Notes on Theory of Computation[1]

Hao Su

May 4, 2025

[1]Actually notes on MIT OCW 18.404J, but indexed according to Sipser's *Introduction to the Theory of Computation* (3rd ed).

Contents

Chapter 1

Regular Languages

Defn 1.1 (1.15) A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called ***alphabet***,
3. $\delta : Q \times \Sigma \to Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***. ◇

Defn 1.2 A ***string*** is a finite sequence of symbols in Σ. A ***language*** is a set of strings (finite or infinite). Definition of FAs ***accepting*** strings and ***recognizing*** languages on page 40. ◇

Defn 1.3 (1.16) A language is called a ***regular language*** if some finite automaton recognizes it. ◇

Rmk 1.4 To construct an FA is to keep track of several different possibilities with one state for each. ◇

Defn 1.5 (1.23) The ***regular operations*** include ***union*** $(A \cup B)$, ***concatenation*** (AB) and ***star*** A^*, where A and B are languages. ◇

Defn 1.6 (1.52) The set of ***regular expressions*** is generated from B by regular operations (1.5), where $B = \{\emptyset\} \cup \{\{a\} | a \in \Sigma\}$. ◇

Thm 1.7 (1.25) The class of regular languages is closed under the union operation. ◇

Proof Idea. We construct an FA that keeps track of the states of *both* given FAs, and terminates when either does. Hence $Q = Q_1 \times Q_2, F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$ and so forth. ⊣

Thm 1.8 (1.26) The class of regular languages is closed under the concatenation operation. ◇

Defn 1.9 (1.37) A ***Nondeterministic*** FA is again a 5-tuple, differing from an DFA in that the transition function δ is of the form $Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$. ◇

Rmk 1.10 Ways to think about nondeterminism:

1. Computational: Fork new parallel thread and accept if any thread leads to an accept state.
2. Mathematical: Tree with branches. Accept if any branch leads to an accept state.
3. Magical: Guess at each nondeterministic step which way to go. Machine always makes the right guess that leads to accepting, if possible. ◇

Thm 1.11 (1.39) Every nondeterministic finite automaton has an equivalent deterministic finite automaton. ◇

Proof Idea. We construct a DFA M' that keeps track of the set of possible states in the given NFA M. Hence $Q' = \mathcal{P}(Q), q_0' = \{q_0\}, F' = \{R \in Q' | R \cap F \neq \emptyset\}$ and so forth. ⊣

Proof Idea for 1.8. We construct an NFA by putting in ε transitions going from F_1 to q_2. The other accomodations are immediate. ⊣

Thm 1.12 The class of regular languages is closed under the star operation. ◇

Proof Idea. We construct an NFA by putting in ε transitions going from F to q_0. The new start state q_0' is in F' and has an ε transition pointing toward q_0. We can not simply put q_0 in F' for that if $q_0 \notin F$, M' might accept some string that M does not. ⊣

Lem 1.13 (1.55) If a language is described by a regular expression, then it is regular. ◇

Proof Idea. We construct NFAs for regular expressions in B (1.6), and composite them the way described when proving that the class of regular languages is closed under regular operations. (This is indeed indution.) ⊣

Lem 1.14 (1.60) If a language is regular, then it is described by a regular expression. ◇

Proof Idea. We construct a special type of NFAs that can be easily converted to a form equivalent to a regular expression, and then describe a procedure that takes a DFA, converts it to an NFA of such type and returns a regular expression. The conversion procedure should maintain the language our automata recognize.

Defn 1.15 (1.64) A *Generalized* NFA is an NFA that allows regular expressions as transition labels. ◇

For convenience assume that (1) one accept state, separate from the start state and (2) one arrow from each state to each state, except only exiting the start state and only entering the accept state. This is the type of NFAs we ask for, and the conversion from DFAs to this type is trivial. After conversion we prove by induction that this NFA is *indeed* equivalent to a regular expression. ⊣

Thm 1.16 (1.70) **Pumping lemma:** For every regular language A, there is a number p (the "pumping length") such that if $s \in A$ and $|s| \geq p$ then $s = xyz$ where

1. $xy^iz \in A$ for all $i \geq 0$
2. $y \neq \varepsilon$
3. $|xy| \leq p$ ◇

Proof Idea. There are finite states, and a long enough input string is going to go through some state twice. This lemma essentially formalizes this. ⊣

Eg 1.17 We show a bunch of examples of proving (by contradiction) non-regularity according to 1.16, where $\Sigma = \{0, 1\}$, L is the language, $s \in L$ and p is the pumping length.

1. $L = 0^k1^k$. We choose $s = 0^p1^p$.
2. $L = \{ww | w \in \Sigma^*\}$. We choose $s = 0^p10^p1$.
3. $L = \{w | w$ has equal numbers of 0s and 1s$\}$. We can easily show that regular languages are closed under intersection, so $L \cap 0^*1^*$ is regular, which is item 1. Thus contradiction. ◇

Exercises and Problems

1.31 Show that if language A is regular, so is its *reverse* $A^{\mathcal{R}}$. ◁

Given a DFA M that recognizes A, we construct an equivalent NFA M' by adding a new state q and putting ε transitions going from F to q. So q is the unique accept state in M'. Swap the start state and q in M' and define a new δ accordingly and we have an NFA that recognizes $A^{\mathcal{R}}$.

1.32 Let

$$\Sigma_3 = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\}.$$

Σ_3 contains all size 3 columns of 0s and 1s. A string of symbols in Σ_3 gives three rows of 0s and 1s. Consider each row to be a binary number and let

$$B = \{w \in \Sigma_3^* \mid \text{the bottom row of } w \text{ is the sum of the top two rows}\}.$$

Show that B is regular. ◁

By 1.31 consider $B^{\mathcal{R}}$. A DFA is straightforward that only has 3 states.

1.41 For languages A and B, let the ***perfect shuffle*** of A and B be the language

$$\{w | w = a_1 b_1 \cdots a_k b_k, \text{ where } a_1 \cdots a_k \in A \text{ and } b_1 \cdots b_k \in B, \text{ each } a_i, b_i \in \Sigma\}$$

Show that the class of regular languages is closed under perfect shuffle. ◁

Similar to the proof of 1.7, we construct an FA that keeps track of the states of both given FAs *and* parity, and terminates when both does *and* the parity is even. Hence $Q = Q_1 \times Q_2 \times \{0, 1\}$, $F = F_1 \times F_2 \times \{0\}$, δ behaves like δ_A on the set of even states and so forth.

1.51 Let x and y be strings and let L be any language. We say that x and y are ***distinguishable by*** L if some string z exists whereby exactly one of the strings xz and yz is a member of L; otherwise, for every string z, we have $xz \in L$ iff $yz \in L$ and we say that x and y are ***indistinguishable by*** L (written $x \equiv_L y$). Show that $\equiv_L$ is an equivalence relation. ◁

Omitted as trivial.

1.52 Myhill-Nerode theorem. Refer to 1.51. Let L be a language and let X be a set of strings. Say that X is ***pairwise distinguishable by*** L if every two distinct strings in X are distinguishable by L. Define the ***index of*** L to be the maximum number of elements in any set that is pairwise distinguishable by L. The index of L may be finite or infinite.

a. Show that if L is recognized by a DFA with k states, L has index at most k.

b. Show that if the index of L is a finite number k, it is recognized by a DFA with k states.

c. Conclude that L is regular iff it has finite index. Moreover, its index is the size of the smallest DFA recognizing it. ◁

Proof. **a.** If α_1 and α_2 is distinguishable by DFA M, the states M ends in upon inputs α_1 and α_2 have to differ. Thus k states indicates the index of L is not greater than k.

b. Let $X = \{s_1, \ldots, s_k\}$ be pairwise distinguishable by L. We construct DFA $M = (Q, \Sigma, \delta, q_0, F)$ with k states recognizing L. Let $Q = \{q_1, \ldots, q_k\}$, $F = \{q_i \mid s_i \in L\}$, the start state q_0 be the q_i such that $s_i \equiv_L \varepsilon$ and define $\delta(q_i, a)$ to be q_j where $s_j \equiv_L s_i a$. M is constructed so that, for any state q_i, $\{s' \mid \delta(q_0, s') = q_i\} = \{s' \mid s' \equiv_L s_i\}$. Hence M recognizes L. (I thought I had to construct according to k *strings* but I was actually given the quotient of *all strings* w.r.t. $\equiv_L$ and that actually defines the DFA *itself*.)

c. This conclusion is immediate by item a and b. ⊣

1.53 Let $\Sigma = \{0, 1, +, =\}$ and

$$ADD = \{x = y + z \mid x, y, z \text{ are binary integers, and } x \text{ is the sum of } y \text{ and } z\}.$$

Show that ADD is not regular. ◁

Suppose ADD is regular and has pumping length p for the sake of contradiction. For $y, z \in 1^p\{0,1\}^*$, of course $|x| > p$. According to 1.16 we have multiple sum of y and z, hence contradiction.

1.59 Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let h be a state of M called its "home". A ***synchronizing sequence*** for M and h is a string $s \in \Sigma^*$ where $\delta'(q, s) = h$ for every $q \in Q$, where δ' is intuitively extended onto strings. Say that M is ***synchronizable*** if it has a synchronizing sequence for some state h. Prove that if M is a k-state synchronizable DFA, then it has a synchronizing sequence of length at most k^3. Can you improve upon this bound? ◁

to do

1.60 Let $\Sigma = \{a, b\}$. $C_k = \Sigma^* a \Sigma^{k-1}$. Describe an NFA with $k + 1$ states that recognizes C_k in terms of both a state diagram and a formal description. ◁

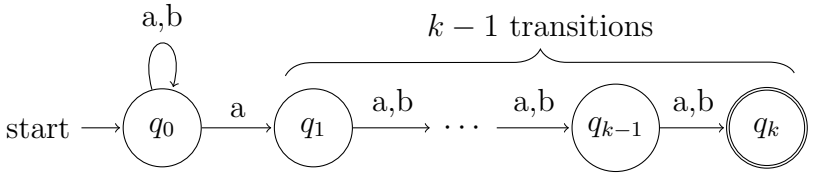

Figure 1.1: State diagram. A formal description is omitted.

1.61 Consider the languages C_k defined in Problem 1.60. Prove that for each k, no DFA can recognize C_k with fewer than 2^k states. ◁

Proof. There are 2^k distinct Myhill-Nerode equivalence classes of strings w.r.t. $\equiv_{C_k}$. They are exactly the equivalence classes of strings with suffix $b^k, b^{k-1}a, b^{k-2}ab, \ldots, a^k$ respectively. By 1.52 we are done. ⊣

Chapter 2

Context-Free Languages

Defn 2.1 (2.2) A ***context-free grammar*** G is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ a finite set such that $\Sigma \cap V = \emptyset$ called ***terminals***,
3. R a finite set of ***rules*** where each rule is a funtion of the form $V \to (V \cup \Sigma)^*$ and
4. $S \in V$ the start variable.

For $u, v \in (V \cup \Sigma)^*$ write

1. $u \Rightarrow v$ (say that u ***yields*** v) if can go from u to v with one substitution step.
2. $u \stackrel{*}{\Rightarrow} v$ (say that u ***derives*** v) if can go from u to v with some number of substitions steps.
3. $u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k = v$ is called a ***derivation*** of v from u. If $u = S$ then "from u" can be omitted.

$L(G) = \{w | (w \in \Sigma^*) \wedge (S \stackrel{*}{\Rightarrow} v)\}$ is called a ***context-free language***. $\diamond$

Context-free means that we can apply substitutions regardless of the contexts.

Defn 2.2 (2.7) A derivation of a string w in a context-free grammar G is a ***leftmost derivation*** if at every step the leftmost remaining variable is the one replaced. w is derived ***ambiguously*** in G if it has two or more different leftmost derivations. G is ***ambiguous*** if it generates some string ambiguously. $\diamond$

Sometimes when we have an ambiguous grammar we can find an unambiguous grammar that generates the same language. Some context-free languages, however, can

be generated only by ambiguous grammars. Such languages are called ***inherently ambiguous***.

Defn 2.3 (2.8) A CFG is in ***Chomsky normal form*** if every rule is of the form $A \to BC$ or $A \to a$, where a is any terminal and A, B and C are any variables (except that B and C may not be the start variable). In addition permit $S \to \varepsilon$, where S is the start variable. $\diamond$

Thm 2.4 (2.9) Any context-free language is generated by a context-free grammar in Chomsky normal form. $\diamond$

Proof Idea. We basically convert an arbitrary CFG into Chomsky normal form. This is nontrivial and demands carefulness. For the proof see page 109. $\dashv$

Defn 2.5 (2.13) A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Γ is the stack alphabet and $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ the transition function. $\diamond$

Our PDA models is nondeterministic. The nondeterministic forks replicate the stack. Also, we can check if the stack is effectively empty by writing at the beginning a custom sign (say \$) at the bottom of it.

Lem 2.6 (2.21) If A is a CFL then some PDA recognizes it. $\diamond$

Proof Idea. PDA begins with starting variable and guesses substitutions. It keeps intermediate generated strings on stack. When done, compare with input. We can only substitute variables when on the top of stack. If a terminal is on the top of stack, pop it and compare with input. $\dashv$

Lem 2.7 (2.27) If a PDA recognizes A then it is a CFL. $\diamond$

Proof. Say that $P = \{Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\}\})$ and construct G. The variables of G are $\{A_{p,q} | p, q \in Q\}$. The start variable is $A_{q_0, q_{accept}}$. The set of rules R of G is described as follows.

1. For each $p, q, r, s \in Q, u \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, if $(r, u) \in \delta(p, a, \varepsilon) \wedge (q, \varepsilon) \in \delta(s, b, u), (A_{p,q} \to aA_{r,s}b) \in R$.
2. For each $p, q, r \in Q, (A_{p,q} \to A_{p,r}A_{r,q}) \in R$.
3. For each $p \in Q, (A_{p,p} \to \varepsilon) \in R$. $\dashv$

Thm 2.8 Pumping Lemma for CFLs: For every CFL A, there is a p such that if $s \in A$ and $|s| \geq p$ then $s = uvxyz$ where

1. $uv^ixy^iz \in A$ for all $i \geq 0$

2. $vy \neq \varepsilon$
3. $|vxy| \leq p$ ◇

Proof Idea. Let b be the length of the longest right hand side of a rule (the max branching of the parse tree). Let h be the height of the parse tree for s. A tree of height h and max branching b has at most b^h leaves, so $|s| \leq b^h$. Let $p = b^{|V|}+1$ where $|V|$ is the number of variables in the grammar. So if $|s| \geq p > b^{|V|}$ then $|s| > b^{|V|}$ and so $h > |V|$. Thus at least $|V| + 1$ variables occur in the longest path. So some variable R must repeat on a path. For item 1 we can cut and paste R arbitrarily. For item 2 we add a requirement that the parse tree has to be smallest, meaning that R to R without generating new symbols is not allowed during the construction of the parse tree. For item 3 choose the lowest repitition of R, for that if $|vxy| > p$, a lower reoccurrence would happen, contradicting "lowest". ⊣

Eg 2.9 Usages of 2.8:

1. $0^k1^k2^k$ is not a CFL. Suppose it is with pumping length p then we can pump $0^p1^p2^p$, which we cannot.
2. ww is not a CFL. Suppose it is with pumping length p then we can pump $0^p1^p0^p1^p$, which we cannot. (Use uv^0xy^0z, or any other pumping choices, as long as you take care of what is pumped.) ◇

Obviously $0^k1^k2^l$ and $0^l1^k2^k$ both are CFLs, thus we have a counterexample showing that the class of CFLs is not closed under intersection. By 2.16 it is closed under union. So it is not closed under complementation by De Morgan's law ($\overline{\overline{A} \cup \overline{B}} = A \cap B$). Why is it closed under union but not closed under intersection? Intuitively, given two CFGs, you can guess which one to use to get the union, but there is no obvious way to use them *both*. Also, given two PDAs, you can guess which of the stacks to keep, but you cannot easily simulate *both* with only one.

Exercises and Problems

2.15 Give a counterexample to show that the following construction fails to prove that the class of context-free languages is closed under star. Let A be a CFL that is generated by the CFG $G = (V, \Sigma, R, S)$. Add the new rule $S \to SS$ and call the resulting grammar G'. This grammar is supposed to generate A^*. ◁

Let $G = (\{S\}, \{a\}, \{S \to a\}, S)$, then $L(G')$ does not contain $\varepsilon \in A^*$. Another counterexample would be $G = (\{S\}, \{a, b\}, \{S \to aSa|b\}, S)$. Then $L(G')$ contains $abba \notin A^*$. This exercise tell us to be careful with the original grammar.

2.16 Show that the class of context-free languages is closed under the regular operations (1.5). ◁

Say that we are given CFGs G_1 and G_2. To get $G_\cup$ we add rule $S \to S_1|S_2$. To get $G_\circ$ we add rule $S \to S_1S_2$. To get G_* (of G_1) we add rule $S \to SS_1|\varepsilon$.

2.17 Refer to 2.16 to give another proof that every regular language is context free by showing how to convert a regular expression directly to an equivalent context-free grammar. ◁

Define a CFG G according to a regular language R. Consider the subset R_G of a regular language that can be generated by G. Obviously R_G is closed under regular operations and contains the symbols that occur in R. By induction theorem we have $R_G = R$, so R is a CFL.

2.18 (a) Let C be a context-free language and R be a regular language. Prove that the language $C \cap R$ is context free. (b) Use part (a) to show that the language $A = \{w|w \in \{a, b, c\}^*$ and contains equal numbers of a's, b's, and c's$\}$ is not a CFL. ◁

(a) We can construct a PDA whose set of states is $Q = Q_C \times Q_R$, the transition function δ defined by $\delta((q_c, q_r), a, x) = \{((q'_c, q'_r), \gamma)|(q'_c, \gamma) \in \delta_C(q_c, a, x), q'_r = \delta_R(q_r, a)\}$ and accept states $F = F_C \times F_R$. (b) If it is, so would be $a^kb^kc^k = A \cap a^*b^*c^*$, which is not by 2.8.

2.22 Let $C = \{x\#y|x, y \in \{0, 1\}^*$ and $x \neq y\}$. Show that C is context-free. ◁

to do

2.24 Show that $E = \{a^ib^j|i \neq j \wedge 2i \neq j\}$ is a context-free language. ◁

to do

2.26 Show that, if G is a CFG in Chomsky normal form, then for any string $w \in L(G)$ of length $n \geq 1$, exactly $2n - 1$ steps are required for any derivation of w. ◁

Each application of a rule of the form $A \to BC$ increases the length of the string by 1 (because B and C are never to derive ε), so we have $n - 1$ steps here. We also need n applications of terminal rules $A \to a$ to convert the variables into terminals. So $2n - 1$ steps are needed as a minimum. Note also that any one more step will lead to a string different from w.

2.27 $G = (V, \Sigma, R, \langle \text{STMT} \rangle)$ is described as follows.

$$\begin{aligned}
\langle \text{STMT} \rangle &\to \langle \text{ASSIGN} \rangle | \langle \text{IF-THEN} \rangle | \langle \text{IF-THEN-ELSE} \rangle \\
\langle \text{IF-THEN} \rangle &\to \texttt{if condition then } \langle \text{STMT} \rangle \\
\langle \text{IF-THEN-ELSE} \rangle &\to \texttt{if condition then } \langle \text{STMT} \rangle \texttt{ else } \langle \text{STMT} \rangle \\
\langle \text{ASSIGN} \rangle &\to \texttt{a:=1}, \\
\Sigma &= \{\texttt{if,condition,then,else,a:=1}\} \\
V &= \{\langle \text{STMT} \rangle, \langle \text{IF-THEN} \rangle, \langle \text{IF-THEN-ELSE} \rangle, \langle \text{ASSIGN} \rangle\}
\end{aligned}$$

(a) Show that G is ambiguous.

(b) Give a new unambiguous grammar for the same language. ◁

to do

2.32 Let $\Sigma = \{1, 2, 3, 4\}$ and $C = \{w \in \Sigma^* \mid \text{in } w, \text{ the number of 1s equals the number of 2s, and the number of 3s equals the number of 4s}\}$. Show that C is not context-free. ◁

Suppose C is with pumping length p. But we can not pump $1^p 3^p 2^p 4^p$.

Chapter 3

The Church-Turing Thesis

Defn 3.1 (3.3) A ***Turing Machine*** (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where

1. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
2. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \in \Gamma$,
3. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is the transition function,
4. $q_{accept} \neq q_{reject}$.

On input w a TM M may halt (enter q_{accept} or q_{reject}) or run forever ("loop"). M ***accepts*** w by entering q_{accept} and ***rejects*** w by entering q_{reject} *or* looping. $\diamond$

Defn 3.2 (3.5) The collection of strings M accepts is the ***language*** M, or the ***language*** recognized by M (written $L(M)$). Call a language ***Turing-recognizable*** (or ***recursively enumerable***) if some TM recognizes it. $\diamond$

Defn 3.3 (3.6) TM M is a ***decider*** if M halts on all inputs. Say that M decides A if $A = L(M)$ and M is a decider. Call a language ***Turing-decidable*** or simply ***decidable*** (or ***recursive***) if some TM decides it. $\diamond$

Thm 3.4 (3.13) Every multitape Turing machine has an equivalent single-tape Turing machine. $\diamond$

Proof Idea. Suppose we have a multi-tape TM M and a single-tape TM S. S can simulate M by storing the contents of multiple tapes on a single tape in "blocks" and record head positions with dotted symbols. To simulate each of M's steps, S can (a) scan entire tape to find dotted symbols, (b) scan again to update according to M's δ and (c) shift to add room as needed. Finally, S accepts/rejects if M does. $\dashv$

Defn 3.5 A ***Nondeterministic*** TM (NTM) is similar to a TM except for its transition function $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\})$. ◇

Thm 3.6 (3.16) Every nondeterministic Turing machine has an equivalent deterministic Turing machine. ◇

Proof Idea. Suppose we have an NTM N and a TM M. M can simulate N by storing each thread's tape in a separate "block" on its tape. M also need to store the head location, and the state for each thread, in the block. If a thread forks, then M copies the block. If a thread accepts then M accepts. ⊣

Defn 3.7 An ***enumerator*** is a TM with a printer. It starts on a blank tape and can print strings $w_1, w_2, w_3, \ldots$, possibly forever. For enumerator E say $L(E) = \{w | E \text{ prints } w\}$. ◇

Thm 3.8 (3.21) A is T-recognizable iff $A = L(E)$ for some enumerator E. ◇

Proof Idea. ($\Leftarrow$) Simulate E with TM M. For input w, whenever E prints x, test if $x = w$. Accept or continue accordingly.
($\Rightarrow$) Simulate TM M with E on *each* possible input w_i. Let E print accordingly whenever M accepts. We can do this in a "time-sharing" scheme, for example let M go through 1 transitions on input w_1, then 2 transitions on input w_1 and w_2 respectively, then 3 transitions on input w_1, w_2 and w_3 respectively and so forth. When switching the input that M is currently working on, keep track of the state and the place of the head on the tape and the input together on a block of the tape. ⊣

Note 3.9 The definition of ***algorithms*** came in the 1936 papers of Alonzo Church and Alan Turing. Church used a notational system called the λ-calculus to define algorithms. Turing did it with his "machines". These two definitions were shown to be equivalent. This equivalence relation (in some sense) between the informal notion of algorithm and the precise definition (TM) has come to be called the ***Church–Turing thesis***. Note that it is about the equivalence between the intuitive and the formal, thus *not* provable. Instead it's a philosophical postulate. ◇

Note 3.10 David Hilbert's tenth problem was to devise an algorithm that tests whether a polynomial has an integral root (*Diophantine eqations*). The set of such polynomials is easily recognizable but was proved to be not decidable in 1970. ◇

Note 3.11 If O is some object, write $\langle O \rangle$ to be an encoding of that object into a string. (For example $\langle M \rangle$, where M is a TM. $\langle O_1, \ldots, O_k \rangle$ is similarly defined.) Furthermore we will use high-level English descriptions of algorithms when we describe

TMs, knowing that we could (in principle) convert those descriptions into states, transition function, etc. ◇

Exercises and Problems

3.12 Question to fill in. ◁

to do

3.14 A ***queue automaton*** is like a push-down automaton except that the stack is replaced by a queue. A queue is a tape allowing symbols to be written only on the left-hand end and read only at the right-hand end. Each write operation (we'll call it a push) adds a symbol to the left-hand end of the queue and each read operation (we'll call it a pull) reads and removes a symbol at the right-hand end. As with a PDA, the input is placed on a separate read-only input tape, and the head on the input tape can move only from left to right. The input tape contains a cell with a blank symbol following the input, so that the end of the input can be detected. A queue automaton accepts its input by entering a special accept state at any time. Show that a language can be recognized by a deterministic queue automaton iff the language is Turing-recognizable. ◁

to do

3.18 Show that a language is decidable iff some enumerator enumerates the language in the standard string order. ◁

to do

Chapter 4

Decidability

Thm 4.1 (4.1) $A_{\text{DFA}} = \{\langle B, w\rangle | B \text{ is a DFA that accepts } w\}$ is decidable. $\diamond$

Proof Idea. We use a TM to simulate B on w, this is clearly a decider. $\dashv$

Thm 4.2 (4.2) Also, A_{NFA} is decidable. $\diamond$

Proof Idea. Convert the NFA to a DFA and use the TM from 4.1 to decide. Actually an NFA might loop for that it takes ε. It actually still will be decidable, but that is something we have to prove. $\dashv$

Thm 4.3 (4.3) $E_{\text{DFA}} = \{\langle A\rangle | A \text{ is a DFA and } L(A) = \emptyset\}$ is decidable. $\diamond$

Proof Idea. We use BFS to see which of the states of A are reachable. $\dashv$

Thm 4.4 (4.5) $EQ_{\text{DFA}} = \{\langle A, B\rangle | A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ is decidable. $\diamond$

Proof Idea. Make DFA C that accepts w where A and B disagree (easily, $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$) and test if $L(C) = \emptyset$. If we are to prove it by feeding strings into the simulated DFAs it would be way more complicated. A basic idea is to try to prove that if $L(A) \neq L(B)$, some strings whose length are at most some bound (say the sum or product of the numbers of states in A and B) are going to behave differently when fed to A and B. $\dashv$

Thm 4.5 (4.7) $A_{\text{CFG}} = \{\langle G, w\rangle | G \text{ is a CFG that generates } w\}$ is decidable. $\diamond$

Proof Idea. By 2.4 and 2.26 this is trivial. So A_{PDA} is also decidable. Note that this is not easy to prove on its own, since PDAs may not halt. ⊣

Thm 4.6 (4.8) $E_{\text{CFG}} = \{\langle G\rangle | G \text{ is a CFG and } L(G) = \emptyset\}$ is decidable. ◇

Proof Idea. Mark all the terminals in G. Then repeat the following until new variables are marked: mark all occurrences of variable A if $A \to B_1 B_2 \cdots B_k$ is a rule and all B_i's were already marked. ⊣

Thm 4.7 (4.9) Every context-free language is decidable. ◇

Proof Idea. It follows immediately from 4.5. Note that we know it is decidable, without knowing *how* to decide it. ⊣

Thm 4.8 $EQ_{\text{CFG}} = \{\langle G, H\rangle | G, H \text{ are CFGs and } L(G) = L(H)\}$ is undecidable. ◇

Thm 4.9 (4.11) $A_{\text{TM}} = \{\langle M, w\rangle | M \text{ is a TM that accepts } w\}$ is undecidable. ◇

Rmk 4.10 A_{TM} is recognizable. ◇

We can prove it by simulating. Note that we do not ask the simulator TM to reject if M rejects by looping (one should prove that this can be carried out effectively, but shows that it cannot) and instead we say nothing about the looping condition, for that if M rejects by looping, our simulator will as well. This simulator is of great historical importance, as the *first* machine described (due to Alan Turing) to operate based on a stored program, and it later came to be known as the Von Neumann architecture.

Exercises and Problems

4.14 Question to fill in. ◁

to do (hard!)

4.18 Let C be a language. Prove that C is T-recognizable iff a decidable language D exists such that $C = \{x | \exists y(\langle x, y\rangle \in D)\}$. ◁

Proof. ($\Leftarrow$) Consider a string x, we enumerate $y \in \Sigma^*$ and see if $\langle x, y_i\rangle \in D$ for at least one i. Thus C is recognizable. *($\Rightarrow$) Say that $L(M) = C$, where M is a TM. Let $D = \{\langle x, y\rangle | M \text{ accepts } x \text{ in } y \text{ steps}\}$, and we are done. ⊣

Comment. A bound that can approach infinity can be useful to turn something recognizable into something decidable.

4.24 A *useless state* in a pushdown automaton is never entered on any input string. Consider the problem of determining whether a pushdown automaton has any useless states. Formulate this problem as a language and show that it is decidable. ◁

to do

4.27 Question to fill in. ◁

to do

Chapter 5

Reducibility

Exercises and Problems

5.21 Question to fill in. ◁

to do

5.22 Show that A is T-recognizable iff $A \leq_m A_{\text{TM}}$. ◁

to do

5.23 Show that A is decidable iff $A \leq_m 0^*1^*$. ◁

to do

5.25 Give an example of an undecidable language B, where $B \leq_m \overline{B}$. Is it possible that B is a T–recognizable language? ◁

to do

5.26 Question to fill in. ◁

to do

Chapter 6

Advanced Topics in Computability Theory

Exercises and Problems

6.1 Give an example in the spirit of the recursion theorem of a program in a real programming language (or a reasonable approximation thereof) that prints itself out. ◁

to do

6.11 Question to fill in. ◁

to do