Guaranteeing resource bounds in computations

Mike Burrell
For the non-computer scientists

- A **computation** is finite and **unambiguous** description of how to turn input into output

![Diagram with nodes and arrows showing computations]

The car is moving and the brakes have locked

- Release brakes for 1ms
Motivation

- Software is required to meet resource constraints
  - E.g., time
- We can test software, but we never get a 100% guarantee from testing
- Can the computer help us?
Problem

- Since the 1930s we’ve known it’s impossible to determine if a computation halts or not
- Since the 1950s we’ve known it’s impossible to determine anything nontrivial about a computation
- These restrictions only hold for universally powerful models of computation!
Hailstone problem

\[ hs(x) = \]
\[
  \text{if } x == 1 \text{ then } 1 \\
  \text{else if } is\text{Odd}(x) \text{ then } 1 + hs(3 \times x + 1) \\
  \text{else } 1 + hs(x / 2)
\]
Hailstone problem

\[hs(x) = \]
\[\text{if } x == 1 \text{ then } 1\]
\[\text{else if } isOdd(x) \text{ then } 1 + hs(3 * x + 1)\]
\[\text{else } 1 + hs(x / 2)\]

E.g.,
\[hs(6) \Rightarrow 1 + hs(3) \Rightarrow 2 + hs(10)\]
\[\Rightarrow 3 + hs(5) \Rightarrow 4 + hs(16) \Rightarrow 5 + hs(8)\]
\[\Rightarrow 6 + hs(4) \Rightarrow 7 + hs(2) \Rightarrow 8 + hs(1)\]
\[\Rightarrow 9\]
Recursively enumerable
Recursively enumerable

- Halting problem
Recursively enumerable

- Halting problem

Recursive
Recursively enumerable

- Halting problem

Recursive

- solving Chess, Go, etc.
- Sudoku
Recursively enumerable

- Halting problem

Recursive

- solving Chess, Go, etc.
- Sudoku

P
Recursively enumerable

- Halting problem

Recursive

- solving Chess, Go, etc.
- Sudoku

P

- sorting
Recursively enumerable

- Halting problem

Recursive

- solving Chess, Go, etc.
- Sudoku

- string matching
- sorting
- graph reachability
- dynamic programming
- parsing
- greatest common divisor
- search trees
- encrypting
- 3D rasterizing
- hashing
- compressing
- matrix operations
- primality
- graph reachability
- dynamic programming
- parsing
- greatest common divisor
- search trees
- encrypting
- 3D rasterizing
- hashing
- compressing
- matrix operations
- primality
Predicting time

• Disallow any side-effects

• No global state

• No variable assignments

• Allow only “safe” recursion

• We can only recurse the thing we recurse on is smaller
Safe recursion

- Enforcing safe recursion does mean everything will halt
  - We can write software which runs in exponential time (or worse!)
  - Gives us little to go on for predicting bounds of polynomial-time computations

- We add further restrictions
  - The number of times you can recurse each time
  - The number of times you can reference a particular value
Limiting recursion is restrictive
Limiting duplicating variables is extremely restrictive
Disallowing side-effects can be restrictive
Idea: let the compiler restructure things to allow, e.g., an object system
Conclusion

- Not all software can be written in a restricted language
- Most can
- That which is can have properties proven about it

The car is moving and the brakes have locked. Will this step finish before you hit the tree? Release brakes for 1ms.