Church Turing Thesis

CS154

Chris Pollett

Mar 22, 2006.

Outline

- Finish talking about RAMs
- Hilbert's Problems
- Describing TMs

Introduction

- Over the last week we have showed that several different computation models can be simulated on a Turing machine.
- Namely, {L,R,S}- TMs, multitape TMs, nondeterministic TMs, and enumerators.
- Last day, we introduced the RAM computational model which is very close to the way a microprocessor might work.
- We showed it could simulate a TM.
- If on the other hand, it can simulated by a TM, it would show that everything at least as far as the computers we use can be done on a TM.
- The Church-Turing Thesis says that every process that can be effectively carried out, can be simulated on a Turing Machine.

Simulating RAMs on TMs

Theorem If L is recognized by a RAM then it is Turing-Recognizable.

Proof: Let P be a RAM program. We will simulate it by a seven tape machine. The first tape will be used to hold the input string and it will never be overwritten. The second tape will be used to represent the content of all the registers. This will be represented by a sequence of semicolon separated pairs i, v. Here i says the register (which may be 0) and v says its value. When a register is updated we copy the pair to the end of our sequence, update the value, then X over the old value. An example sequence might be: 0, 101; XXX 1, 10; _

The states of M are split into m groups where m in the number of instructions in P. Each group implements one instruction. Tape 3 is used to store the current program counter. This is initially 1. At the start of the simulation tape 2 is initialize to the input configuration of a register machine based on the contents of the input tape. Thereafter, at the start of simulating an instruction. The program counter is read and the start state of the group of states of M for that instruction is entered. (see next slide)

Proof Continued

An instruction is then processed, tape 2 is updated, and the program counter on tape 3 is updated, then the next step can be simulated and so on. Most instructions are reasonably straightforward to carry out: To process an instruction that uses indirect addressing of the form (j), tape 4 is used to store the value k of the register j so that we can then go access register k on tape 2. For operations like Add and Sub, tapes 5 and 6 are used to store the operands and tape 7 is used to compute the result. If the RAM halts, the contents of register 0 (the accumulator) are looked up on tape 2, and the TM accepts if the value is positive.

Hilbert's Problems

- Our next example of a computational model which can be simulated by a Turing Machine is a little less obvious.
- In 1900, David Hilbert proposed 23 problems to motivate mathematicians for the next century. The 10th problem of this list was to give a decision procedure for Diophantine equations over the integers.
- Given a polynomial P in the variables $x_1,..x_m$; $z_1,\ldots,z_m$. he wanted some way to tell if $P(\underline{x}; \underline{z}) = 0$ for some integer settings of the variables $z_1,\ldots,z_m$.

More on Diophantine Equations

- One can use Diophantine equations to check the graphs of functions:

  $w - xy = 0$ iff $xy = w$

- We can simulate AND, OR, =, Even(x):

  $P = 0$ AND $Q = 0$ iff $P^2 + Q^2 = 0$

  $P = 0$ OR $Q = 0$ iff $P*Q = 0$

  $P = Q$ iff $P - Q = 0$

  Even(x) iff for some z, $2z = x$.

- It turns out Matiyasevich proved any Turing Recogizable language can be simulated as a solution to a Diophantine equation.

Yet More on Diophantine Equations

- One can construct a TM which recognizes the language:

{P | P is a polynomial with an integral root }

- Idea is machine tries all settings of the variables so that the absolute value of these settings is less than n for each fixed n.

Describing TMs

- In computer science we are interested in algorithms.
- The Church-Turing thesis says we can implement any reasonable algorithm on a Turing Machine.
- Now that we have such a machine model, when we need to specify an algorithm, we will typically use one of three levels of description:
  - **formal** -- we completely write out the Turing Machine in terms of states, alphabets and transitions, etc for the problem.
  - **implementation-level** -- using English sentences we say what a Turing machine implementing the algorithm would do, done to how it moves its tape head etc and stores things on the tape.
  - **high-level** -- using English sentences to describe the algorithm ignoring implementation level details about what exactly is stored on the tape and how the tape head in each situation.

Example High Level Description

- Let A = {<G> | G is a connected undirected graph}
- By <G> we mean an encoding of the graph G into some alphabet. For instance, if G is the ordered pair V,E and and V has vertices 1,2,3,4 and edges between 1 and 2 and 3 and 4. We could use the ACSII string <{1,2,3,4}, {{1,2}, {3,4}}> to encode G. This particular <G> is not in A.
- A high level description of a machine for A is:

On input <G>:

1. Select the first node of G and mark it.
2. Repeat the following stage until no new nodes are marked:
   a) For each node in G, mark it if it is attached by an edge to a node that it already marked.
3. Scan all nodes of G to determine, whether they are all marked. If they are, accept; otherwise, reject.