COMP 200: Elements of Computer Science
Fall 2004
Lecture 28: November 5, 2004

Universal Turing Machines

On the Board

Second Exam: available today (spares outside DH 2065)

Final Project Discussion

1. Five to ten page written report; graded on technical content as well as writing (grammar, persuasiveness)
2. Ten to 15 minute oral report in class.

Back to Turing Machines

Last class, we talked about Turing machines as a simplification of the model that we use to think about computing.

1. All input data can be represented with finite strings of symbols drawn from some finite alphabet.
2. All storage that the computer needs can be represented as a single tape of infinite length with only relative addressing (the ability to move left or right from the cell currently being read, as opposed to integer names for every cell — so called, random-access memory).
3. An effective algorithm can be represented with a finite-length string of symbols (and these can be translated into a finite state controller, such as the one for palindrome recognition in Figure 9.24 of the book).
4. The set of commands needed to implement algorithms over symbols is extremely simple — the ability to read and write single symbols from the tape, to move right or left on the tape, and to move from state to state in the controller based on the symbol read from the tape.

Despite these simplifications, we showed that a Turing machine could perform a moderately difficult task — recognizing palindromes.

Programming a TM

Several Turing-machine programming tricks come in handy if we try to think about programming more complex actions.

- The controller can always find scratch space out on the tape. It just looks for an area that has all blanks and is big enough. Since the tape is infinite, such a space must exist.
- The controller can leave behind special characters on the tape to mark positions. The alphabet is finite but need not be small.
- The controller can copy the input data to another location on the tape if it needs to modify or destroy it. This lets us create a Turing machine that leaves the input symbols intact.
- Multi-dimensional data-structures (arrays, for example) can be linearized for storage on a linear tape.

Universal Turing Machines

Turing claimed that any effective algorithm can be implemented as a TM — that is, a finite state controller can be designed that causes the Turing machine to implement the algorithm. (Alonzo Church made a similar claim for his ***lambda calculus.*** The lambda calculus is actually the basis for the Scheme programming language that we used earlier in the course. The equivalence of the lambda calculus and TMs has led people to refer to the claim of generality as the Church-Turing thesis. It is a thesis, rather than a theorem, because it contains some squishy terms — such as "effective".)

How can a Turing machine be universal? (implement *any* effective algorithm?) If we can represent the finite state control of a Turing machine in a symbolic form, then we can built a controller that ***interprets*** such a description and performs the actions that the described machine would take.

- represent transitions in the finite state control as a tuple:

  <source, destination, trigger symbol, write symbol, tape motion>

  and require that the interpreter use finite storage (leaving infinite storage for the interpreted machine).
- tape marker at the start and end of the stored program, say "!"
- read an input symbol, back up to the start of the program, find the current state transitions, check against the input program; then write the tape symbol and take the transition.

Actually, the claim that Turing machines are universal is fairly ***robust***. We can modify the model in many ways without losing its power.

- One-way infinite tape — we can fold the tape in half, using even numbered cells for one direction and odd-numbered cells for the other. Bottom line — infinity is big enough.
- Machine cannot modify its inputs — can copy data elsewhere on the tape.
- Machine must clean up its tape — only input and output, separated by some designated character.
- Multiple tape machines have the same power as single tape machines. (Again, infinity is big enough. Can encode k tapes onto a single tape by alternating them. Stacking them one after another doesn't work as well because expanding one tape is expensive.)
- Two-dimensional tapes have the same property — we can linearize them into a single tape.

Turing Machines and the P versus NP question:

- Deterministic finite state control (at most one transition per alphabet symbol per state) limits the set of algorithms solvable in polynomial time on a TM to PTIME — the set P in the discussion of P and NP.
- Nondeterministic finite state control lets the TM solve NP-complete problems in polynomial time. Unfortunately, the nondeterministic controller is not sequential — it makes oracular guesses — so it cannot be considered sequential. The other model for an NDTM uses a construction to simulate the nondeterminism on a deterministic controller. The construction can require exponential growth in the controller size (and, perhaps in the running time on a TM). Thus, it is an open question as to whether nondeterministic control and deterministic control are equivalent. After all, does P = NP?

Numbering the TMs

Turing machines are important tools in such activities as proving lower bounds on an algorithmic problem's complexity, in reasoning about what can be computed effectively (in polynomial time or space) and in reasoning about what can be computed at all.

One of the simplifications that Turing made was to recognize that a string of symbols can adequately represent any kind of input data. The converse of this observation is that any string of symbols can be viewed as a number in the base that corresponds to the cardinality (set size) of the input alphabet. Just as the digits 901 represent a number (we assume base ten) and the characters FF represent a base 16 (hexadecimal number), so the contents of a TM input tape can be viewed as representing a number.

Given some encoding of TMs for input to some universal TM, it must be true that all the TMs that can serve as input form unique numbers; they are countable and we can define an order on them.

This observation leads to one of the most profound observations in the theory of computability — we cannot create a TM that takes as input a TM and its input and returns YES if the input TM halts on that input and returns NO if the input TM does not halt on that input. This problem, called ***the halting problem***, is one of a handful of classic examples of a problem whose solution cannot be computed. (Proof by diagonalization)