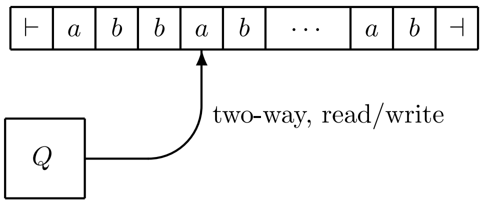
Homework 8

1. Describe a TM that accepts the set $\{a^n \mid n \text{ is a power of } 2\}$. Your description should be at the level of the descriptions in Lecture 29 of the TM that accepts $\{ww \mid w \in \Sigma^*\}$ and the TM that implements the sieve of Eratosthenes. In particular, do not give a list of transitions.

2. A *linear bounded automaton* (LBA) is exactly like a one-tape Turing machine, except that the input string $x \in \Sigma^*$ is enclosed in left and right endmarkers $\vdash$ and $\dashv$ which may not be overwritten, and the machine is constrained never to move left of the $\vdash$ nor right of the $\dashv$. It may read and write all it wants between the endmarkers.



   (a) Give a rigorous formal definition of deterministic linearly bounded automata, including a definition of configurations and acceptance. Your definition should begin as follows: "A *deterministic linearly bounded automaton (LBA)* is a 9-tuple

   $$M = (Q,\ \Sigma,\ \Gamma,\ \vdash,\ \dashv,\ \delta,\ s,\ t,\ r),$$

   where Q is a finite set of *states*, ... "

   (b) Let M be a linear bounded automaton with state set Q of size k and tape alphabet Γ of size m. How many possible configurations are there on input x, $|x| = n$?

   (c) Argue that the halting problem for deterministic linear bounded automata is decidable. (*Hint*: You need to be able to detect after a finite time if the machine is in an infinite loop. Presumably the result of part (b) would be useful here.)

   (d) Prove by diagonalization that there exists a recursive set that is not accepted by any LBA.

3. Let A be any regular set. Show that the set

   $$\{x \mid \exists y\ |y| = |x|^2 \text{ and } xy \in A\}$$

   is regular.

Homework 9

1. Prove that the following question is undecidable. Given a Turing machine M and state q of M, does M ever enter state q on some input? (This problem is analogous to the problem of identifying *dead code*: given a PASCAL program containing a designated block of code, will that block of code ever be executed?)

2. Prove that it is undecidable whether two given Turing machines accept the same set. (This problem is analogous to determining whether two given PASCAL programs are equivalent.)

3. Prove that the emptiness problem for deterministic linearly bounded automata (i.e., whether $L(M) = ?$) is undecidable. (*Hint*: Think VALCOMPS.)

? should be the empty set symbol

4. Prove that an r.e. set is recursive iff there exists an enumeration machine that enumerates it in increasing order.

5. For $A, B \subseteq \Sigma^*$, define

$$\begin{aligned} A/B &\overset{\text{def}}{=} \{x \mid \exists y \in B \; xy \in A\}, \\ A \leftarrow B &\overset{\text{def}}{=} \{x \mid \forall y \in B \; xy \in A\}. \end{aligned}$$

   (a) Show that if A is regular and B is any set whatsoever, then A/B and $A \leftarrow B$ are regular.

   (b) Show that even if we are given a finite automaton for A and a Turing machine for B, we cannot necessarily construct an automaton for A/B or $A \leftarrow B$ effectively.

Homework 10

1. Show that neither the set

   $$\mathrm{TOTAL} \overset{\mathrm{def}}{=} \{M \mid M \text{ halts on all inputs}\}$$

   nor its complement is r.e.

2. Consider one-tape Turing machines that are constrained not to overwrite the input string. They may write all they want on the blank portion of the tape to the right of the input string.

   (a) Show that these machines accept only regular sets. (If you're thinking, "Hey, why not just copy the input string out to the blank portion of the tape," think again ...)

   (b) Show that, despite (a), it is impossible to construct an equivalent finite automaton effectively from such a machine.

3. Show that it is undecidable whether the intersection of two CFLs is nonempty. (*Hint*: Use a variant of VALCOMPS in which every other configuration is reversed:

   $$\alpha_0\#\mathbf{rev}\,\alpha_1\#\alpha_2\#\mathbf{rev}\,\alpha_3\#\cdots\#\alpha_n.$$

   Express this set as the intersection of two CFLs.)

Homework 11

1. Recursive enumerability is intimately linked with the idea of *unbounded existential search.* Often an algorithm for accepting an r.e. set can be characterized in terms of searching for a *witness* or *proof* that a given input x is in the set.

   A binary relation R on strings over $\{0,1\}$ is called *recursive* if the set

   $$\{x\#y \mid R(x,y)\}$$

   is a recursive set. Here # is just another input symbol different from 0 or 1.

   Show that a set $A \subseteq \{0,1\}^*$ is r.e. if and only if there exists a recursive binary relation R such that

   $$A \quad = \quad \{x \in \{0,1\}^* \mid \exists y \; R(x,y)\}.$$

2. Show that it is undecidable whether the intersection of two given CFLs is again a CFL. (*Hint*: Use Homework 10, Exercise 3.)

Miscellaneous Exercises

Turing Machines and Effective Computability

96. Give a Turing machine with input alphabet $\{a\}$ that on input a^m halts with a^{m^2} written on its tape. Describe the operation of the machine both informally and formally. Be sure to specify all data.

97. The Euclidean algorithm computes the greatest common divisor (GCD) of two nonnegative integers:

```
procedure gcd(m,n):
if n = 0 then return(m)
         else return(gcd(n,m mod n))
```

Give a Turing machine with input alphabet $\{a, \#\}$ that on input $a^m\#a^n$ halts with $a^{\mathrm{gcd}(m,n)}$ written on its tape. Describe the operation of the machine both informally and formally. Be sure to specify all data.

98. Prove that the class of r.e. sets is closed under union and intersection.

99. A *queue machine* is like a Turing machine, except that it has a queue instead of a tape. It has a finite queue alphabet Γ and a finite input alphabet $\Sigma \subseteq \Gamma$. If $x \in \Sigma^*$ is the input, the machine starts in its start state s with $x\$$ in the queue, where $\$$ is a special symbol in

$\Gamma - \Sigma$. In each step, it removes a symbol from the front of the queue. Based on that symbol and the current state, it pushes a string $z \in \Gamma^*$ onto the back of the queue and enters a new state according to the transition function δ. It accepts by emptying its queue.

^{S}(a) Give a rigorous formal definition of these machines, including a definition of configurations and acceptance. Your definition should begin as follows: "A *queue machine* is a sextuple

$$M = (Q,\ \Sigma,\ \Gamma,\ \$,\ \delta,\ s),$$

where ... "

^{***HS}(b) Prove that queue machines and Turing machines are equivalent in power.

100. A *one-counter automaton* is an automaton with a finite set of states Q, a two-way read-only input head, and a separate counter that can hold any nonnegative integer. The input $x \in \Sigma^*$ is enclosed in endmarkers $\vdash, \dashv \notin \Sigma$, and the input head may not go outside the endmarkers. The machine starts in its start state s with its counter empty and with its input head pointing to the left endmarker $\vdash$. In each step, it can test its counter for zero. Based on this information, its current state, and the symbol its input head is currently reading, it can either add one to its counter or subtract one, move its input head either left or right, and enter a new state. It accepts by entering a distinguished final state t.

(a) Give a rigorous formal definition of these machines, including a definition of acceptance. Your definition should begin as follows: "A *one-counter automaton* is a septuple

$$M = (Q,\ \Sigma,\ \vdash,\ \dashv,\ s,\ t,\ \delta),$$

where ... "

*(b) Prove that the membership problem for deterministic one-counter automata is decidable: given M and x, does M accept x?

^{H}(c) Prove that the emptiness problem is undecidable: given a one-counter automaton M, is $L(M) = ?$?

the first ? should be the empty set symbol

101. A *ray automaton* consists of an infinite number of deterministic finite automata $A_0, A_1, A_2, \ldots$ arranged in a line. The automata all have the same set of states Q, the same start state s, and the same transition function δ except A_0, which has a different transition function δ_0 since it has no left neighbor. They all start simultaneously in their initial state s and execute synchronously. In each step, each A_i moves

to a new state, depending on its own current state and the current states of its immediate left and right neighbors, according to its transition function. The ray automaton is said to *halt* if A_0 ever enters a distinguished final state t. There is no input alphabet.

(a) Give a rigorous formal definition of ray automata, including a definition of execution and halting. Your definition should begin as follows: "A *ray automaton* is a quintuple

$$\mathcal{A} \quad = \quad (Q,\ s,\ t,\ \delta_0,\ \delta),$$

where Q is a finite set of *states*, ... "

(b) Prove that the halting problem for ray automata is undecidable.

(c) Is the halting problem for ray automata semidecidable? Why or why not?

102. A *deterministic two-dimensional Turing machine* is like a Turing machine except that instead of a one-dimensional tape it has a two-dimensional tape that is like a chessboard, infinite in all directions. It has a finite input alphabet Σ and a finite tape alphabet Γ containing Σ as a subset. If $x \in \Sigma^*$ is the input, $|x| = n$, the machine starts in its start state s with x written in tape cells $(0,1)$, $(0,2)$, ... , $(0,n)$, the origin $(0,0)$ containing a special symbol $O \in \Gamma - \Sigma$, and all other cells (i,j) containing a special blank symbol $xy \in \Gamma - \Sigma$. It has a read/write head initially pointing to the origin. In each step, it reads the symbol of Γ currently occupying the cell it is scanning. Depending on that symbol and the current state of the finite control, it writes a symbol of Γ on that cell, moves one cell either north, south, east, or west, and enters a new state, according to its transition function δ. It accepts its input by erasing the entire board; that is, filling all cells with xy.

the weird xy symbol should be the blank symbol

[H](a) Give a rigorous formal definition of these machines, including a definition of configurations, the next configuration relation, and acceptance. Try to be as precise as possible. Your definition should begin as follows: "A *two-dimensional Turing machine* is a 7-tuple

$$M \quad = \quad (Q,\ \Sigma,\ \Gamma,\ xy,\ O,\ s,\ \delta),$$

where Q is a finite set of *states*, ... "

(b) Argue that two-dimensional Turing machines and ordinary Turing machines are equivalent in the sense that each can simulate the other. Describe the simulations *informally* (i.e., no transitions) but in sufficient detail that transitions implementing your description could readily be written down.

103. A nondeterministic Turing machine is one with a multiple-valued transition relation. Give a formal definition of these machines. Argue that every nondeterministic TM can be simulated by a deterministic TM.

104. Show that the type 0 grammars (see Lecture 36) generate exactly the r.e. sets.

105. For $A, B \subseteq \Sigma^*$, define

$$A/B \overset{\text{def}}{=} \{x \in \Sigma^* \mid \exists y \in B\ xy \in A\}.$$

(a) Show that if A and B are r.e., then so is A/B.

$^{*\mathrm{H}}$(b) Show that every r.e. set can be represented as A/B with A and B CFLs.

106. Is it decidable, given $M\#y$, whether the Turing machine M ever writes a nonblank symbol on its tape on input y? Why or why not?

107. Is it decidable for TMs M whether $L(M) = \mathbf{rev}\ L(M)$? Give proof.

108. Tell whether the following problems are decidable or undecidable. Give proof.

(a) Given a TM M and a string y, does M ever write the symbol # on its tape on input y?

(b) Given a CFG G, does G generate all strings except ϵ?

(c) Given an LBA M, does M accept a string of even length?

(d) Given a TM M, are there infinitely many TMs equivalent to M?

109. Tell whether or not the following sets are r.e. Give proof.

(a) $\{(M, N) \mid M$ takes fewer steps than N on input $\epsilon\}$

(b) $\{M \mid M$ takes fewer than 481^{481} steps on some input$\}$

(c) $\{M \mid M$ takes fewer than 481^{481} steps on at least 481^{481} different inputs$\}$

(d) $\{M \mid M$ takes fewer than 481^{481} steps on all inputs$\}$

110. Show that the set $\{M \mid M$ accepts at least 481 strings$\}$ is r.e. but not co-r.e.

111. One of the following sets is r.e. and the other is not. Which is which? Give proof for both.

     (a) $\{M \mid L(M) \text{ contains at least 481 elements}\}$
     (b) $\{M \mid L(M) \text{ contains at most 481 elements}\}$

112. Show that the set

     $$\{M \mid M \text{ halts on all inputs of length less than 481}\}$$

     is r.e., but its complement is not.

S113. Let M range over Turing machine descriptions. Show that neither the set

     $$\text{REG} \overset{\text{def}}{=} \{M \mid L(M) \text{ is a regular set}\}$$

     nor its complement is recursively enumerable.

114. Let $|M|$ denote the length of the description of the Turing machine M. Are the following problems decidable? Give proof.

     (a) Does a given Turing machine M take at least $|M|$ steps on some input?
     (b) ... on all inputs?

115. Tell whether the following problems are decidable or undecidable, and give proof:

     (a) whether a given TM runs for at least 481^{481} steps on input a^{481};
     (b) whether a given TM ever reenters its start state on any input;
     *(c) whether a given Turing machine will ever move its head left more than ten times on input a^{481};
     *(d) whether a given Turing machine will ever print more than 481 nonblank symbols on input a^{481}.

116. Think for two minutes about why the following problems are undecidable, but don't write anything down:

     (a) whether two given C++ programs compute the same function;
     (b) whether a given C++ program will ever get into an infinite loop on some input;

(c) whether a given μ-recursive function is total;

(d) whether a given λ-term reduces to normal form.

117. Show that the following problems of pairs of Turing machines are undecidable:

 (a) whether $L(M) = L(N)$;

 (b) whether $L(M) \subseteq L(N)$;

 (c) whether $L(M) \cap L(N) = ?$;

 (d) whether $L(M) \cap L(N)$ is a recursive set;

 (e) whether $L(M) \cap L(N)$ is finite.

? should be the empty set symbol

**118. Formalize and prove the following extension of Rice's theorem that has the results of Exercise 117 as special cases: every nontrivial property of *pairs* of r.e. sets is undecidable.

119. Let G and G' denote context-free grammars over $\{a, b\}$. Prove that the following problems are undecidable:

 H(a) whether $L(G) = L(G')$;

 (b) whether $L(G) \subseteq L(G')$;

 *(c) whether $L(G) = L(G)L(G)$.

120. One of the following problems is decidable and the other is not. Which is which? Give proof for both.

 (a) Given a CFL L and a regular set R, is $L \subseteq R$?

 (b) Given a CFL L and a regular set R, is $R \subseteq L$?

H121. Prove that the following problems are undecidable:

 (a) whether a given CFL is a DCFL;

 (b) whether the intersection of two given CFLs is a CFL;

 (c) whether the complement of a given CFL is a CFL;

 **(d) whether the union of two given DCFLs is a DCFL.

122. Prove that it is undecidable whether a given LBA halts on all inputs.

H123. Show that the finiteness problem for Turing machines reduces to the finiteness problem for LBAs.

124. Prove that it is undecidable whether a given LBA accepts a regular set.

125. Consider the following context-sensitive productions.

$$\begin{aligned} S &\rightarrow bSb, \\ S &\rightarrow AcA, \\ Ab &\rightarrow A, \\ Ab &\rightarrow b, \\ bA &\rightarrow b, \\ bA &\rightarrow A. \end{aligned}$$

Let G be the grammar given by all the rules except for the last, and let G' the grammar given by all the rules including the last. One of $L(G)$ and $L(G')$ is regular, and the other is context-free but not regular. Which is which, and why?

126. Give a set over a *single letter alphabet* in each of the following classes, or explain why such a set does not exist:

(a) regular;
(b) DCFL but not regular;
(c) CFL but not DCFL;
(d) recursive but not CFL;
(e) r.e. but not recursive;
(f) not r.e.

127. Prove that every infinite regular set contains a non-r.e. subset.

$^{*\mathrm{H}}$128. Prove that every infinite r.e. set contains an infinite recursive subset.

*129. In this exercise we will prove a kind of fixpoint theorem for Turing machines known as the *recursion theorem.*

If M is a TM, let $M(x)$ denote the contents of M's tape at the point that M halts on input x, provided M does indeed halt on input x. If M does not halt on input x, then $M(x)$ is undefined.

A partial function $\sigma : \Sigma^* \rightarrow \Sigma^*$ is said to be a *computable function* if $\sigma(x) = M(x)$ for some Turing machine M. In addition, σ is a *total computable function* if M is total.

Let M_x be the TM whose encoding over Σ^* is x.

Theorem **(Recursion theorem)** *Let $\sigma : \Sigma^* \to \Sigma^*$ be any total computable function. Then there exists a string u such that*

$$L(M_u) \quad = \quad L(M_{\sigma(u)}).$$

(a) Let $\sigma : \Sigma^* \to \Sigma^*$ be a given total computable function, say computable by a total TM K. Let N be a TM that on input x computes a *description* of a machine that does the following on input y:

- constructs M_x;
- runs M_x on input x;
- if it halts, runs K on $M_x(x)$;
- interprets the result of that computation, $K(M_x(x))$, as the description of a TM, and simulates that TM on the original input y, accepting or rejecting as that machine accepts or rejects, respectively.

Argue that N is total and that

$$L(M_{N(x)}) \quad = \quad L(M_{\sigma(M_x(x))}).$$

(b) Let v be a description of the machine N; that is, $N = M_v$. Argue that $N(v)$ is the desired fixpoint of σ.

^{H}130. Give a short proof of Rice's theorem using the recursion theorem (see Miscellaneous Exercise 129).

^{**H}131. A TM is *minimal* if it has the fewest states among all TMs that accept the same set. Prove that there does not exist an infinite r.e. set of minimal TMs.

132. ^{H}(a) Show that there does not exist an r.e. list of Turing machines such that every machine on the list is total (i.e., halts on all inputs) and every recursive set is represented by some machine on the list.

^{**H}(b) Show that there exists an r.e. list of Turing machines such that every machine on the list accepts a recursive set and every recursive set is represented by some machine on the list.

**133. In addition to the usual constructs of **while** programs (simple assignment, conditional, while loop, sequential composition), add a print statement

print x **and halt**

that prints the current value of a variable and halts. Call two programs *equivalent* if for all initial values of the variables, one program halts iff the other does, and whenever they both halt, they print the same value.

One of the following problems is decidable and the other is undecidable. Which is which? Justify your answers.

(a) Given a program, does there exist an equivalent program with at most one **while** loop?

(b) Given a program, does there exist an equivalent program with no **while** loops?

134. This question is for those who know something about propositional logic. A *propositional Horn clause* is a disjunction of literals with at most one positive literal. The clause

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_n \vee Q$$

is often written as

$$P_1 \wedge P_2 \wedge \cdots \wedge P_n \quad \to \quad Q,$$

and the clause

$$\neg P_1 \vee \neg P_2 \vee \cdots \vee \neg P_n$$

can be written as

$$P_1 \wedge P_2 \wedge \cdots \wedge P_n \quad \to \quad \perp,$$

where $\perp$ denotes falsity. Any single positive literal Q is also a Horn clause.

(a) Show that the emptiness problem for context-free languages (i.e., given a context-free grammar G, deciding whether $L(G) = \textbf{?}$) reduces to the satisfiability problem for finite conjunctions of Horn clauses, and vice versa.

? should be the empty set symbol

(b) Since the satisfiability of propositional formulas is decidable, what can we conclude about the decidability of the emptiness problem for CFLs?

H135. Show that the finiteness problem for regular sets and context-free languages (i.e., whether a given machine/grammar accepts/generates a finite set) is decidable.

136. Show that FIN $\leq_T$ REG. In other words, suppose you are given an oracle that will always answer questions of the form "Is $L(M)$ a regular set?" truthfully. Show how to use such an oracle to decide questions of the form "Is $L(M)$ finite?"

**137. Prove Theorem J.1.

^{*H}138. Let $\mathrm{HP}_1 \stackrel{\mathrm{def}}{=} \mathrm{HP}$, and let HP_{n+1} be the halting problem for oracle Turing machines with oracle HP_n, $n \geq 1$; that is,

$$\begin{aligned}\mathrm{HP}_{n+1} \stackrel{\mathrm{def}}{=} \{M\#x \mid\ & M \text{ is an oracle TM with oracle } \mathrm{HP}_n,\\ & M \text{ halts on input } x\}.\end{aligned}$$

The oracle need not be represented in the description of the oracle machine M. Show that $\mathrm{HP}_n \in \Sigma_n^0 - \Pi_n^0$.

139. Show that the integer square root function is primitive recursive. On input n, the function should return the greatest integer less than or equal to the square root of n.

^{H}140. A language B is said to be *computable in linear time* if there exists a deterministic Turing machine M and a constant $c > 0$ such that $L(M) = B$ and M always halts within cn steps on inputs of length n. Show that there exists a recursive set that is not computable in linear time.

141. Show that the Turing reducibility relation $\leq_T$ is reflexive and transitive and that $\leq_m$ refines $\leq_T$.

142. Prove that the following sets are $\leq_m$-complete for the given classes:

(a) EMPTY is $\leq_m$-complete for Π_1^0;

*(b) TOTAL is $\leq_m$-complete for Π_2^0;

**(c) COF is $\leq_m$-complete for Σ_3^0;

**(d) the set

$$\mathrm{REG} \stackrel{\mathrm{def}}{=} \{M \mid L(M) \text{ is a regular set}\}$$

is $\leq_m$-complete for Σ_3^0.

^{*H}143. Prove that there exists a total computable function $f : \mathbf{N} \to \mathbf{N}$ that is not provably total in Peano arithmetic.

70. Prove inductively that

$$\begin{aligned} S \xrightarrow[G]{*} x &\iff \#a(x) = 2\#b(x), \\ A \xrightarrow[G]{*} x &\iff \#a(x) = 2\#b(x) + 1, \\ B \xrightarrow[G]{*} x &\iff \#a(x) = 2\#b(x) - 2. \end{aligned}$$

Think about the graph of the function $\#a(y) - 2\#b(y)$ for prefixes y of x.

79. Everything you need can be found in Lecture 10.

80. Use Parikh's theorem (Theorem H.1) and the theorem on ultimate periodicity (Theorem 12.3).

83. Use Parikh's theorem (Theorem H.1) and the fact that the complement of a semilinear subset of $\mathbf{N}^k$ is semilinear.

86. Show that the condition (P) of the pumping lemma for regular sets given in Lecture 11 implies the condition of the pumping lemma for context-free languages given in Lecture 22. Give a non-context-free set satisfying (P). A slightly modified version of the hint for Miscellaneous Exercise 43 should do.

87. (c) Build a PDA that pushes and pops antimatter.

90. (d) Consider the set

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

93. (a) Let

$$M = (Q,\ \Sigma,\ \Gamma,\ \delta,\ \bot,\ \dashv,\ s,\ \mathbf{?})$$

be a DPDA for D that accepts by empty stack. Prove that for $p, q \in Q$ and $a \in \Sigma$,

$$\{\gamma \in \Gamma^* \mid (p, a, \gamma) \xrightarrow[M]{*} (q, \epsilon, \epsilon)\}$$

is a regular set.

(b) See the end of Lecture 27.

99. (b) Simulating a queue machine with a Turing machine is easy. The other direction is tricky. Make the queue of the queue machine contain a representation of the configuration of the

Turing machine. The hard part is simulating a left move of the Turing machine. You need to go all the way around the queue. You might try breaking your solution into two steps:

(i) First invent a new kind of Turing machine, a *one-way* Turing machine. These machines can only move right on the tape. When they see the right endmarker, they magically jump back to the left endmarker. Show that one-way machines can simulate ordinary TMs. Simulate a left move of the ordinary TM by pushing a marker all the way around to the right.

(ii) Simulate one-way machines with queue machines.

100. (c) Think VALCOMPS.

102. (a) The infinite checkerboard should be $\mathbf{Z} \times \mathbf{Z}$, where $\mathbf{Z}$ is the set of integers $\{\ldots, -2, -1, 0, 1, 2, 3, \ldots\}$. Tape contents should be modeled by functions $f : \mathbf{Z} \times \mathbf{Z} \to \Gamma$, which assign a tape symbol in Γ to each cell $(i, j) \in \mathbf{Z} \times \mathbf{Z}$.

105. (b) Think VALCOMPS.

119. (a) Take G to be $S \to aS \mid bS \mid \epsilon$.

121. Think VALCOMPS.

123. Think VALCOMPS.

128. Use Exercise 4 of Homework 9.

130. Let P be a nontrivial property of the r.e. sets. Then there exist TMs $M_\top$ and $M_\perp$ such that $P(L(M_\top)) = \top$ and $P(L(M_\perp)) = \perp$. Show that if it were decidable for TMs M whether $P(L(M)) = \top$, then we could construct a total computable map σ with no fixpoint, contradicting the recursion theorem (see Miscellaneous Exercise 129).

131. Use the recursion theorem (see Miscellaneous Exercise 129).

132. (a) Diagonalize.

(b) Let $\leq$ be an arbitrary computable linear order on the set of input strings. Given M, let M' be a machine that on input x simulates M on all $y \leq x$.

135. Use the pumping lemma.

138. Diagonalize.

140. Construct a list of total Turing machines that run in linear time such that every set computable in linear time is accepted by some machine on the list. Build a machine that diagonalizes over this list.

143. Diagonalize.

78. Let P be a PDA for L and M a DFA for R. Build a PDA for L/R that on input x scans x and simulates P, then when it comes to the end of the input x, guesses the string y and continues to simulate P (from the same configuration where it left off) but also runs M simultaneously on the guessed y starting from the start state. It accepts if both L and M accept. Thus it accepts its original input x if it was successfully able to guess a string y such that $xy \in L(P)$ and $y \in L(M)$; that is, if there exists y such that $xy \in L$ and $y \in R$.

Here is an alternative proof using homomorphisms. Suppose the alphabet is $\{a, b\}$. Let $\{a', b'\}$ be another copy of the alphabet disjoint from $\{a, b\}$. Let h be the homomorphism that erases marks; that is, $h(a) = h(a') = a$ and $h(b) = h(b') = b$. Let g be the homomorphism that erases the unmarked symbols and erases the marks on the marked symbols; that is, $g(a) = g(b) = \epsilon$, $g(a') = a$, $g(b') = b$. Then

$$L/R \quad = \quad g(h^{-1}(L) \cap \{a', b'\}^* R). \tag{7}$$

This is a CFL, since CFLs are closed under homomorphic preimage, intersection with regular set, and homomorphic image.

To see (7), first consider the set $h^{-1}(L)$. This is the set of all strings that look like strings in L, except that some of the symbols are marked. Now intersect with the regular set $\{a', b'\}^* R$. This gives the set of strings of the form $x'y$ such that the symbols of x' are marked, those of y are unmarked, $xy \in L$, and $y \in R$. Now apply g to this set of strings. Applied to a string $x'y$ as described above, we would get x. Therefore, the resulting set is the set of all x such that there exists y such that $x'y \in h^{-1}(L) \cap \{a', b'\}^* R$; in other words, such that $xy \in L$ and $y \in R$. This is L/R.

99. (a) A *queue machine* is a sextuple

$$M \quad = \quad (Q,\ \Sigma,\ \Gamma,\ \$,\ s,\ \delta),$$

where

- Q is a finite set of *states*,
- Σ is a finite *input alphabet*,
- Γ is a finite *queue alphabet*,
- $\$ \in \Gamma - \Sigma$ is the *initial queue symbol*,
- $s \in Q$ is the *start state*, and
- $\delta : Q \times \Gamma \to Q \times \Gamma^*$ is the *transition function.*

A *configuration* is a pair $(q, \gamma) \in Q \times \Gamma^*$ giving the current state and current contents of the queue. The *start configuration* on input x is the pair $(s, x\$)$. The *next configuration relation* $\xrightarrow[M]{1}$ is defined as follows: if

$$\delta(p, A) \quad = \quad (q, \gamma),$$

then

$$(p, A\alpha) \quad \xrightarrow[M]{1} \quad (q, \alpha\gamma).$$

The relation $\xrightarrow[M]{*}$ is the reflexive transitive closure of $\xrightarrow[M]{1}$. An *accept configuration* is any configuration of the form (q, ϵ), where $q \in Q$ and ϵ is the null string. The queue machine M is said to *accept* $x \in \Sigma^*$ if

$$(s, x\$) \quad \xrightarrow[M]{*} \quad (q, \epsilon) \qquad \text{for some } q \in Q.$$

(b) To simulate a queue machine U on a Turing machine T, let T maintain the contents of U's queue on its tape and simulate the action of U, shuttling back and forth from the front to the back of the simulated queue. Each simulated step of U consists of T moving to the front of the simulated queue, erasing the first symbol and remembering it in the finite control, then moving to the back of the simulated queue to write symbols. The simulated queue migrates to the right on T's tape, but that's okay, because there's plenty of room. The machine T accepts if its tape ever becomes completely blank, which indicates that the simulated queue of U is empty.

The simulation in the other direction is much harder. Given a Turing machine T, we build a queue machine U that simulates moves of T, using the queue to maintain a description of T's current configuration. We will represent configurations by strings such as

$$\vdash a \ b \ a \ a \ b \ a \ q \ b \ b \ a \ \$$$

for example; exactly one of the symbols is a state of T (q in this example), and its position in the string indicates the position of the tape head of T, which is immediately to the right of the state. If the state is just before the \$, this models T scanning a blank cell to the right of the portion of its tape represented in the configuration.

The queue machine U will also have an "internal queue" that will hold two symbols of this configuration; since this is only a finite amount of information, the internal queue can be encoded in the finite control of U. The remaining portion of the configuration will be held in the external queue. Since configurations have at least three symbols (state, left endmarker,

\$), even if the tape is blank, the external queue will never be prematurely emptied.

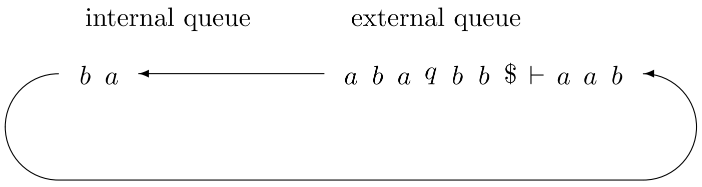
Let x be the input string. The queue machine U starts with $x\$$ in its external queue. It enters a state representing an internal queue of

$s \vdash$

The internal and external queues concatenated together represent the start configuration of T on input x:

$s \vdash x \ \$$

A **rotate** operation consists of rotating the symbols on the internal and external queues as follows:



Formally, this is done by popping the first element off the front of the external queue and pushing the front element of the internal queue onto the back of the external queue, then changing state to reflect the new contents of the internal queue. In this example, after one rotation we would have

$a\ a \qquad\qquad\qquad\qquad b\ a\ q\ b\ b\ \$ \vdash a\ a\ b\ b$

on the internal and external queues, respectively.

We simulate a step of T as follows. If the rightmost symbol of the internal queue is not a state of T, we **rotate** until this becomes true. In this example, after three more steps we would have

$a\ q \qquad\qquad\qquad\qquad b\ b\ \$ \vdash a\ a\ b\ b\ a\ a\ b$

Now we have the symbol that T is scanning (say b) at the head of the external queue and the current state of T (say q) rightmost on the internal queue. We read b at the head of the queue. If b is not \$, we simulate a move of T as follows.

- If $\delta_T(q, b) = (p, c, R)$ and the leftmost symbol of the internal queue is d, we push d onto the back of the external queue and make $(c\,p)$ the new internal queue.
- If $\delta_T(q, b) = (p, c, L)$ and the leftmost symbol of the internal queue is d, we push p onto the back of the external queue and make $(d\,c)$ the new internal queue.

In the example above, this would give

$a\ p \qquad\qquad\qquad\qquad b\ \$ \vdash a\ a\ b\ b\ a\ a\ b\ a$

if $\delta_T(q, b) = (p, a, R)$, and

$a \; a \qquad\qquad\qquad b \; \$ \vdash a \; a \; b \; b \; a \; a \; b \; q$

if $\delta_T(q, b) = (p, a, L)$. If the symbol at the head of the external queue is \$, then this indicates that the tape head of T is scanning a blank symbol to the right of the portion of the tape represented in the configuration. For example,

$b \; q \qquad\qquad\qquad \$ \vdash a \; a \; b \; b \; b \; a \; a \; b \; a \; b$

In this case, when we pop \$ we don't simulate a move of T immediately; first we insert an extra blank symbol xy between the state and the \$. We do this by pushing both symbols in the internal queue onto the back of the external queue and making the new internal queue (xy \$). In this example, the resulting queue contents would be

the weird xy symbol should be the blank symbol

xy \$ $\qquad\qquad\qquad \vdash a \; a \; b \; b \; b \; a \; a \; b \; a \; b \; b \; q$

We continue to simulate moves of T. If T ever enters its accept state, then U goes into a little subroutine that just empties its external queue, thereby accepting.

113. Let

$$\text{REG} \;=\; \{M \mid L(M) \text{ is regular}\}.$$

A corollary of Rice's theorem for r.e. sets (Theorem 34.2) is that any semidecidable property of the r.e. sets is monotone. That is, if P is a property of r.e. sets such that $\{M \mid P(L(M))\}$ is r.e., A and B are r.e. sets, $A \subseteq B$, and $P(A)$, then $P(B)$. Applying this corollary with $P(C)$ = "C is regular," A = **?**, and $B = \{a^n b^n \mid n \geq 0\}$ gives immediately that REG is not r.e. Similarly, applying the corollary with $P(C)$ = "C is not regular," $A = \{a^n b^n \mid n \geq 0\}$, and $B = \Sigma^*$ gives immediately that REG is not co-r.e.

? should be the empty set symbol