

# Problems in Foundations of Computer Science

Dan Dougherty

September 8, 2009 – 22: 49

Some problems, as indicated, have been borrowed from the textbooks by Hopcroft, Motwani and Ullamn, Kozen, and Sipser

- John Hopcroft, Rajeev Motwani, and Jeffry Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2007.
- Dexter Kozen. *Automata and Computability*. Springer-Verlag, New York, NY, 1997.
- Michael Sipser. *Introduction to the Theory of Computation*. PWS, Boston, MA, 1997.

## Notation

There are a few notations for standard concepts which differ according to which author one reads. We use the following notation in these problems.

- The complement of set  $S$  is denoted  $\bar{S}$ . The difference between sets  $X$  and  $Y$  is denoted  $X - Y$ .
- The empty string is denoted  $\lambda$ . (Some authors use  $\epsilon$ , some authors use  $\lambda$ .)
- The reversal of a string  $w$  is denoted  $w^R$ .

## Contents

<b>1</b>	<b>Strings, languages, regular expressions</b>	<b>3</b>
<b>2</b>	<b>Context-free grammars and languages</b>	<b>7</b>
<b>3</b>	<b>Ambiguity</b>	<b>11</b>
<b>4</b>	<b>Normal forms for CFGs.</b>	<b>14</b>
<b>5</b>	<b>Closure properties of context-free languages</b>	<b>16</b>
<b>6</b>	<b>Pushdown automata and parsing.</b>	<b>18</b>
<b>7</b>	<b>Proving languages not context-free</b>	<b>19</b>
<b>8</b>	<b>Decision problems for CFGs</b>	<b>21</b>
<b>9</b>	<b>Finite Automata</b>	<b>23</b>
<b>10</b>	<b>Proving languages not regular</b>	<b>28</b>
<b>11</b>	<b>Closure properties of regular languages</b>	<b>30</b>
<b>12</b>	<b>Decision problems about regular languages</b>	<b>32</b>
<b>13</b>	<b>The Myhill-Nerode theorem and minimization of automata</b>	<b>35</b>
<b>14</b>	<b>Turing machines</b>	<b>41</b>
<b>15</b>	<b>Undecidability</b>	<b>43</b>
<b>16</b>	<b>Reducibility</b>	<b>47</b>
<b>17</b>	<b>Post's Correspondence Problem</b>	<b>51</b>
<b>18</b>	<b>P and NP</b>	<b>53</b>
<b>19</b>	<b>Poly-time Reducibility</b>	<b>55</b>
<b>20</b>	<b>NP-completeness of satisfiability</b>	<b>57</b>
<b>21</b>	<b>More on NP-completeness</b>	<b>58</b>

# 1 Strings, languages, regular expressions

**Problem 1.1.** What's the difference between an alphabet and a language?

**Problem 1.2.** What's the difference between a string and a word?

**Problem 1.3.** Can an element of a language  $A \subseteq \Sigma^*$  be infinite?

**Problem 1.4.** Can a language be infinite?

**Problem 1.5.** Does it make sense to talk about taking the union of two languages?

**Problem 1.6.** Does it make sense to talk about taking the union of two strings?

**Problem 1.7.** Does it make sense to talk about taking the concatenation of two strings?

**Problem 1.8.** Suppose  $A$  and  $B$  are languages. We do in fact sometimes speak of the "concatenation"  $AB$  of these languages. What does this have to do with the "concatenation" of two strings??

**Problem 1.9.** Let  $A = \{aa, bb\}$  and  $B = \{\lambda, b, ab\}$

1. List the strings in the set  $AB$
2. List the strings in the set  $BB$
3. How many strings of length 6 are there in  $A^*$ ?
4. List the strings in  $B^*$  of length 3 or less.
5. List the strings in  $A^*B^*$  of length 4 or less.

**Problem 1.10.** Give an example of a language  $A$  such that  $A^* = A$ . OK,  $A = \Sigma^*$  is an easy example. Find another example!

Can there be a *finite*  $A$  satisfying  $A^* = A$ ?

Is it possible to have  $A^* \subset A$ , where  $\subset$  means *proper* subset?

**Problem 1.11.** For each part decide if there can be a language  $A$  with the given property. If so, give an example, if not, explain why not.

1.  $AA = A$ .
2.  $AA \subseteq A$  but  $AA \neq A$ .
3.  $AA \not\subseteq A$
4.  $A \subseteq AA$  but  $AA \neq A$ .

**Problem 1.12** (Decision problems as languages). For each decision problem below, indicate the corresponding language (as a subset of  $\Sigma^*$ , where  $\Sigma$  depends on the problem). Since most of these correspond to an infinite language you won't be able to actually *list* the language, so just write down enough strings in the language to make it clear that you know what the language is.

1. INPUT: a string  $w$  over  $\Sigma = \{0, 1\}$   
QUESTION: does  $w$  have even length?
2. INPUT: a string  $w$  over  $\Sigma = \{0, 1\}$   
QUESTION: does  $w$ , in binary notation, code an odd number?
3. INPUT: a string  $w$  over the ASCII alphabet  
QUESTION: is  $w$  a legal Scabble word?
4. INPUT: a string  $w$  over the ASCII alphabet  
QUESTION: is  $w$  a syntactically correct English sentence?
5. INPUT: a string  $w$  over the ASCII alphabet  
QUESTION: is  $w$  a syntactically correct Java program?

**Problem 1.13.** Let  $A$  and  $B$  be languages. Suppose you have available an algorithm  $\mathcal{D}_A$  which, given any string  $w$ , will answer “yes” or “no” as to whether  $w \in A$ . Further suppose you have an algorithm  $\mathcal{D}_B$  will can test membership in  $B$  in the same way.

Give pseudocode for an algorithm  $\mathcal{E}$  to solve the following problem.

INPUT: a string  $w$

QUESTION: is  $w \in AB$ ?

**Problem 1.14.** Let  $A$  be a language and suppose you have available an algorithm  $\mathcal{D}_A$  which, given any string  $w$ , will answer “yes” or “no” as to whether  $w \in A$ .

Give pseudocode for an algorithm  $\mathcal{E}$  to solve the following problem.

INPUT: a string  $w$

QUESTION: is  $w \in A^*$ ?

**Problem 1.15.** Prove or disprove

- $A(B \cup C) = AB \cup AC$
- $A(B \cap C) = AB \cap AC$

**Problem 1.16.** Let  $A$  and  $B$  be arbitrary languages. Prove:

1.  $(A \cup B)^* = A^*(BA^*)^*$
2.  $(A \cup B)^* = B^*(AB^*)^*$

**Problem 1.17.** A language  $L$  is said to be *idempotent* if  $LL \subseteq L$ . In this problem you will prove that (modulo a detail about  $\lambda$ ) for any  $A$ ,  $A^*$  is the smallest idempotent language containing  $A$ .

Specifically, prove the following hold for any language  $A$ .

1. The language  $A^*$  is idempotent.
2. If  $B$  is any idempotent language with  $A \subseteq B$  and  $\lambda \in B$ , then  $A^* \subseteq B$ .

**Problem 1.18** (*A useful equation*). Let  $A$ ,  $B$  and  $C$  be arbitrary languages. Prove:

$$\text{if } B \cup AC \subseteq C \text{ then } A^*B \subseteq C$$

This yields a useful technique for proving that the language denoted by one regular expression is a subset of the language denoted by another.

**Problem 1.19.** The following result is known as Arden's Lemma<sup>1</sup>

**Theorem 1.20.** *Let  $A$  be a language such that  $\lambda \notin A$ . Then the equation  $X = AX + B$  has the unique solution  $X = A^*B$ .*

Verify this for yourself in several cases (make up sample  $A$  and  $B$  and check the solution).

Prove Arden's Lemma.

Suppose we removed the condition " $\lambda \notin A$ ". Would it still be true that  $A^*B$  is a solution to the equation? Give an example to show why the condition is needed.

**Problem 1.21.** What's the difference between a regular expression and a language?

**Problem 1.22.** Can a regular expression be infinite?

**Problem 1.23.** *Reading regular expressions I*

For each of the following regular expressions, give two strings which are in the language they define, and give two strings which are not in the language they define. Assume that the alphabet  $\Sigma$  is always  $\{a, b\}$ .

1.  $a^*b^*$
2.  $a(ba)^*b$
3.  $a^* + b^*$
4.  $(aaa)^*$
5.  $aba + bab$
6.  $(\lambda + a)b$

**Problem 1.24.** *Reading regular expressions II*

For each of the following regular expressions  $\alpha$  give the shortest non- $\lambda$  string in  $L(\alpha)$ . (There may be more than one answer to a given question...)

---

<sup>1</sup> D.N. Arden. Delayed logic and finite state machines. In Theory of Computing Machine Design, pp.1-35, U. of Michigan Press, Ann Arbor. 1960.

1.  $ba + (a + bb)a^*b$ .
2.  $(aa + bb + (ab + ba)(aa + bb)^*(ab + ba))^*$
3.  $((aa + bb)^* + (aab + bba)^*)^*$

**Problem 1.25.** (from Kozen) *Algebraic laws for regular expressions*

For each pair of regular expressions below, say whether they are equal (i.e., denote the same language). if so, give a proof. If not, give an example of a string in one but not the other.

1.  $(0 + 1)^*$  and  $0^* + 1^*$
2.  $0(120)^*12$  and  $01(201)^*2$
3.  $\emptyset^*$  and  $\lambda^*$
4.  $(0^*1^*)^*$  and  $(0^*1)^*$
5.  $(01 + 0)^*0$  and  $0(10 + 0)^*$

**Problem 1.26.** *Closure under reverse*

Prove that if  $L$  is regular then so is  $L^R$  (the set of reversals of string in  $L$ ), using regular expressions. That is, suppose you are given a regular expression denoting  $L$  and show how to write a regular expression denoting  $L^R$ . Your argument is a proof by induction over regular expressions. This is a much easier proof than using automata.

## 2 Context-free grammars and languages

**Problem 2.1.** What's the difference between a grammar and a language?

**Problem 2.2.** (from Kozen) Consider the following grammar  $G$ :

$$\begin{aligned}S &\rightarrow ABS \mid AB, \\A &\rightarrow aA \mid a \\B &\rightarrow bA\end{aligned}$$

Which of the following strings are in  $L(G)$ ? Prove your answers.

1.  $aabaab$
2.  $aaaaba$
3.  $aabbaa$
4.  $abaaba$

**Problem 2.3.** *Regular languages are context-free I*

In this problem you show that every regular language is context-free.

Suppose  $M = (Q, \Sigma, \delta, q_0, F)$  is a DFA. Let  $G$  be the following context-free grammar:

- The non-terminals of  $G$  are the states of  $Q$ .
- The set of terminals of  $G$  is  $\Sigma$ .
- The start symbol of  $G$  is  $q_0$
- The production of  $G$  are given by
  - whenever  $\delta(q, a) = r$  in  $M$ , we have the production

$$q \rightarrow ar$$

in  $G$ , and

- for each accepting state  $q \in F$  of  $M$  we have the production

$$q \rightarrow \lambda$$

in  $G$

Prove that  $L(G) = L(M)$ .

Does it matter that we started with a DFA rather than an NFA above?

Recall the techniques we used, based on Arden's Lemma, for constructing a regular expression corresponding to a DFA or NFA. How does this construction compare to the equations we built as the first step in that process?

Take several examples of finite automata and do this construction on them...

**Proving correctness of grammars** In each of the next few problems you are asked to prove that a certain CFG generates a certain language.

In order to prove that CFG  $G$  generates language  $L$ , you must do two things:

1. Prove that  $L(G) \subseteq L$ .

To do this you typically use induction on the length of a derivation in  $G$ .

2. Prove that  $L \subseteq L(G)$ .

To do this you typically use induction on the length of a word in  $L$ .

Often the proof of (ii) is significantly harder than the proof of (i). Sometimes what you have to do is find some way to characterize the strings in  $L$  that helps you put the kind of “structure” on them that the grammar induces.

**Problem 2.4.** *Equal as and bs*

Let  $E$  be the set of strings over  $\{a, b\}$  with an equal number of  $as$  and  $bs$ .

Let  $G$  be the following CFG:

$$S \rightarrow \lambda \mid SS \mid aSb \mid bSa$$

Prove that  $L(G)$  is  $E$ .

**Problem 2.5.** *More as than bs*

Let  $G$  be the following grammar.

$$S \rightarrow aS \mid aSbS \mid \lambda$$

Prove that  $L(G)$  is the set of strings  $w$  over  $\{a, b\}$  such that every prefix of  $w$  has at least as many  $as$  as  $bs$ .

**Problem 2.6.** *Generating palindromes*

For this problem: let  $G$  be the grammar

$$S \rightarrow aSa \mid bSb \mid \lambda$$

Prove that  $L(G)$  is the set of even-length palindromes over  $\{a, b\}$ . (A palindrome is a string which is equal to its own reversal.)

*Hint:* Let  $E$  denote the set of even-length palindromes over  $\{a, b\}$ . As suggested above, you need to:

1. Prove that  $E \subseteq L(G)$ , that is, every palindrome is generated by the grammar. (Induct on the length of the palindrome.)
2. Prove that  $L(G) \subseteq E$ , that is, every string generated by the grammar is a palindrome. (Induct on the length of the derivation.)

**Problem 2.7.** *Balanced parentheses*

Let  $\Sigma$  be the alphabet  $\{a, b\}$ . But think of these as standing for the left and right parentheses symbols, respectively (it's kind of confusing to try to read long strings of parentheses). If  $u$  is a string, let us use the notation  $\#a(u)$  to stand for the number of occurrences of the symbol  $a$  in  $u$ , and of course  $\#b(u)$  is the number of occurrences of  $b$  in  $u$ .

Consider the language *Bal* over  $\Sigma$  defined as follows: a string  $w$  is in *Bal* if

- $\#a(w) = \#b(w)$ , and
- for every initial substring  $u$  of  $w$ ,  $\#a(u) \geq \#b(u)$ .

Another way to express the conditions above is to define, for any string  $u$ , the quantity  $\#a(u) - \#b(u)$ , and say that as  $u$  ranges over larger and larger initial substrings of  $w$ , this quantity never goes below 0 and must equal 0 when  $u$  reaches  $w$  itself.

Convince yourself that *Bal* is the language which we intuitively describe as strings of “balanced” parentheses.

Now let  $G$  be the following grammar.

$$S \rightarrow SS \mid aSb \mid \lambda$$

Prove that  $G$  generates the set *Bal* of “balanced parentheses”.

**Problem 2.8.** *Equal as and bs again*

Let  $E$  be the set of strings over  $\{a, b\}$  with an equal number of *as* and *bs*.

Let  $G$  be the following grammar.

$$S \rightarrow aSbS \mid bSaS \mid \lambda$$

Prove that  $L(G)$  is  $E$ .

*Yes, this is the same language as in problem 2.4. Very different grammars can define the same language...*

*Hint:* As in problem 2.7 consider the quantity  $\#a(u) - \#b(u)$  as  $u$  ranges over prefixes of a word  $w$ . Characterize the strings in the language of equal numbers of *as* and *bs* in terms of this function. Then you can proceed as in problem 2.7.

**Problem 2.9.** Let  $L$  be a context-free language over a one-letter alphabet  $\Sigma$ . prove that  $L$  is regular.

**Problem 2.10.** Consider the language  $L$  over  $\Sigma = \{a, b\}$ :  $L = \{x \mid x \text{ is not of the form } ww\}$ . Show that  $L$  is context-free.

*Hint.* This is not easy. The following exercises gradually build up to a solution of that problem.

1. Write a CFG for  $X = \{uav \mid u, v \in \{a, b\}^*, |u| = |v|\}$ .

2. Observe that a CFG for  $Y = \{ubv \mid u, v \in \{a, b\}^*, |u| = |v|\}$  is a trivial variation on the one for  $X$ .
3. Describe the languages  $XY$  and for  $YX$ . Observe that it is now easy to write a CFG for the concatenations  $XY$  and for  $YX$ .
4. Now note that a string is in  $\{x \mid x \text{ is not of the form } ww\}$  precisely if

- $|x|$  is odd, OR
- $x$  looks like

$$y_1 \cdots y_{i-1} a y_{i+1} \cdots y_m z_1 \cdots z_{i-1} b z_{i+1} \cdots z_m$$

OR

- $x$  looks like

$$y_1 \cdots y_{i-1} b y_{i+1} \cdots y_m z_1 \cdots z_{i-1} a z_{i+1} \cdots z_m$$

5. Put the pieces together.

### 3 Ambiguity

**Problem 3.1.** Define:  $G$  is an ambiguous context-free grammar.

**Problem 3.2.** Define:  $L$  is an inherently ambiguous context-free language.

**Problem 3.3.** Does it make sense to talk about a language being ambiguous?

**Problem 3.4.** Does it make sense to talk about a grammar being inherently ambiguous?

**Problem 3.5.** Does it make sense to talk about a parse tree being ambiguous?

**Problem 3.6.** *Basic ambiguity*

For each grammar, decide whether it is ambiguous or not. If it is, prove it (by exhibiting a string with two different parse trees). Then find an unambiguous grammar generating the same set of strings.

*Hint.* Remember that it is not true that **every** ambiguous grammar can be replaced by an unambiguous grammar generating the same strings. But in this problem you may be confident that the ambiguous grammars below do have unambiguous counterparts.

1.  $S \rightarrow aS \mid Sa \mid \lambda$
2.  $S \rightarrow aS \mid Sa \mid b$
3.  $S \rightarrow aSb \mid aSbb \mid \lambda$
4.  $S \rightarrow aSbS \mid \lambda$
5.  $S \rightarrow aS \mid aSbS \mid \lambda$
6.  $S \rightarrow aSb \mid SS \mid \lambda$
7.  $S \rightarrow AbB$   
 $A \rightarrow aA \mid \lambda$   
 $B \rightarrow aB \mid bB \mid \lambda$
8.  $S \rightarrow AaSbB \mid \lambda$   
 $A \rightarrow aA \mid a$   
 $B \rightarrow bB \mid \lambda$

**Problem 3.7.** *Ambiguity and arithmetic I*

The grammars in this problem and the next are slight variations on the grammars described in class, simplified to avoid the distracting verbosity there in generating the identifiers,  $I$ .

Let  $G$  be the following grammar.

$$\begin{aligned} E &\rightarrow E + E \mid E * E \mid I \\ I &\rightarrow a \mid b \mid c \end{aligned}$$

Let  $G^*$  be the following grammar.

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * I \mid I \\ I &\rightarrow a \mid b \mid c \end{aligned}$$

1. The grammar  $G$  is ambiguous not only because the precedence of the operators  $+$  and  $*$  is not determined, but also because the associativity of the individual operators  $+$  and  $*$  is not determined. The grammar  $G^*$  is unambiguous, and ensures that  $+$  and  $*$  are parsed as left-associative operators.

Change the grammar of Figure  $G^*$  so that it is still unambiguous and generates the same strings, but with  $+$  parsed as being right-associative and  $*$  as left-associative.

2. Suppose we add a new terminal “ $-$ ” to the language, intended to denote subtraction. Suppose that we enrich the grammar  $G^*$  to include subtraction as a binary operator, so that if  $E_1$  and  $E_2$  are expressions, then so is  $E_1 - E_2$ .

Enrich the grammar  $G^*$  to get an unambiguous grammar generating arithmetic expressions including subtraction. Your grammar should make subtraction left-associative and at the same level of precedence as  $+$ .

3. Suppose we add a new terminal  $\uparrow$  to the language, intended to denote exponentiation. That is, if  $U$  and  $W$  are expressions, then  $U \uparrow W$  is to be an expression, intuitively denoting “ $U$  raised to the power  $W$ ”.

Enrich the grammar  $G^*$  to get an unambiguous grammar generating arithmetic expressions including exponentiation. Your grammar should induce the usual rules of precedence involving exponentiation, that is, that  $\uparrow$  binds more tightly than any other operator.

Note that exponentiation is not associative. Does repeated exponentiation conventionally group to the right or to the left? For example, what is the conventional value of  $2 \uparrow 3 \uparrow 2$ ? Be sure your grammar induces the correct grouping!

4. Extend the grammar  $G^*$  to allow the comparison operators  $=$  and  $<$ . They should all be left-associative and at the same level of precedence, below that of  $+$ . (That is, an expression like  $a + b = c$  should have a parse tree with the  $=$  appearing higher than the  $+$ .)

**Problem 3.8.** *Ambiguity and arithmetic II (from HMU)*

The following grammar generates *prefix* arithmetic expressions over  $+$ ,  $*$ , and  $-$ .

$$\begin{aligned} E &\rightarrow +EE \mid *EE \mid -EE \mid I \\ I &\rightarrow a \mid b \mid c \end{aligned}$$

Build a parse tree for the term  $+* - abab$ .

Is this grammar ambiguous? Explain why or why not.

**Problem 3.9.** *Ambiguity and Boolean logic*

Here is a grammar  $G$  generating formulas of propositional logic over the alphabet  $\Sigma = \{\wedge, \vee, \equiv, \neg, (, ), p, q, r, s, \dots\}$

$$\begin{aligned} B &\rightarrow B \wedge B \mid B \vee B \mid B \equiv B \mid \neg B \mid (B) \mid I \\ I &\rightarrow p \mid q \mid r \mid s \mid \dots \end{aligned}$$

This grammar is ambiguous.

Write an unambiguous grammar  $G'$  generating the same strings, enforcing the rules that

- the binary operators are left-associative,
- $\neg$  has highest precedence, followed by  $\wedge$ , then  $\vee$ , then  $\equiv$  (but parentheses can override these precedences).

For example, the parse tree for  $p \equiv \neg q \wedge q \vee r$  should have  $\equiv$  as the operator at the top of the tree, with  $\vee$  at the next level,  $\wedge$  at the level below that, and  $\neg$  applied to the symbol  $q$  alone.

**Problem 3.10.** *Ambiguity and programming languages*

Consider the following grammar over the alphabet

$$\Sigma = \{\text{if, then, else, statement, condition}\}$$

$$\begin{aligned} S &\rightarrow \text{if } C \text{ then } S \\ S &\rightarrow \text{if } C \text{ then } S \text{ else } S \\ S &\rightarrow \text{statement} \\ C &\rightarrow \text{condition} \end{aligned}$$

If we interpret  $S$  as a variable standing for statements in a programming language and we interpret  $C$  as a variable standing for boolean conditions, then the above grammar generates “if-then” and “if-then-else” statements. (of course in a grammar specifying more details we would expand statement and condition...)

1. Show that this grammar is ambiguous.
2. This ambiguity is known as the “dangling else” problem; explain what this ambiguity means in terms in the meaning of real programming language statements.
3. The convention in most programming languages is that a statement which is ambiguous in the sense that you discovered above should be interpreted using the following rule: *an else is always paired with the closest preceding if if that doesn't already have an else paired with it.* Give an unambiguous grammar which enforces this rule.

## 4 Normal forms for CFGs.

**Problem 4.1.** Give a grammar with no  $\lambda$ - or unit productions generating the set  $L(G)$ , where  $G$  is the following grammar

$$\begin{aligned} S &\rightarrow aSbb \mid ST \mid c \\ T &\rightarrow bTaa \mid S \mid \lambda \end{aligned}$$

Note that  $L(G)$  does not contain  $\lambda$ .

**Note added Feb 9** The original grammar had a bug: it generated no terminal strings! I have fixed it here, by adding  $S \rightarrow c$ ; the solution below is for the repaired grammar.

**Problem 4.2.** Build Chomsky normal form grammars equivalent to the grammars below. (None of these grammars generates  $\lambda$ .)

1.

$$\begin{aligned} S &\rightarrow XY \\ X &\rightarrow aXbb \mid abb \\ Y &\rightarrow cY \mid c \end{aligned}$$

2.

$$\begin{aligned} S &\rightarrow aSa \mid aBa \\ B &\rightarrow Bb \mid b \end{aligned}$$

3.

$$\begin{aligned} S &\rightarrow ASc \mid AScc \mid ABc \mid ABcc \\ A &\rightarrow Aa \mid a \\ B &\rightarrow bB \mid b \end{aligned}$$

4.

$$S \rightarrow aSa \mid bSb \mid a \mid b$$

**Problem 4.3.** *Building Chomsky Normal Form (from Kozen)*

Give grammars in Chomsky Normal Form for each of the following languages.

1.  $\{a^n b^{2n} c^k \mid k, n \geq 1\}$

2.  $\{a^n b^k a^n \mid k, n \geq 1\}$

3.  $\{a^k b^m c^n \mid k, m, n \geq 1, 2k \geq n\}$

**Problem 4.4.** from Sudkamp: Exercises from Chapter 4, numbers 18 through 23, and exercise 24 (a).

## 5 Closure properties of context-free languages

### Problem 5.1. Union and CFGs

Let  $G_1$  and  $G_2$  be context-free grammars, each with the same terminal alphabet  $\Sigma$ , but with disjoint sets of non-terminals  $V_1$  and  $V_2$ . Show how to build a grammar  $G$  such that  $L(G) = L(G_1) \cup L(G_2)$ . Why did we need to assume that  $V_1$  and  $V_2$  were disjoint?

### Problem 5.2. Union and CFLs

Show by a direct construction (inspired by problem 5.1) that the context-free languages are closed under union. That is, show that if  $G_1$  and  $G_2$  be context-free grammars with the same terminal alphabet  $\Sigma$  then there is a context-free grammar  $G$  such that  $L(G) = L(G_1) \cup L(G_2)$ .

There is a subtlety here: note that we do *not* assume in the current problem statement that  $G_1$  and  $G_2$  have disjoint sets of non-terminals. Make sure you deal with this annoyance.

### Problem 5.3. More CFL closure properties

In the same manner as problem 5.2, show that the context-free languages are closed under

1. concatenation,
2. asterate (that is, \*-closure)
3. reversal.

### Problem 5.4. CFL intersection

Show that the context-free languages are *not* closed under intersection.

### Problem 5.5. CFL complement

Show that the context-free languages are *not* closed under complement.

### Problem 5.6. CFL and regular closure

Suppose  $L_1$  and  $L_2$  are context-free languages and suppose that  $R$  is a regular language. For each of the following languages, say whether it is guaranteed to be context-free. If your answer is “yes” prove it; If your answer is “no” give specific languages which comprise a counterexample. (Recall that  $A - B$  abbreviates  $A \cap \overline{B}$ .)

1.  $L_1 - R$
2.  $R - L_1$
3.  $L_1 - L_2$

**Problem 5.7.** Let  $R$  be a regular language and let  $N$  be a language which is *not* context-free. Let  $X = R \cup N$ .

1. Show by means of examples that we cannot conclude whether  $X$  is context-free or not.

2. Prove that if  $R \cap N = \emptyset$  then  $X$  is not context-free.

**Problem 5.8.** *CFL subset*

1. Prove or disprove: If  $L$  is context-free and  $K \subseteq L$  then  $K$  is context-free.
2. Prove or disprove: If  $L$  is context-free and  $L \subseteq K$  then  $K$  is context-free.

## 6 Pushdown automata and parsing.

### Generalities:

**Problem 6.1.** Be able to: given a PDA  $P$  and a string  $w$  show the sequence of IDs arising in a computation of  $P$  on  $w$ .

**Problem 6.2.** Be able to: given a grammar  $G$  build a PDA  $M$  such that  $L(M) = L(G)$ .

**Problem 6.3.** The language  $D = \{w \mid w \text{ can be written as } xx \text{ for some string } x\}$  is not context-free (over the alphabet  $\Sigma = \{a, b\}$ ).

Show that  $\bar{D}$ , the complement of  $D$ , is context-free.

As a hint, note that  $\bar{D}$  can be written as

$$\begin{aligned} & \{w \mid \text{the length of } w \text{ is odd}\} \\ & \cup \{xax'by' \mid x, y, x', y' \in \Sigma^*, |x| = |x'|, |y| = |y'|\} \\ & \cup \{xbyx'ay' \mid x, y, x', y' \in \Sigma^*, |x| = |x'|, |y| = |y'|\} \end{aligned}$$

Try building a PDA accepting this language.

## 7 Proving languages not context-free

**Problem 7.1.** *Using the pumping lemma.*

Prove that

$$L = \{a^n b^n c^i \mid i \leq n\}$$

is not context-free.

**Problem 7.2.** *More pumping*

Show that each of the following languages is not context-free.

1.  $\{a^n b^m \mid n^2 \geq m\}$
2.  $\{a^n b^m \mid n^2 \leq m\}$
3.  $\{a^n b^m \mid n \geq m^2\}$
4.  $\{a^n b^m \mid n \leq m^2\}$

**Problem 7.3.** *Taxonomy*

For each of the following languages, tell whether it is

- regular
- context-free, but not regular
- not context-free

In each case, prove your answer.

What does this mean? To prove a language regular you can exhibit a finite automaton (any flavor) or a regular expression, possibly in conjunction with using some of the known closure properties. To prove a language context-free you can exhibit a PDA or grammar, possibly in conjunction with using some of the known closure properties. To prove a language *not* regular or context-free, you can use a pumping lemma argument.

1.  $\{w \in \{a, b, c\}^* \mid w \text{ has an equal number of } a\text{s, } b\text{s and } c\text{s}\}$
2.  $\{a^n \mid n \text{ is a power of } 2\}$
3.  $\{w \in \{0, 1\}^* \mid w \text{ represents a power of } 2 \text{ in binary}\}$
4.  $\{a^n b^m \mid n = m\}$
5.  $\{a^n b^m \mid n \neq m\}$
6.  $\{a^n b^m \mid n \leq m\}$
7.  $\{a^n b^m c^k d^l \mid n = m \text{ or } k = l\}$

8.  $\{a^n b^m c^k d^l \mid n = m \text{ and } k = l\}$
9.  $\{a^n b^m c^k d^l \mid n = k \text{ or } m = l\}$
10.  $\{a^n b^m c^k d^l \mid n = k \text{ and } m = l\}$
11.  $\{a^n b^m c^k d^l \mid n > k \text{ or } m > l\}$
12.  $\{a^n b^m c^k d^l \mid n > k \text{ and } m > l\}$
13. The set of all strings  $w$  over  $\{a, b\}$  satisfying:  $w$  has an equal number of  $a$ s and  $b$ s and each prefix of  $w$  has at most 1 more  $a$  than  $b$  and at most 1 more  $b$  than  $a$ .

**Problem 7.4.** Prove that

$$L = \{a^i j c^k \mid i < j < k\}$$

is not context-free.

**Problem 7.5.** Let  $L$  be

$$\{a^i b^j c^i \mid i \geq 0\}$$

Show that  $\bar{L}$  is context-free.

**Problem 7.6.** 1. Show the following to be a regular language:  $\{a^n b^n c^n \mid n \leq 999\}$

2. Show the following to be a non-context-free language:  $\{a^n b^n c^n \mid n > 999\}$

## 8 Decision problems for CFGs

### Problem 8.1. *CFG membership*

Give pseudocode for an algorithm to solve the following problem.

INPUT: a context-free grammar  $G$  with no unit or erasing rules, and a string  $x$  of terminals.

QUESTION: is  $x \in L(G)$ ?

Now consider the same problem, but do not assume that the given grammar  $G$  has no unit or erasing rules.

### Problem 8.2. *CFG emptiness*

Consider the following problem.

INPUT: a context-free grammar  $G$

QUESTION: is  $L(G)$  empty?

Show that this problem is decidable.

### Problem 8.3. *CFG infiniteness*

Consider the following problem.

INPUT: a context-free grammar  $G$

QUESTION: is  $L(G)$  infinite?

### Problem 8.4. *CFG boundedness*

Consider the following problem.

INPUT: a context-free grammar  $G$

QUESTION: does  $L(G)$  contain at least 100 strings?

### Problem 8.5. *The CFG all-strings problem*

Spend 5 minutes thinking about whether there is an algorithm to solve the following problem.

INPUT: an arbitrary context-free grammar  $G$

QUESTION: is  $L(G) = \Sigma^*$ ?

**Problem 8.6.** Assume the following fact: Over the alphabet  $\Sigma = \{a, b\}$ , the language  $D = \{w \mid w \text{ can be written as } xx \text{ for some string } x\}$  is not context-free.

Prove that the language  $C = \{a^n b^m a^n b^m \mid n, m \geq 0\}$  is not context-free.

### Problem 8.7.

1. Let  $L$  be a language over an alphabet  $\Sigma$ . Let  $a$  and  $b$  be elements of  $\Sigma$  and define the function  $h$  from  $\Sigma^*$  to  $\Sigma^*$  by:  $h(x)$  = the result of replacing all occurrences of  $a$  in  $x$  by  $b$ . Having defined  $h$ , let  $h(L)$  be the language obtained by applying  $h$  to each member of  $L$ , that is,  $h(L) = \{h(x) \mid x \in L\}$ .

Prove that if  $L$  is context-free then  $h(L)$  is context-free. (*Hint*: consider a CFG  $G$  such that  $L(G) = L$ ; show how to build a CFG  $G'$  such that  $L(G') = h(L)$ ).

2. More generally, suppose that  $h$  is *any* function mapping elements of  $\Sigma$  to elements of  $\Sigma$ . Extend  $h$  to strings:  $h : \Sigma^* \rightarrow \Sigma^*$  is defined by  $h(x)$  = the result of replacing all occurrences of each symbol  $c \in \Sigma$  by the symbol  $h(c)$ . Finally, extend  $h$  to languages by:  $h(L) = \{h(x) \mid x \in L\}$ .

Prove that if  $L$  is context-free then  $h(L)$  is context-free. (*Hint*: consider a CFG  $G$  such that  $L(G) = L$ ; show how to build a CFG  $G'$  such that  $L(G') = h(L)$ ).

[This phenomenon can be pushed even further, to consider more general mappings from words into words, called “homomorphisms,” which preserve context-free-ness...]

**Problem 8.8.** *PDA non-emptiness*

Show that the following problem is decidable.

INPUT: a PDA  $M$

QUESTION: is  $L(M) \neq \emptyset$

Your solution should take the form of pseudocode for an algorithm to solve the problem. What is the time complexity of your algorithm?

*Hint.* Don’t work with  $M$  directly. Use some basic theory of context-free languages.

**Problem 8.9.** *Using the CYK algorithm I*

Let  $G$  be the following grammar.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow AB \mid b \end{aligned}$$

Use the Cocke-Younger-Kasami algorithm to determine whether the string  $aab$  is in  $L(G)$ .

**Problem 8.10.** *Extending the CYK algorithm I*

Modify the Cocke-Younger-Kasami algorithm to count the number of parse trees of a given string.

**Problem 8.11.** *Extending the CYK algorithm II*

Suppose that to each production in a Chomsky Normal Form grammar  $G$  we associate a cost, and say that the cost of a derivation of a string  $w$  is the sum of the costs of the individual production steps.

Give an algorithm for finding a minimum-cost derivation of a string  $w$  in such a grammar. What is the complexity of your algorithm?

*Hint.* Modify the CYK algorithm.

## 9 Finite Automata

**Problem 9.1.** Can an automaton be infinite?

**Problem 9.2.** Does it make sense to talk about a language accepting a string?

**Problem 9.3.** For a given  $k$  let  $\Sigma_k$  denote the alphabet  $\{0, 1, \dots, k-1\}$ . For given  $k$  and  $m$  let  $L_{k,m}$  be the set of words  $x$  over  $\Sigma_k$  that, when viewed as a representing an integer in base- $k$  notation, code a multiple of  $m$ . Show that each  $L_{k,m}$  is regular, by showing that there is a DFA  $M$  with  $L(M) = L_{k,m}$ .

It is simplest to agree that the empty word codes the number 0.

*Note.* I suggest first solving the problem fixing  $m = 2$  and letting  $k$  vary, then fixing  $k = 2$  and letting  $m$  vary, then finally doing the general case.

**Problem 9.4.** Let  $\Sigma = \{a, b\}$ . Let  $Q = \{p, q\}$ . Build all of the DFAs with input alphabet  $\Sigma$  and state space  $Q$ . Then for each DFA you built, describe the language it accepts. Which pairs of DFAs accept the same language?

*Hints on organizing your work.* By “all” we mean “up to renaming states.” That is, do not bother to distinguish two machines that differ only by swapping the names “ $p$ ” and “ $q$ .” This means that without loss of generality you may assume that the start state is  $p$ . So if each machine is denoted as  $(Q, \Sigma, \delta, p, F)$  this means that you have a choice of (i) the  $\delta$  function, and (ii) what  $F$  is.

There are four choices for  $F$ , right?

I suggest (i) draw a picture for each  $\delta$  function, then (ii) describe the four different DFAs (and their languages) that arise from the different choices for  $F$ .

**Problem 9.5.** Give DFAs to accept the following languages over the alphabet  $\{a, b\}$ .

1.  $\{w \mid w \text{ does not contain } bb \text{ as a substring}\}$ .
2.  $\{w \mid w \text{ every odd position of } w \text{ is the symbol } b\}$ .
3.  $\emptyset$ , the empty set.
4.  $\{\lambda\}$ . This is the language consisting of one string, the empty string  $\lambda$ .

**Problem 9.6.** *Building NFAs*

Give NFAs to accept the following languages over the alphabet  $\{a, b\}$ .

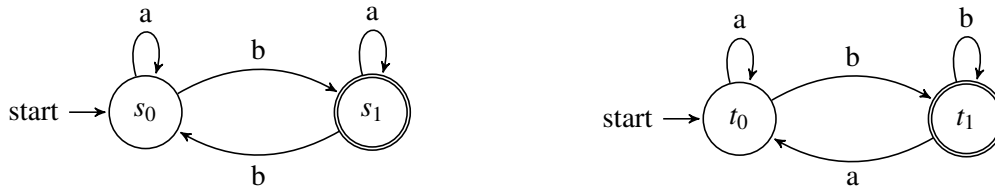
1.  $\{w \mid w \text{ ends in } aa\}$ . Make a machine with three states.
2.  $\{w \mid w \text{ contains } abab \text{ as a substring}\}$ . Make a machine with five states.
3.  $\{w \mid \text{the third symbol from the end of } w \text{ is an } a\}$ .

**Problem 9.7.**

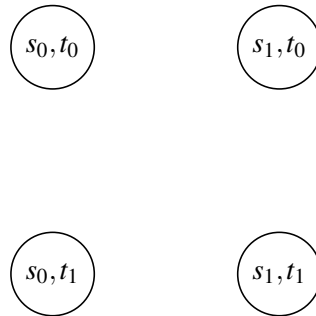
1. Let  $K$  be the language containing a single string. Show that  $K$  is regular. *Hint.* Use an NFA.

2. Let  $K$  be a finite language, that is,  $K$  is a language containing finitely many strings. Show that  $K$  is regular.

**Problem 9.8.** Let  $M$  and  $N$  be the following DFAs.



We want to build a DFA  $P$  such that  $L(P) = L(M) \cap L(N)$ . Below is the state space used in the standard product construction that we studied:



1. What is the start state of  $P$ ? *Mark the diagram and also answer here:*
2. What is the set of accepting states of  $P$ ? *Use double circles in the diagram and also list them here:*
3. Draw and label the transitions of  $P$ . *Just mark the diagram.*
4. Explain briefly and precisely what changes need to be made to the construction in Problem 9.8 in order to obtain a DFA  $Q$  such that  $L(Q) = L(M) \cup L(N)$ .

**Problem 9.9.** Fix  $\Sigma = \{0, 1\}$ . Draw the obvious 2-state DFA that accepts the set of strings of even length. Draw the obvious 2-state DFA that accepts the set of strings with an even number of 0s.

Using the product construction, draw a DFA that accepts the set of strings  $w$  such that  $w$  has even length and  $w$  has an even number of 0s.

Using the product construction, draw a DFA that accepts the set of strings  $w$  such that  $w$  has even length or  $w$  has an even number of 0s (or both, of course).

**Problem 9.10.** The product construction is typically described defined for DFAs, but it actually works just fine for NFAs as well. MAke sure this is clear to you; make sure you can perform the product construction on NFAs.

**Problem 9.11.** Be able to do the construction that, given an NFA  $N$  with  $\lambda$ -transitions, returns a DFA  $M$  such that  $L(M) = L(N)$ .

**Problem 9.12.** Be able to construct, given an arbitrary regular expression  $E$ , an NFA  $N$  with  $\lambda$ -transitions such that  $L(N) = L(E)$ .

**Problem 9.13.** *Complementation and automata*

1. Suppose  $M$  and  $M'$  are DFAs with the same state set, start state, and transitions but with complementary notions accepting and non-accepting:

$$M = (Q, \Sigma, \delta, s, F)$$
$$M' = (Q, \Sigma, \delta, s, Q - F)$$

We know that the language accepted by  $M'$  is precisely the complement  $\bar{L}$  of  $L$ . Show by an example that this claim is *not* true if  $M$  is an NFA rather than a DFA.

2. Prove that if  $N$  is an NFA accepting the language  $L$ , then there is an NFA accepting the language  $\bar{L}$ .
3. Explain why this does *not* contradict the previous part.

**Problem 9.14.** *Reverse*

If  $L$  is a language let us write  $L^R$  for the language consisting of the reverses of the strings in  $L$ . Show that if  $L$  is regular so is  $L^R$ .

*Hint:* Start the proof like this:

Let  $M = (Q, \Sigma, \delta, q_0, F)$  be a DFA such that  $L(M) = L$ . We define the automaton  $M'$  as follows ...

A formal proof that the machine  $M'$  really does accept  $L^R$  is not asked for here. What *is* being asked for is a precise definition of  $M'$ . For starters, what kind of an automaton is  $M'$ ?

**Problem 9.15.** *NFA simulation*

Given an algorithm for simulating an NFA efficiently (that is, without constructing an equivalent DFA first).

Formally: If  $N$  be an NFA, show how to write an algorithm which on input  $w$  will decide whether or not  $w \in L(N)$ . If  $N$  has  $n$  states your algorithm should run in time polynomial in  $n$  and the length of  $w$ .

**Problem 9.16.** “for-all” NFAs

As you know, a standard NFA  $N$  accepts a string  $w$  if *there exists* an accepting run of  $N$  on  $w$ . But we can imagine a new kind of finite automaton, called an *for-all-NFA*. Such a machine  $A$  has the same “hardware” as an NFA:

$$A = (Q, \Sigma, \delta, s, F)$$

as before, with  $\delta$  mapping (state, symbol) pairs into sets of states. The difference is that we declare that a string  $w$  is accepted by a for-all-NFA if *every* run of the machine on  $w$  ends in a state in  $F$ .

Prove that for-all-NFAs accept precisely the regular languages. That is, first prove that for every regular language  $L$  there is an for-all-NFA  $A$  with  $L(A) = L$ ; then prove that for any for-all-NFA  $A$  the language  $L(A)$  is regular.

**Problem 9.17.** *Hamming*

If  $x$  and  $y$  are bit strings, the *Hamming distance*  $H(x, y)$  between them is the number of places in which they differ. If  $x$  and  $y$  have different lengths then we declare their Hamming distance to be infinite. This is an important concept in analyzing transmission of signals in the presence of possible errors.

If  $x$  is a string and  $L$  is a language over  $\{0, 1\}$  then we define

$$H(x, L) \stackrel{\text{def}}{=} \min\{H(x, y) \mid y \in L\}$$

Now for any language  $L$  over  $\{0, 1\}$  and any  $k \geq 0$  we may define

$$N_k(L) \stackrel{\text{def}}{=} \{x \mid H(x, L) \leq k\}$$

For example, if  $L$  were the language  $\{00, 001\}$  then  $N_0(L) = L$ ,  $N_1(L) = L \cup \{10, 01, 101, 011, 000\}$ , and  $N_2(L)$  is the set of all bit strings of length 2 or three except for the string 110.

Prove that if  $L \subseteq \{0, 1\}^*$  is regular, then so is  $N_1(L)$ .

*Hint.* Suppose  $L$  is  $L(M)$  for a DFA with state set  $Q$ . Build an NFA for  $N_1(L)$  with state set  $Q \times \{0, 1\}$ . The second component tells how many “errors” you have seen so far. Use non-determinism generously!

Is there anything special about a Hamming distance of 1 here, or is the result true for any distance?

**Problem 9.18.** *Prefixes (from Sipser)*

A string  $x$  is a *proper prefix* of a string  $z$  if there is a non-empty word  $y$  such that  $xy = z$ . Suppose  $L$  is a language; we may define the following languages based on  $L$ :

- $\text{NoPrefix}(L) = \{w \in L \mid \text{no proper prefix of } w \text{ is in } L\}$ .
- $\text{NoExtend}(L) = \{w \in L \mid w \text{ is not a proper prefix of any string of } L\}$ .

Suppose  $L$  is regular. Is  $\text{NoPrefix}(L)$  guaranteed to be regular?

Suppose  $L$  is regular. Is  $\text{NoExtend}(L)$  guaranteed to be regular?

[This is a fairly hard problem.]

**Problem 9.19.** *Halves (from Sipser)*

Suppose  $L$  is a language; we may define the following language based on  $L$ :

$$\text{Half}L = \{x \mid \exists y|y| = |x| \text{ and } xy \in L\}$$

Suppose  $L$  is regular. Is  $\text{Half}L$  guaranteed to be regular?

[This is a hard problem.]

**Problem 9.20.** (from Kozen ) *Conversion between machines and regular expressions*

Give deterministic finite automata accepting the languages denoted by the following regular expressions.

1.  $(00 + 11)^*(01 + 10)(00 + 11)^*$
2.  $(000)^*1 + (00)^*1$
3.  $(0(01)^*(1 + 00) + 1(10)^*(0 + 11))^*$

## 10 Proving languages not regular

**The game method** There is a way of presenting the standard non-regularity proofs *which is mathematically equivalent to the proofs as found in the text* but which may be a little easier to think about. This is the presentation we did in class, which uses a games metaphor.

**How to prove a language  $L$  not regular:** Prove that you can always win the following game against your opponent Mac.

1. Mac picks a number  $p$ .
2. You pick a  $w \in L$  whose length is at least  $p$ .
3. Mac divides your word into three pieces:  $w = xyz$ . He is constrained to do this in such a way that  $y$  is not  $\lambda$ , and  $xy$  has length no more than  $p$ .
4. You pump up (or down) the  $y$  part to get a word *not* in  $L$ . That is, you choose an  $i \geq 0$  such that the new word  $xy^iz$  is not in  $L$ .

If you can argue that you can always do this, that is, you can make moves in steps 2 and 4 no matter what Mac does in steps 1 and 3, then you have proved that  $L$  cannot be regular.

### Problem 10.1. Easy non-regular languages

Prove that the following languages are not regular.

1.  $L = \{a^n b^n \mid n \geq 1\}$
2. the set of balanced parentheses, over the language whose symbols are “(“ and “)””.
3.  $L = \{a^n b a^n \mid n \geq 1\}$
4.  $L = \{a^n b^m c^n \mid n, m \geq 0\}$
5.  $L = \{a^n b^m \mid n \leq m\}$
6.  $L = \{a^n b^{2^n} \mid n \geq 1\}$

### Problem 10.2. More interesting pumping lemma applications

Use the Pumping Lemma to prove that the following languages are not regular.

1.  $A = \{a^n \mid n \text{ is a perfect square}\}$
2.  $B = \{a^n \mid n \text{ is a perfect cube}\}$
3.  $C = \{a^n \mid n \text{ is a power of } 2\}$
4. the set  $D$  of strings of  $a$ s and  $b$ s whose length is a perfect square

5.  $E = \{ww \mid w \in \Sigma^*\}$

6.  $F = \{ww^R \mid w \in \Sigma^*\}$

**Problem 10.3.** *Infinite subset of non-regular*

Prove that no infinite subset of  $\{a^n b^n \mid n \geq 0\}$  is regular.

## 11 Closure properties of regular languages

### Problem 11.1. Closure properties: summary

Be able to prove each of the following:

1. If  $L$  is regular then the complement  $\bar{L}$  is regular.
2. If  $L$  and  $K$  are regular then intersection  $L \cap K$  is regular.
3. If  $L$  and  $K$  are regular then their union  $L \cup K$  is regular.
4. If  $L$  and  $K$  are regular then their concatenation  $LK$  is regular.
5. If  $L$  is regular then its asterate  $L^*$  is regular.

### Problem 11.2. Applying closure properties

Let  $R$  be a regular language and let  $N$  be a language which is not regular.

1. Suppose  $X$  is a language such that  $X = R \cap N$ . Does it follow that  $X$  is necessarily regular? If so, say why. Does it follow that  $X$  is necessarily non-regular? If so, say why. If your answers to the two previous questions was no, name a regular  $R$  and non-regular  $N$  satisfying  $X = R \cap N$  with  $X$  non-regular, and name a regular  $R$  and non-regular  $N$  satisfying  $X = R \cap N$  with  $X$  regular.
2. Suppose  $X$  is a language such that  $N = R \cap X$ . Does it follow that  $X$  is necessarily regular? If so, say why. Does it follow that  $X$  is necessarily non-regular? If so, say why. If your answers to the two previous questions was no, name a regular  $R$  and non-regular  $N$  satisfying  $N = R \cap X$  with  $X$  non-regular, and name a regular  $R$  and non-regular  $N$  satisfying  $N = R \cap X$  with  $X$  regular.
3. Suppose  $X$  is a language such that  $R = N \cap X$ . Does it follow that  $X$  is necessarily regular? If so, say why. Does it follow that  $X$  is necessarily non-regular? If so, say why. If your answers to the two previous questions was no, name a regular  $R$  and non-regular  $N$  satisfying  $R = N \cap X$  with  $X$  non-regular, and name a regular  $R$  and non-regular  $N$  satisfying  $R = N \cap X$  with  $X$  regular.
4. Suppose  $X$  is a language such that  $R = \bar{X}$ . Does it follow that  $X$  is necessarily regular? If so, say why. Does it follow that  $X$  is necessarily non-regular? If so, say why. If your answers to the two previous questions was no, name a regular  $R$  satisfying  $R = \bar{X}$  with  $X$  non-regular, and name a regular  $R$  satisfying  $R = \bar{X}$  with  $X$  regular.
5. Suppose  $X$  is a language such that  $N = \bar{X}$ . Does it follow that  $X$  is necessarily regular? If so, say why. Does it follow that  $X$  is necessarily non-regular? If so, say why. If your answers to the two previous questions was no, name a non-regular  $N$  satisfying  $N = \bar{X}$  with  $X$  non-regular, and name a non-regular  $N$  satisfying  $N = \bar{X}$  with  $X$  regular.

### Problem 11.3. Bounded exponents.

For each of the following two languages, either prove that it is regular or prove that it is not regular.

1.  $A = \{a^i b^j \mid i \geq j \text{ and } j \leq 100\}$
2.  $B = \{a^i b^j \mid i \geq j \text{ and } j \geq 100\}$

**Problem 11.4.** *Regular subset*

1. Prove or disprove: If  $L$  is regular and  $K \subseteq L$  then  $K$  is regular.
2. Prove or disprove: If  $L$  is regular and  $L \subseteq K$  then  $K$  is regular.

**Problem 11.5.** Let  $M$  and  $N$  be DFAs (or NFAs). Suppose we build the product of  $M$  and  $N$  in the same way we do for the intersection of  $A$  and  $B$  except, rather than taking the accepting states to be those pairs  $(q, r)$  such that  $q$  is accepting in  $M$  and  $r$  is accepting in  $N$ , we define the accepting states to be those pairs  $(q, r)$  such that  $q$  is accepting in  $M$  **or**  $r$  is accepting in  $N$ . Call the resulting automaton  $P$ . Note that if we start with DFAs then  $P$  is a DFA, otherwise  $P$  is an NFA.

Prove that  $L(P) = L(M) \cup L(N)$ .

**Problem 11.6.** Let  $M$  be an NFA with  $L(M) = A$ ; let  $N$  be an NFA with  $L(N) = B$ ; suppose that  $M$  has  $m$  states and  $N$  has  $n$  states.

Here are two proposed ways to define a DFA accepting the language  $A \cup B$ :

1. build an NFA in the obvious way: put together a copy of  $M$  with a copy of  $N$  and non-deterministically branch to one or the other at the start; then use the subset construction to build a DFA  $P_1$ ;
2. use the subset construction on  $M$  and on  $N$  separately to get DFAs  $Q_1$  and  $Q_2$  accepting  $A$  and  $B$ ; then build the product of  $M$  and  $N$  in the same way we do for the intersection of  $A$  and  $B$  except, rather than taking the accepting states to be those pairs  $(q, r)$  such that  $q$  is accepting in  $M$  and  $r$  is accepting in  $N$ , we define the accepting states to be those pairs  $(q, r)$  such that  $q$  is accepting in  $M$  **or**  $r$  is accepting in  $N$ .

Are each of these methods correct, that is, is it true that  $L(P_i) = A \cup B$  for  $i = 1, 2$ ?

What are the relative merits of the two constructions? Pay attention to the running time of the algorithms and to the size of DFAs  $P_i$ .

**Problem 11.7.** Let  $M$  be a DFA with  $L(M) = A$ ; let  $N$  be a DFA with  $L(N) = B$ ; suppose that  $M$  has  $m$  states and  $N$  has  $n$  states.

Here are three proposed ways to define a DFA accepting the language  $A \cup B$ :

1. build an NFA in the obvious way: put together a copy of  $M$  with a copy of  $N$  and non-deterministically branch to one or the other at the start; then use the subset construction to build the DFA  $P_1$
2. build the product of  $M$  and  $N$  in the same way we do for the intersection of  $A$  and  $B$  except, rather than taking the accepting states to be those pairs  $(q, r)$  such that  $q$  is accepting in  $M$  and  $r$  is accepting in  $N$ , we define the accepting states to be those pairs  $(q, r)$  such that  $q$  is accepting in  $M$  **or**  $r$  is accepting in  $N$ ; call this  $P_2$ ;

3. build  $M'$  such that  $L(M') = \overline{A}$  by simply reversing the notions of accepting and non-accepting in  $M$ ; build  $N'$  with  $L(N') = \overline{B}$  by doing the same thing with  $N$ ; then use the standard product construction on  $M'$  and  $N'$  to obtain a DFA  $P'$ ; and finally reversing the notions of accepting and non-accepting in  $P'$  to obtain  $P_3$

Are each of these methods correct, that is, is it true that  $L(P_i) = A \cup B$  for  $i = 1, 2, 3$ ?

What are the relative merits of the three constructions? Pay attention to the running time of the algorithms and to the size of DFAs  $P_i$ .

**Problem 11.8.** *Closure saves pumping*

Assume that the following language is not regular.  $A = \{a^n \mid n \text{ is a perfect square}\}$

Prove that the set  $D$  of strings of  $as$  and  $bs$  whose length is a perfect square is not regular.

**Problem 11.9.** *Closure saves pumping II*

For this problem, *assume* that the language  $Eq = \{a^n b^n \mid n \geq 0\}$  is not regular.

Prove that the following languages over  $\{a, b\}$  are not regular *without doing a direct Pumping Lemma construction*.

1.  $Neq = \{a^k b^l \mid k \neq l\}$ . (Caution: this set is not the same set as  $\overline{Eq}$ )
2. For this problem let us introduce the temporary notation  $w^o$  to stand for the result of taking a string  $w$  and replacing  $as$  by  $bs$  and vice versa.  
Let  $G = \{ww^o \mid w \in \{a, b\}^*\}$ . Show  $G$  is not regular.
3.  $K = \{w \mid w \text{ has an unequal number of } as \text{ and } bs\}$  (Caution: this set is also not the same set as  $\overline{Eq}$ )

## 12 Decision problems about regular languages

Note: to say that a certain problem is *decidable* is to say that there exists an algorithm which solves the problem. As we will see eventually not all problems one is interested in are decidable.

**Problem 12.1.** *DFA non-emptiness*

Give pseudocode for an algorithm to solve the following problem. What is the time complexity of your algorithm?

INPUT: a DFA  $M$

QUESTION: is  $L(M) \neq \emptyset$

Note that it does *not* suffice to say:

*For each string  $x$ ,*

*run  $M$  on  $x$ ;*  
*if  $x$  is accepted then return YES*

This is not a correct solution since this algorithm will not halt in the case that  $L(M)$  is indeed empty.

**Problem 12.2.** *DFA universal*

Give pseudocode for an algorithm to solve the following problem. What is the time complexity of your algorithm?

INPUT: a DFA  $M$

QUESTION: is  $L(M) = \Sigma^*$ ?

**Problem 12.3.** *DFA subset*

Give pseudocode for an algorithm to solve the following problem. What is the time complexity of your algorithm?

INPUT: two DFAs  $M_1$  and  $M_2$

QUESTION: is  $L(M_1) \subseteq L(M_2)$ ?

**Problem 12.4.** *DFA equality*

Give pseudocode for an algorithm to solve the following problem. What is the time complexity of your algorithm?

INPUT: DFAs  $M_1$  and  $M_2$

QUESTION: is  $L(M_1) = L(M_2)$ ?

**Problem 12.5.** *DFA intersection*

Give pseudocode for an algorithm to solve the following problem. What is the time complexity of your algorithm?

INPUT: DFAs  $M_1$  and  $M_2$

QUESTION: do  $L(M_1)$  and  $L(M_2)$  have at least one string in common?

**Problem 12.6.** *DFA neither*

Give pseudocode for an algorithm to solve the following problem. What is the time complexity of your algorithm?

INPUT: DFAs  $M_1$  and  $M_2$

QUESTION: is there any string that is neither in  $L(M_1)$  nor  $L(M_2)$ ?

**Problem 12.7.** *DFA infinity*

Give pseudocode for an algorithm to solve the following problem. What is the time complexity of your algorithm? (You should be able to construct a polynomial-time algorithm.)

INPUT: a DFA  $M$

QUESTION: is  $L(M)$  infinite?

**Problem 12.8.** *DFA 100*

Give pseudocode for an algorithm to solve the following problem. What is the time complexity of your algorithm?

INPUT: a DFA  $M$

QUESTION: does  $L(M)$  have at least 100 strings?

**Problem 12.9.** *Decision problems for NFAs*

Now suppose that for each of the questions above the input is given as  $\lambda$ -NFAs instead of DFAs.

Are the problems still decidable? (That is, are there algorithms to answer them?) Prove or disprove, as appropriate!

For which questions does the *complexity* change significantly?

**Problem 12.10.** *Decision problems for regular expressions*

Same questions as in Problem 12.9, but this time suppose that inputs are presented via regular expressions.

**Problem 12.11.** *Conversion complexity*

Know the complexity results for the conversions among representations of regular languages. Know the complexity results for the two fundamental decision problems concerning regular languages.

## 13 The Myhill-Nerode theorem and minimization of automata

### Problem 13.1. DFA state equivalence

Consider the two automata below. The table shown is the transition function, the start state is indicated by an arrow, and the accepting states are starred.

	a	b
→ 1	1	4
2	3	1
*3	4	2
*4	3	5
5	4	6
6	6	3
7	2	4
8	3	1

	a	b
→* 1	3	5
*2	8	7
3	7	2
4	6	2
5	1	8
6	2	3
7	1	4
8	5	1

1. For each machine tell which states are accessible.
2. List the equivalence classes for the equivalence relation  $\approx$  (the relation we defined which captures the collapsing criterion in the minimization algorithm.)
3. Give the automaton obtained by collapsing equivalent states and removing inaccessible states.

### Problem 13.2. Minimization

Minimize the following DFAs.

1.

	a	b
→ * 1	2	5
* 2	1	4
3	7	2
4	5	7
5	4	3
6	3	6
7	3	1

2.

	a	b
→ * 1	2	6
* 2	1	7
3	5	2
4	2	3
5	3	1
6	7	3
7	6	5

3.

	a	b
→ 1	1	3
* 2	6	3
3	5	7
* 4	6	1
5	1	7
* 6	2	7
7	3	3

4.

	a	b
→ * 1	2	5
* 2	1	6
3	4	3
4	7	1
5	6	7
6	5	4
7	4	2

**Problem 13.3.** *Minimization practice*

1. Construct the minimal DFA for the language  $(a + b) * ab$ .
2. Construct the minimal DFA for the language  $((a + b)(a + b))^* + (a + b)*b$
3. Construct the minimal DFA for the language consisting of all strings containing  $aba$  as a subword.

Don't be clever and invent a minimal DFA from scratch - treat these as exercises in finding minimal DFAs systematically. That is, make a naive DFA (perhaps by starting with an NFA) and use the minimization algorithm.

**Problem 13.4.** *The Myhill-Nerode Theorem*

(This is an extended extra-credit problem...)

Fix an alphabet  $\Sigma$ . Let  $K$  be any language (regular or not). Define the relation  $\cong_K$  on words by:

$$x \cong_K y \quad \text{if} \quad \forall w (xw \in K \text{ if and only if } yw \in K)$$

Show that  $\cong_K$  is an equivalence relation on  $\Sigma^*$ , that is, it is reflexive, symmetric, and transitive.

Since  $\cong_K$  is an equivalence relation, it partitions  $\Sigma^*$  into classes.

Let us write  $[x]_K$  for the equivalence class of a word  $x$ . That is,

$$[x]_K = \{y \mid x \cong_K y\}$$

Do at least a few of the following

- Let  $A = \{x \in \{a, b\}^* \mid x \text{ has odd length}\}$  Find the equivalence classes of  $\cong_A$ .
- Let  $B = \{a^n b^m \mid n, m \geq 0\}$ . Find the equivalence classes of  $\cong_B$ .
- Let  $C = \{a^n b^n \mid n \geq 0\}$ . Find the equivalence classes of  $\cong_C$ .
- Let  $D = \{x \in \{a, b\}^* \mid x \neq \lambda \text{ and } x \text{ begins and ends with the same symbol}\}$ . Find the equivalence classes of  $\cong_D$ .
- Let  $E = \{w \mid \exists x w = xx\}$ . Find the equivalence classes of  $\cong_E$ .
- Let  $F = \{w \in \{a, b\}^* \mid w \text{ has an equal number of } a\text{s and } b\text{s}\}$ . Find the equivalence classes of  $\cong_F$ .
- Let  $G = \{a^n b^m \mid n, m \geq 0 \text{ and } n + m \text{ is divisible by } 3\}$ . Find the equivalence classes of  $\cong_G$ .
- For fixed  $k$  let  $H_k = \{w \in \{a, b\}^* \mid \text{the } k\text{th symbol from the end of } w \text{ is } a\}$ . Find the equivalence classes of  $\cong_{H_k}$ .

You should have observed above that the regular language examples generated finitely many equivalence classes while the non-regular examples generated infinitely many equivalence classes. Let's explore this further. Some convenient notation: for a language  $K$  let  $\text{Index}(K)$  stand for the number of equivalence classes of the relation  $\cong_K$ . (This is a slight abuse of notation since  $\text{Index}(K)$  might be infinite.)

- Suppose  $K$  is a language that is accepted by a DFA  $M$ , with start state  $s$ . Let  $x$  and  $y$  be words such that  $\delta(s, x) = \delta(s, y)$ . Show that  $x \cong_K y$ .
- Show that if  $K$  is a regular language, accepted by a DFA with  $p$  states, then  $\cong_K$  has at most  $p$  equivalence classes. (Use the previous result).

Now we have shown that if a language  $K$  is regular then  $\text{Index}(K)$  is finite. In fact the converse holds as well: we (you) will prove this next.

First let's pause and note that we already have a powerful tool at our disposal. For example we can prove that a language  $K$  is non-regular as soon as we can show that  $\text{Index}(K)$  is infinite: no Pumping Lemma required. As another application, consider the languages  $H_k$  defined in a problem above (the set of words whose  $k$ th symbol from the end is a particular pre-determined letter). It is pretty easy to show that, over a two-letter alphabet,  $\text{Index}(H_k)$  is  $2^{k+1} - 1$ . So the size of the smallest DFA for  $H_k$  is no less than  $2^{k+1} - 1$ . Since it is easy to build an NFA accepting  $H_k$  with  $k + 1$  states, we may conclude that in the worst case an exponential blowup in the number of states is unavoidable when converting NFAs to DFAs.

OK, on to showing that if  $\text{Index}(K)$  is finite then  $K$  is regular, and indeed has an accepting DFA with  $\text{Index}(K)$ -many states.

Let  $K$  be a language and suppose that  $\text{Index}(K)$  is finite, say  $\text{Index}(K) = n$ . We can write the equivalence classes of  $\cong_K$  as

$$[x_1]_K, [x_2]_K, \dots, [x_n]_K$$

for certain words  $x_1, x_2, \dots, x_n$ . Important: the choice of these  $x_i$  is not canonical, that is, there are usually many different "names" for the same equivalence class. Indeed, whenever  $x \cong_K y$  we have that  $[x]_K$  is the same set as  $[y]_K$ .

- Mini-exercise: go back to the examples above where you generated the equivalence classes for language  $A$ , (there were two of them, right?) and write them as

$$[x_1]_A, [x_2]_A$$

for several different choices of  $x_1$  and  $x_2$ . Do the same for  $B, C, D, \dots$  until you really understand this idea or until you get tired.

The following technical observation will be crucial later.

- Prove that if  $[x]_K = [y]_K$  then for every  $a \in \Sigma$ ,  $[xa]_K = [ya]_K$ .

Now here is the beautiful idea. We want to show that if  $\text{Index}(K)$  is finite then  $K$  is regular. It suffices to show that if  $\text{Index}(K)$  is finite then we can build a DFA accepting  $K$ . The trick is to let the states of the DFA be the equivalence classes of  $\cong_K$ !

Given  $K$  we define  $M_K = (Q, \Sigma, \delta, s, F)$  as follows.

- $Q = \{[x]_K \mid x \in \Sigma^*\}$
- For states  $[x]_K$  and input characters  $a$  we have  $\delta([x]_K, a) = [xa]_K$ . That is,  $[x]_K \xrightarrow{a} [xa]_K$
- $s = [\lambda]_K$
- $F = \{[x_k] \mid x \in K\}$

The claim is that  $L(M_K) = K$ .

Before we do anything else we must show that the above is well-defined. By this we mean that we have to show that the definition of  $\delta$ , which ostensibly depends on the choice of names for states (equivalence classes) does not *really* depend on the names. In other words we must show that if  $[x]_K$  and  $[y]_K$  are the same state, then our definition of the  $\delta$ -function treats them the same. This amounts to proving that in writing

$$[x]_K \xrightarrow{a} [xa]_K,$$

if we replace the name  $x$  by some  $y$  which also names  $[x]_K$ , that is,  $[x]_K = [y]_K$ , then

$$[y]_K \xrightarrow{a} [ya]_K.$$

- Prove this.

Now that we know that the definition of  $M_K$  makes sense, it remains to

- Prove that  $L(M_K) = K$ .

*Hint.* Consider an arbitrary word  $w$  and suppose

$$s \xrightarrow{w} q$$

You want to show that  $q \in F$  if and only if  $w \in K$ .

First unravel what the above means in terms of the particular  $M_K$  we are working with. (What's  $s$ ? What's  $F$ ? What can you say about  $\xrightarrow{w}$  based on the definition of  $\rightarrow$ ?)

- Prove that the DFA constructed above has minimal size among all DFAs accepting  $K$ .

(This isn't hard, it is a direct consequence of something you proved earlier in this marathon exercise.)

Can there be more than one minimal DFA for a given language? Of course in a trivial sense the answer is yes: given any DFA we can get a "different" one doing the same job just by changing the names of the states. This isn't interesting. What we want to mean by "different" is "having a different structure". This is fairly clear intuitively, but deserves a careful definition.

Standard terminology: two DFAs  $N_1$  and  $N_2$  are *isomorphic* (Greek: "same structure") if there is a one-to-one and onto mapping  $r$  (a "renaming") from the states of  $N_1$  to the states of  $N_2$  that preserves the  $\delta$ -functions; formally

$$\text{for all input symbols } a \text{ and states } q \text{ in } N_1, \quad q \xrightarrow{a} p \quad \text{implies} \quad r(q) \xrightarrow{a} r(p)$$

- Mini-exercise: we wrote "implies" above but show that the assertion above actually has "if and only if" as a consequence.

Now we can re-phrase the earlier question precisely as (i) *can there be two or more non-isomorphic minimal DFAs for a given language?*

Another question that comes to mind is: (ii) *how are the DFAs we built out of equivalence classes related to the "minimized" DFAs we built by the quotient construction?*

These questions seem unrelated at first glance but in fact we can answer them both with the following result.

- Let  $K$  be a regular language. Let  $M$  be any DFA such that  $L(M) = K$ , and let  $M'$  be the result of the quotient construction on  $M$ . Then  $M'$  is isomorphic to  $M_K$ .

*Hint.* Build an isomorphism from  $M'$  to  $M_K$ . You don't have to be clever! The collapsing construction does all the work for you.

This immediately answers the question labelled (ii) just above. With a little more thought we can answer (i), to wit:

- Let  $K$  be a language and Let  $M_1$  and  $M_2$  each be minimal-state-size DFAs accepting  $K$ . Then  $M_1$  and  $M_2$  are isomorphic.

*Hint.* This is a corollary of the previous.

**Problem 13.5.** *Another Myhill-Nerode exercise*

Let  $R$  be the set denoted by the regular expression

$$a^*b^* + b^*a^*$$

1. Find the minimal DFA  $M$  with  $L(M) = R$
2. Find a regular expression denoting each of the equivalence classes of the Myhill-Nerode relation  $\cong_R$

**Problem 13.6.** Let  $\Sigma = \{a, b, c\}$ . Let  $K$  be the language consisting of all words  $w$  over  $\Sigma$  such that the  $n$ th letter from the end of  $w$  is a  $b$ .

$$K = (a \cup b \cup c)^* b (a \cup b \cup c) \dots (a \cup b \cup c)$$

where there are  $(n - 1)$  occurrences of  $(a \cup b \cup c)$  after the  $b$ .

Describe the equivalence classes of the Myhill-Nerode relation  $\cong_R$ . There are  $2^n$  of them!

**Problem 13.7.** *Complexity of NFA-to-DFA*

Prove that the worst-case complexity of translating from NFAs to DFAs is exponential, by showing the following. For each  $n$  there is a NFA  $N$  with  $n + 1$  states such that the smallest DFA accepting the same language as  $N$  has  $2^n$  states. *Hint:* use problem [13.6](#)

## 14 Turing machines

**Problem 14.1.** Exercise 9 from the online notes (That is, be able to trace a computation given a Turing machine and an input.)

**Problem 14.2.** 1. Give a formal definition of a TM which behaves as follows.

- The input alphabet is  $\{a, b\}$
- The tape alphabet is  $\{a, b, \sqcup\}$
- The machine scans the tape once, replacing *as* by *bs* and vice-versa, and goes into a halting-accepting state when it first sees a blank.

Your main job is to decide what states are needed, and write the above description as a set of quintuples comprising the  $\delta$  move relation.

2. Write down explicitly the sequence of configurations (IDs) that results when this machine is started on input *abbab*.

**Problem 14.3.** Give a formal definition of a Turing machine  $M$  over the input alphabet  $\Sigma = \{0, 1\}$  such that

- $L(M) = \{1^n \mid n > 0\}$ ;
- $M$  fails to halt on input word  $w$  whenever there are two consecutive 0s in  $w$
- $M$  halts normally in state  $q_{rej}$  for all other words

**Problem 14.4.** Say that a Turing machine is “simple” if, at each step, it may

- change the state and tape symbol but remain scanning the same tape square, *or*
- change the state and move left or right, but not change the tape symbol

Show that simple TMs can simulate ordinary Turing machines. Precisely, show that given a TM  $M$  there exists a simple TM  $M'$  such that  $L(M') = L(M)$ .

**Problem 14.5.** A Turing machine must move the tape head either left or right at each step. Sometimes this restriction is annoying. Consider the following variation on Turing machines: at each step the tape head can move left, or move right, or remain in the same place. The notions of computation and language acceptance for these new machines are the obvious trivial generalizations of the definitions for ordinary TMs.

Show that this variation adds no computing power, in the sense that given any machine  $M$  of the new type there exists an ordinary TM  $M'$  such that  $L(M') = L(M)$ . The obvious idea for the construction works; the real content of this problem is to have you write down the construction rigorously.

**Problem 14.6.** Let  $D = (Q, \Sigma, \delta, q_s, F)$  be a DFA. Show that there exists a TM  $M$  with such that  $L(M) = L(D)$ . Careful: it is not correct to say something like “A DFA *is* a TM ...!”. There are some small subtleties, which is what you have to deal with in this problem.

**Problem 14.7.** (from Kozen) Show that the following problems are decidable, by giving pseudocode for an algorithm that solves them. In each case the input is a Turing machine  $M$ .

1. Question: Does  $M$  take more than 1000 steps on input  $\lambda$ ?
2. Question: Does there exist an input  $w$  such that  $M$  take more than 1000 steps on  $w$ ?
3. Question: Does  $M$  take more than 1000 steps on *all* inputs  $w$ ?
4. Question: On input  $\lambda$ , does  $M$  ever move its head more than 1000 tape cells away from where it started?

**Problem 14.8.** Informally, a pushdown automaton is, as you recall, an NFA equipped with a stack memory. It is easy to see that PDAs are not as powerful as Turing machines, since there exists a TM to recognize the set  $\{a^n b^n c^n \mid n \geq 0\}$  (we know this since it is easy to see that there is a C program to accept this language).

Let's define a "PDPDA" to be just like a PDA except that it has two stacks.

1. Make a formal definition of PDPDA. Take as your model the definition of standard PDA which follows.

**Definition.** A pushdown automaton  $M$  is a tuple  $(Q, \Sigma, \Gamma, \delta, s, \perp)$ , where

- $Q$  is a finite set of *states*, with  $s \in Q$ ,
- $\Sigma$  is a finite *input alphabet*,
- $\Gamma$  is a finite *stack alphabet*, with  $\perp \in \Gamma$ , and
- $\delta$  is a *move relation*:  $\delta \subseteq (Q \times (\Sigma \cup \{\lambda\}) \times \Gamma) \times (Q \times \Gamma^*)$

For simplicity assume that the two stacks have the same stack alphabet, and that at each move both stacks can be altered.

2. Show that PDPDAs are as powerful as Turing machines. That is, describe how an arbitrary TM can be simulated by a PDPDA. You do not need very gory detail here, but be precise.

## 15 Undecidability

### Notation.

- It is convenient below to use the following notation

$$\begin{aligned}U &= \{\langle w, y \rangle \mid y \in L(w)\} \\U_1 &= \{w \mid \langle w, w \rangle \in U\} \\&= \{w \mid w \in L(w)\}\end{aligned}$$

- Below, when we speak of a *total* program we mean: a program that halts on all inputs.

### Problem 15.1. Sets and decision problems

For each of the following sets write the corresponding decision problem (in INSTANCE/QUESTION style).

1. The set of strings of even length.
2. The set of primes in binary.
3. The set  $\{a^n b^n \mid n \geq 0\}$ .
4. The universal set  $U$ .
5. The set  $U_1$ .
6. The set  $\overline{U_1}$ .
7. The set of total programs.

### Problem 15.2. Decision problems and sets

For each of the following problems identify the corresponding language.

1. INPUT: A program  $m$   
QUESTION: Does  $m$  have at least 999 lines?
2. INPUT: A program  $m$   
QUESTION: Does  $m$  accept  $\lambda$ ?
3. INPUT: A program  $m$   
QUESTION: Does  $m$  accept every string?
4. INPUT: A program  $m$   
QUESTION: Is  $L(m)$  regular?
5. INPUT: A program  $m$

QUESTION: Is  $L(m)$  decidable?

6. INPUT: A program  $m$

QUESTION: Is there some program  $n$  with fewer lines than  $m$  such that  $L(n) = L(m)$ ?

*Context.* This is essentially the problem of asking whether a given program can be shortened without changing its meaning.

**Problem 15.3.** *Fundamental theorems*

Be able to state precisely *and prove* the theorems that

1. a language  $A$  is decidable if and only if  $A$  is RE and  $\bar{A}$  is RE.
2. the set  $U_1$  is RE but not decidable.
3. the universal set  $U$  is RE but not decidable

**Problem 15.4.** *Decidable closure properties*

Suppose that  $A$  and  $B$  are decidable sets. Prove the following

1.  $\Sigma^* - A$  is decidable.
2.  $A \cap B$  is decidable.
3.  $A \cup B$  is decidable.
4.  $AB$  is decidable.
5.  $A^*$  is decidable.

**Problem 15.5.** *RE closure properties*

Suppose that  $A$  and  $B$  are RE sets. Prove the following

1.  $A \cap B$  is RE.
2.  $A \cup B$  is RE.
3.  $AB$  is RE.
4.  $A^*$  is RE.

*Hint.* In each case you should proceed by outlining a not-necessarily-total program for the language in question (for instance,  $A \cup B$ ), based on the existence of such programs for the component languages (for instance  $A$  and  $B$ ).

Be careful! For example, here is a *wrong* answer to the problem of showing  $A \cup B$  RE given  $A$  RE and  $B$  RE:

We know there is a program  $M$  with  $L(M) = A$  and a program  $N$  with  $L(N) = B$ .  
Here is a program which recognizes  $A \cup B$ :

On input  $w$ , run  $M$  on  $w$ . If this accepts, print yes. If not, run  $N$  on  $w$ . If this accepts, print yes.

This is wrong wrong wrong because there is no reason to assume that  $M$  will necessarily halt and return control to the top level and allow you to go on to simulate  $N$ . You have to be more clever than that.

**Problem 15.6.** *RE subset*

1. Prove or disprove: If  $L$  is RE and  $K \subseteq L$  then  $K$  is RE.
2. Prove or disprove: If  $L$  is RE and  $L \subseteq K$  then  $K$  is RE.

**Problem 15.7.** Prove or give a specific counterexample: If  $A \cup B$  is RE then  $A$  and  $B$  are RE.

What if we replace “RE” by decidable? What if we replace “ $\cup$ ” by “ $\cap$ ”?

**Problem 15.8.** *RE partition (from HMU)*

Suppose  $L_1, L_2, \dots, L_k$  are recursively enumerable languages over the alphabet  $\Sigma$  such that

1.  $L_i \cap L_j = \emptyset$  when  $i \neq j$ , and
2.  $L_1 \cup L_2 \cup \dots \cup L_k = \Sigma^*$ .

Prove that  $L_1$  is decidable. Indicate clearly how each of the hypotheses (a) and (b) are used in your argument.

Is it really necessary to have *both* condition 1 and condition 2 in order to conclude 3? If so, give counterexamples showing that the result fails if one or the other condition fails. If not, explain why not.

**Problem 15.9.** *RE splitting (from HMU)*

Suppose that  $L$  is RE but not decidable. (So  $\bar{L}$  is not RE...) Consider the language

$$L' \stackrel{\text{def}}{=} \{0x \mid x \text{ is in } L\} \cup \{1x \mid x \text{ is not in } L\}$$

Can you say for certain (without knowing more about  $L$ ) whether  $L'$  is decidable, RE, or non-RE? Justify your answer.

**Problem 15.10.** Let  $X$  be decidable and let  $Y$  be semi-decidable but not decidable. Define

- $Z_1 = X - Y$
- $Z_2 = Y - X$

1. Exactly one of  $Z_1$  or  $Z_2$  is guaranteed to be semi-decidable. Which is it?
2. For the  $Z_i$  which is guaranteed to be semi-decidable, prove it. Here you may quote without proof any closure properties you know.

3. For the  $Z_i$  which is not guaranteed to be semi-decidable, give a recursive set  $X$  and an semi-decidable set  $Y$  which demonstrate this.

**Problem 15.11.** *Taxonomy*

For each of the following situations, either give an example of a language  $L$  fitting the description, or explain why the situation is impossible.

1.  $L$  is decidable and  $\bar{L}$  is decidable.
2.  $L$  is decidable and  $\bar{L}$  is RE but not decidable.
3.  $L$  is decidable and  $\bar{L}$  is not RE.
4.  $L$  is RE but not decidable and  $\bar{L}$  is decidable.
5.  $L$  is RE but not decidable and  $\bar{L}$  is RE but not decidable.
6.  $L$  is RE but not decidable and  $\bar{L}$  is not RE.
7.  $L$  is not RE and  $\bar{L}$  is decidable.
8.  $L$  is not RE and  $\bar{L}$  is RE but not decidable.
9.  $L$  is not RE and  $\bar{L}$  is not RE.

## 16 Reducibility

### Basic facts about reducibility

**Problem 16.1.** (On being careful about the direction of the reduction...)

We know that if  $A \leq_m B$  and  $B$  is decidable then  $A$  is decidable. Show by examples that if  $A \leq_m B$  and  $A$  is decidable then  $B$  may or may not be decidable.

**Problem 16.2.** Prove that if  $A \leq_m B$  and  $B \leq_m C$  then  $A \leq_m C$ .

**Problem 16.3.** 1. There is only one set  $A$  such that  $A \leq_m \Sigma^*$ . What is it? (Prove your answer).

2. There is only one set  $A$  such that  $A \leq_m \emptyset$ . What is it? (Prove your answer).

**Problem 16.4.** Prove that If  $A \leq_m B$  then  $\bar{A} \leq_m \bar{B}$ . (Easy.)

**Problem 16.5.** Let  $A$  and  $B$  be languages, neither of which is  $\emptyset$  and neither of which is  $\Sigma^*$ . Prove that if  $A$  is decidable then  $A \leq_m B$ .

Note that we do not assume anything about the decidability of  $B$ .

### Easy reducibility

**Problem 16.6.** Prove that the following language is not decidable.

$$\{0p \mid p \in U_1\}$$

This is the set of all programs in  $U_1$  but with the single character “0” prepended to each one.

**Problem 16.7.** Let  $A$  be any non-decidable set. Show that the set  $A^R$  of reversals of strings in  $A$  is not decidable.

**Problem 16.8.** Let  $A$  be any set; then let  $A_{\text{even}}$  be the set all even length strings in  $A$ . Prove that if  $A_{\text{even}}$  is not decidable then  $A$  is not decidable.

*Hint.* You need a little tricky observation to start: explain why there must be at least one string  $w$  which is not in  $A$ . Then this  $w$  will be handy in the rest of the proof.

**Problem 16.9.** Prove or disprove: If  $A$  is not decidable then  $A_{\text{even}}$  is not decidable.

(Compare Problem 16.8)

**Problem 16.10.** Prove that the following language is not decidable.

$$2U_1 = \{pp \mid p \in U_1\}$$

Note that  $2U_1$  is not the same as the concatenation  $U_1U_1$

**Problem 16.11.** Prove: If  $A \leq_m B$  and  $B$  is semidecidable then  $A$  is semi-decidable.

**Problem 16.12.** Show by example that the following claim *fails*: If  $A$  is undecidable then  $AA$  is undecidable.

(Compare problem 16.10.)

## Miscellaneous

**Problem 16.13.** We know that the following problem is not decidable.

INPUT: a context-free grammar  $G$

QUESTION: is  $L(G) = \Sigma^*$ ?

Show that in fact it is not even semi-decidable.

*Hint.* Don't make this too hard; it's really an easy observation based on a fundamental fact about CFGs.

**Problem 16.14.** Consider the question: is it true that for every  $A$  we have  $A \leq_m \bar{A}$ ? That seems like a reasonable conjecture. But it's always good to test things on the "extreme" cases, so you consider  $A = \emptyset$ , and then you realize that  $\emptyset \leq_m \Sigma^*$  is pretty obviously false.

So now consider the more refined conjecture: if  $A \neq \emptyset$  and  $A \neq \Sigma^*$  then we have  $A \leq_m \bar{A}$ . Prove or disprove.

*Hint.* Recall Problem 16.11

## Undecidability and programs

**Problem 16.15.** Let  $Tot$  be the set of (codes for) "total programs:" those program that halt on all inputs. Prove that  $U_1 \leq_m Tot$ .

**Problem 16.16.** *Program emptiness*

Without using Rice's Theorem, prove that the following problem is undecidable.

INPUT: A program  $M$ .

QUESTION: Is  $L(M) = \emptyset$ ?

**Problem 16.17.** *Program equivalence*

Assume that the following problem is undecidable.

INPUT: A program  $M$ .

QUESTION: Is  $L(M) = \emptyset$ ?

Prove that the following problem is undecidable.

INPUT: Two programs  $M_1$  and  $M_2$ .

QUESTION: Does  $L(M_1) = L(M_2)$ ?

*Hint.* Use proof by contradiction. Show how, if you had an algorithm to solve this problem, you could solve a certain problem that we already know to be undecidable. (This is not an m-reducibility problem)

**Problem 16.18.** *Program inclusion*

Assume that the following problem is undecidable.

INPUT: A program  $M$ .

QUESTION: Is  $L(M) = \emptyset$ ?

Without using Rice's Theorem, prove that the following problem is undecidable.

INPUT: Two programs  $M_1$  and  $M_2$ .

QUESTION: Is  $L(M_1) \subseteq L(M_2)$ ?

**Problem 16.19.** For each of the following problems do two things: (i) determine whether Rice's Theorem applies or not, then (ii) determine whether the problem is decidable.

1. INPUT: A program  $M$   
QUESTION: Does  $M$  have at least 999 lines?
2. INPUT: A program  $M$   
QUESTION: Does  $M$  take at least 999 steps on input  $\lambda$ ?
3. INPUT: A program  $M$   
QUESTION: Does there exist an input on which  $M$  takes at least 999 steps?
4. INPUT: A program  $M$   
QUESTION: Does  $M$  take at least 999 steps on *all* inputs?
5. INPUT: A program  $M$   
QUESTION: Does  $M$  accept  $\lambda$ ?
6. INPUT: A program  $M$   
QUESTION: Does  $M$  accept any string?
7. INPUT: A program  $M$   
QUESTION: Does  $M$  accept every string?
8. INPUT: A program  $M$   
QUESTION: Is  $L(M)$  regular?
9. INPUT: A program  $M$   
QUESTION: Is  $L(M)$  finite?
10. INPUT: A program  $M$

QUESTION: Is  $L(M)$  a semi-decidable language?

(a trick question, mostly here to provide context for the next question...)

11. INPUT: A program  $M$

QUESTION: Is  $L(M)$  a decidable language?

12. INPUT: A program  $M$

QUESTION: Is there some program  $n$  with fewer lines than  $M$  such that  $L(N) = L(M)$ ?

*Context.* This is essentially the problem of asking whether a given program can be shortened without changing its meaning.

## 17 Post's Correspondence Problem

Recall the definition of the Post Correspondence Problem:

**Definition** Fix an alphabet  $\Sigma$ . *Post's Correspondence Problem (PCP)* is as follows.

INPUT: Two lists  $\langle w_1, \dots, w_k \rangle$  and  $\langle x_1, \dots, x_k \rangle$  of words over  $\Sigma$ ; note that the number of words in the lists are the same.

QUESTION: Does there exist a sequence of integers  $i_1, \dots, i_m$  such that

$$w_{i_1} \cdots w_{i_m} = x_{i_1} \cdots x_{i_m}$$

That is, we interpret the sequence  $i_1, \dots, i_m$  as indices to build a simultaneous concatenation of words from the two lists, and ask for a sequence such that the resulting concatenated words are the same.

We refer to  $k$ , the number of words in each list, as the *size* of the problem instance.

It is helpful to think of the two lists  $\langle w_1, \dots, w_k \rangle$  and  $\langle x_1, \dots, x_k \rangle$  as defining a set of  $k$  “dominoes”, the first domino having  $w_1$  on top and  $x_1$  on the bottom, the second domino having  $w_2$  on top and  $x_2$  on the bottom, and so forth. Under this intuition we are asking for a sequence of dominoes such that the words obtained by reading across the tops and the bottoms are the same.

**Theorem** Post's Correspondence Problem is undecidable.

Needless to say, the fact that PCP is undecidable as a decision problem does not mean that one cannot analyze individual instances, or even certain families of instances. In this set of problems you will play with some PCP instances, develop some intuition for the general problem, and prove a few results about special cases.

**Problem 17.1.** (*Warm Up*) For each instance of the PCP below, either give a solution or argue why no solution exists.

1.  $\langle abc, aca, b \rangle$  and  $\langle ab, ca, acab \rangle$
2.  $\langle abc, abba, c, bbba, abcc \rangle$  and  $\langle ab, c, ccc, cbbb, aab \rangle$
3.  $\langle c, b, a, aba, bab \rangle$  and  $\langle cb, aba, bab, b, a \rangle$

**Problem 17.2.** Show that the subclass of PCP instances in which each  $w_i$  and each  $x_i$  has length either 1 or 2 is decidable, by giving pseudocode for a decision procedure.

**Problem 17.3.** In this problem assume that the alphabet  $\Sigma$  is  $\{a\}$ . Show that the PCP is decidable over this alphabet, by giving pseudocode for a decision procedure.

**Problem 17.4.** (*Hard.*) In this problem assume that the alphabet  $\Sigma$  is  $\{a, b\}$ . Show that the subclass of PCP instances of size  $k = 2$  is decidable, by giving pseudocode for a decision procedure.

**Problem 17.5.** Suppose  $k$  is some number such that over the alphabet  $\Sigma = \{a, b\}$ , the subclass of PCP instances of size  $k$  is decidable. Show that the subclass of PCP instances of size  $k$  over *any* alphabet would be decidable.

**Problem 17.6.** Consider the following two decision problems.

1. INPUT: A context-free grammar  $G$  and a DFA  $M$   
QUESTION: Is  $L(M) \subseteq L(G)$ ?
2. INPUT: A context-free grammar  $G$  and a DFA  $M$   
QUESTION: Is  $L(G) \subseteq L(M)$ ?

One of these problems is decidable and one is undecidable. Which is which? Prove your answer in each case.

## 18 P and NP

**Problem 18.1.** Suppose  $P$  is a program that runs in time  $O(n^k)$ ; suppose  $Q$  is a program that runs in time  $O(n^{k+2})$ ; Give a tight big-Oh expression for the running time of the following programs.

Your answer should look like “ $O(n^{\text{something}})$ ”, with no other arithmetic expressions inside the big-Oh.

1. on input  $x$ ;  
run  $P$  on  $x$ ;  
if this rejects, REJECT;  
else run  $Q$  on  $x$  and return the same answer.
2. on input  $x$ ;  
run  $P$  on  $x$ ;  
run  $P$  on  $x$ ;  
run  $Q$  on  $x$  and return the same answer.
3. on input  $x$ ;  
for  $i = 0$  to 9999:  
run  $P$  on  $x^i$  ; // that is,  $x$  concatenated with itself  $i$  times  
if this accepts, break out of this loop and ACCEPT;
4. on input  $x$ ;  
for  $i = 0$  to  $|x|$ :  
run  $P$  on the prefix of  $x$  given by the first  $i$  characters;  
if this accepts, break out of this loop and ACCEPT;

**Problem 18.2.** For each of the following pairs of functions  $f$ ,  $g$ , determine whether  $f$  is  $o(g)$ ,  $g$  is  $o(f)$ , or  $f$  is  $\Theta(g)$ .

1.  $f(n) = n^2$ ,  $g(n) = 3n^2 + \log n$
2.  $f(n) = n^{10}$ ,  $g(n) = 2^n$
3.  $f(n) = n^{10}$ ,  $g(n) = 2^{n/2}$
4.  $f(n) = \sqrt{n}$ ,  $g(n) = 2^{\sqrt{n}}$
5.  $f(n) = n^{10}$ ,  $g(n) = 2^{(\log n)^2}$
6.  $f(n) = 10n$ ,  $g(n) = n \log n$

**Problem 18.3.** The following problem is easily seen to be in **NP**; it is not know whether it is in **P**.

INPUT: A triple of numbers  $\langle n, l, k \rangle$  represented in binary notation

QUESTION: Is there an integer  $j$ ,  $l \leq j \leq k$ , dividing  $n$ ?

However, show that the following problem is indeed in **P**.

INPUT: A triple of numbers  $\langle n, l, k \rangle$  represented in *unary* notation

QUESTION: Is there an integer  $j$ ,  $l \leq j \leq k$ , dividing  $n$ ?

**Problem 18.4.** Prove that the class  $P$  is closed under intersection, union, and complement.

**Problem 18.5.** Prove that if  $A \in P$  and  $B \in P$  then  $AB \in P$ .

(You have to be just slightly clever.)

**Problem 18.6.** Prove that if  $A \in P$  then  $A^* \in P$ .

*Hint.* Think dynamic programming.

**Problem 18.7.** Prove that the class  $NP$  is closed under intersection, union, concatenation, and asterate. Do you think it is closed under complement? Discuss.

*Hint.* The proof for asterate is easier than the corresponding proof for the class **P**. No need for dynamic programming!

**Problem 18.8.** 1. Prove or disprove: If  $L \in P$  and  $K \subseteq L$  then  $K \in P$ .

2. Prove or disprove: If  $L \in P$  and  $L \subseteq K$  then  $K \in P$ .

3. Prove or disprove: If  $L \in NP$  and  $K \subseteq L$  then  $K \in NP$ .

4. Prove or disprove: If  $L \in NP$  and  $L \subseteq K$  then  $K \in NP$ .

**Problem 18.9.** Prove or give a specific counterexample: If  $A \subseteq B$  and  $B$  is in **NP** then  $A$  is in **NP**.

## 19 Poly-time Reducibility

**Problem 19.1.** Let  $G = (V, E)$  be a graph. A set  $W$  of vertices is an *independent set* in  $G$  if no two vertices in  $W$  are connected by an edge.

This motivates the following problem, the Independent Set Problem.

INPUT: a graph  $G$  and an integer  $k$

QUESTION: does  $G$  have an independent set of size at least  $k$ ?

Draw a graph  $G$  with six vertices such that the instance  $\langle G, 4 \rangle$  gets a no answer but the instance  $\langle G, 3 \rangle$  gets a yes answer.

**Problem 19.2.** Let  $G = (V, E)$  be a graph. A set  $W$  of vertices is a *vertex cover* for  $G$  if every edge in  $E$  uses at least one vertex from  $W$ .

This motivates the following problem, the Vertex Cover Problem.

INPUT: a graph  $G$  and an integer  $k$

QUESTION: does  $G$  have a vertex cover of size no more than  $k$ ?

1. Draw a graph  $G$  with six vertices such that the instance  $\langle G, 3 \rangle$  gets a yes answer.
2. Draw a graph  $G$  with six vertices such that the instance  $\langle G, 1 \rangle$  gets a yes answer.

**Problem 19.3.** Let  $G = (V, E)$  be an undirected graph, and let  $W \subseteq V$ . Prove that each of the following statements implies the other.

1.  $W$  is a vertex cover for  $G$
2.  $(V - W)$  is an independent set for  $G$ .
3.  $(V - W)$  is a clique in  $\tilde{G}$

**Problem 19.4.** Prove that any one of these problems is  $\leq_p$ -reducible to each of the others:

VertexCover    IndependentSet    Clique

**Problem 19.5.** Suppose  $f$  and  $g$  are functions from  $\Sigma^*$  to  $\Sigma^*$  which each run in polynomial time. Prove that the composition  $(f \circ g)$  runs in polynomial time. Be rigorous.

**Problem 19.6.** Prove that if  $A \leq_p B$  and  $B \leq_p C$  then  $A \leq_p C$ . If you do not find yourself using the result of Problem 19.5 you are doing something wrong.

**Problem 19.7.** Some elementary facts about  $\leq_p$ :

1. There is only one set  $A$  such that  $A \leq_p \Sigma^*$ . What is it? (Prove your answer).

2. There is only one set  $A$  such that  $A \leq_p \emptyset$ . What is it? (Prove your answer).

**Problem 19.8.** Prove or disprove by counterexample. If  $A \leq_p B$  then  $\bar{A} \leq_p \bar{B}$ .

**Problem 19.9.** Let  $A$  and  $B$  be languages, neither of which is  $\emptyset$  and neither of which is  $\Sigma^*$ . Prove that if  $A \in \mathbf{P}$  then  $A \leq_p B$ .

Note that we do not assume anything about the complexity of membership in  $B$ .

**Problem 19.10.** Prove that if  $A \leq_p B$  and  $B \in NP$  then  $A \in NP$ .

**Problem 19.11.** Prove or give a specific counterexample: If  $A \subseteq B$  and  $B$  is **NP**-complete then  $A$  is **NP**-complete..

## 20 NP-completeness of satisfiability

**Problem 20.1.** Tell whether each of the following formulas of propositional logic are satisfiable.

1.  $(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y)$
2.  $(x \wedge y) \vee (x \wedge \neg y) \vee (\neg x \wedge y) \vee (\neg x \wedge \neg y)$
3.  $(x \wedge (\neg x \vee y)) \wedge (\neg x \vee y)$

**Problem 20.2.** Show that the following problem is in **P**.

INPUT: a Boolean formula  $\alpha$  in *disjunctive* normal form

QUESTION: is  $\alpha$  satisfiable?

**Problem 20.3.** Show that the following problem is in **P**.

INPUT: a Boolean formula  $\alpha$  in *conjunctive* normal form

QUESTION: is  $\alpha$  a tautology?

Recall that a tautology is a propositional logic formula which is true under *every* truth assignment.

**Problem 20.4.** TAUT is the following problem

INPUT: a propositional logic formula 'a' in *disjunctive* normal form

QUESTION: is 'a' a tautology?

Prove that  $\text{TAUT} \leq_p \overline{\text{SAT}}$  and  $\overline{\text{SAT}} \leq_p \text{TAUT}$ .

**Problem 20.5.** Let  $\phi$  be a Boolean formula in conjunctive normal form. Let  $x$  be any literal. Define the operation  $\text{bind}_x$  on  $\phi$  as follows. The formula  $\text{bind}_x(\phi)$  is constructed by

1. remove every clause which contains the literal  $x$ ; then
2. delete  $\bar{x}$  (that is, "not  $x$ ") from every remaining clause.

If we ever generate the empty clause, which can happen in step (b) above in case some clause of  $\phi$  consists of  $x$  alone, then we say that  $\text{bind}_x(\phi)$  FAILS, otherwise it SUCCEEDS.

Prove that  $\phi$  is satisfiable if and only if either  $\text{bind}_x(\phi)$  succeeds and is satisfiable or  $\text{bind}_{\bar{x}}(\phi)$  succeeds and is satisfiable.

**Problem 20.6.** 3SAT is NP-complete. In this problem we ask the obvious question: what's special about 3?

Let 2SAT be the following problem.

*Input:* A Boolean formula  $\phi$  in conjunctive normal form in which each clause has no more than 2 literals

*Question:* Is  $\phi$  satisfiable?

Prove that 2SAT is in **P**. Give careful pseudocode, and carefully defend the claim that your algorithm runs in polynomial time.

*Hint.* This requires some creativity. One solution is based on judicious use of the *bind* operator defined in problem 20.5.

Another approach is based on graph theory. First note that the clause  $(x \vee y)$  is logically equivalent to the implication  $\bar{x} \rightarrow y$ . This is suggestive: consider the directed graph whose nodes are literals and whose edges are determined by the clauses in  $\phi$ . Show that the unsatisfiability of  $\phi$  is equivalent to the existence of certain paths in this graph, and that one can test for such paths in polynomial time.

Be sure to justify that your algorithm is correct!

## 21 More on NP-completeness

**Problem 21.1.** Spend at least 10 minutes browsing the following url:

[http://en.wikipedia.org/wiki/List\\_of\\_NP-complete\\_problems](http://en.wikipedia.org/wiki/List_of_NP-complete_problems)

**Problem 21.2.** Explain the difference between *NP*-hard and *NP*-complete.

**Problem 21.3.** Recall that 3SAT is **NP**-complete. Also recall that in class we showed that  $3\text{SAT} \leq_p \text{Clique}$ .

Prove that each of VertexCover, IndependentSet, and Clique is **NP**-complete.

**Problem 21.4.** Be able to show that SAT is **NP**-complete (by showing that  $3\text{SAT} \leq_p \text{SAT}$ ).

**Problem 21.5.** Be able to show that IntegerLinearProgramming is **NP**-complete (by showing that  $3\text{SAT} \leq_p \text{IntegerLinearProgramming}$ ).

**Problem 21.6.** Consider the variation on the SubsetSum problem in which integers are represented in *unary* notation; call this problem UnarySubsetSum. Prove that UnarySubsetSum is in **P**.

*Hint.* Dynamic programming.

**Problem 21.7.** Most researchers believe that **NP** is not closed under complement, but so far no one can prove it.

Suppose you read in the New York Times tomorrow that someone has finally proven that **NP** is not closed under complement. Could you conclude anything about whether **P** and **NP** are equal? Explain.

**Problem 21.8.** Most researchers believe that not all languages in **NP** are **NP**-complete, but so far no one can prove it. A prime candidate for being non-**NP**-complete is Graph Isomorphism, the problem of deciding whether two given graphs are isomorphic. It is easy to see that Graph Isomorphism is in **NP**, but no one has shown that Graph Isomorphism is **NP**-hard.

Suppose you read in the New York Times tomorrow that someone has finally proven that Graph Isomorphism is not **NP**-complete. Could you conclude anything about whether **P** and **NP** are equal? Explain. (You do not need to know anything about graphs or isomorphisms to solve this problem.)

**Problem 21.9.** Suppose you read in the New York Times tomorrow that someone has finally proven that every problem in **NP** is **NP**-complete. Could you conclude anything about whether **P** and **NP** are equal? Explain.

**Problem 21.10.** Assuming HamCycle is NP-complete, prove that the TravelingSalesperson problem (TSP) NP-complete.

*Hint.* First show that TSP is in NP. Then show that HamCycle  $\leq_p$  TSP. Note that TSP is a problem about undirected graphs where all pairs of vertices are connected and all edges have weights, while in HamCycle not all vertices are connected and the edges have no weights.

**Problem 21.11.** Here's a reasonable complaint about what we're doing these days: "All this stuff about decision procedures doesn't really help me do what I want. To know *there exists* a solution to some problem isn't enough. I want to *actually find* solutions."

Fair enough. In this problem we show that for CNF-SAT, if you could somehow *decide* it quickly then in fact you can find solutions quickly. (By the way this phenomenon holds for most problems in **NP**.)

Suppose that  $R$  were an algorithm which could decide CNF-SAT in polynomial time. Show how to use  $R$  to construct satisfying assignments for satisfiable formulas in polynomial time. Give careful pseudocode, and carefully defend the claim that your algorithm runs in polynomial time.

*Hint.* Use problem 20.5.

**Problem 21.12.** More, in the spirit of Problem 21.11.

1. Let's look at Hamiltonian paths. Suppose **P** = **NP**. Build a PTIME algorithm which will construct a Hamiltonian path whenever one exists. Do a precise big-Oh analysis of your algorithm's complexity. *Hint.* Suppose  $R$  were a PTIME algorithm for deciding whether a graph  $G$  had a Hamiltonian path. Use  $R$  to build your path incrementally. For the complexity analysis, assume  $R$  runs in time  $O(n^k)$  then say what exponent your algorithm has. Be careful to explain what  $n$  measures!
2. Let's look at factoring. This one is a little harder. Suppose **P** = **NP**. Build a PTIME algorithm which will construct factors of an integer  $n$  whenever it is composite. Do a precise big-Oh analysis of your algorithm's complexity (assume of course that integers are represented in binary).

*Hint.* Think about the following language.

$$F = \{ \langle n, a, b \rangle \mid n \text{ has a factor between } a \text{ and } b \}$$

Argue that  $F \in \mathbf{NP}$ . Then, under the assumption **P** = **NP**, you may postulate a PTIME algorithm  $R$  for  $F$ . Use this to find factors.

Note that if there is a PTIME algorithm to find one factor of any composite number then there is a PTIME algorithm to find all factors of any composite number (do you see why?). So you may take your task in this problem to “find one factor of a given input number, if it is not prime.”

3. Make up another example for yourself. Take an **NP** decision problem which underlies a real computation with results that one might want to do and show that, if **P = NP** then we can construct the results efficiently.