More Tape Theorems, Universal Machines

CS254

Chris Pollett

Sep 11, 2006.

Outline

- Linear Speedup
- RAMs
- Nondeterminism,
- Universal Turing Machines

In our results today we will use the books version of TMs. These can write and move in one step -- unlike the model last day which could only do one or the other in one step.

Linear Speedup

Thm. Let L be in TIME(f(n)). Then for any $\varepsilon > 0$, L is also in TIME(f´(n)) where $f'(n) = \varepsilon f(n) + n + 2$.

Proof. Let $M=(K,\Sigma, \partial, s)$ be a k-tape machine that decides L in TIME(f(n)). Let k´ be 2 if k=1 and be k otherwise. We will get the speedup by making a simulating k´ tape machine $M'=(K',\Sigma', \partial', s')$ that encodes several of moves of M by one move of M´. To be specific let $\Sigma'=\Sigma\cup\Sigma^m$ --we'll fix m in a moment. To begin M´ scans the input $x= x_1\ldots x_n$ writing for each m adjacent symbols a single symbol in Σ'. At the end of this $m[|x|/m]+2$ step process the second tape has n/m squares with symbols. We then delete the first tape and use the second tape as the input from now on. The machine M´ then simulates m steps of M using 6 or fewer steps. At the beginning of a simulating stage the state of M´consists of a k+1 tuple $(q, j_1,..,j_k)$ where q is M's state at the start of the stage, and the j_i's represent where within the current m-block on each tape M's head would be. For each tape M´ then scan one square left, two right, one left. It remembers in its controls the symbols it saw to the left and right. Using this information M´ can now completely predict where M would go in its next M moves. It then updates the at most two tape square it needs to update and starts simulating the next. If M accepts or reject, so does M´. Simulating f(n) steps takes time $|x|+2 +6[f(|x|)/m]$ steps. So for $m = [6/\varepsilon]$ the theorem holds.

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)

Simulating RAMs on TMs

Theorem If L is recognized by a RAM in time f(n) then it is in TIME($O(f(n)^3)$).

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 runtime for results comes because this tape can be shown (see book) to grow as $O(f(n)^2)$.

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.