GEM 1501 Problem Solving With Computers

Lecture 9:

Algorithmic Universality

Martin Henz

Summary of Previous Lecture

- Undecidability
  - Examples: Tiling, word correspondence, halting problem
  - Proof of undecidability of halting problem
  - Diagonalization
  - Certificates
  - Highly undecidable problems

Are all computers the same?

- Computers differ in their speed, architecture, design etc.
- Are there problems that can be solved with one computer, but not with the other?
- In practice: yes
- In theory: no
- Given enough time and memory, any computer can "simulate" any other computer

---

Simplifications

- Data: All data are strings!
- Control: Tape, gearbox, narrow eye
- Operations: Change the symbol under the eye
- Result: The Turing Machine

The Turing Machine

- Finite set of states
- Finite alphabet of symbols
- Infinite tape
- Head that can read and write symbols on the tape
- State-transition diagram

Detecting Palindromes

- Input tape: `...##abba##...`
- Ideas:
  - For each letter at the beginning, find the corresponding letter at the end.
  - Erase corresponding letters, replacing them by `#`.
  - Report "yes" if the entire string is erased.
  - Report "no" if no corresponding letter is found.

What problems can be solved with Turing machines?

- Turing machines can detect palindromes, add, multiply etc.
- What other problems can they solve?
- Answer: Turing machines can solve any effctively solvable algorithmic problem!
- This statement is called the Church/Turing thesis (1930s).

Evidence for the CT thesis

- Many models of computation exist (Turing machines, Lambda calculus, etc)
- All have been proven to be equivalent.
- In practice, computer scientists routinely translate programs from one computational framework to another, and never encounter fundamental problems.
- The CT thesis is called "thesis" and not "theorem", because it is difficult to describe formally what a computer is.

Another model: Counter programs

- Operations: $X \leftarrow 0, X \leftarrow Y + 1, X \leftarrow Y - 1$
- Control statement: `if` $X = 0$ `goto` G

Example: Multiplying numbers

```
   U <- 0
   Z <- 0
A: if X = 0 goto G
   X <- X-1
   V <- Y+1
   V <- V-1
B: if V=0 goto A
   V <- V-1
   Z <- Z+1
   if U=0 goto B
```

Robustness

- Counter programs and Turing machines are equivalent.
- We can always translate one to the other.
- This process is called simulation.
- As very simple models, Turing machines are used in lower-bound proofs.

Polynomial Equivalence

- There are problems for which counter programs take exponential time, whereas Turing machines run in polynomial time.
- Counter programs are exponentially slower than Turing machines.
- Add the following two statements to counter programs:
  - $X \leftarrow X \times 10$
  - $X \leftarrow X/10$
- We call counter programs with these additional statements "extended counter programs"

Tractability is Robust

- Every polynomial algorithm can run in polynomial time on Turing machines and extended counter programs; they are *polynomially equivalent*.
- Extended counter programs are polynomially equivalent to Turing machines.
- Extended counter programs, Turing machines and all other "reasonable" models of computation are polynomially equivalent.
- Intractability is robust with respect to the choice of the computer.

Turing Machines and P vs. NP

- Nondeterministic Turing machines are Turing machines that can "guess" the right possibility of a number of alternatives.
- Nondeterministic Turing machines can solve NP-complete problems in polynomial time.
- Nondeterminstic Turing machines are not polynomially equivalent to normal Turing machines.
- If we could show that an NP-complete problem cannot be solved using a Turing machine, we would know that $P \neq NP$.

Other kinds of machines

- Finite state machines: One-way Turing machines
- Finite state machines cannot count
- Proof using the Pigeon hole principle

Next Week

- Parallelism, concurrency
- Probabilistic algorithms