EECS 2001N : Introduction to the Theory of Computation

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/2001N`
Also on Moodle

Turing Machines - Simulating FA

How can we show that TM's can simulate DFA's?

- Custom designed TM
- Generic TM

Turing Machines - Simulating A Specific DFA

- Intuitively, the states of the TM can be the same as those of the FA
- However, since the TM has a tape containing the input, we have to make sure that the head moves to the right pointing to the next input character at each step, updating states appropriately
- The TM also has to sense end of the input (could be a blank, or a $) and depending on the state of the DFA, move to q_{accept} or q_{reject}

Turing Machines - Simulating Any DFA

Input: Description of DFA B and an input w, i.e.,
$B = (Q, \Sigma, \delta, q_0, F)$ and $w \in \Sigma^*$.
The TM performs the following steps:

- Check if B and w are valid, if not: "reject"
- Copy B to a tape, w to another
- Simulate B on w. The head on the tape containing B points to $q \in Q$, the state of the DFA, and the head on the tape containing w points to i, $i = 0, 1, .., |w|$, the position on the input.
- While we increase i from 0 to $|w|$, we update q according to the input letter $_i$ and the transition function value $\delta(q, w_i)$
- If B accepts w: "accept"; otherwise "reject"

Turing Machines - Simulating Other Machines

- The previous proof was important for another reason, and we will return to it
- We can ask: what else can a TM simulate?
- Very surprising answer: **any** TM
- We will show that a TM can simulate a given TM on a given input!
- Is it weird for a TM to be an input to another TM?
  No. A Java program to count the number of lines or characters in a file can take a Java program as input.

Universal Turing Machines

- The input is a TM description and an input
- Can we follow the same strategy as we did for simulating any FA?
  - Yes!
  - Tape 1 has the machine description, tape 2 has the contents of the tape of the input machine and tape 3 has the state of the input machine
  - In a loop, until tape 3 has a halting state:
    Scan tape 1 to find the correct transition, and update tapes 2 and 3

Universal Turing Machines - Implications

- This is the equivalent of writing “programs” to run on a general purpose computing model
- We can “construct” one TM, and every other TM can “run” on it
- From this point of view any TM is an “algorithm” that is “implemented” on a universal TM
- Recall Church-Turing Thesis: The intuitive notion of computing and algorithms is captured by the Turing machine model

Turing Machines - Implications on Mathematics

- In 1900, David Hilbert (1862–1943) proposed his Mathematical Problems (23 of them)
- Hilbert's 10th problem: Determination of the solvability of a Diophantine equation
  Given a Diophantine equation with any number of unknown quantities and with integer coefficients: To devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in integers
- Let $P(x_1, \ldots, x_k)$ be a polynomial in k variables with integral coefficients. Does P have an integral root $(x_1, \ldots, x_k) \in \mathbb{Z}^k$?

Turing Machines - Implications on Mathematics

- Examples:
  $P(x, y, z) = 6x^3yz + 3xy^2 - x^3 - 10$ has integral root
  $(x, y, z) = (5, 3, 0)$
  $P(x, y) = 21x^2 - 81xy + 1$ does not have an integral root
- Hilbert's "... a process according to which it can be determined by a finite number of operations ..." needed to be defined in a proper way
- Matijasevic proved that Hilbert's 10th problem is unsolvable in 1970

Decidability

- We are now ready to tackle the question: **What can computers do and what can they not?**
- We do this by considering the question: *Which languages are TM-decidable, TM-recognizable, or neither?*
- Assuming the Church-Turing thesis, these are fundamental properties of the languages (problems)

Describing TM's

Three Levels of Describing algorithms:

- formal (state diagrams, CFGs, etc)
- implementation (pseudo-code)
- high-level (coherent and clear English)

Describing input/output format: TM's allow only strings in Σ^* as input/output. If our inputs X and Y are of another form (graph, Turing machine, polynomial), then we use $\langle X, Y \rangle$ to denote "some kind of encoding in Σ^*"

Examples of Decidable Problems

- First we look at several decidable problems
- Then we develop the tools to prove that some problems are provably not decidable

Decidability of Regular Languages - DFA

- We showed earlier that a TM can simulate a DFA
- Another way to look at this is:
  The acceptance problem for DFA is

  $$A_{DFA} = \{\langle B, w\rangle | B \text{ is a DFA that accepts } w\}$$

  A_{DFA} is a TM-decidable language
- Note that this language deals with all possible DFAs and inputs w, not a specific instance

Decidability of Regular Languages - NFA

The acceptance problem for NFA is

$$A_{NFA} = \{\langle B, w\rangle | B \text{ is a NFA that accepts } w\}$$

A_{NFA} is a TM-decidable language

- Use our earlier results on finite automata to transform the NFA B into an equivalent DFA C. We saw an algorithm to do this, and that algorithm can be implemented on a TM
- Use the TM C of the previous slide on $\langle C, w\rangle$
- This can all be done with one big, combined TM

Note: Similar reasoning can be done for regular expressions

Emptiness-testing of Regular Languages

Another problem relating to DFAs is the emptiness problem:

$$E_{DFA} = \{\langle A\rangle | A \text{ is a DFA with } L(A) = \emptyset\}$$

- How can we decide this language? This language concerns the behavior of the DFA A on **all possible** strings
- Idea: check if an accept state of A is reachable from the start state of A

Emptiness-testing of Regular Languages - 2

Algorithm for E_{DFA} on input $A = (Q, \Sigma, \delta, q_0, F)$:

- If A is not a proper DFA: "reject"
- Mark the start state of A, q_0
- Repeat until no new states are marked:
  Mark any states that can be δ-reached from any state that is already marked
- If no accept state is marked, "accept";
  else "reject"

Equivalence-testing of DFA

$$EQ_{DFA} = \{\langle A, B\rangle | A, B \text{ are DFA with } L(A) = L(B)\}$$

- Idea: Look at the symmetric difference between the two languages $(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$
- This expression uses standard DFA transformations: union, intersection, complement

Equivalence-testing of DFA - 2

Algorithm for EQ_{DFA} on input $\langle A, B \rangle$:

- If A or B are not proper DFA: "reject"
- Construct a third DFA C that accepts the language $(L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$ (using standard transformations)
- Decide with the Emptiness-testing TM o to check whether or not $C \in E_{DFA}$
  If $C \in E_{DFA}$ then "accept"
  If $C \notin E_{DFA}$ then "reject"