Variations on the TM Model

The goal of this section is not to teach you how to program TM's but to show how powerful they are. Eventually we show they are as powerful as a computer which has unlimited time and space.

Programming Techniques for TM's

- Store data in a state, e.g., remembering the first symbol you've seen.
- Think of the tape as having multiple tracks, e.g. marking the data.
- Example: the subroutine for `Shift`

TM's with multiple tapes

A k-tape TM is like a 1-tape TM except there are k read-write heads and k tapes.

Initially,

- the input is on the first tape and the others are blank;
- a head is allowed to stay put.
- the heads are at the start of the tapes.

TM's with multiple tapes

A move of the machine depends on the state and the symbol scanned by EACH of the heads.

In one move:

- the TM enters a new state;
- on each tape a new symbol is written;
- each head may move L, R, or S (remain stationery)
- $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$ where k is the number of tapes.

It might seem that having more tapes makes some problems easier to program.

Example

Given the string w, output ww, using a 2-tape machine.

1. Move both heads to the right, copying w to second tape.

2. Move second head to the left.

3. Move both heads to the right, copying w onto first tape, after first occurrence of w.

Example 2

Devise a 2-tape machine to accept $\{wcw \mid w \in \{0,1\}^*\}$.

A 1-tape TM can simulate a k-tape TM

Let M be a k-tape TM with alphabet Σ. We show how to build S, a 1-tape TM which simulates M.

S has an expanded alphabet which includes the original tape symbols supplemented by the same symbols but with a $'$ over them and a special $\#$ symbol.

S contains a copy of the contents of the first tape, then the second, then the third, etc, separated by $\#$ marks. It replaces one symbol a for each tape with a' if the head for that tape is reading that symbol.

To simulate a step in M, S must scan the tape until it sees all three marked places and remember in its state the contents of each and then carry out the correct moves based on the transition function of M. In particular, it may need to shift whole chunks of tape contents to make room for a longer configuration.

Simulation Overhead

It is important to understand the cost of a simulation, in terms of time and space. E.g., when simulating a k-tape machine, each simulation step requires two whole scans of the tape – one to read the k tape-head positions, and one to update. The update could involve k shifts, which could require moving up to the whole tape.

So if the original machine takes time $O(t(n))$, it could use up to $O(t(n))$ tape cells, and the simulation tape has $O(t(n))$ cells. So each simulated step takes $O(t(n)) + O(t(n)) + k \cdot O(t(n)) = O(t(n))$ simulation steps. Since there are $O(t(n))$ steps to simulate, the entire simulation takes time $O(t(n)^2)$.

We will consider running time in more detail later in the course.

Two-Way Infinite Tapes

How can a one-way (semi-infinite) tape simulate a two-way infinite tape?

One solution: Use two tapes – one to represent the "right half" and one to represent the "left half". How do transitions work?

Nondeterministic Turing machines

A *nondeterministic Turing machine–NTM* is like an ordinary TM except that the transition function δ is defined differently: for each state q and tape symbol X, $\delta(q, X)$ is a set of triples of state, tape symbol to be written, and direction to move:

$$\{(q_1, Y_1, D_1), (q_2, Y_2, D_2), ..., (q_k, Y_k, D_k)\}$$

for any finite integer k. The NTM can choose at any step which triple to determine the next move.

The NTM accepts a string w if some sequence of moves from the initial ID with w as input leads to the accept state. If other sequences on w lead to nonaccept states, it's irrelevant.

Example

$L = \{\text{composite numbers}\}$, where the input is given as a binary number.

1. Nondeterministically choose two numbers p, q and write their binary representation next to the input.

2. Multiply p and q and replace p and q on the tape with their product.

3. Check if the two integers n and pq are the same. If they're the same, go to a final state.

Nondeterministic and deterministic TM's

If a nondeterministic TM N accepts a language then we can construct a deterministic TM which accepts that language.

Idea: Construct D so that it systematically looks at all possible computations of N searching for one that is accepted. If it finds one, then it accepts.

What if the first computation we try doesn't halt?

We need to *dovetail*. D first tries all possible computations of N consisting of 1 step, all with 2 steps, etc.

A 3-tape machine D

We describe D as a 3-tape machine to simulate N. We know any 3-tape machine can be simulated by a standard 1-tape machine. Let b be the maximum number of choices in each step of N.

IDEA: How to explore the computation paths? Use breadthfirst search of tree of possible computations Each node in tree is numbered by the path used to get there.

1. Tape 1 contains the input sequence w and is fixed.

2. Tape 3 contains the address of the node we are exploring.

3. On tape 2 D runs the simulation from the root to the node whose address is on tape 3.

4. If node is reached and it is an accepting configuration, accept.

5. Else replace address on Tape 3 with *lexicographically next* string and go to step 3..

Question: What is the cost of this simulation?

Theorem: A language is Turing recognizable iff some nondeterministic TM recognizes it.

A Turing machine is a *decider* if all branches halt on all inputs.

Theorem: A language is decidable iff some nondeterministic TM decides it.

Enumerators

An *enumerator* is a variant of a TM which prints out strings. A language is enumerated if all its strings are printed out by an enumerator, possibly with repetitions.

Theorem: A language is Turing recognizable iff some enumerator enumerates it.

Proof; ($\leftarrow$) First, assume we have an enumerator for a language A. Use it to construct a TM M which recognizes the language.

- Given an input w, M simulates E. Each time E outputs a string, it compares it to w.

- M accepts if w is output.

Other direction: Assume we have a TM M which recognizes A. We construct an enumerator E.

1. E ignores any input.

2. For $i = 1, 2, 3, ...$

   (a) For each possible string $s \in \{0, 1\}^*$
      i. Run M on s for i steps.
      ii. If M accepts, print s.

Simulating a TM by a computer

- Use a table of transitions to look up changes in state
- How to simulate infinite tape?
  Keep disks in two stacks. The further down the stack, the further away from the location of the TM head.

Simulating a computer by a TM

Our assumptions about a computer:

- Storage: unlimited length words, and addresses 0,1,2,...
- Computer program is stored in words in memory; indirect addressing is permitted
- Each instruction contains a finite number of words and changes the value of at most one word.
- Any operation can be performed on any word. (No need for registers.)

The TM

We use a multitape TM, with the following tapes:

- Tape 1: start of tape symbol followed by the address, followed by the word at that address. The addresses are ordered 0,1,2,etc.
- Tape 2: instruction counter holds address.
- Tape 3: memory address or contents of memory address
- Tape 4: Computer's input file
- Tape 5: Scratch

Simulating an instruction cycle

- Search the first tape for the address matching the instruction counter
- When the instruction address is found, its value determines an operation and possibly another address containing a value.
- If the instruction requires a value of some address, then copy that address onto third tape and find its value and copy it onto the third tape.
- Execute the instruction using that value.
  - If more space in a word is needed, the entire nonblank tape to the right must be shifted over.
  - Example: Jump.
- After performing the instruction, if it's not a jump, add 1 to the instruction counter and begin again.

Running times

- Can we do this efficiently?
- We can't allow multiplication of arbitrarily large numbers in one operation. Otherwise, the computer could square a number at every step, starting with 2, and after n steps it would have a number of size 2^{2^n}, which would have 2^n bits and require exponential time for the TM.
- So either limit wordsize (e.g., to 64 bits,) or allow words to grow only by one bit with each operation.

Running times

- In our simulation, after $t(n)$ steps by the computer, the largest word is $t(n)$ bits, so $t(n)$ cells are needed to represent the computer word in the TM, and the nonblank tape is $t(n)^2$ long.

- Then each lookup and each shift takes the TM only $t(n)^2$ steps, and in general, for usual instructions, there's a cost of only $t(n)^2$ per step of the computer.

- $t(n)$ steps of the computer can be simulated with $t(n)^3$ steps on the TM

- Then the single tape TM can simulate the computer with $t(n)^6$ steps.