Turing Machines, part III

March 27, 2020
Agenda

- Warm up
- Mid Term 2
- HW6
- Lecture Topic: Variants of TM
Post-Spring Break Schedule at a glance

Week 3/23 – 3/27
- Mon 3/23
- Wed 3/25
- Fri 3/27: HW6 out

Week 3/30 – 4/3
- Mon 3/30
- Wed 4/1
- Fri 4/3: HW6 in

Week 4/6 – 4/10
- Mon 4/6
- Wed 4/8
- Fri 4/10: Mid Term 2 (take-home)

Week 4/13 – 4/17
- Mon 4/13
- Wed 4/15
- Fri 4/17: HW7 out

Week 4/20 – 4/24
- Mon 4/20
- Wed 4/22
- Fri 4/24: HW 7 in, HW8 out

Week 4/27 – 5/1
- Mon 4/27
- Wed 4/29
- Fri 5/1: HW 8 in, Wrap-up

Week 5/4 – 5/8
- Thurs 5/7: Final due (take-home)
HW6

Has six problems:

- Problem 1: CFG to PDA conversion
- Problem 2: CFG to Chomsky Normal Form
- Problem 3: Show that a language is not context free
- Problem 4: Use Table Filling Algorithm to minimize states in a DFA (lecture of 3/13)
- Problem 5: TM, give informal description
- Problem 6: TM, give state diagram

Due by: **Fri 4/3, 11:59pm**, on OSBLE
Variants of Turing Machine

- The Turing Machine model the way we defined it thus far is robust
  - Finite Automata and PDA are (to a degree) robust too, but TMs are supremely so
- Quick example of TM robustness
  - Variation 0: tape head allowed to “stay”, not just move Left/Right
  - Result: TM with only \{L,R\} can easily simulate \{L,R,S\} by doing L+R for an S
- We will see that several other more involved \textit{variants} can also be nicely simulated
Variation I: Multitape TM

- Theorem I: *Every multi-tape TM has an equivalent single-tape TM*

- Proof:
  - Show how to convert a multi-tape TM $M$ to an equivalent single-tape TM $S$
Variation I: Multitape TM

Simulation Idea:

- Say \( M \) has \( k \) tapes
- \( S \) simulates the effect of \( k \) tapes by storing their information on its single tape
- **Delineate tapes**: \( S \) uses the new symbol \# as a delimiter to separate contents of the different tapes
- **Track tape heads**: \( S \) writes a dot above a tape symbol to mark the place where the head on that tape would be
Variation I: Multitape TM
Complete/formal simulation

\[ S = \text{“On input } w = w_1 \cdots w_n:\]

1. First \( S \) puts its tape into the format that represents all \( k \) tapes of \( M \). The formatted tape contains

\[ \# \cdot w_1 \cdot w_2 \cdots \cdot w_n \cdot \# \cdot \# \cdot \# \cdots \# . \]

2. To simulate a single move, \( S \) scans its tape from the first \( \# \), which marks the left-hand end, to the \((k + 1)\)st \( \# \), which marks the right-hand end, in order 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 until the rightmost \( \# \), one unit to the right. Then it continues the simulation as before.”
Variation II: Nondeterministic TMs

- **Nondeterminism**: at any point in a computation, the machine may proceed according to several possibilities.
- IOGW, the transition function of a nondeterministic TM has the form

\[ \delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}). \]

- The computation of a nondeterministic TM can be viewed as a tree whose branches correspond to different possibilities of the machine.
- If some branch leads to the **accept** state, the machine accepts the input.
Variation II: Nondeterministic TMs

- **Theorem II:** *Every nondeterministic TM has an equivalent deterministic TM*

- **Proof Idea**
  - Simulate a NDTM $N$ with a DTM $D$
  - The idea behind the simulation is to have $D$ try all possible branches of $N$’s nondeterministic computation.
  - If $D$ ever finds the accept state on one of the branches, $D$ accepts. Otherwise $D$’s simulation will not terminate.
  - Use **Breadth First Search** to explore the tree.
Variation II: Nondeterministic TMs: Details of the proof

- The simulating DTM D has three tapes (this is equivalent to having a single tape, by Theorem I)
  - Tape 1: contains the input string, never altered
  - Tape 2: maintains a copy of N’s tape on some branch of its ND computation
  - Tape 3: keeps track of D’s location in N’s ND computation tree

\[\text{Diagram:} \quad \begin{array}{c}
D \\
0 \ 0 \ 1 \ 0 \ # \ 0 \ 1 \ x \ \square \ \ldots \ \text{input tape} \\
x \ x \ # \ 0 \ 1 \ x \ \square \ \ldots \ \text{simulation tape} \\
1 \ 2 \ 3 \ 3 \ 2 \ 3 \ 1 \ 2 \ 1 \ 1 \ 3 \ \square \ \ldots \ \text{address tape}
\end{array}\]
Variation II: Nondeterministic TMs: Details of the proof

- Let us first consider the data representation on tape 3.
- Every node in the tree can have at most \( b \) children, where \( b \) is the size of the largest set of possible choices by \( N \)'s transition function.
- To every node in the tree we assign an address that is a string over the alphabet \( T_b = \{1, 2, \ldots, b\} \).
- E.g. we assign the address 231 to the node we arrive at by starting at the root, going to its 2\(^{nd}\) child, going to that node's 3\(^{rd}\) child, and finally going to that node's 1\(^{st}\) child.

![Diagram](image)
Variation II: Nondeterministic TMs: Details of the proof

- Each symbol in the string tells us which choice to make next when simulating a step in one branch in N’s computation.
- Sometimes a symbol may not correspond to any choice if too few choices are available for a configuration. In that case, the address is invalid and doesn’t correspond to any node.
- Tape 3 contains a string over $T_b$. It represents the branch of N’s computation from the root to the node addressed by that string unless the address is invalid.
- The empty string is the address of the root of the tree.
Variation II: Nondeterministic TMs:
Full description of D

1. Initially, tape 1 contains the input $w$, and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2 and initialize the string on tape 3 to be $\varepsilon$.
3. Use tape 2 to simulate $N$ with input $w$ on one branch of its nondeterministic computation. Before each step of $N$, consult the next symbol on tape 3 to determine which choice to make among those allowed by $N$’s transition function. If no more symbols remain on tape 3 or if this nondeterministic choice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, accept the input.
4. Replace the string on tape 3 with the next string in the string ordering. Simulate the next branch of $N$’s computation by going to stage 2.
Variation III: Enumerators

- Loosely defined, an enumerator is a TM with an attached printer.
- The TM can use that printer as an output device to print strings.
- Every time the TM wants to add a string to the list, it sends the string to the printer.

![Diagram of control and printer with strings on the work tape.](image-url)
Variation III: Enumerators

- An enumerator $E$ starts with a blank input on its work tape.
- If the enumerator doesn’t halt, it may print an infinite list of strings.
- The language enumerated by $E$ is the collection of all the strings that it eventually prints out.
- Moreover, $E$ may generate the strings of the language in any order, possibly with repetitions.
Variation III: Enumerators

Theorem III: A language is Turing-recognizable if and only if some enumerator enumerates it.

Proof:

First direction: we show that if we have an enumerator $E$ that enumerates a language $A$, a TM recognizes $A$.

The TM works in the following way.

$M = “On input w:
1. Run $E$. Every time that $E$ outputs a string, compare it with $w$.
2. If $w$ ever appears in the output of $E$, accept.”

Clearly, $M$ accepts those strings that appear on $E$’s list.
Variation III: Enumerators

Theorem III: A language is Turing-recognizable if and only if some enumerator enumerates it.

Proof

The other direction: If TM $M$ recognizes language $A$, we can construct the following enumerator for $A$. Say that $s_1, s_2, s_3, \ldots$ is a list of all possible strings in $\Sigma^*$.

- $E =$ “Ignore the input,
  1. Repeat the following for $i = 1, 2, 3$
  2. Run $M$ for $i$ steps on each input, $s_1, s_2, \ldots, s_i$
  3. If any computations accept, print out the corresponding $s_j$.”

If $M$ accepts a particular string $s$, eventually it will appear on the list generated by $E$. In fact, it will appear on the list infinitely many times because $M$ runs from the beginning on each string for each repetition of step 1. This procedure gives the effect of running $M$ in parallel on all possible input strings.
Equivalence with other models

- We saw several variants of the TM model and their equivalence
- Many other models of general purpose computation have been proposed
- Some are very much like TMs, but others are quite different
- All share the essential feature of TMs – namely, unrestricted access to unlimited memory – distinguishing them from weaker models such as FA
- Remarkably, all models with that feature turn out to be equivalent in power, so long as they satisfy reasonable requirements (e.g. the ability to perform only a finite amount of work in a single step)
- This phenomenon is analogous to “equivalence of programming languages”
- This analogy has profound implication – definition of algorithm -- the subject of our next lecture!