More Simulations

CS154

Chris Pollett

Apr 25, 2007.

Outline

- Multi-tape Turing Machines
- RAMS

k-tape Turing Machine

- One way you might try to improve the power of a TM is to allow multiple tapes.

Definition A k-tape TM, where $k \geq 1$ is an integer, is a six-tuple $M=(K,\Sigma,\partial,s,\_,F)$ where K,Σ,s are as in the 1-tape case. Now, however, the transition functions is a map

$\partial : K \times \Sigma^k \rightarrow K \times \Sigma^k \times \{L,R\}^k$

- Basically the heads on each tape can move independently of each other.
- For example, with a two tape machine an algorithm for palindrome testing is easy.
  - We set up the transition function so it first copies the first tape input to the second tape.
  - Then it rewinds the first tape and leaves the second tape at the end of the input.
  - Then the first tape moves right while the second tape moves left and we compare the two tape symbol by symbol. If they don't match we halt and reject. If the second tape gets back to the _ then we accept.

TIME and SPACE classes.

- We shall use the k-tape model of TM as our basic model to study time and space complexity.
- Let f:**N --> N**. We say that machine M operates within time f(n) if for any input string x, the time require by M on input x is at most f(|x|). Here |x| is the length of x as a string. We can make a similar definition for space.

Defn. We say that a language L is in **TIME**(f(n)) (resp. SPACE(f(n))) if it is decided by some k-tape TM in time f(n) (resp. space f(n)).

- For example, the algorithm for palindrome in time TIME(3(n+2)).
- You can show for a single tape machine for palindrome you need at least time $\Omega(n^2)$.
- How well can a 1-tape machine simulate a k-tape machine?

Simulating k-tape by 1-tape

Thm. Given any k-tape machine M that operates within time f(n), we can construct a 1-tape machine M´operating within time $O((f(n))^2)$.

Proof. Let $M=(K,\Sigma,\partial,s,\_,F)$ be a k tape machine.

- The idea is M´ alphabet, Σ', is going to be expanded to include symbol #´ to denote the last used square of a tape. And we are going to add to Σ' a symbol $\underline{b}$ for each symbol b in Σ.
- A configuration of M can now be written as:

$$(q, \#w_1\underline{a}_1v_1\#'w_2\underline{a}_2v_2\#'\ldots\#'w_k\underline{a}_kv_k\#')$$

- So except for the state which we can keep track of in K´ the rest of the state is a string over Σ'.
- We will use new states K´ to keep track of the state of M during a simulation step.
- To simulate M, we first convert the input into the initial configuration of M viewed as a string.
- Then to simulate a step we scan left to right the current configuration string, noting what symbol is being read by each tape in our finite control.
- Next we rewind the tape and we then do passes again to update each tapes configuration.
- In the worst case we need to expand the number of tape square of each tape by 1. So we could need (k(f(|x|)+1)+1, passes to simulate 1 step.
- So simulating f(|x|) steps take at most f(|x|)((k(f(|x|)+1)+1) times which is $O((f(n))^2)$.

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). 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 $i_1 * (i_2 \bmod 2^k)$)
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

Runtime of a RAM

- Let D be a set of finite sequences of integers.
- A program P computes a function f from D to the integers if for all I in D, the programP halts with f(I) in its accumulator
- Let len(i) be the binary length of integer i.
- The length of the input is defined as $len(I) = \sum_j len(i_j)$.
- We say P *runs in time* g(n) if for all I, such that len(I) = n, it runs in at most g(len(I)) steps.

Simulation Set-up

Theorem If L is in TIME(f(n)), then there is a RAM

which computes it in times O(f(n)).

Proof: Let M be the TM recognizing L. We assume one black space is added to any input and the tape alphabet has been encoded as integers. 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 begin 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 $\delta(q, j) = (p, k, D)$. 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)