EECS 2001A : Introduction to the Theory of Computation

Suprakash Datta

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

Turing Machines - Decidability

- A language $L = L(M)$ is **decided** by the TM M if on *every* input w, the TM finishes in a halting configuration.
  That is: q_{accept} for $w \in L$ and q_{reject} for all $w \notin L$.
- A language L is Turing-decidable if and only if there is a TM M that decides L
- Also called: a *recursive* language

Turing Machines - Recognizability

- A language $L = L(M)$ is **recognized** by the TM M if on *every* input $w \in L$, the TM finishes in the halting configuration q_{accept}
- On an input $w \notin L$, the machine M can halt in the rejecting state q_{reject}, or it can 'loop' indefinitely
- A language L is Turing-recognizable if and only if there is a TM M such that $L = L(M)$
  Recall: The language that consists of all inputs that are **accepted** by a TM M is denoted by $L(M)$
- Also called: a *recursively enumerable* language

Turing Machines - Variants

- Multiple tapes
- 2-way infinite tapes
- Non-deterministic TM's

Multi-tape Turing Machines (Ch 3.2)

Theorem 3.13: Let $k \geq 1$ be an integer. Any k-tape Turing machine can be converted to an equivalent one-tape Turing machine.

- Proving and understanding these kinds of robustness results is essential for appreciating the power of the Turing Machine model
- From this theorem it follows that:
  A language L is TM-recognizable if and only if some multi-tape TM recognizes L.

Proof of Theorem 3.13

- Take a 2-tape TM M and construct an equivalent one-tape TM N
  "N can **simulate** M"
- Tape alphabet of N: $\Gamma \cup \{\dot{x} | x \in \Gamma\} \cup \{\#\}$
- Idea: the contents of the two tapes will be maintained on one tape separated by $\#$ and the dotted version of a character will be used to indicate the location of the head

Proof of Theorem 3.13 - contd.

N simulates the computation of M in each step

- At the start of the step, the tape head of N is on the leftmost symbol $\#$
- N "remembers" the state of M in its state
- In each step, N moves right until it has read both dotted symbols
- The second and then the first dotted symbol is changed as M would change them
- In either case above the contents of the tape may have to be shifted
- Finally, N remembers the new state of M and moves to the leftmost symbol $\#$

2-way Infinite Tape Turing Machines

- For every 2-way infinite tape TM M, there is a 2-tape TM M' such that $L(M) = L(M)$
- Suppose the cells are numbered 0,1,2,.... and -1,-2,....
- Idea: Store the contents of cell 0 and everything to its right on the first tape of M' and everything to the left of cell 0 on the second tape, and simulate the computation of M as usual

Non-deterministic Turing Machines

A Non-deterministic one-tape Turing Machine M is defined by a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$:

- finite set of states Q
- finite input alphabet Σ
- finite **tape** alphabet Γ
- start state $q_0 \in Q$
- accept state $q_{accept} \in Q$
- reject state $q_{reject} \in Q$
- transition function $\boxed{\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})}$

Non-deterministic Turing Machines - 2

- Just like multi-tape TM's, nondeterministic TM's are not more powerful than simple TMs
- Every nondeterministic TM has an equivalent 3-tape TM, which in turn has an equivalent 1-tape TM (stated at Theorem 3.16 in the text)
- Hence: "A language L is recognizable if and only if some nondeterministic TM recognizes it."
- The Turing machine model is extremely robust!

Non-deterministic Turing Machines - 3

- A non-deterministic TM's computation may be thought of as a tree of configurations rather than a path
- If there is (at least) one accepting leaf in this tree, then the TM accepts
- We have to traverse this tree using a deterministic TM
- Bad idea: "depth first" exploration. The TM may explore never-halting paths
- Good idea: "breadth first" exploration. For time steps 1,2,..., we list all possible configurations of the non-deterministic TM. The simulating TM accepts when it reaches an accepting configuration

Non-deterministic Turing Machines - 4

- Let M be the non-deterministic TM on input w
- The simulating TM uses three tapes:
  T_1 contains the input w
  T_2 the tape content of M on w at a node
  T_3 describes a node in the tree of M on w
- Inititally, T_1 contains w, T_2 and T_3 are empty
- Simulate M on w via the deterministic path to the node of tape 3.
  If the node accepts, "accept"
- Increase the node value on T_3, go to previous step

The Church Turing Thesis (Ch 3.3)

- The Church-Turing thesis marks the end of a long sequence of developments that concern the notions of "way-of-calculating", "procedure", "solving", "algorithm"
- Statement: The following computation models are equivalent, i.e., any one of them can be converted to any other one:
  1. One-tape Turing machines
  2. k-tape Turing machines, for any $k \geq 1$
  3. Non-deterministic Turing machines
  4. Java programs
  5. C++ programs
  6. Python programs

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 Turing 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 - 2

- 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"

Context Free Language Problems

- $A_{CFG} = \{\langle G, w\rangle | G \text{ is a CFG that generates } w\}$
- $E_{CFG} = \{\langle G\rangle | G \text{ is a CFG with } L(G) = \emptyset\}$
- $EQ_{CFG} = \{\langle G, H\rangle | G, H \text{ are CFGs with } L(G) = L(H)\}$

Acceptance of Context Free Languages

Recall: Chomsky Normal Form

- A CFG $G = (V, \Sigma, R, S)$ is in CNF if every rule is of the form
  $A \rightarrow BC$, or
  $A \rightarrow x$
  with variables $A \in V$ and $B, C \in V \backslash \{S\}$, and $x \in \Sigma$
  For the start variable S we also allow $S \rightarrow \epsilon$

- Chomsky NF grammars are easier to analyze

- The derivation $S \Rightarrow^* w$ requires $2|w| - 1$ steps (apart from $S \rightarrow \epsilon$)

Acceptance of Context Free Languages - 2

The language

$$A_{CFG} = \{\langle G, w\rangle | G \text{ is a CFG that generates } w\}$$

is TM-decidable.
Proof: Perform the following algorithm:

- Check if G and w are proper, if not "reject"
- Rewrite G to G' in Chomsky normal form
- Take care of $w = \epsilon$ case via $S \to \epsilon$ check for G'
- List all G' derivations of length $2|w| - 1$
- Check if w occurs in this list:
  if so "accept"; if not "reject"

Emptiness of Context Free Languages

The language

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

is TM-decidable.
Proof: Perform the following algorithm:

- Check if G is proper, if not "reject"
- Let $G = (V, \Sigma, R, S)$, define set $T = \Sigma$
- Repeat $|V|$ times:
  Check all rules $B \to X_1 \ldots X_k$ in R
  If $B \notin T$ and $X_1 \ldots X_k \in T^k$ then add B to T
- If $S \in T$ then "reject", otherwise "accept"

Equality of Context Free Languages

Is the language

$$EQ_{CFG} = \{\langle G, H\rangle | G, H \text{ are CFG's with } L(G) = L(H)\}$$

TM-decidable?

- For DFA's we could use the emptiness decision procedure to solve the equality problem
- For CFG's this is not possible because CFGs are not closed under complementation or intersection
- We suspect this problem is undecidable, but need machinery to **prove** this

Beyond Decidability: TM Recognizable Languages

- We know that TM-decidable languages are also TM-recognizable
- We will see that there exist problems that are not TM-decidable but are TM-recognizable
- A common example is the Halting Problem (Ch 5)