Yet More Turing Machines

CS154

Chris Pollett

Mar 20, 2006.

Outline

- Equivalence with Enumerators
- RAMs

Equivalence with Enumerators

Theorem A language is Turing-recognizable if and only if some enumerator enumerates it.

Proof: Suppose we have an enumerator E that enumerates a language A. A high level description of a TM that recognizes A is:

M= "On input w:

1. Run E. Every time that E outputs a string, compare it with w.
2. If w ever appears in the output of E, accept."

On the other hand, suppose M recognizes A. Order all strings in Σ^* in some fixed way: s_1, s_2 ,… . Then we can construct an enumerator for A as follows:

E="Ignore the input.

1. Repeat the following for i= 1,2,3,…
2. Run M for i steps on each input s_1, s_2, … s_i.
3. If any computation accepts, print out the corresponding s_j.

Random Access Machines (RAMs)

- Consist of a program which acts on an arrays of registers.
- Each register is capable of storing an arbitrarily large integers (either positive or negative).
- Register 0 is called the accumulator. It is where any computations will be done.
- A RAM program consists of a finite sequence of instructions $P=(p_1, p_2, \ldots p_n)$.
- The machine has a program counter which says what instruction is to be executed next.
- This initially starts at the first instruction. An input $w=w_1,\ldots,w_n$ is initially placed symbol-wise into registers 1 through n. (We assume some encoding of the alphabet into the integers). Register n+1 has the encoding for the blank square '_' for the end of the input. All other registers are 0.
- A step of the machine consists of looking at the instruction pointed to by the program counter, executing that instruction, then adding 1 to the program counter if the instruction does not modify the program counter and is not the halt instruction.
- Upon halting, the output of the computation is the contents of the accumulator. For languages, we say w is in the language of the RAM, if the accumulator is positive, and is not in the language otherwise.

Allowable Instructions

- Each instruction is from the list:
  1. Read j /* read into register j into accumulator */
  2. Read (j) /* look up value v of register j then read register v into the accumulator*/
  3. Store j /* store accumulator's value into register j */
  4. Store (j) /* look up value v of register j then store acumulators values into register v*/
  5. Load x /* set the accumulator's value to   x */
  6. Add j /* add the value of register j to the accumulator's value */
  7. Sub j /* subtract the value of register j from the acummulator's value */
  8. Half /* divide accumulator's value by 2 round down (aka shift left)*/
  9. Jump j /* set the program counter to be the jth instruction */
  10. JPos j /* if the accumulator is positive, then set the program counter to be j */
  11. JZero j /* if the accumulator is zero, then set the program counter to be j */
  12. JNeg j /* if the accumulator is negative, then set the program counter to be j */
  13. HALT /* stop execution */

Example Program for Multiplication

Suppose we input into register 1 and 2 two number i_1 and i_2 we would like to multiply these two numbers:

1. Read 1 //(Register 1 contains i_1 ; during the kth iteration
2. Store 5 // Register 5 contains $i_1 2^k$. At the start k=0)
3. Read 2
4. Store 2 //(Register 2 contains $\lfloor i_2/2^k \rfloor$ just before we increment k)
5. Half //(k is incremented, and the k iteration begins)
6. Store 3 // (Register 3 contains half register 2)
7. Add 3 //(so now accumulator is twice register 3)
8. Sub 2 //(accumulator will be 0 if low order bit of what was stored in register 2 is 0)
9. JZero 13
10. Read 4 //(the effect is we add register 5 to register 4
11. Add 5 // only if the kth least significant bit of i_2 is 0)
12. Store 4 //(Register 4 contains i_1*(i_2 mod 2^k))
13. Read 5
14. Add 5
15. Store 5 //(see comment of instruction 3)
16. Read 3
17. JZero 19
18. Jump 4 //(if not, we repeat)
19. Read 4 //(the result)
20. Halt

Simulation Set-up

Theorem If L is Turing-recognized then there is a RAM which recognizes it.

Proof: Let M be the TM recognizing L. The RAM first moves the input to Registers 4 through n+3. This is actually a little tricky to do. First, the RAM reads registers 1 and 2. Since the tape alphabet of M is finite, the RAM can "remember" these values without having to write them to some other register by branching to different subroutines to execute. As these values are now remembered, register 1 can be used as a counter to count up. By doing Read (1) instructions and incrementing register 1, until the encoding of the '_' for the end of the input is found we scan to the end of the input. We look one register before this, read it, and store it into register 2. Adding 3 to register 1, we can then load the accumulator with register 2's values and do a Store (1). This moves the nth symbol to register n+3. By subtracting 4 from register 1 and doing a read (1) we can get the next symbols to move and so on. We are now almost ready to being the simulation.

More Proof

Register 1 is used to hold the current tape square number being read by the TM in the simulation and this is initially set to 4 , the first square of the input. Register 3 holds a special start of tape symbol. The program now tries to simulate steps of the Turing machine. The program has a sequence of instructions simulating each state q of M and has a subsequence of these instruction, N(q,j), for handling the transition for state q while reading the jth type of alphabet symbol.

N(q, j)

Suppose δ(q, j) = (p, k, D). Here D is a direction. To simulate this we do:

```
N(q,j)      Load j
N(q,j)+1    Store 2
N(q,j)+2    Read (1)
N(q,j)+3    Sub 2 //(if the tape position we are reading has value j this will be 0)
N(q,j)+4    JZero N(q, j) + 6
N(q,j)+5    Jump N(q, j+1) //(if we are not reading a ‘j’ check if we are reading a
            ‘j+1’)
N(q,j)+6    Load k
N(q,j)+7    Store (1)
N(q,j)+8    Load -1, 0, 1 //(depending on which direction the move was)
N(q,j)+9    Add 1
N(q,j)+10   Store 1
N(q,j)+11   Jump N(p, k)
```