More About Turing Machines

“Programming Tricks”
Restrictions
Extensions
Closure Properties
Overview

◆ At first, the TM doesn’t look very powerful.
  ▶ Can it really do anything a computer can?
◆ We’ll discuss “programming tricks” to convince you that it can simulate a real computer.
Programming Trick: Multiple Tracks

- Think of tape symbols as vectors with k components.
- Makes the tape appear to have k tracks.
- Let input symbols be blank in all but one track.
- Thus, a multi-track TM is the same as a single-track (single-tape) TM.
Picture of Multiple Tracks

Represents one symbol [X,Y,Z]

Represents input symbol 0

Represents the blank

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Y</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>Z</td>
<td>B</td>
</tr>
</tbody>
</table>

Represents one symbol [X,Y,Z]
Multi-track Programming Trick 1: Marking

A common use for an extra track is to mark certain positions.

Almost all cells hold B (blank) in this track, but several hold special symbols (marks) that allow the TM to find particular places on the tape.
Multi-track Programming Trick 2: Semi-infinite Tape

◆ Consider a version of a TM that never moves left from the initial position.
◆ Let this position be 0; positions to the right are 1, 2, ... and positions to the left are –1, –2, ...
◆ New TM has two tracks.
  ◦ Top holds positions 0, 1, 2, ...
  ◦ Bottom holds a marker, positions –1, –2, ...
Simulating Infinite Tape by Semi-infinite Tape

State remembers whether simulating upper or lower track. Reverse directions for lower track.

Put * here at the first move. You don’t need to do anything, because these are initially B.
Multitape Turing Machines

- Allow a TM to have $k$ tapes (each with its own head) for any fixed $k$.
- Move of the TM depends on the state and the symbols under the head for each tape.
- In one move, the TM can change state, write symbols under each head, and move each head independently.
Simulating k Tapes by One

- Use 2k tracks.
- Each tape of the k-tape machine is represented by a track.
- The head position for each track is represented by a mark on an additional track.
Picture of Multitape Simulation

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>head for tape 1</td>
</tr>
<tr>
<td>. . . A B C A C B . . .</td>
<td>tape 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>head for tape 2</td>
</tr>
<tr>
<td>. . . U V U U U W V . . .</td>
<td>tape 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Nondeterministic TM’s

◆ Allow the TM to have a choice of move at each step.
  ▶ Each choice is a state-symbol-direction triple, as for the deterministic TM.

◆ The TM accepts its input if any sequence of choices leads to an accepting state.
Simulating a NTM by a DTM

- The DTM maintains on its tape a queue of ID’s of the NTM.
- A second track is used to mark certain positions:
  1. A mark for the ID at the head of the queue.
  2. A mark to help copy the ID at the head and make a one-move change.
Picture of the DTM Tape

Where you are copying ID<sub>k</sub> with a move

Front of queue

ID<sub>0</sub> # ID<sub>1</sub> # ... # ID<sub>k</sub> # ID<sub>k+1</sub> ... # ID<sub>n</sub> # New ID

Rear of queue
Operation of the Simulating DTM

◆ The DTM finds the ID at the current front of the queue.
◆ It looks for the state in that ID so it can determine the moves permitted from that ID.
◆ If there are m possible moves, it creates m new ID’s, one for each move, at the rear of the queue.
Operation of the DTM – (2)

◆ The m new ID’s are created one at a time.

◆ After all are created, the marker for the front of the queue is moved one ID toward the rear of the queue.

◆ However, if a created ID has an accepting state, the DTM instead accepts and halts.
Why the NTM -> DTM Construction Works

- There is an upper bound, say k, on the number of choices of move of the NTM for any state/symbol combination.

- Thus, any ID reachable from the initial ID by n moves of the NTM will be constructed by the DTM after constructing at most \((k^{n+1} - k)/(k-1)\)ID’s.

\[\text{Sum of } k + k^2 + \ldots + k^n\]
Why? – (2)

◆ If the NTM accepts, it does so in some sequence of $n$ choices of move.
◆ Thus the ID with an accepting state will be constructed by the DTM in some large number of its own moves.
◆ If the NTM does not accept, there is no way for the DTM to accept.
Taking Advantage of Extensions

◆ When we discuss construction of particular TM’s that take other TM’s as input, we can assume the input TM is as simple as possible.
  ▶ E.g., one, semi-infinite tape, deterministic.
◆ But the simulating TM can have many tapes, be nondeterministic, etc.
Real Computers

◆ A real computer is often modeled as a Random Access Machine (RAM), where memory is an address space indexed with integer addresses.

◆ Imagine a computer with a store for an unbounded number of key-value pairs.
  ▶ Generalizes an address space.
Simulating a Name-Value Store by a TM

- The TM uses one of several tapes to hold an arbitrarily large sequence of key-value pairs in the format `#key*value#`...
- Mark, using a second track, the left end of the sequence.
- A second tape can hold a key whose value we want to look up.
Lookup

◆ Starting at the left end of the store, compare the lookup key with each key in the store.
◆ When we find a match, take what follows between the * and the next # as the value.
Suppose we want to insert key-value pair \((k, v)\), or replace the current value associated with key \(k\) by \(v\).

Perform lookup for key \(k\).

If not found, add \(k*v#\) at the end of the store.
Insertion – (2)

◆ If we find \( k^*v' \), we need to replace \( v' \) by \( v \).
◆ If \( v \) is shorter than \( v' \), you can leave blanks to fill out the replacement.
◆ But if \( v \) is longer than \( v' \), you need to make room.
Insertion – (3)

- Use a third tape to copy everything from the first tape at or to the right of v'.
- Mark the position of the * to the left of v' before you do.
- Copy from the third tape to the first, leaving enough room for v.
- Write v where v' was.
Closure Properties of Recursive and RE Languages

- Both closed under union, concatenation, star, reversal, intersection, inverse homomorphism.
- Recursive closed under difference, complementation.
Union

- Let $L_1 = L(M_1)$ and $L_2 = L(M_2)$.
- Assume $M_1$ and $M_2$ are single-semi-infinite-tape TM’s.
- Construct 2-tape TM $M$ to copy its input onto the second tape and simulate the two TM’s $M_1$ and $M_2$ each on one of the two tapes, “in parallel.”
Union – (2)

- Recursive languages: If $M_1$ and $M_2$ are both algorithms, then $M$ will always halt in both simulations.
- Accept if either accepts.
- RE languages: accept if either accepts, but you may find both TM’s run forever without halting or accepting.
Picture of Union/Recursive

Remember: = “halt without accepting"
Picture of Union/RE

Input $w$

$M_1 \xrightarrow{\text{Accept}} M \xrightarrow{\text{OR}} M_2 \xrightarrow{\text{Accept}}$
Intersection/Recursive – Same Idea

Input $w$

$M_1$ → Accept
    → Reject

$M_2$ → Accept
    → Reject

$M$ → AND → Accept
    → OR → Reject
Intersection/RE

Input $w$

$M_1$ -> Accept

$M_2$ -> Accept

$M$ -> Accept

AND -> Accept
Difference, Complement

◆ **Recursive languages**: both TM’s will eventually halt.

◆ **Accept if** $M_1$ **accepts and** $M_2$ **does not.**
  
  ▶ **Corollary**: Recursive languages are closed under complementation.

◆ **RE Languages**: can’t do it; $M_2$ may never halt, so you can’t be sure input is in the difference.
Let $L_1 = L(M_1)$ and $L_2 = L(M_2)$.

Assume $M_1$ and $M_2$ are single-semi-infinite-tape TM’s.

Construct 2-tape Nondeterministic TM $M$:

1. Guess a break in input $w = xy$.
2. Move $y$ to second tape.
3. Simulate $M_1$ on $x$, $M_2$ on $y$.
4. Accept if both accept.
Let’s not use a NTM.
Systematically try each break \( w = xy \).
\( M_1 \) and \( M_2 \) will eventually halt for each break.
Accept if both accept for any one break.
Reject if all breaks tried and none lead to acceptance.
Star

Given $w=x_1x_2x_3...x_n$ and Turing Machine $M$

For $k=1$ to $n$

For each way to write $w=w_1w_2...w_k$

- If all the $w_i$ are in $L(M)$, accept.

Else, reject.
Reversal

- Start by reversing the input.
- Then simulate TM for L to accept w if and only if $w^R$ is in L.
- Works for either Recursive or RE languages.