Stupid Turing Machine Tricks

Often it is useful to think of the state and the tape symbol as having structure.

- The components of a tape symbol are *tracks*.
- Usually one component of the state is the *control*, responsible for running the "program" of the TM; other components hold data.

Example

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$. Suppose the program of M needs to swap the contents of adjacent tape cells sometimes.

- Some of the states in Q will be of the form $[q, X]$ and $[p, X]$, where X is any symbol in Γ. We also need states r to begin and s to end.
- In state r, M will pick up the symbol scanned into the data portion of the state.
  - $\delta(r, X) = ([q, X], X, R)$ for all X in Γ.
- In control state q, M deposits the symbol in its data component and picks up the symbol that was there, going to control state p.
  - $\delta([q, X], Y) = ([p, Y], X, L)$ for all X and Y in Γ.
- In control state p, M deposits its data and enters state s, moving right.
  - $\delta([p, X], Y) = (s, X, R)$ for all X and Y in Γ.

Example: Multiple Tracks

A common use for multiple tracks is to use one track for data the other for a single "mark."

- Symbols of Γ are pairs $[A, X]$, where X is the "real" symbol, and A is either B (blank) or $*$.
  - Input symbol a is identified with $[B, a]$.
  - The blank is $[B, B]$.
- Here's a program to find the $*$, assuming it is somewhere to the left of the present position.
  1. $\delta(q, [B, X]) = (q, [B, X], L)$
  2. $\delta(q, [*, X]) = (p, [B, X], R)$

Other TM Models

While regular or CF languages are classes of languages that we defined by convenient notations (RE's, CFG's, etc.), no one supposed that they represented "everything we can compute."

- The purpose of the TM was to define "everything we can compute."

  ❖ For convenience, we use recognition of languages as the space of possibly computable things; other spaces, e.g., computing arithmetic functions, yield the same conclusions.

- Thus, it would be awkward if we could find another notion of "everything" that was different from the TM.

  ❖ A real computer is an important special case. Do real computers and TM's define the same set of computable things?

- Our next steps are to consider potentially more powerful notions of computing and see that the TM model we defined can simulate them.

  ❖ These models are: multitape TM, nondeterministic TM, multistack machines, counter machines, "real" computers.

Multitape TM's

Allow the TM to have some finite number of tapes k, with a head for each tape.

- Move is a function of the state and the symbol scanned by each tape head.
- Action = new state, new symbol for each tape, and a head motion (L, R, or S, for "stationary").
- First tape holds the input, other tapes are initially blank.

Many Tapes to One Tape Simulation

To simulate k tapes, use one tape with $2k$ tracks.

- One track holds the contents of each tape.
- Another track holds a mark representing the head position of that tape, as

	$*$		
W	X	Y	Z

- To simulate one move of the multitape TM, the one-tape TM must remember how many *'s are to its left.

1. Move left, then right, visiting all the *'s to see what each tape head is scanning.

2. Decide on the multitape TM's move, based on the scanned symbols and its state (remembered in the state of the one-tape TM).

3. Visit each * again, making the necessary adjustments: change symbols and move *'s one cell left or right, as needed.

- Important observation for when we study polynomial time TM's: If the multitape TM makes $T(n)$ moves when the input is of length n, then the one-tape TM makes $O\big(T^2(n)\big)$ moves.

  - Thus, if the multitape TM takes polynomial time, so does the one-tape TM.

  - Key point in proof: The *'s can't get more than $T(n)$ cells apart, so one move is simulated in $O\big(T(n)\big)$ moves of the one-tape TM.

Nondeterministic TM

Let the TM have a finite set of choices of move.

- As with the (nondeterministic) PDA, there is no "mix-and-match"; if (p, X, L) and (q, Y, R) are choices, we cannot go to state q, print X on the cell and move right, e.g., unless (q, X, R) is another choice.

Nondeterministic to Deterministic Simulation

Let the NTM have one tape, but first simulate with a multitape DTM; later convert the multitape DTM to a one-tape DTM.

- Use one tape of DTM to hold a *queue* of ID's of the NTM, separated by special markers (*).

- When an ID reaches the front of the queue, find all its next ID's, and add them to the back of the queue.

- Accept if you ever reach an ID with an accepting state.

  - Note that the queue discipline is important, so that the DTM eventually reaches every ID that the NTM can enter.

  - In contrast, if we used a stack discipline, the NTM might reach acceptance, but the

> DTM would go off on some infinite chase of ID's and never reach the accepting ID of the NTM.

MultiStack Machines

Like PDA, but with more than one stack.

- One stack is not enough to simulate a TM; you get only CFL's.
  - ❖ While we haven't emphasized the point, all the examples in Section 7.2.3 of non-CFL's are recognized by TM's.
- But 2 stacks is enough!
- Key idea: use one stack to hold what is to the left of the tape head, use the other to hold what is to the right.

Counter Machines

Two equivalent ways to think of a counter:

1. A stack with a bottom-marker, say Z_0, and one other symbol, say X, that can be placed on the stack.
   - ❖ Thus, stack always looks like $XX \cdots XZ_0$.
2. A device that holds a nonnegative integer, with the operations add 1, subtract 1, and test-if-0.

- 1 counter = subset of CFL's, including all regular languages and some nonregular languages like $\{0^n 1^n \mid n \geq 1\}$.
- 2 counters = TM!
  - ❖ Proof in two stages: 3 counters simulate 2 stacks, then 2 counters simulate 3 counters.

2 Stacks to 3 Counters

Suppose a stack has $r - 1$ symbols. Think of the stack contents as a base-r number, with the symbols as digits 1 through $r - 1$.

- Use one counter for each stack, plus one "scratch" counter.
- Multiply and divide by r using two counters.
  - ❖ Subtract r from one, add 1 to the other, or vice-versa.
- Push X = multiply by r, then add digit represented by X.

- Pop = divide by r, throw away the remainder.
- Read top symbol = move from one counter to the other, counting in the state modulo r to determine the remainder.

3 Counters to 2 Counters

Key idea, represent counters i, j, and k by the integer $2^i 3^j 5^k$.

- Store this number on one counter, use the other counter as scratch.
- Test if $i = 0$ by moving count from one counter to the other, counting modulo 2 in the state.
  - ❖ $i = 0$ if and only if the number is not divisible by 2.
- Tests for $j = 0$ and $k = 0$ analogous.
- Adding to i, j, k are multiplications of the count by 2, 3, 5, respectively.
- Subtractions are similarly divisions.

Real Computers

In one sense, a real computer has a finite number of states, and thus is *weaker* than a TM.

- We have to postulate an infinite supply of tapes, disks, or some periferal storage device to simulate an infinite TM tape.
- Assume human operator to mount disks, keep them stacked neatly on the sides of the computer.

TM to Real Computer

Computer can simulate finite control, and mount one disk that holds the region of the TM tape around the tape head.

- When the tape head moves off this region, the computer prints an order to have its disk moved to the top of the left or right pile, and the top of the other pile mounted.

Real Computer to TM

Simulation is at the level of stored instructions and words of memory.

- TM has one tape that holds all the used memory locations, and their contents.

- Other TM tapes hold the instruction counter, memory address, computer input file, and scratch.
- Instruction cycle of computer simulated by:
  1. Find the word indicated by the instruction counter on the memory tape.
  2. Examine the instruction code (a finite set of options), and get the contents of any memory words mentioned in the instruction, using the scratch tape.
  3. Perform the instruction, changing any words' values as needed, and adding new address-value pairs to the memory tape, if needed.

Comparison of Running Times

- If the computer can do a multiplication of words whose length is not limited (e.g., to 64 bits, as on most computers), then the length of the longest value can double at each step, and it takes $O(2^{T(n)})$ steps of the TM to simulate $T(n)$ steps of the computer.
- However, if we limit the length of words to, say, 64, or we allow arbitrarily long words but only instructions that add at most 1 to the length in one step (e.g., addition), then $O\big(T^3(n)\big)$ steps of the TM suffice to simulate $T(n)$ computer steps.
  - Thus, polynomial-time computer program becomes polynomial-time TM.
  - Why? Memory tape can only grow to $O\big(T^2(n)\big)$. Thus, one step takes $O\big(T^2(n)\big)$ on the TM, and $T(n)$ steps take $O\big(T^3(n)\big)$.