Variants of Turing Machines

- Many alternative definitions of TMs exist
- The original model and its reasonable variants have the same power
  - Recognize the same languages
- E.g., a TM that can move left, right, or “stay put”
  - No new power though because could always simulate the “stay put” move with two moves, one to the left and one to the right, in the original TM
  - This example is key to showing equivalence of TM variants

Previously:
Introduce Turing machines

Today:
Assignment 5 back
TM variants, relation to algorithms, history

Later
Exam 2 due Friday, 5pm
Assignment 6 out

A lot of models for computation turn out to be equivalent (especially variants of Turing machines). To show they are equivalent, give a way to simulate one model with the other.
Multi-tape TMs

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)
- First tape holds the input, other tapes are initially blank

Every multi-tape TM has an equivalent single-tape TM

Proof: show how to convert multi-tape TM $M$ to an equivalent single tape TM $S$

- If $M$ has $k$ tapes, $S$ simulates the effect of $k$ tapes by storing the same information on its single tape
- Uses a new symbol # as a delimiter to separate the contents of the different tapes
- $S$ must also keep track of the locations of the heads on each tape
  - Writes a tape symbol with a dot above it to mark where the head on that tape would be
  - Dotted symbols are simply new symbols added to the tape alphabet

Definition of $S$

On input $w = w_1 \ldots w_n$:

1. First $S$ puts its tape into the format that represents all $k$ tapes of $M$. The formatted tape contains
   \[
   \# w_1 \# w_2 \# \ldots \# w_n \# 0 1 0 1 0 \ldots
   \]

2. To simulate a single move, $S$ scans its tape from the first # (marks the left end) to the $(k+1)$th # (marks the right end) to determine the symbols under the virtual heads. Then $S$ makes a second pass to update the tapes according to the way that $M$'s transition function dictates.

3. If at any point $S$ moves one of the virtual heads to the right onto a #, this action signifies that $M$ has moved the corresponding head onto the previously unread blank portion of that tape. So $S$ writes a blank symbol on this tape cell and shifts the tape contents, from this cell to the rightmost #, one unit to the right. Then it continues the simulation as before.
**Nondeterministic TM**

- Let the TM have a **finite set of choices of move**
- Components of NDTM same as TM
- Transition function may indicate that more than one move is possible

**NDTM**

- **Arbitrarily chooses move** when more than one possibility exists
- **Accepts if there is at least one computation that terminates in an accepting state**
  - Existence of other computations that halt in non-accepting states or fail to halt altogether is irrelevant

**NDTM Computation**

Evolution of the NTM represented by a tree of configurations

If there is (at least) one accepting leaf, then the TM accepts

**NTM**

$L_{NDTM} = \{wcz \mid c \text{ is immediately preceded or followed by the string } ab\}$

Processes input until a c is encountered; then, can stay in state $q_1$, enter state $q_2$ to determine if the c is followed by ab, or enter $q_3$ to determine if the c is preceded by ab
Simulating Nondeterministic TMs with Deterministic Ones

We want to search every path down the tree for accepting configurations.

**Bad idea:** depth-first
- This approach can get lost in never-halting paths.

**Good idea:** breadth-first
- For time step 1, 2, ..., we list all possible configurations of the non-deterministic TM.
- The simulating TM accepts when it lists an accepting configuration.

### Breadth-First

Let $b$ be the maximum number of children for a node.

Any node in the tree can be uniquely identified by a string $\in \{1, \ldots, b\}^*$.

Example: location of the rejecting configuration is $(3, 1)$.

With the lexicographical listing $\varepsilon$, $(1)$, $(2)$, ..., $(b)$, $(1, 1)$, $(1, 2)$, ..., $(1, b)$, $(2, 1)$, ... etc., we cover all nodes.

---

**Every nondeterministic TM has an equivalent deterministic TM**

*Proof:*

Let $M$ be the nondeterministic TM on input $w$.
Let $M'$ be the deterministic TM.

The simulating TM $M'$ uses three tapes:
1. $T_1$ contains the input $w$.
2. $T_2$ simulates the tape content of $M$ on $w$ at a given node.
3. $T_3$ keeps track of the current location in the nondeterministic computation tree.

Steps in the simulation:
1. Empty string (symbolizing address of the root) is written on $T_3$.
2. Input string on $T_1$ is copied to $T_2$.
3. The computation of $M$ defined by the sequence on $T_3$ is simulated on $T_2$.
4. If no more symbols remain on $T_3$, a nondeterministic choice is invalid, or a rejecting configuration is found, abort this branch and go to step 6.
5. If an accepting configuration is found, the computation of $M'$ halts and accepts the input.
6. If the computation did not halt in step 4, the next sequence is generated on $T_3$ and the computation continues at step 3.

---

**Robustness**

Just like $k$-tape TMs, nondeterministic Turing machines are not more powerful than simple TMs:

Every nondeterministic TM has an equivalent 3-tape Turing machine, which – in turn – has an equivalent 1-tape Turing machine.

Hence

A language $L$ is recursively enumerable (Turing recognizable) if and only if some nondeterministic TM recognizes it.

A language $L$ is recursive (decidable) if and only if some nondeterministic TM decides it.
Turing’s World example

Turing’s World example

Turing’s World example