

Notes on Theory of Computation¹

Hao Su

May 13, 2025

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

Contents

1	Regular Languages	1
2	Context-Free Languages	7
3	The Church-Turing Thesis	12
4	Decidability	15
4.1	Decidable Languages	15
4.2	Undecidability	16
5	Reducibility	19
6	Advanced Topics in Computability Theory	25

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. $\delta : Q \times \Sigma \rightarrow Q$ 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. \dashv

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

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 $Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$. \diamond

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. \diamond

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

Proof Idea. We construct a DFA M' that keeps track of the set of possible states in the given NFA M . Hence $Q' = \mathcal{P}(Q)$, $q'_0 = \{q_0\}$, $F' = \{R \in Q' \mid R \cap F \neq \emptyset\}$ and so forth. \dashv

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

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

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. \dashv

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

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 induction.) \dashv

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

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. \diamond

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. \dashv

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. $|xy| \leq p$ \diamond

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. \dashv

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. $L = 0^k 1^k$. We choose $s = 0^p 1^p$.
2. $L = \{ww \mid w \in \Sigma^*\}$. We choose $s = 0^p 10^p 1$.
3. $L = \{w \mid w \text{ has equal numbers of 0s and 1s}\}$. We can easily show that regular languages are closed under intersection, so $L \cap 0^* 1^*$ is regular, which is item 1. Thus contradiction. \diamond

Exercises and Problems

1.31 Show that if language A is regular, so is its *reverse* A^R . \triangleleft

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^R .

1.32 Let

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

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. \triangleleft

By 1.31 consider B^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

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

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

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. \triangleleft

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. \triangleleft

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 $X = \{s_1, \dots, s_k\}$ be pairwise distinguishable by L . We construct DFA $M = (Q, \Sigma, \delta, q_0, F)$ with k states recognizing L . Let $Q = \{q_1, \dots, q_k\}$, $F = \{q_i \mid s_i \in L\}$, 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. \dashv

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. \triangleleft

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? \triangleleft

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. \triangleleft

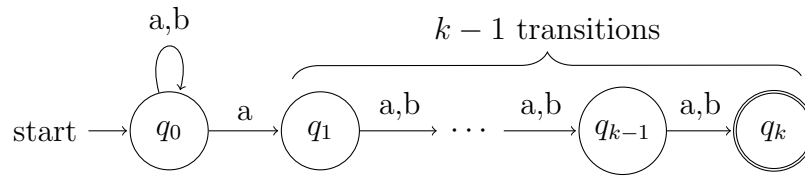


Figure 1.1: State diagram. A formal description is omitted.

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. \triangleleft

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 $b^k, b^{k-1}a, b^{k-2}ab, \dots, a^k$ respectively. By 1.52 we are done. \dashv

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 function of the form $V \rightarrow (V \cup \Sigma)^*$ 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. $u \xRightarrow{*} v$ (say that u *derives* v) if can go from u to v with some number of substitutions steps.
3. $u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_k = v$ is called a *derivation* of v from u . If $u = S$ then “from u ” can be omitted.

$L(G) = \{w | (w \in \Sigma^*) \wedge (S \xRightarrow{*} w)\}$ 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 $A \rightarrow BC$ or $A \rightarrow a$, 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 $S \rightarrow \varepsilon$, 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 $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ 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 $(r, u) \in \delta(p, a, \varepsilon) \wedge (q, \varepsilon) \in \delta(s, b, u)$, $(A_{p,q} \rightarrow aA_{r,s}b) \in R$.
2. For each $p, q, r \in Q$, $(A_{p,q} \rightarrow A_{p,r}A_{r,q}) \in R$.
3. For each $p \in Q$, $(A_{p,p} \rightarrow \varepsilon) \in R$. \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. $uv^i xy^i z \in A$ for all $i \geq 0$

2. $vy \neq \varepsilon$

3. $|vxy| \leq p$ \diamond

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 repetition of R , for that if $|vxy| > p$, a lower reoccurrence would happen, contradicting “lowest”. \dashv

Eg 2.9 Usages of 2.8:

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

Obviously $0^k 1^k 2^l$ and $0^l 1^k 2^k$ 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 $S \rightarrow SS$ and call the resulting grammar G' . This grammar is supposed to generate A^* . \triangleleft

Let $G = (\{S\}, \{a\}, \{S \rightarrow a\}, S)$, then $L(G')$ does not contain $\varepsilon \in A^*$. Another counterexample would be $G = (\{S\}, \{a, b\}, \{S \rightarrow aSa|b\}, S)$. 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). \triangleleft

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

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. \triangleleft

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 $A = \{w|w \in \{a, b, c\}^* \text{ and contains equal numbers of } a\text{'s, } b\text{'s, and } c\text{'s}\}$ is not a CFL. \triangleleft

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

to do

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

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 . \triangleleft

Each application of a rule of the form $A \rightarrow BC$ 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 $A \rightarrow a$ 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.

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

(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 $\Sigma \subseteq \Gamma$,
3. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ 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 $\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$. \diamond

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

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. \dashv

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, \dots , possibly forever. For enumerator E say $L(E) = \{w \mid E \text{ prints } w\}$. \diamond

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

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. \dashv

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. \diamond

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

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. $\langle O_1, \dots, O_k \rangle$ 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. \diamond

Exercises and Problems

3.12 Question to fill in. \triangleleft

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. \triangleleft

to do

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

to do

Chapter 4

Decidability

4.1 Decidable Languages

Thm 4.1 (4.1) $A_{\text{DFA}} = \{\langle B, w \rangle \mid B \text{ is a DFA that accepts } w\}$ 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) $E_{\text{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$ is decidable. \diamond

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

Thm 4.4 (4.5) $EQ_{\text{DFA}} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ 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) $A_{\text{CFG}} = \{\langle G, w \rangle \mid G \text{ is a CFG that generates } w\}$ is decidable. \diamond

Proof Idea. By 2.4 and 2.26 this is trivial. \dashv

Comment. 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) $E_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ is decidable. \diamond

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

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

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

4.2 Undecidability

Thm 4.8 \mathbb{R} is uncountable. \diamond

Proof. We prove by contradiction and via diagonalization. Say that each real number is assigned an index $i \in \mathbb{N}$. Then $r \in \mathbb{R}$ in its decimal representation that differs from the n th number in the n th digit for any $n \in \mathbb{N}$ is not i th number for any i . Hence $r \notin \mathbb{R}$ and contradiction. \dashv

Cor 4.9 The set of all languages over Σ is not countable. \diamond

Proof. Σ^* is countable. We can assign each language an infinite *binary* decimal, with each digit of it representing if a string is present in the language. \dashv

Observe that the set \mathcal{M} of all turing machines is countable. So some languages are not recognizable, a fortiori decidable.

Note 4.10 David Hilbert's first problem asked if there is a set of intermediate size between \mathbb{N} and \mathbb{R} . Gödel and Cohen showed that we cannot answer this question by using the standard axioms of mathematics. We are even not sure if "infinite sets" has mathematical "reality". \diamond

Rmk 4.11 A_{TM} is recognizable. \diamond

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 5.1 shows that it cannot, for that we can not tell if a TM is looping on some input) 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.

Thm 4.12 (4.11) $A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM that accepts } w\}$ is undecidable. \diamond

Proof. Assume some TM H decides A_{TM} . We use H to construct TM

- $D =$ “On input $\langle M \rangle$
1. Simulate H on input $\langle M, \langle M \rangle \rangle$.
 2. Accept if H rejects and reject if H accepts.”

Then D accepts $\langle D \rangle$ iff D does not accept $\langle D \rangle$. Hence contradiction. \dashv

Comment. Surprisingly $\langle M, \langle M \rangle \rangle$ is of some real world applications, for example self-hosting languages whose compilers can be written in themselves. Check the following figure to see that this proof uses *diagonalization*. To employ it we essentially attempt to show contradiction by constructing some item (by countability) that differs from all items of its type by some *decidable* process.

TMs	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	\dots	$\langle D \rangle$
M_1	acc	rej	acc	acc	\dots	acc
M_2	rej	rej	rej	rej	\dots	rej
M_3	acc	acc	acc	acc	\dots	acc
M_4	rej	rej	acc	acc	\dots	rej
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots	\vdots
D	rej	acc	rej	rej	\dots	?

Figure 4.1: Undecidability hinges on diagonalization.

Thm 4.13 A language is decidable iff both it and its complement are Turing-recognizable. \diamond

Proof. We construct a decider by simulating both TMs in parallel (alternately). \dashv

Cor 4.14 $\overline{A_{\text{TM}}}$ is unrecognizable. \diamond

Comment. 4.13 can be stated more generally: Let D and L be two languages, and L a decidable one. Then $D \cap L$ is decidable iff both it and $\overline{D} \cap L$ is recognizable.

Exercises and Problems

4.14 Show that

$$\{\langle G \rangle \mid G \text{ is a CFG over } \{0, 1\} \text{ and } 1^* \cap L(G) \neq \emptyset\}$$

is decidable. \triangleleft

to do (hard!)

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

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. $\ast(\Rightarrow)$ Say that $L(M) = C$, where M is a TM. Let $D = \{\langle x, y \rangle \mid M \text{ accepts } x \text{ in } y \text{ steps}\}$, and we are done. \dashv

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. \triangleleft

to do

4.27 Show that

$$E = \{\langle M \rangle \mid M \text{ is a DFA that accepts some string with more 1s than 0s}\}$$

is decidable. \triangleleft

to do

Chapter 5

Reducibility

Thm 5.1 (5.1) $H_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ halts on input } w\}$ is undecidable. \diamond

Proof. Assume that R decides H_{TM} . Construct TM

- $S =$ “On input $\langle M, w \rangle$
1. Simulate R on input $\langle M, w \rangle$. If M does not halt reject.
 2. Simulate M on w .
 3. Accept if M accepts and reject if M rejects.”

Then S decides A_{TM} . Hence contradiction. \dashv

Comment. This proof uses the *reducibility* method. If we have two statements A and B , then A is reducible to B means that to show A , it *suffices* to show B . Thus here A_{TM} is reducible to H_{TM} .

Thm 5.2 (5.2) $E_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$ is undecidable. \diamond

Proof. Assume that R decides E_{TM} and let W decide $\{w\}$. Construct TM

- $S =$ “On input $\langle M, w \rangle$
1. Construct TM M_w that accepts k iff both M and W accept k , where W is a decider for $\{w\}$.
 2. Simulate R on input $\langle M_w \rangle$.
 3. Accept if R rejects and reject if R accepts.”

Then S decides A_{TM} . Hence contradiction. \dashv

Comment. We are reducing A_{TM} to E_{TM} . Note that M_w works like M except that it always rejects $x \neq w$.

Defn 5.3 (5.17) A function $f : \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some Turing machine M , on every input w , *halts* with $f(w)$ on its tape. \diamond

Defn 5.4 (5.20) A is **mapping reducible** to B , written $A \leq_m B$, if there is a computable function f where $w \in A \Leftrightarrow f(w) \in B$. f is then called a **reduction** from A to B . \diamond

Thm 5.5 (5.22, 5.28) If $A \leq_m B$ and B is decidable (or recognizable), then A is decidable (or recognizable). \diamond

Its proof is omitted as trivial. And the following is its contraposition.

Cor 5.6 (5.23, 5.29) If $A \leq_m B$ and A is undecidable (or unrecognizable), then B is undecidable (or unrecognizable). \diamond

Mapping reducibility is useful to prove unrecognizability (often reducing $\overline{A_{\text{TM}}}$), while “reducibility to a decider” is useful to prove undecidability (often reducing A_{TM}).

Eg 5.7 In 5.2, we essentially constructed $f : \langle M, w \rangle \mapsto \langle M_w \rangle$, where $\langle M, w \rangle \in A_{\text{TM}} \Leftrightarrow \langle M_w \rangle \in \overline{E_{\text{TM}}}$. Thus $A_{\text{TM}} \leq_m \overline{E_{\text{TM}}}$ and $\overline{A_{\text{TM}}} \leq_m E_{\text{TM}}$. By 5.6 E_{TM} is unrecognizable. \diamond

Thm 5.8 (5.30) Let

$$EQ_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2) \}.$$

Then both EQ_{TM} and $\overline{EQ_{\text{TM}}}$ is unrecognizable. \diamond

Proof. Let $T_{M,w}$ be a TM that simulates M on w , T_{rej} a TM that always rejects and T_{acc} a TM that always accepts. Construct $f_1 : \langle M, w \rangle \mapsto \langle T_{M,w}, T_{rej} \rangle$. It follows that $\overline{A_{\text{TM}}} \leq_m EQ_{\text{TM}}$. Similarly, $f_2 : \langle M, w \rangle \mapsto \langle T_{M,w}, T_{acc} \rangle$ shows that $A_{\text{TM}} \leq_m \overline{EQ_{\text{TM}}}$, and we are done. \dashv

Defn 5.9 A **configuration** of a TM is a snapshot of a TM at some time, written triple (q, p, t) , with q, p and t representing the current state, the head position and the tape contents respectively. \diamond

Encode configuration (q, p, t) as the string t_1qt_2 where $t = t_1t_2$ and the head position is on the first symbol of t_2 .

Defn 5.10 An (accepting) **computation history** for TM M on input w is a sequence of configurations C_1, C_2, \dots, C_{acc} that M enters until it accepts. \diamond

Encode a computation history as the string $C_1\#C_2\#\cdots\#C_{acc}$, where each C_i is encoded as a string.

Defn 5.11 (5.6) A **linearly bounded automaton** (LBA) is a 1-tape TM that cannot move its head off the input portion of the tape. \diamond

Thm 5.12 (5.9) $A_{\text{LBA}} = \{\langle B, w \rangle \mid \text{LBA } B \text{ accepts } w\}$ is decidable. \diamond

Proof Idea. For inputs of length n , an LBA can have only $|Q| \times n \times |\Gamma|^n$ different configuration. Therefore, if an LBA runs for longer, it must repeat some configuration and will never halt. So for a decider for A_{LBA} , just simulate it for this many steps. \dashv

Thm 5.13 (5.10) $E_{\text{LBA}} = \{\langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset\}$ is undecidable. \diamond

Proof. Assume TM R decides E_{LBA} . Construct TM

- $S =$ “On input $\langle M, w \rangle$
1. Construct LBA $B_{M,w}$ that accepts its input x iff x is an accepting computation history for M on w .
 2. Simulate R on input $\langle B_{M,w} \rangle$.
 3. Accept if R rejects and reject if R accepts.”

Then S decides A_{TM} . Hence contradiction. \dashv

Comment. It is not immediately straightforward that $B_{M,w}$ is *capable* of doing that. But note that it obviously takes *finite* memory to check if a string is an accepting computation history for M on w , for that after confirming that a configuration is valid, whatever is before that configuration *does not* matter. So we may even add however much need finite memory to the right of the tape as input (in the form of \sqcup 's) to make sure our $B_{M,w}$ is able to do that.

Thm 5.14 (5.15) An instance of the *Post Correspondence Problem* (PCP) consists of a finite collection of dominos

$$P = \left\{ \left[\begin{array}{c} t_1 \\ b_1 \end{array} \right], \left[\begin{array}{c} t_2 \\ b_2 \end{array} \right], \dots, \left[\begin{array}{c} t_k \\ b_k \end{array} \right] \right\},$$

where each t_i, b_i are strings over some alphabet Σ . A *match* is a nonempty sequence of indices (repetitions permitted) i_1, i_2, \dots, i_ℓ such that

$$t_{i_1} t_{i_2} \cdots t_{i_\ell} = b_{i_1} b_{i_2} \cdots b_{i_\ell}.$$

Then $\text{PCP} = \{\langle P \rangle \mid P \text{ is an instance of PCP with a match.}\}$ is undecidable. \diamond

Proof. Assume TM R decides PCP . Construct TM

- $S =$ “On input $\langle M, w \rangle$
1. Construct PCP instance $P_{M,w}$ where a match corresponds to a computation history for M on w .
 2. Simulate R on input $\langle P_{M,w} \rangle$.
 3. Accept if R accepts and reject if R rejects.”

Then S decides A_{TM} . Hence contradiction.

Now we construct $P_{M,w}$, where *the only* match is a computation history for M on w .

1. Assume for simplicity that the match starts with a starting domino:

$$\left[\frac{u_1}{v_1} \right] = \left[\frac{\#}{\#q_0w_1w_2 \cdots w_n\#} \right]$$

2. Transition dominos: For each transition $\delta(q, a) = (r, b, R)$ (handle left moves similarly), include:

$$\left[\frac{qa}{br} \right]$$

along with identity dominos and blank-handling dominos:

$$\left[\frac{a}{a} \right], \quad \left[\frac{\#}{\#} \right], \quad \left[\frac{\#}{\sqcup\#} \right]$$

3. Accepting dominos: To terminate a valid computation history:

$$\left[\frac{aq_{\text{accept}}}{q_{\text{accept}}} \right], \quad \left[\frac{q_{\text{accept}}a}{q_{\text{accept}}} \right], \quad \left[\frac{q_{\text{accept}}\#\#}{\#} \right] \quad \dashv$$

Comment. Believe it or not, we are *forcing* the match to make a computation history. To resolve the assumption we can convert the strings a little bit to force starting using the first domino. See page 233 for this technical detail. Note that PCP is recognizable, for that the sequences of dominos is countable.

The proof of 3.10 was by showing that A_{TM} is reducible to

$$D = \{ \langle p \rangle \mid \text{polynomial } p(x_1, x_2, \dots, x_k) = 0 \text{ has an integer root} \}.$$

Similar to 5.14, on input $\langle M, w \rangle$, we construct a polynomial with several variables, where to get an integer root of that polynomial, one of the variables has to be assigned some encoding of a computation history of the TM of M on w , and the other are to help to decode that encoding.

Thm 5.15 $ALL_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^* \}$ is undecidable. \diamond

Proof. Assume TM R decides ALL_{CFG} . Construct TM

$S =$ “On input $\langle M, w \rangle$

1. Construct PDA $B_{M,w}$ that accepts its input x iff x is *not* an accepting computation history for M on w .
2. Simulate R on input $\langle B_{M,w} \rangle$.
3. Accept if R rejects and reject if R accepts.”

Then S decides A_{TM} . Hence contradiction.

$B_{M,w}$ is constructed so that it *nondeterministically* pushes some C_i and pop to compare with C_{i+1} (with M in its mind) and accepts if some step is invalid, or the input starts wrongly or the ending configuration is not accepting. We reverse the even-indexed C_i to facilitate the comparing using stack. \dashv

Comment. Similarly ALL_{LBA} is undecidable. Note that PDAs are not capable of checking if strings are computation histories, which resonates with 4.6.

The computation history method is useful for showing the undecidability of problems involving testing for the existence of some object.

Exercises and Problems

5.1 Show that

$$EQ_{CFG} = \{ \langle G, H \rangle \mid G, H \text{ are CFGs and } L(G) = L(H) \}$$

is undecidable. \triangleleft

to do

5.21 Question to fill in. \triangleleft

to do

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

(\Leftarrow) This is trivial by 5.5.

(\Rightarrow) Say that $L(R) = A$, then $f : w \mapsto \langle R, w \rangle$, where $w \in A$ shows that $A \leq_m A_{TM}$.

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

(\Leftarrow) 0^*1^* is decidable, so this is trivial.

(\Rightarrow) This is also trivial in a sense: Say that R decides A . We modify R so that it leaves 01 on the tape if accepts and otherwise 10.

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

to do

No. In that case both B and \overline{B} would both be recognizable, thus both decidable.

5.26 Question to fill in. \triangleleft

to do

Chapter 6

Advanced Topics in Computability Theory

Lem 6.1 (6.1) There is a computable function $q : w \mapsto \langle P_w \rangle$, where $w \in \Sigma^*$ and P_w is a TM that prints out w and then halts. \diamond

Rmk 6.2 We describe in spirit and not formally a TM *SELF*, that halts with $\langle SELF \rangle$ on the tape. *SELF* will be the combination of two TMs *A* and *B*. $A = P_{\langle B \rangle}$. And *B* on input $\langle M \rangle$ computes $q(\langle M \rangle)$, combines the result with $\langle M \rangle$ to make a complete TM, and finally prints the description of it and halt. \diamond

An English implementation would be:

Print out two copies of the following, the second one in quotes:

"Print out two copies of the following, the second one in quotes:"

The second line is *A* and the first *B*. *A* provides a description of *B*. *B* knows what itself (the instructions, not a string in quotes) is upon that description and computes *A* that will generate that description. Finally *B* prints them together. Consider that *B* would be fixed when deciding *A*, adding quotes to whatever it is.

Thm 6.3 Recursion Theorem Let TM *T* compute $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. Then there is a TM *R* that computes $r : \Sigma^* \rightarrow \Sigma^*$, where for all $w \in \Sigma^*$, $r(w) = t(\langle R \rangle, w)$. \diamond

Proof Idea. Similar to 6.2, we describe without formality a TM *R* which is the combination of three TMs *A*, *B* and *T*, where *T* is given. $A = P_{\langle BT \rangle}$. *B* on input $\langle M \rangle$ computes $q(\langle M \rangle)$ and combines the result with $\langle B \rangle$ and $\langle T \rangle$ into a single TM. Finally *T* acts however it was intended to. \dashv

Take 6.3 in the following fashion: consider a TM T that computes $t(\langle M \rangle, w)$ (indicating, if $\langle M \rangle$ is the description of a TM, then compute within it and w and otherwise just print error) then there magically exists an R that computes $r(w) = t(\langle R \rangle, w)$. That means if we want to construct a TM that computes upon w and some TM M , M can be exactly the TM we are to construct!

Second Proof of 4.12. Assume that H decides A_{TM} . Construct TM

- $B =$ “On input $\langle w \rangle$
1. Obtain via 6.3 $\langle B \rangle$.
 2. Simulate H on $\langle B, w \rangle$.
 3. Do the opposite of what H says.”

Thus H fails to decide A_{TM} . Hence contradiction and we are done. \dashv

Thm 6.4 (6.7) $MIN_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a minimal TM under some encoding}\}$ is not recognizable. \diamond

Proof. Assume that E enumerates MIN_{TM} . Construct TM

- $C =$ “On input $\langle w \rangle$
1. Obtain via 6.3 $\langle C \rangle$.
 2. Run E until some TM D appears that is longer than C .
 3. Simulate D on input w .”

Hence contradiction and we are done. \dashv

Comment. MIN_{TM} is very special, in that any infinite subset of it is not recognizable.

Thm 6.5 Let $t : \Sigma^* \rightarrow \Sigma^*$ be a computable function. Then there is a TM F such that $t(\langle F \rangle)$ describes a TM equivalent to F . \diamond

Proof. Construct

- $F =$ “On input $\langle w \rangle$
1. Obtain via 6.3 $\langle F \rangle$.
 2. Compute $t(\langle F \rangle)$ to obtain the description of a TM G .
 3. Simulate G on input w .”

Thus G will be equivalent to F . \dashv

Comment. F is a fixed point w.r.t. t .

Thm 6.6 Gödel’s First Incompleteness Theorem In any reasonable formal system, some true statements are not provable. \diamond

Proof Idea. Suppose otherwise. If we can always prove (that is, to decidably check the truthfulness of something) $\langle M, w \rangle \in \overline{A_{TM}}$ when it is true, then $\overline{A_{TM}}$ is recognizable, and hence contradiction. \neg

Comment. To prove is essentially the same as to compute. There are things we can not recognize, then sure enough facts exists that we can not verify.

Consider the statement “This statement is unprovable”. It has to be true, thus unprovable.

Eg 6.7 Let ϕ be the statement $\langle R, 0 \rangle \in \overline{A_{TM}}$ where R is the following TM:

- $R =$ “On any input
1. Obtain via 6.3 $\langle R \rangle$ and use it to obtain ϕ .
 2. Enumerate all proofs and accept if one is found for ϕ .

A proof or the falsity of ϕ each leads to a contradiction. Thus we have an example statement that is true but without proof. \diamond

Exercises and Problems

6.1 Give an example in the spirit of the recursion theorem of a program in a real programming language that prints itself out. \triangleleft

```

1 #include <stdio.h>
2 char *program = "#include <stdio.h>%cchar *program = %c%s%c;%c
   cint main()%c%c          printf(program, 10, 34, program,
   34, 10, 10, 10, 10, 10, 10);%c          return 0;%c}%c";
3 int main()
4 {
5     printf(program, 10, 34, program, 34, 10, 10, 10, 10,
6         10, 10);
7     return 0;
8 }
```

Listing 6.1: A piece of C that prints itself.

10 is the ASCII code for NEWLINE and 34 for ". To relate to 6.2, consider that A is the string `*program`. B , the “printer”, upon seeing a string, combines it with what ever will generate it. So the first `program` in line 5 is what B sees, and the second is what generates the string. The combination is done in a nested fashion.

6.11 Question to fill in.

◁

to do