

GEM 1501 Problem Solving With Computers

Lecture 8:

Undecidability; Algorithmic Universality

Martin Henz

Summary of Previous Lecture

- Undecidability
- Examples: Tiles, snakes, word correspondence
- Proving undecidability

Program Behavior

- Many problems in computer science are undecidable.
- Examples:
 - Do programs terminate?
 - Do given program lines get executed?
 - Will given programs encounter a certain error (e.g. division by zero)?

Proving Undecidability

- Consider the Halting Problem, i.e. the question whether a given program terminates on a given set of input or not.
- Assume that there is a program Q that decides the halting problem.
- Construct a new program S that applies Q to a given program using the program itself as input. S does not terminate if the answer is “yes” and it does terminate if the answer is “no”.
- Apply S to itself and derive a contradiction!

Diagonalization Proofs

- Table containing halting information Q for program P and input I
- Let us say S takes as input the index of a program.
- S itself must have an index, let's say σ .
- What is the value of the table at position σ ?

Certificates for Undecidable Problems

- Many undecidable problems have one-way certificates.
- Yes-certificates and no-certificates.
- Examples:
 - Word correspondence problem
 - Halting problem
 - Snake problem

Problems with two-way certificates are decidable

- If there are both yes and no-certificates, we can construct an algorithm for deciding the answer.
- Idea: Try all possible yes-certificates and all possible no-certificates in an alternating fashion.

One-way certificates

- All problems with only a one-way certificate are equivalent.
- If we would be able to decide one of them, we could decide all.
- There are problems with no one-way certificates.
- Example (totality problem): Does a given program halt on all possible inputs?

Highly undecidable problems

- Some problems are undecidable, even with “oracles” to decide one-way undecidable problems.
- Example: Tiling with an infinite number of one given tile.

Four fundamental levels of algorithmic behavior

- Highly undecidable problems
- Undecidable problems
- Intractable problems
- Tractable 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 effectively 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: Cunter 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.

Other kinds of machines

- Finite state machines: cannot count
- Stack machines: can count, but not calculate

Next Week

- Parallelism, concurrency
- Probabilistic algorithms