COMS3261: Computer Science Theory

Fall 2013

Mihalis Yannakakis

Lecture 20, 11/18/13

"Real" Computers vs. TMs

- Typical RC: Memory = set of addresses 0,1,2,… storing words (caches, RAM, discs etc)
- program: instructions stored in memory in machine (or assembly) code

  operations(memory locations of operands) → mem loc (for result)

  example: +, - , *, conditional jumps etc

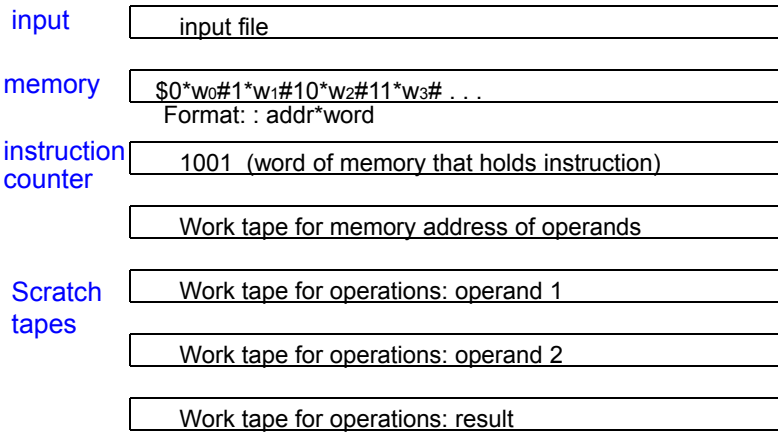
  Includes indirect addressing

Turing Machine → Real Computer

- If RC has bounded memory then finite automaton, though huge.
- But realistically can think of having removable storage (eg. unbounded supply of discs) -> unbounded memory. Then:
- Can easily simulate any Turing machine : RC just keeps track of state of TM and tape contents. “Main memory” of RC has portion of tape around tape head, and then have two stacks of discs for further portions of tape that an operator can mount if needed

Real Computer → Turing Machine

Tapes:



RC $\rightarrow$TM

Instruction Cycle:

1. Match number i in instruction counter with address in tape 2 $\rightarrow$ instruction = w_i = opcode addresses
2. Copy the addresses of the locations needed for the instruction on tape 4, fetch the contents and copy on work tapes
3. Perform operation (+,-,x) in work tape and copy to memory tape
4. Update instruction counter.

Time of Simulation

- If word length is unlimited and can multiply in one RC cycle (unrealistic) then length (#bits) doubles with every multiplication: $a \times b \rightarrow$ #bits = #bits(a) + #bits(b)

  $\Rightarrow$ after t steps, get numbers with 2^t bits
- More realistic: Bound word length by a constant (e.g. 64), or allow only instructions that add ≤ 1 bit, or charge RC for each operation time ~ #bits of operands
- Then TM can operate on k-bit numbers in $O(k^2)$ steps

  $\Rightarrow$ t steps of RC can be simulated by $O(t^3)$ steps of TM

Pf. In model where each operation of RC adds at most 1 bit, after t steps all words $\leq$ t+c bits (c=initial length) $\Rightarrow$ tape grows only to $O(t^2)$ (t words) $\Rightarrow$ each operation takes $O(t^2)$ time on TM

Polynomial time overhead

TM vs RC

- Implications of the simulation:

1. Turing Machines are a simple useful model for determining which problems can be solved by computers (are computable/decidable) and which cannot be solved
   since any RC can be simulated by a TM
2. TM are useful also to make gross distinctions in the time complexity of problems,
   eg. polynomial time vs. exponential time –
   since the overhead of the simulation is polynomial

For finer distinctions in lower complexities, eg. n vs nlogn vs n^2 we use a finer model with random access memory

Undecidability

- Will show there are undecidable languages
- Key:

- Inputs are strings

- TM specifications (programs) can be written also as strings and treated themselves as input

$\Rightarrow$ Can apply TM's on their own codes and make them do inconsistent things like "This sentence is false"

- In particular, the following are undecidable:
- Does this TM accept (the code for) itself as input?
- Does this TM accept this input?

Encoding of TMs

- Encoding: Mapping every TM to a string over a fixed alphabet, e.g. {0,1} (and hence then $\rightarrow$ natural number)
- Want code <M>=code(M) so that another TM (program) can read <M> and know what the components of M are:
  $M = (Q,\Sigma,\Gamma, \delta,q_1,F)$
- Example: Write program in ASCII and encode in {0,1}
- Many ways to do it.

Encoding of TMs

- Assume for now that 1-tape deterministic TM
- States: Assume $Q= \{q_1 , q_2,\ldots\}$
  Encode each state q_i in unary as 0^i
- Input and Tape symbols: Assume $\Gamma= \{X_1, X_2,\ldots\}$ where say $X_1,..X_{|\Sigma|}$ is Σ and $X_{|\Sigma|+1}$ is B (blank)
  Encode each symbol X_i of Γ as 0^i,
- Head directions: L , R mapped to 0, 00
- Use 1's as separators
- Final states: codes of states in F separated by 1
- Example: $F=\{q_2,q_4,q_7\} \rightarrow$ <F> = $0^2 10^4 10^7$

Encoding of TMs (ctd)

- Transitions:
- Code of a transition $\delta(q_i,X_j) = (q_k,X_l,D_m)$: $0^i10^j10^k10^l10^m$
- Code of transition list δ : $C_1 11 C_2 11 C_n$ where there are say n transition rules with codes $C_1,\ldots C_n$
- Example:
- $\delta(q_1,X_1)=(q_2,X_3,L)$, $\delta(q_1,X_2) = (q_1,X_1,R)$, $\delta(q_2,X_1) = (q_3,X_2,L)$
- $<\delta> = 01010^2 10^3 10 11 010^2 101010^2 11 0^2 1010^3 10^2 10$

- Code of TM: $110^{|Q|}110^{|\Sigma|}110^{|\Gamma|}11<F>11<\delta>$
- Can treat it as a string, or a binary number (positive integer)
- Multitape TMs, Nondeterministic TMs: Similar

Countable Set

- Countable set: can map 1-1 to positive integers (mapping f does not have to be onto)
- Can list them in order of f(.) as : $a_1, a_2, \ldots$

- Some other countable sets:
- All (+/-) integers: 0,1,-1,2,-2, 3,-3..
- i.e. a_i = 0 if i=1; else i/2 if i even; else –(i-1)/2 if i odd >1

- Pairs of numbers, triples,…
  ex: $(i,j,k) \rightarrow 10^i10^j10^k$ or $(i,j,k) \rightarrow 2^i3^j5^k$

- Rational numbers: i/j with i,j relatively prime (= pairs i,j)

Strings , TMs → Numbers

- Binary strings: prepend a 1: w → 1w
- So: ε→1, 0→10 =2, 1→11=3, 00→100 =4, … etc
- enumerates by length, and lexicographically for same length
- Each number i≥1 ↔ binary string w_i

- Strings over an arbitrary finite alphabet Σ: can do same
  - or treat as r-base number where r =|Σ|+1 (Σ ={a_1,…,a_{r-1}})
  - or code Σ in {0,1} → binary string → number

- TMs mapped to numbers. Some numbers i not legal codes.
- If i is legal TM code then let M_i = TM with code i; if not, let M_i = a default TM (e.g. 1 state, no moves) that accepts ∅

Diagonalization Language

- Some sets are not countable (can't be mapped to numbers)
- Real numbers (even just real numbers in [0,1]): Cantor
- Set of languages over {0,1} (any alphabet with ≥ 2 symbols)

- Diagonalization argument

- Diagonalization language
- L_d = { w_i | w_i is not in L(M_i) } = set of binary strings w that are not accepted by the TM whose code is w (if w is an illegal code then in L_d: default TM accepts ∅ }

Diagonalization Language L_d is not r.e.

- Table: Row i $\leftrightarrow$ TM Mi; column j $\leftrightarrow$ wj ;
  entry (i,j) = 1 if $w_i \in L(M_i)$, =0 otherwise

TMs	1	2	3	4	5	strings
1	0	1	1	0	1	
2	1	1	0	0	0	
3	1	1	0	1	1	
4	0	0	1	0	0	

diagonal

Characteristic vector of L_d = complement of diagonal:
For all i, w_i is in L_d iff i-th entry of diagonal is 0 $\Rightarrow$ L_d differs from $L(M_i)$ in string w_i $\Rightarrow$ no TM accepts L_d

Recursive vs RE languages (reminder)

- Can assume wlog that TM halts when it accepts
- But TM may reject either by running forever or by halting at a nonaccepting state.
- Recursive language L: $\exists$ TM M that halts on every input and L=L(M) (M accepts w $\Leftrightarrow$ w $\in$L) (M =algorithm for L)
- Recursively enumerable language L: L=L(M) for some TM M (M may not halt on some inputs not in L)
- Decidable problem $\Leftrightarrow$ recursive language
- Undecidable problem: Not a recursive language; could be recursively enumerable

- L_d is not r.e.

Relationship