
  • It’s the value of not(safely(safely)). Oops!

• Resolution: you can’t have a function like halts.
  • In any language, no matter how smart you are.
  • Determining whether or not a program halts is undecidable
  • The only way you can figure out what a program does it to run it!
  • Related to Cantor’s proof of more reals than integers, Goedel’s incompleteness proof, Russell’s paradox
    o All of these are “diagonalization” arguments

• This has huge real-life consequences.
  o Microsoft design of plug-ins (requiring burglars to sign in)
  o Virtualization
  o Virus issues
• Computer scientists tend to informally say that all programming languages are the same,
  o i.e. anything you can do in one language you can do in another
• There is a mathematically precise way to express this
  o Turing equivalence, see CS3810
  o Taught by John Hopcroft, Turing-award winner
• Weaker languages can actually be better
  o PDF versus postscript

• How do you tell if a problem is undecidable?
• It’s not always obvious, though there is one great (sound) heuristic

• Consider the following child’s game:
  o We are given types of blocks over symbols, such as a,b,c
  o Infinite set of blocks of each type
  o Find a sequence of blocks so that the top symbols and the bottom ones are the same
• Example 1:

\[
\begin{array}{ccc}
  a & ab & bba \\
  baa & aa & bb \\
i = 1 & i = 2 & i = 3
\end{array}
\]

• Solution: 3, 2, 3, 1

\[
\begin{array}{cccc}
  bba & ab & bba & a \\
  bb & aa & bb & baa \\
i_1 = 3 & i_2 = 2 & i_3 = 3 & i_4 = 1
\end{array}
\]
• Example 2:

Solution: 1, any number of 2, 3

Can we write a program to solve this? It depends!
• For a binary alphabet, it is decidable (first example)
• For an alphabet with 7 or more characters it is undecidable
• For 3 (second example) or more characters it is unknown!

Suppose we can use no more than k blocks (including copies). Is it decidable?
• Yes – it is finite!
• But it is actually NP-hard, so can’t do better than brute force