Computer Language Theory

Chapter 3: The Church-Turing Thesis

Chapter 3.1

Turing Machines

Turing Machines: Context

- Models
  - Finite Automata:
    - Models for devices with little memory
  - Pushdown Automata:
    - Models for devices with unlimited memory that is accessible only in Last-In-First-Out order
  - Turing Machines:
    - Only model thus far that can model general purpose computers

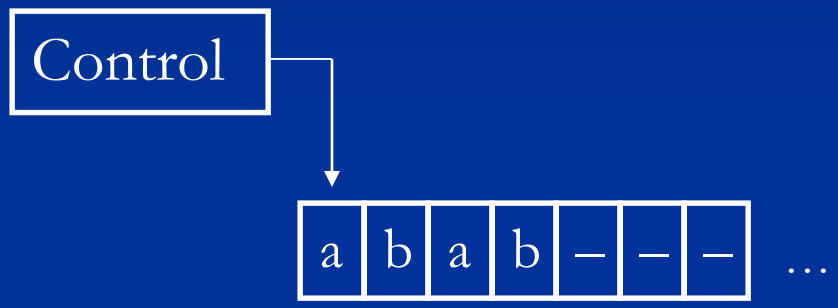
Turing Machines Overview

- Introduced by Alan Turing in 1936
- Unlimited memory
  - Infinite tape that can be moved left/right and read/written
  - Much less restrictive than stack of a PDA
- A Turing Machine can do everything a real computer can do (even though a simple model!)
- But a Turing Machine cannot solve all problems

What is a Turing Machine?

- Informally:
  - Contains an infinite tape
  - Tape initially contains the input string and blanks everywhere else
  - Machine can read and write from tape and move left and right after each action
  - The machine continues until it enters an accept or reject state at which point it immediately stops and outputs accept or reject
    - Note this is very different from FAs and PDAs
  - The machine can loop forever
    - Why can't a FA or PDA loop forever?
      - Answer: it will terminate when input string is fully processed and will only take one "action" for each input symbol

Turing Machine



Designing Turing Machines

- Design a TM to recognize the language:
  $B = \{w\#w \mid w \in \{0,1\}^*\}$
  - Will focus on informal descriptions, as with PDAs
    - But even more so in this case
  - Imagine that you are standing on an infinite tape with symbols on it and want to check to see if the string belongs to B?
    - What procedure would you use given that you can read/write and move the tape in both directions?
    - You have a finite control so cannot remember much and thus must rely on the information on the tape
    - Try it!

Turing Machine Example 1

- M1 to Recognize B = $\{w\#w \mid w \in \{0,1\}^*\}$
- M1 loops and in each iteration it matches symbols on each side of the #
  - It does the leftmost symbol remaining
  - It thus scans forward and backward
  - It crosses off the symbol it is working on. We can assume it replaces it with some special symbol x.
  - When scanning forward, it scans to the "#" then scans to the first symbol not crossed off
  - When scanning backward, it scans past the "#" and then to the first crossed off symbol.
  - If it discovers a mismatch, then reject; else if all symbols crossed off then accept.
- What are the possible outcomes?
  - Accept or Reject. Looping is not possible.
    - Guaranteed to terminate/halt since makes progress each iteration

Sample Execution

- What happens for the string 011000#011000?

  The tape head is at the red symbol

  0 1 1 0 0 0 # 0 1 1 0 0 0 - -

  X 1 1 0 0 0 # 0 1 1 0 0 0 - -

  …

  X 1 1 0 0 0 # X 1 1 0 0 0 - -

  X 1 1 0 0 0 # X 1 1 0 0 0 - -

  X X 1 0 0 0 # X 1 1 0 0 0 - -

  …

  X X X X X X # X X X X X X - -

Formal Definition of a Turing Machine

- The transition function δ is key:
  - $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
    - A machine is in a state q and the head is over the tape at symbol a, then after the move we are in a state r with b replacing the a and the head has moved either left or right

Formal Definition of a Turing Machine

- A Turing Machine is a 7-tuple
  $\{Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\}$, where
- Q is a set of states
- Σ is the input alphabet not containing the blank
- Γ is the tape alphabet, where blank$\in\Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function
- q_0, q_{accept}, and q_{reject} are the start, accept, and reject states
  - Do we need more than one reject or accept state?
  - No: since once enter such a state you terminate

TM Computation

- As a TM computes, changes occur in:
  - the state
  - the content of the current tape location
  - the current head location
- A specification of these three things is a configuration

Turing Recognizable & Decidable Languages

- The set of strings that a Turing Machine M accepts is the language of M, or the language recognized by M, L(M)
- Definitions:
  - A language is Turing-recognizable if some Turing machine recognizes it
    - Called recursively enumerable by some texts
  - A Turing machine that halts on all inputs is a decider. A decider that recognizes a language decides it.
  - A language is Turing-decidable or simply decidable if some Turing machine decides it.
    - Called recursive by some texts
- Notes:
  - Decidable if Turing-recognizable and always halts (decider)
  - Every decidable language is Turing-recognizable
  - It is possible for a TM to halt only on those strings it accepts

Turing Machine Example II

- Design a TM M2 that decides $A = \{0^{2^n} | n \geq 0\}$, the language of all strings of 0s with length 2^n.
- Without designing it, do you think this can be done? Why?
  - Simple answer: we could write a program to do it and therefore we know a TM could do it since we said a TM can do anything a computer can do
- Now, how would you design it?
- Solution:
  - English: divide by 2 each time and see if result is a one
  1. Sweep left to right across the tape, crossing off every other 0.
  2. If in step 1:
     - the tape contains exactly one 0, then accept
     - the tape contains an odd number of 0's, reject immediately
     - Only alternative is even 0's. In this case return head to start and loop back to step 1.

Sample Execution of TM M2

0 0 0 0 - -	Number is 4, which is 2^2
x 0 0 0 - -	
x 0 x 0 - -	Now we have 2, or 2^1
x 0 x 0 - -	
x 0 x 0 - -	
x x x 0 - -	
x x x 0 - -	Now we have 1, or 2^0
x x x 0 - -	Seek back to start
x x x 0 - -	Scan right; one 0, so accept

Turing Machine Example III

- Design TM M3 to decide the language:

  $C = \{a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 1\}$

  - What is this testing about the capability of a TM?
    - That it can do (or at least check) multiplication
    - As we have seen before, we often use unary
  - How would you approach this?
    - Imagine that we were trying 2 x 3 = 6

Turing Machine Example III

- Solution:
  1. First scan the string from left to right to verify that it is of form $a^+b^+c^+$; if it is scan to start of tape* and if not, reject. Easy to do with finite control/FA.
  2. Cross off the first a and scan until the first b occurs. Shuttle between b's and c's crossing off one of each until all b's are gone. If all c's have been crossed off and some b's remain, reject.
  3. Restore** the crossed off b's and repeat step 2 if there are a's remaining. If all a's gone, check if all c's are crossed off; if so, accept; else reject.

* Some subtleties here. See book. Can use special symbol or backup until realize tape is stuck and hasn't actually moved left.

**How restore? Have a special cross-off symbol that incorporates the original symbol– put an X thru the symbol

Transducers

- We keep talking about recognizing a language, not generating a language. This is common in language theory.
- But now that we are talking about computation, this may seem strange and limiting.
  - Computers typically transform input into output
  - For example, we are more likely to have a computer perform multiplication than check that the equation is correct.
  - Turing Machines can also generate/transduce
  - How would you compute c^k given $a^i b^j$ and $i \times j = k$
    - In a similar manner. For every a, you scan through the b's and for each you go to the end of the string and add a c. Thus by zig-zagging a times, you can generate the appropriate number of c's.

Turing Machine Example IV

- Solve the element distinctness problem:

  Given a list of strings over {0, 1} each separated by a #, accept if all strings are different.

  $E = \{\#x1\#x2\# \ldots \# xn \mid \text{each } xi \in \{0,1\}^* \text{ and } xi \neq xj \text{ for each } i \neq j\}$

- How would you do this?

Turing Machine Example IV

- Solution:
  1. Place a mark on top of the left-most symbol. If it was a blank, accept. If it was a # continue; else reject
  2. Scan right to next # and place a mark on it. If no # is encountered, we only had x1 so accept.
  3. By zig-zagging, compare the two string to the right of the two marked #s. If they are equal, reject.
  4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so accept.
  5. Go to step 3

Decidability

- All of these examples have been decidable.
- Showing that a language is Turing recognizable but not decidable is more difficult
  - We cover that in Chapter 4
- How do we know that these examples are decidable?
  - You can tell that each iteration you make progress toward the ultimate goal, so you must reach the goal
  - This would be clear just from examining the "algorithm"
  - Not hard to prove formally. For example, perhaps n symbols to start and if erase a symbol each and every iteration, will be done in n iterations

Chapter 3.2

Variants of Turing Machines

Variants of Turing Machines

- We saw only a few variants of FA and PDA
  - Deterministic and non-deterministic
- There are many variants of Turing Machines
  - Mainly because they are all equivalent, so that makes things more convenient without really changing anything
  - This should not be surprising, since we already stated that a Turing Machine can compute anything that is computable

TM Variant I

- Our current TM model *must* move the tape head left or right after each step. Often it is convenient to have it stay put. Is the variant that has this capability equivalent to our current TM model? Prove it!
  - This one is quite trivial
- Proof: we can convert any TM with the "stay put" feature to one without it by adding two transitions: move "right" then "left"
  - To show that two models are equivalent, we only need to show that one can simulate another
  - Two machines are equivalent if they recognize the same language

Variant II: MultiTape TMs

- A multitape Turing machine is like an ordinary TM with several tapes.
  - Each tape has its own head for reading and writing
  - Initially tape 1 has input string and rest are blank
  - The transition function allows reading, writing, and moving the heads on some or all tapes simultaneously
  - Multitape is convenient (think of extra tapes as scratch paper) but does not add power.

Proof of Equivalence of Variant II

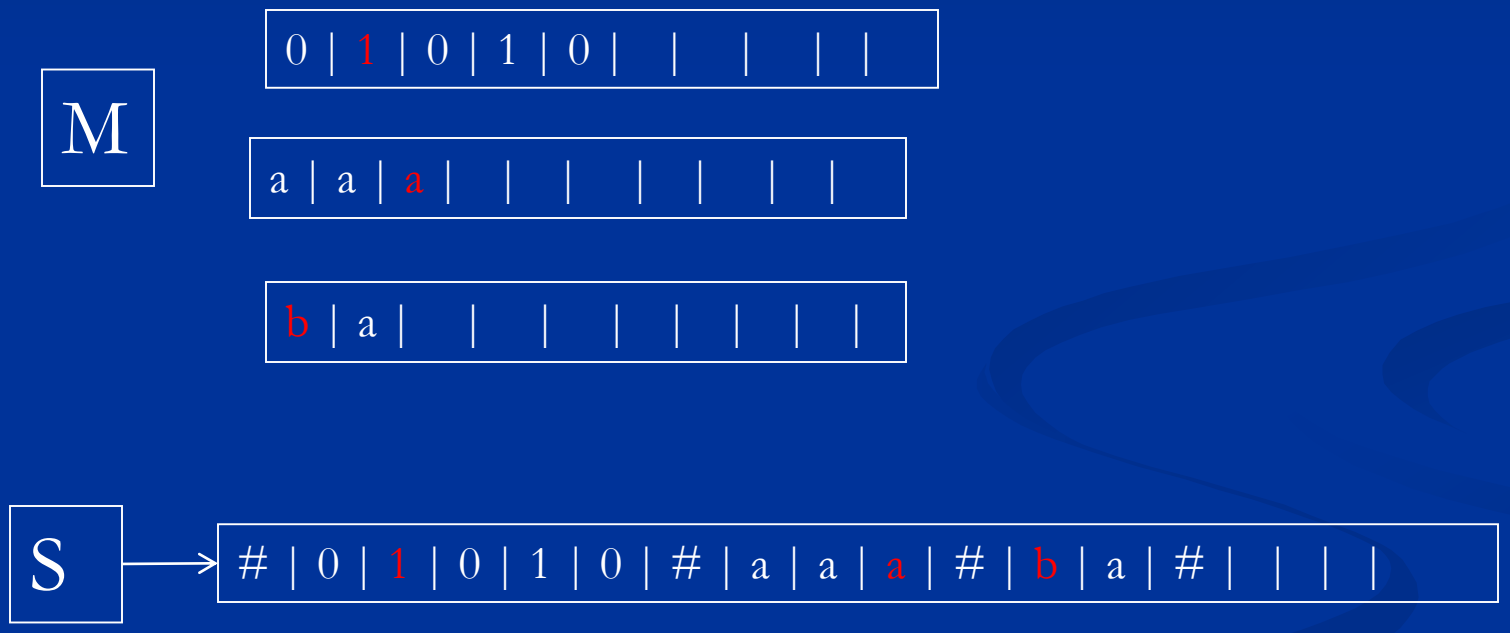
- We show how to convert a multitape TM M with k tapes to an equivalent single-tape TM S
  - S simulates the k tapes of M using a single tape with a # as a delimiter to separate the contents of the k tapes
  - S marks the location of the k heads by putting a dot above the appropriate symbols.
    - Using powerpoint I will use the color red instead

Proof of Equivalence of Variant II

- On input of w = w1, w2…, wn, S will look like:
  #w1w2…wn#-#-# …. #
- To simulate a single move, S scans its tape from the first # to the (k+1)st # in order to determine the symbols under the virtual heads. The S makes a second pass to update the heads and contents based on M's transition function
- If at any point S moves one of the virtual heads to the right onto a #, this action means that M has moved the head to a previously blank portion of the tape. S write a blank symbol onto this cell and shifts everything to the right on the entire tape one unit to the right.
  - Not very efficient. I suppose you could start with blank space on each virtual tape but since there is no finite limit to the physical tape length, one still needs to handle the case that one runs out of space

Example of Variant II

In Multitape case, red indicates where the tape head is located.
In Single tape case, red indicates the symbol with a dot on it.



Variant III: Nondeterministic TM

- Proof of Equivalence: simulate any non-deterministic TM N with a deterministic TM D (proof idea only)
  - D will try all possible branches
  - We can view the branches as representing a tree and we can explore this tree
    - Using depth-first search is a bad idea. Will fully explore one branch before going to the next. If that one loops forever, will never even try most branches.
    - Use breadth-first search. This method guarantees that all branches will be explored to any finite depth and hence will accept if any branch accepts.
      - The DTM will accept if the NTM does
  - Text then goes on to show how this can be done using 3 tapes, one tape for input, one tape for handling current branch, and one tape for tracking position in computation tree

Enumerators

- An enumerator E is a TM with a printer attached
  - The TM can send strings to be output to the printer
  - The input tape is initially blank
  - The language enumerated by E is the collection of strings printed out
  - E may not halt and print out infinite numbers of strings
  - Theorem: A language is Turing-recognizable if and only if some enumerator enumerates it

Proof of Enumerator Equivalence

- First we prove one direction
  - If an enumerator E enumerates a language A then a TM M recognizes it
    - Show how we can turn an enumerator into a recognizer
    - M = "On input w"
      - Run E. Every time E outputs a string compare it to w.
      - If w ever appears in the output of E, accept.
    - Clearly M accepts any string enumerated by E

Proof of Enumerator Equivalence

- Now we prove other direction
  - If a TM M recognizes a language A, we can construct an enumerator E for A as follows:
    - Let s1, s2, s3, … be the list of all possible strings in Σ^*
    - For i = 1, 2, …
      - Run M for i steps on each input s1, s2, …, si.
      - If a string is accepted, then print it.
    - Why do we need to loop over i?
      - We need to do breadth first search so eventually generate everything without getting stuck.

Equivalence with Other Models

- Many variants of TM proposed, some of which may appear very different
  - All have unlimited access to unlimited memory
  - All models with this feature turn out to be equivalent assuming reasonable assumptions
    - Assume only can perform finite work in one step
- Thus TMs are universal model of computation
  - The classes of algorithms are same independent of specific model of computation
- To get some insight, note that all programming languages are equivalent
  - For example, assuming basic constructs can write a compiler for any language with any other language

Chapter 3.3

Definition of an Algorithm

What is an Algorithm?

- How would you describe an algorithm?
- An algorithm is a collection of simple instructions for carrying out some task
  - A procedure or recipe
  - Algorithms abound in mathematics and have for thousands of years
    - Ancient algorithms for finding prime numbers and greatest common divisors

Hilbert's Problems

- In 1900 David Hilbert proposed 23 mathematical problems for next century
- Hilbert's 10^{th} problem:
  - Devise an algorithm for determining if a polynomial has an integral root (i.e., polynomial will evaluate to 0 with this root)
  - Instead of algorithm Hilbert said "a process by which it can be determined by a finite number of operations"
  - For example, $6x^3yz^2 + 3xy^2 - x^3 - 10$ has integral root $x=5$, $y=3$, and $z=0$.
  - He assumed that a method exists.
    - He was wrong

Church-Turing Thesis

- It could not really be proved that an algorithm did not exist without a clear definition of what an algorithm is
- Definition provided in 1936
  - Alonzo Church's λ-calculus
  - Alan Turing's Turing Machines
  - The two definitions were shown to be equivalent
  - Connection between the information notion of an algorithm and the precise one is the Church-Turing thesis
    - The thesis: the intuitive notion of algorithm equals Turing machine
- In 1970 it was shown that no algorithm exists for testing whether a polynomial has integral roots
  - But of course there is an answer: it does or doesn't

More on Hilbert's 10th Problem

- Hilbert essentially asked if the language D is decidable (not just Turing-recognizable)
  - D = {p| p is a polynomial with an integral root}
  - Can you come up with a procedure to answer this question?
    - Try all possible integers. Starting from negative infinity is hard, so start and 0 and loop out: 0, 1, -1, 2, -2, …
    - For multivariate case, just lots of combinations
    - Is this decidable, Turing recognizable, or neither?
    - What is the problem here?
      - May never terminate
      - You will never know whether it will not terminate or will accept shortly
      - So, based on this method Turing-recognizable but not decidable
        - Could be another method that is decidable

More on Hilbert's 10th Problem

- For univariate case, there is actually an upper bound on the root of the polynomial
  - So in this case there is an algorithm and the problem is decidable
  - For multivariate cases has been proven that it is not decidable
- Think about how significant it is that you can prove something cannot be computed
  - Doesn't mean that you are not smart or clever enough
  - We will look into this in Chapter 4

Ways of Describing Turing Machines

- As we have seen before, we can specify the design of a machine (FA, PDA) formally or informally.
  - The same hold true with a Turing Machine
  - The informal description still describes the implementation of the machine– just more informally
- With a TM we can actually go up one more level and not describe the machine (e.g., tape heads, etc.).
  - Rather, we can describe in algorithmically
  - We will either describe them informally (but at the implementation level), or algorithmically

Turing Machine Terminology

- This is for the algorithmic level
- The input to a TM is always a string
  - Other objects (e.g., graphs, lists, etc) must be encoded as a string
  - The encoding of object O as a string is <O>
- We implicitly assume that the TM checks the input to make sure it follows the proper encoding and rejects if not proper

Example: Algorithmic Level

- Let A be the language of all strings representing graphs that are connected (i.e., any node can be reached by any other).
  - A = {<G>| G is a connected undirected graph
- Give me a high level description of TM M

Example Continued

M = "On input <G> the encoding of a graph G:

1. Select and mark the first node in G
2. Repeat the following until no new nodes are marked

   For each node in G, mark it if it is attached by an edge to a node that is already marked
3. Scan all nodes of G to determine whether they are all marked. If they are, then accept; else reject