Solutions for Section 2.2

Exercise 2.2.1(a)

States correspond to the eight combinations of switch positions, and also must indicate whether the previous roll came out at D, i.e., whether the previous input was accepted. Let 0 represent a position to the left (as in the diagram) and 1 a position to the right. Each state can be represented by a sequence of three 0's or 1's, representing the directions of the three switches, in order from left to right. We follow these three bits by either a indicating it is an accepting state or r, indicating rejection. Of the 16 possible states, it turns out that only 13 are accessible from the initial state, 000r. Here is the transition table:

杠杆可能出现 8 种情况，影响着最终状态。并且也要说明，前面一个大理石球是否从 D 滚出，也就是说，前一个输入是否被接受。令 0 代表向左方的状态（如图表）， 1 代表向右方。这三个杠杆的每一个状态都可以用三个数（0 或 1）组成的序列表示。这个序列后面跟着字母 a 或者 r。a 代表接受状态，r 代表拒绝状态。16 种可能的状态中，只有 13 种是从初始状态 000r 可达的。下面它的有穷自动机的转移表。

	A	**B**
->000r	100r	011r
*000a	100r	011r
*001a	101r	000a
010r	110r	001a
*010a	110r	001a
011r	111r	010a
100r	010r	111r
*100a	010r	111r
101r	011r	100a
*101a	011r	100a
110r	000a	101a
*110a	000a	101a
111r	001a	110a

Exercise 2.2.2

The statement to be proved is δ-hat(q,xy) = δ-hat(δ-hat(q,x),y), and we proceed by induction on the length of y.

证明：通过对|y|进行归纳，来证明$\hat{\delta}(q , xy)=\hat{\delta}(\hat{\delta}(q , x) , y)$ ，具体过程如下：

Basis: If y = ε, then the statement is δ-hat(q,x) = δ-hat(δ-hat(q,x),ε). This statement follows from the basis in the definition of δ-hat. Note that in applying this definition, we must treat δ-hat(q,x) as if it were just a state, say p. Then, the statement to be proved is p = δ-hat(p,ε), which is easy to recognize as the basis in the definition of δ-hat.

基础: $|y|=0$，则 y=ε。那么需证$\hat{\delta}(q,x)=\hat{\delta}(\hat{\delta}(q ,x),ε)$，记 $p=\hat{\delta}(q,x)$,命题变为 $p=\hat{\delta}(p ,ε)$, 由$\hat{\delta}$的定义知这显然成立。

Induction: Assume the statement for strings shorter than y, and break y = za, where a is the last symbol of y. The steps converting δ-hat(δ-hat(q,x),y) to δ-hat(q,xy) are summarized in the following table:

归纳: 假设命题对于比 y 短的串成立, 且 y = za, 其中 a 是 y 的结尾符号。$\hat{\delta}(\hat{\delta}(q,x),y)$ 到$\hat{\delta}(q,xy)$ 的变换总结在下表中:

Expression 表达式	**Reason 原因**
$\hat{\delta}(\hat{\delta}(q,x),y)$	Start 开始
$\hat{\delta}(\hat{\delta}(q,x),za)$	y=za by assumption 由假设 y=za
δ($\hat{\delta}(\hat{\delta}(q,x),z),a)$	Definition of δ-hat, treating δ-hat(q,x) as a state $\hat{\delta}$的定义, 把$\hat{\delta}(q,x)$ 看作是一个状态
δ($\hat{\delta}(q,xz),a)$	Inductive hypothesis 归纳假设
$\hat{\delta}(q,xza)$	Definition of δ-hat $\hat{\delta}$的定义
$\hat{\delta}(q,xy)$	y=za

Exercise 2.2.4(a)

The intuitive meanings of states A, B, and C are that the string seen so far ends in 0, 1, or at least 2 zeros.

状态 A, B，C 分别表示以 1 , 0 和 00 结尾的串的状态。

	0	**1**
->A	B	A
B	C	A
*C	C	A

Exercise 2.2.6(a)

The trick is to realize that reading another bit either multiplies the number seen so far by 2 (if it is a 0), or multiplies by 2 and then adds 1 (if it is a 1). We don't need to remember the entire number seen --- just its remainder when divided by 5. That is, if we have any number of the form 5a+b, where b is the remainder, between 0 and 4, then 2(5a+b) = 10a+2b. Since 10a is surely divisible by 5, the remainder of 10a+2b is the same as the remainder of 2b when divided by 5. Since b, is 0, 1, 2, 3, or 4, we can tabulate the answers easily. The same idea holds if we want to consider what happens to 5a+b if we multiply by 2 and add 1.

对于一个二进制整数，如果读入一个比特 0，其值等于原数乘以 2；否则等于原数乘以 2 再加以 1。而任意一个数均可写成形如 5a+b，其中 a 任意，0<= b <=4，那么输入 0，原数变为 2(5a+b) = 10a+2b，由于 10a 是 5 的倍数，，因此 10a+2b 除以 5 的余数与 2b 相同。输入 1，则得 2(5a+b)+1 类似。因此对于所有的数只要记住它被 5 除的余数就可以。由于 b 是 0, 1, 2, 3 或者 4, 我们可以容易得到该 DPA 的转移表，具体如下：

The table below shows this automaton. State qi means that the input seen so far has remainder i when divided by 5.

其中状态 qi 代表输入串被 5 除的余数 i 的状态。

	0	**1**
->*q0	q0	q1
q1	q2	q3
q2	q4	q0
q3	q1	q2
q4	q3	q4

There is a small matter, however, that this automaton accepts strings with leading 0's. Since the problem calls for accepting only those strings that begin with 1, we need an additional state s, the start state, and an additional ``dead state'' d. If, in state s, we see a 1 first, we act like q0; i.e., we go to state q1. However, if the first input is 0, we should never accept, so we go to state d, which we never leave. The complete automaton is:

但是上述自动机仍接受以 0 开头的字符串。因为题目要求只接受以 1 开头的串，可增加一个初始状态 s 和“死亡状态”d。在状态初始状态 s, 若看到 1 ，则转到状态 q1；若看到 0 , 则直接转到状态 d，识别终止。所求自动机如下:

	0	**1**
->s	d	q1
*q0	q0	q1
q1	q2	q3
q2	q4	q0
q3	q1	q2
q4	q3	q4
d	d	d

Exercise 2.2.9

Part (a) is an easy induction on the length of w, starting at length 1.

Basis: |w| = 1. Then δ-hat(q_0,w) = δ-hat(q_f,w), because w is a single symbol, and δ-hat agrees with δ on single symbols.

Induction: Let w = za, so the inductive hypothesis applies to z. Then δ-hat(q_0,w) = δ-hat(q_0,za) = δ(δ-hat(q_0,z),a) = δ(δ-hat(q_f,z),a) [by the inductive hypothesis] = δ-hat(q_f,za) = δ-hat(q_f,w).

证明：a) 通过对 w 长度的归纳证明。

基础: 若|w| = 1，则 w 是一个符号，此时需证 $\hat{\delta}$(q_0,w) = $\hat{\delta}$(q_f,w), 而对于单个符号扩展转移函数 $\hat{\delta}$ 与转移函数 δ 的作用是一样的，得证。

归纳: 令 w = za, 假设对于 z 命题 $\hat{\delta}$(q_0,z) = $\hat{\delta}$(q_f,z)成立。那么 $\hat{\delta}$(q_0,w) = $\hat{\delta}$(q_0,za) = δ($\hat{\delta}$(q_0,z),a) = δ($\hat{\delta}$ (q_f,z),a) [由归纳假设] = $\hat{\delta}$ (q_f,za) = $\hat{\delta}$ (q_f,w).

For part (b), we know that δ-hat(q_0,x) = q_f. Since xε, we know by part (a) that δ-hat(q_f,x) = q_f. It is then a simple induction on k to show that δ-hat(q_0,x^k) = q_f.

Basis: For k=1 the statement is given.

Induction: Assume the statement for k-1; i.e., δ-hat(q_0,xSUP>k-1) = q_f. Using Exercise 2.2.2, δ-hat(q_0,x^k) = δ-hat(δ-hat(q_0,x^{k-1}),x) = δ-hat(q_f,x) [by the inductive hypothesis] = q_f [by (a)].

b) x 是属于 L(A)的非空串，也即串 x 被接收，因此 $\hat{\delta}(q_0,x) = q_f$ ，则由 a)知 $\hat{\delta}(q_f,x) = \hat{\delta}(q_0,x) = q_f$ 。现在通过对 k 的归纳来证明 $\hat{\delta}(q_0,x^k) = q_f$ 。

基础: k=1 时，需证 $\hat{\delta}(q_0,x) = q_f$ ，由已知可得。

归纳：假设对于 k-1 命题成立，也就是说，$\hat{\delta}(q_0,x^{k-1}) = q_f$ 。由练习 2.2.2，$\hat{\delta}(q_0,x^k) = \hat{\delta}(\hat{\delta}(q_0,x^{k-1}),x) = \hat{\delta}(q_f,x)$ [由归纳假设] = q_f [由(a)]。

Exercise 2.2.10

The automaton tells whether the number of 1's seen is even (state A) or odd (state B), accepting in the latter case. It is an easy induction on |w| to show that dh(A,w) = A if and only if w has an even number of 1's.

Basis: |w| = 0. Then w, the empty string surely has an even number of 1's, namely zero 1's, and $\hat{\delta}(A,w) = A$.

Induction: Assume the statement for strings shorter than w. Then w = za, where a is either 0 or 1.

Case 1: a = 0. If w has an even number of 1's, so does z. By the inductive hypothesis, $\hat{\delta}(A,z) = A$. The transitions of the DFA tell us $\hat{\delta}(A,w) = A$. If w has an odd number of 1's, then so does z. By the inductive hypothesis, δ-hat(A,z) = B, and the transitions of the DFA tell us δ-hat(A,w) = B. Thus, in this case, δ-hat(A,w) = A if and only if w has an even number of 1's.

Case 2: a = 1. If w has an even number of 1's, then z has an odd number of 1's. By the inductive hypothesis, δ-hat(A,z) = B. The transitions of the DFA tell us δ-hat(A,w) = A. If w has an odd number of 1's, then z has an even number of 1's. By the inductive hypothesis, δ-hat(A,z) = A, and the transitions of the DFA tell us δ-hat(A,w) = B. Thus, in this case as well, δ-hat(A,w) = A if and only if w has an even number of 1's.

这个自动机表示，状态 A 表示偶数个 1，状态 B 表示奇数个 1，不管串有偶数个还是奇数个 1，都会被接受。当且仅当串 w 中有偶数个 1 时，$\hat{\delta}(A,w) = A$.。用

归纳法证明如下

基础: |w| = 0。空串当然有偶数个 1 ，即 0 个 1，且$\hat{\delta}$ (A,w) = A.

归纳：假设对于比 w 短的串命题成立。令 w = za, 其中 a 为 0 或 1。

情形 1： a = 0. 如果 w 有偶数个 1, 则 z 有偶数个 1。由归纳假设，$\hat{\delta}$ (A,z) = A。

由转移表的 DFA 知$\hat{\delta}$(A,w) = A.如果 w 有奇数个 1, 则 z 有奇数个 1. 由归纳假设，

$\hat{\delta}$ (A,z) = B, 由转移表的 DFA 知$\hat{\delta}$ (A,w) = B. 因此这种情况下$\hat{\delta}$(A,w) = A 当且仅当 w 有偶数个 1。

情形 2： a = 1. 如果 w 有偶数个 1, 则 z 有奇数个 1。由归纳假设，$\hat{\delta}$ (A,z) = B.

由转移表的 DFA 知$\hat{\delta}$ (A,w) = A. 如果 w 有奇数个 1, 则 z 有偶数个 1。由归纳

假设，$\hat{\delta}$(A,z) = A, 由转移表的 DFA 知$\hat{\delta}$ (A,w) = B. 因此这种情况下$\hat{\delta}$(A,w) = A 当且仅当 w 有偶数个 1.

综合上述情形，命题得证。

Solutions for Section 2.3

Exercise 2.3.1

Here are the sets of NFA states represented by each of the DFA states A through H: A = {p}; B = {p,q}; C = {p,r}; D = {p,q,r}; E = {p,q,s}; F = {p,q,r,s}; G = {p,r,s}; H = {p,s}.
下表就是利用子集构造法将 NFA 转化成的 DFA。其中构造的子集有：A = {p}; B = {p,q}; C = {p,r}; D = {p,q,r}; E = {p,q,s}; F = {p,q,r,s}; G = {p,r,s}; H = {p,s}.

	0	**1**
->A	B	A
B	D	C
C	E	A
D	F	C
*E	F	G
*F	F	G
*G	E	H
*H	E	H

Exercise 2.3.4(a)

The idea is to use a state qi, for i = 0,1,...,9 to represent the idea that we have seen an input i and guessed that this is the repeated digit at the end. We also have state qs, the initial state, and qf, the final state. We stay in state qs all the time; it represents no guess having been made. The transition table:
记状态 qi 为已经看到 i 并猜测 i 就是结尾将要重复的数字，i = 0,1,...,9 。初始状态为 qs，终止状态为 qf。我们可以一直停留在状态 qs，表示尚未猜测。转移表如下：

	0	**1**	**...**	**9**
->qs	{qs,q0}	{qs,q1}	...	{qs,q9}
q0	{qf}	{q0}	...	{q0}
q1	{q1}	{qf}	...	{q1}
...	...	...	...	...
q9	{q9}	{q9}	...	{qf}
*qf	{}	{}	...	{}

Solutions for Section 2.4

Exercise 2.4.1(a)

We'll use q0 as the start state. q1, q2, and q3 will recognize abc; q4, q5, and q6 will recognize abd, and q7 through q10 will recognize aacd. The transition table is:

记 q0 为初始状态。q1, q2 和 q3 识别 abc; q4, q5 和 q6 识别 abd, q7 到 q10 识别 aacd. 转移表如下：

	a	**b**	**c**	**d**
->q0	{q0,q1,q4,q7}	{q0}	{q0}	{q0}
q1	{}	{q2}	{}	{}
q2	{}	{}	{q3}	{}
*q3	{}	{}	{}	{}
q4	{}	{q5}	{}	{}
q5	{}	{}	{}	{q6}
*q6	{}	{}	{}	{}
q7	{q8}	{}	{}	{}
q8	{}	{}	{q9}	{}

q9	{}	{}	{}	{q10}
*q10	{}	{}	{}	{}

Exercise 2.4.2(a)

The subset construction gives us the following states, each representing the subset of the NFA states indicated: A = {q0}; B = {q0,q1,q4,q7}; C = {q0,q1,q4,q7,q8}; D = {q0,q2,q5}; E = {q0,q9}; F = {q0,q3}; G = {q0,q6}; H = {q0,q10}. Note that F, G and H can be combined into one accepting state, or we can use these three state to signal the recognition of abc, abd, and aacd, respectively.

由子集构造法可得以下 DFA 的状态，其中每一个状态都是 NFA 状态的子集： A = {q0}; B = {q0,q1,q4,q7}; C = {q0,q1,q4,q7,q8}; D = {q0,q2,q5}; E = {q0,q9}; F = {q0,q3}; G = {q0,q6}; H = {q0,q10}.注意到 F, G 和 H 可以整合到一个接受状态中，或者我们可以用这三个状态来分别标记已识别 abc, abd 和 aacd。

	a	**b**	**c**	**d**
->A	B	A	A	A
B	C	D	A	A
C	C	D	E	A
D	B	A	F	G
E	B	A	A	H
*F	B	A	A	A
*G	B	A	A	A
*H	B	A	A	A

Solutions for Section 2.5

Exercise 2.5.1

For part (a): the closure of p is just {p}; for q it is {p,q}, and for r it is {p,q,r}.
(a): 根据状态的 ε 闭包的的性质。求得，
p 的 ε 闭包：{p}； q 的 ε 闭包：{p,q}； r 的 ε 闭包：{p,q,r}。

For (b), begin by noticing that a always leaves the state unchanged. Thus, we can think of the effect of strings of b's and c's only. To begin, notice that the only ways to get from p to r for the first time, using only b, c, and ε-transitions are bb, bc, and c. After getting to r, we can return to r reading either b or c. Thus, every string of length 3 or less, consisting of b's and c's only, is accepted, with the exception of the string b. However, we have to allow a's as well. When we try to insert a's in these strings, yet

keeping the length to 3 or less, we find that every string of a's b's, and c's with at most one a is accepted. Also, the strings consisting of one c and up to 2 a's are accepted; other strings are rejected.

b) 由于输入 a 状态总是保持不变，因此只需考虑输入 b 和 c 的情况。可以看出，从状态 p 第一次到 r 且只经过 b，c 和 ε 转移的路径为 bb, bεc 和 c ；到 r 之后，读入 b 仍可回到 r，读入 c 回到 p ，则可通过继续读入串 bb, bc 和 c 回到 r。

因此，每一个由 b 和 c 组成的长度小于等于 3 的串可以被接受，除了串 b 不能接受。向这些串中插入 a，并保持长度小于等于 3，就会得到所有由 a，b，c 组成的，至多含有一个 a 的可被接受的串。由一个 c 和两个 a 组成的任意串也是可以被接受的。其它的串均被拒绝。

There are three DFA states accessible from the initial state, which is the ε closure of p, or {p}. Let A = {p}, B = {p,q}, and C = {p,q,r}. Then the transition table is:

由初始状态，即 p 的 ε 闭包或者{p}，有 3 个状态可以达到。令 A = {p}, B = {p,q}, C = {p,q,r}。转移表如下：

	a	**b**	**c**
->A	A	B	C
B	B	C	C
*C	C	C	C

Solutions for Section 3.1

Exercise 3.1.1(a)

The simplest approach is to consider those strings in which the first a precedes the first b separately from those where the opposite occurs. The expression: **c*a(a+c)*b(a+b+c)* + c*b(b+c)*a(a+b+c)***

首先考虑第一个 a 在第一个 b 的前面，然后再考虑相反的情况。表达式为：**c*a(a+c)*b(a+b+c)* + c*b(b+c)*a(a+b+c)***

Exercise 3.1.2(a)

(Revised 9/5/05) The trick is to start by writing an expression for the set of strings that have no two adjacent 1's. Here is one such expression: **(10+0)*(ε+1)**

To see why this expression works, the first part consists of all strings in which every 1 is followed by a 0. To that, we have only to add the possibility that there is a 1 at the end, which will not be followed by a 0. That is the job of (ε+1).

首先写出没有两个 1 相邻的串的集合，如下：**(10+0)*(ε+1)** 。表达式的第一部分表示每个 1 之后都紧跟一个 0 的这样的串组成。为了表示结尾可能是 1 的情况，则可在串尾处加上 (ε+1)。

Now, we can rethink the question as asking for strings that have a prefix with no adjacent 1's followed by a suffix with no adjacent 0's. The former is the expression we developed, and the latter is the same expression, with 0 and 1 interchanged. Thus, a solution to this problem is **(10+0)*(ε+1)(01+1)*(ε+0)**. Note that the ε+1 term in the middle is actually unnecessary, as a 1 matching that factor can be obtained from the **(01+1)*** factor instead.

题目要求的串可由两部分组成，也就是，前缀没有相邻的 1，后缀没有相邻的 0。前半部分也就是已经给出的**(10+0)*(ε+1)**，根据对称性后半部分可将上式的 1 和 0 交换得到。所求即为**(10+0)*(ε+1)(01+1)*(ε+0)**。注意中间的 ε+1 项没有作用，因为 1 可以由后面的**(01+1)*** 项得到。因此最后得到的正则表达式为 **(10+0)*(01+1)*(ε+0)**

Exercise 3.1.4(a)

This expression is another way to write ``no adjacent 1's." You should compare it with the different-looking expression we developed in the solution to Exercise 3.1.2(a). The argument for why it works is similar. **(00*1)*** says every 1 is preceded by at least one 0. **0*** at the end allows 0's after the final 1, and (ε+**1**) at the beginning allows an initial 1, which must be either the only symbol of the string or followed by a 0. 你可以与练习 3.1.2(a)中我们给出的不同样子的表达式作比较。为什么起作用的原因是类似的。

这个表达式是 “没有相邻的 1”的另一种描述方式。**(00*1)*** 表示每个 1 的前面都至少有一个 0 做前缀。最后的 **0*** 允许在最后一个 1 后面有 0。开头的(ε+**1**)允许初始为 1，要么串就只有这一个符号，要么后面跟着的就是 0。

Exercise 3.1.5

The language of the regular expression ε. Note that ε* denotes the language of strings consisting of any number of empty strings, concatenated, but that is just the set containing the empty string.

正则表达式 ε。ε*表示由任意多个空串组成的串，也是只包含空串的集合。

Solutions for Section 3.2

Exercise 3.2.1

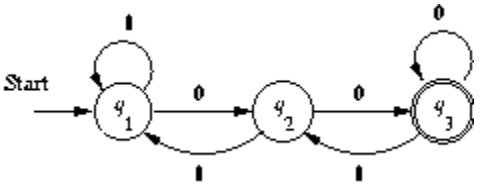
Part (a): The following are all R^0 expressions; we list only the subscripts. R11 = ε+**1**; R12 = **0**; R13 = phi; R21 = **1**; R22 = ε; R23 = **0**; R31 = phi; R32 = **1**; R33 = ε+**0**.

a) 下面就是所有 R^0 的表达式；我们只写出下标： R11 = ε+**1**；R12 = **0**; R13 = Φ (phi); R21 = **1**; R22 = ε; R23 = **0**; R31 = Φ (phi); R32 = **1**; R33 = ε+**0**.

Part (b): Here all expression names are $R^{(1)}$; we again list only the subscripts. R11 = **1***; R12 = **1*0**; R13 = phi; R21 = **11***; R22 = ε+**11*0**; R23 = **0**; R31 = phi; R32 = **1**; R33 = ε+**0**.

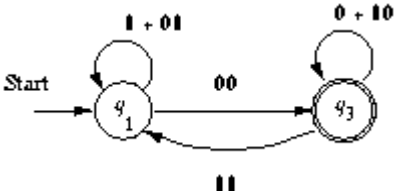
b) 下面就是所有 $R^{(1)}$ 的表达式；我们只写出下标：R11 = **1***; R12 = **1*0**; R13 = phi; R21 = **11***; R22 = ε+**11*0**; R23 = **0**; R31 = phi; R32 = **1**; R33 = ε+**0**.

Part (e): Here is the transition diagram 转移图:



If we eliminate state q2 we get:

如果消除状态 q2，有:

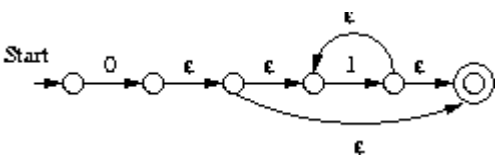


Applying the formula in the text, the expression for the ways to get from q1 to q3 is: **[1 + 01 + 00(0+10)*11]*00(0+10)***

由课本中的公式，q1 到 q3 的正则表达式：**[1 + 01 + 00(0+10)*11]*00(0+10)***

Exercise 3.2.4(a)

利用定理 3。7 每个用正则表达式来定义的语言也可用穷自动机来定义



Exercise 3.2.6(a)

(Revised 修改 1/16/02) LL* or L^+.

Exercise 3.2.6(b)

The set of suffixes of strings in L. (以)L 中串(作为)后缀/下标的集合。

Exercise 3.2.8

Let $R^{(k)}_{ijm}$ be the number of paths from state i to state j of length m that go through no state numbered higher than k. We can compute these numbers, for all states i and j, and for m no greater than n, by induction on k.
令 $R^{(k)}_{ijm}$ 为从状态 i 到状态 j，长度为 m，且没有经过编号大于 k 的路径的个数。对于所有状态 I 和 j，以及 m（m≤n），通过对 k 归纳来计算这个个数。

Basis: R^{0}_{ij1} is the number of arcs (or more precisely, arc labels) from state i to state j. R^{0}_{ii0} = 1, and all other R^{0}_{ijm}'s are 0.

基础: k=0，R^{0}_{ij1} 是由状态 i 到状态 j 的箭弧（更准确的说，是箭弧标号）的个数。R^{0}_{ii0} = 1,其他的 R^{0}_{ijm}'s 都为 0。

Induction: $R^{(k)}_{ijm}$ is the sum of $R^{(k-1)}_{ijm}$ and the sum over all lists (p1,p2,...,pr) of positive integers that sum to m, of $R^{(k-1)}_{ikp1} * R^{(k-1)}_{kkp2} *R^{(k-1)}_{kkp3} *...* R^{(k-1)}_{kkp(r-1)} * R^{(k-1)}_{kjpr}$. Note r must be at least 2.

归纳: $R^{(k)}_{ijm}$ 是 $R^{(k-1)}_{ijm}$ 的和，$R^{(k-1)}_{ikp1} * R^{(k-1)}_{kkp2} *R^{(k-1)}_{kkp3} *...* R^{(k-1)}_{kkp(r-1)} * R^{(k-1)}_{kjpr}$。(p1,p2,...,pr)是所有和为 m 的正整数序列，r 大于等于 2。

The answer is the sum of $R^{(k)}_{1jn}$, where k is the number of states, 1 is the start state, and j is any accepting state.

答案就是 $R^{(k)}_{1jn}$ 的总和，其中 k 是状态个数，1 为开始状态，j 是任意接受状态。

Solutions for Section 3.4

Exercise 3.4.1(a)

Replace R by {a} and S by {b}. Then the left and right sides become {a} union {b} = {b} union {a}. That is, {a,b} = {b,a}. Since order is irrelevant in sets, both languages are the same: the language consisting of the strings a and b.

将 R 替换为{a} ，S 替换为{b}。 等式变为{a} + {b} = {b} + {a}. 也就是 {a,b} = {b,a}. 因为集合中元素的顺序是无关紧要的，所以，等式两边是一样的：由串 a 和 b 构成的语言。

Exercise 3.4.1(f)

Replace R by {a}. The right side becomes {a}*, that is, all strings of a's, including the empty string. The left side is ({a}*)*, that is, all strings consisting of the

concatenation of strings of a's. But that is just the set of strings of a's, and is therefore equal to the right side.

将 R 替换为{a} 。右边变为{a}*, 代表 a 组成的所有串，包含空串。左边是({a}*)*, 代表由 a 组成的串构成的串，也就是由 a 构成的串。当然相等。

Exercise 3.4.2(a)

Not the same. Replace R by {a} and S by {b}. The left side becomes all strings of a's and b's (mixed), while the right side consists only of strings of a's (alone) and strings of b's (alone). A string like ab is in the language of the left side but not the right.

不等。将 R 替换为{a} ，S 替换为{b}。 左边表示所有由 a 和 b（可混合）构成的串。而右边表示只有 a 构成的串和只有 b 构成的串。像 ab 这样的串就只属于左边的语言，而不属于右边。

Exercise 3.4.2(c)

Also not the same. Replace R by {a} and S by {b}. The right side consists of all strings composed of zero or more occurrences of strings of the form a...ab, that is, one or more a's ended by one b. However, every string in the language of the left side has to end in ab. Thus, for instance, ε is in the language on the right, but not on the left.

不等。举反例，将 R 替换为{a} ，S 替换为{b}。右边表示由 0 个或多个形如 a...ab 组成的串，也就是，一个或多个 a 后面紧跟一个结尾的 b。但是，左边的串必须以 ab 结尾。因此，ε 属于右边的语言，但不属于左边。

Solutions for Section 4.1

Exercise 4.1.1(c)

Let n be the pumping-lemma constant (note this n is unrelated to the n that is a local variable in the definition of the language L). Pick w = 0^n10^n. Then when we write w = xyz, we know that |xy| <= n, and therefore y consists of only 0's. Thus, xz, which must be in L if L is regular, consists of fewer than n 0's, followed by a 1 and exactly n 0's. That string is not in L, so we contradict the assumption that L is regular.

令 n 为泵引理常数(这个 n 与语言 L 的定义中的局部变量 n 无关)。设 w = 0^n10^n。把 w 打断为 w = xyz，满足|xy| <= n, 则 y 只由 0 构成。若 L 是正则的，那么 xz 一定在 L 中。但 xz 由少于 n 个 0，后面跟着一个 1 和恰好 n 个 0 构成。这个串不在 L 中。所以 L 不是正则语言。

Exercise 4.1.2(a)

Let n be the pumping-lemma constant and pick w = 0^{n^2}, that is, n^2 0's. When we write

w = xyz, we know that y consists of between 1 and n 0's. Thus, xyyz has length between $n^2 + 1$ and $n^2 + n$. Since the next perfect square after n^2 is $(n+1)^2 = n^2 + 2n + 1$, we know that the length of xyyz lies strictly between the consecutive perfect squares n^2 and $(n+1)^2$. Thus, the length of xyyz cannot be a perfect square. But if the language were regular, then xyyz would be in the language, which contradicts the assumption that the language of strings of 0's whose length is a perfect square is a regular language.

令 n 为泵引理常数，设 $w = 0^{n2}$,也就是 n^2 个 0。 将 w 打断为 w = xyz。y 是由大于 1 小于 n 个 0 组成的。因此 xyyz 的长度介于 $n^2 + 1$ 和 $n^2 + n$ 之间。n^2 的下一个完全平方数是 $(n+1)^2 = n^2 + 2n + 1$。因为 xyyz 的长度介于 n^2 和$(n+1)^2$ 之间。所以 xyyz 的长度不是完全平方数。若语言是正则的，那么 xyyz 也应该是正则的。xyyz 的长度应该是完全平方数。矛盾。所以语言不是正则的。

Exercise 4.1.4(a)

We cannot pick w from the empty language. 我们无法从空集中找到 w ，因为 y 非空。

Exercise 4.1.4(b)

If the adversary picks n = 3, then we cannot pick a w of length at least n.

如果对手选择 n=3，那么我们无法找到长度至少为 n 的 w 。

Exercise 4.1.4(c)

The adversary can pick an n > 0, so we have to pick a nonempty w. Since w must consist of pairs 00 and 11, the adversary can pick y to be one of those pairs. Then whatever i we pick, xy^iz will consist of pairs 00 and 11, and so belongs in the language.

对手选择 n > 0, 我们就要选择一个非空的 w。因为 w 必须由成对的 00 和 11 组成，对手可以选择这些对中的一个。那么不管我们选择什么样的 i ， xy^iz 都由 00 和 11 组成，所以属于语言。

Solutions for Section 4.2

Exercise 4.2.1(a)

aabbaa.

Exercise 4.2.1(c)

The language of regular expression **a(ab)*ba**.

正则表达式语言 a(ab)*ba

Exercise 4.2.1(e)

Each b must come from either 1 or 2. However, if the first b comes from 2 and the second comes from 1, then they will both need the a between them as part of h(2) and h(1), respectively. Thus, the inverse homomorphism consists of the strings {110, 102, 022}.

每个 b 必须来自 1 或 2 。如果第一个 b 来自 2 和第二个来自 1，那么将需要两个 a 在他们之间分别作为 h(2) 和 h(1)的一部分，这种情况不行。因此，逆同态组成的字符串{110, 102, 022} 。

Exercise 4.2.2
Start with a DFA A for L. Construct a new DFA B, that is exactly the same as A, except that state q is an accepting state of B if and only if δ(q,a) is an accepting state of A. Then B accepts input string w if and only if A accepts wa; that is, L(B) = L/a.

首先从 L 的一个 DFA A 出发。我们构造一个 DFA B，满足当且仅当 δ(q,a)是 A 的一个接受状态时，状态 q 才是 B 的一个接受状态。因此，当且仅当 A 接受 wa 时，B 接受输入串 w；即 L(B) = L/a。(DFA 的语言是正则的)

Exercise 4.2.5(b)

We shall use D_a for ``the derivative with respect to a.'' The key observation is that if epsilon is not in L(R), then the derivative of RS will always remove an a from the portion of a string that comes from R. However, if epsilon is in L(R), then the string might have nothing from R and will remove a from the beginning of a string in L(S) (which is also a string in L(RS). Thus, the rule we want is:

我们用 D_a 表示导数 a'。 关键是，如果 ε 不在 L(R)中, 则 RS 的导数总是能从 R 的字符串中去掉 a。如果 ε 在 L(R)中, 则 R 可能是空，这时从 L(S)的字符串头去掉 a . 因此，总的规则就是:

If epsilon is not in L(R), then $D_a(RS) = (D_a(R))S$. Otherwise, $D_a(RS) = D_a(R)S + D_a(S)$.

如果 ε 不在 L(R)，则 $D_a(RS) = (D_a(R))S$. 否则， $D_a(RS) = D_a(R)S + D_a(S)$.

Exercise 4.2.5(e)

L may have no string that begins with 0. L 中没有以 0 开始的字符串.

Exercise 4.2.5(f)

This condition says that whenever 0w is in L, then w is in L, and vice-versa. Thus, L must be of the form L(0*)M for some language M (not necessarily a regular language) that has no string beginning with 0.

这个条件说明如果 0w 在 L 中,则 w 就在 L 中, 反之亦然。因此，L 必须是这样的形式 L(0*)M，其中语言 M 没有包含以 0 开始的串。

In proof, notice first that $D_0(L(0^*)M = D_0(L(0^*))M$ union $D_0(M) = L(0^*)M$. There are two reasons for the last step. First, observe that D_0 applied to the language of all strings of 0's gives all strings of 0's, that is, L(0*). Second, observe that because M has no string that begins with 0, $D_0(M)$ is the empty set [that's part (e)].

证明： 首先 $D_0(L(0^*)M)= (D_0(L(0^*))M$ (union) $D_0(M)) = L(0^*)M$ 。存在两个条件。首先, D_0 是所有由 0 组成的字符串语言的导数得到的由 0 组成的串，即 L(0*)。 其次，由于 M 没有以 0 开头的串, $D_0(M)$ 就是一个空集。

We also need to show that every language N that is unchanged by D_0 is of this form. Let M be the set of strings in N that do not begin with 0. If N is unchanged by D_0, it follows that for every string w in M, 00...0w is in N; thus, N includes all the strings of L(0*)M. However, N cannot include a string that is not in L(0*)M. If x were such a string, then we can remove all the 0's at the beginning of x and get some string y that is also in N. But y must also be in M.

仍需证没有被 D_0 作用的每个语言 N 都满足这种形式。 令 M 是 N 的一个不以 0 开头的子集。若 N is unchanged by D_0, 则对于每一个在 M 中的 w, 00...0w 存在于 N 中； 因此，N 包含了所有 L(0*)M，N 不能包含 L(0*)M 之外的串。如果 x 是这样的串,我们去掉 x 串前面的 0 得到 y，则 y 在 N 中。并且 y 必须存在于 M 中。

Exercise 4.2.8

Let A be a DFA for L. We construct DFA B for half(L). The state of B is of the form [q,S], where:
令 A 是 L 的一个 DFA 。我们为 half(L)构造一个 DFA B 。状态 B 满足形式 [q,S]， 其中：

- q is the state A would be in after reading whatever input B has read so far.
- q 是这样一个状态： A 在读入 B 已经读入的任意输入后将要进入的状态。
- S is the set of states of A such that A can get from exactly these states to an accepting state by reading any input string whose length is the same as the length of the string B has read so far.
- S 是 A 的一个状态集，使得 A 能够通过读入任意的长度和 B 已经读入的串长一样的串，从这些状态而到达一个接收状态。

It is important to realize that it is not necessary for B to know how many inputs it has read so far; it keeps this information up-to-date each time it reads a new symbol. The rule that keeps things up to date is: $\delta_B([q,S],a) = [\delta_A(q,a),T]$, where T is the set of states p of A such that there is a transition from p to any state of S on any input symbol. In this manner, the first component continues to simulate A, while the second component now represents states that can reach an accepting state following a path that is one longer than the paths represented by S.

首先对于 B 来说，没有必要知道已经输入多少个串；它要保证每次读入一个新的符号都要更新。保证更新的规则: $\delta_B([q,S],a) = [\delta_A(q,a),T]$, 其中 T 是 A 的一个状态集 p 使得输入任意一个符号都能够使 p 转换到状态 S。在这种情况下，第一个条件继续模拟 A,第二个条件表示一个接受状态，这是根据一个比 S 表示的路径还要长的路径。

To complete the construction of B, we have only to specify:

为了完成 B 的构造，我们指出:

- The initial state is $[q_0,F]$, that is, the initial state of A and the accepting states of A. This choice reflects the situation when A has read 0 inputs: it is still in its initial state, and the accepting states are exactly the ones that can reach an accepting state on a path of length 0.
- The accepting states of B are those states [q,S] such that q is in S. The justification is that it is exactly these states that are reached by some string of length n, and there is some other string of length n that will take state q to an accepting state.
- 始状态 $[q_0,F]$,也是, A 的初始状态和接受状态。这满足了当 A 的输入是 0 时的情况：它还是在初始状态，它的接收状态是能够通过 0 长度路经到达的接收状态。
- B 的接收状态是状态 [q,S]，使得 q 在 S 中。这是因为这些状态是通过符号长度为 n 的串到达的状态，并且存在另一个长度为 n 的串使得状态 q 到达一个接受状态。

Exercise 4.2.13(a)

Start out by complementing this language. The result is the language consisting of all strings of 0's and 1's that are not in 0*1*, plus the strings in L_{0n1n}. If we intersect with 0*1*, the result is exactly L_{0n1n}. Since complementation and intersection with a regular set preserve regularity, if the given language were regular then so would be L_{0n1n}. Since we know the latter is false, we conclude the given language is not regular.

从补语言出发。补语言由所有的通过 0's 和 1's 组成的串，除了 0*1*, 加上 L_{0n1n} 中的串。 如果我们和 0*1*作交运算，结果就是 L_{0n1n}。因为补运算和交运算保

持正则，如果给定的语言是正则的，则 L_{0n1n} 也是正则的。因为后者不是正则的，所以我们得到给定的语言不是正则的。

Exercise 4.2.14(c)

Change the accepting states to be those for which the first component is an accepting state of A_L and the second is a nonaccepting state of A_M. Then the resulting DFA accepts if and only if the input is in L - M.

把接收状态转换成那些第一部分是 A_L 的接收状态并且第二部分是 A_M 的非接受状态。当且仅当输入是在 L – M 中是，接受得到的 DFA 。

Solutions for Section 4.3

Exercise 4.3.1

Let n be the pumping-lemma constant. Test all strings of length between n and 2n-1 for membership in L. If we find even one such string, then L is infinite. The reason is that the pumping lemma applies to such a string, and it can be ``pumped" to show an infinite sequence of strings are in L.

令 n 是泵引理中的常量。测试 L 中长度在 n 和 2n-1 之间的所有符号串。如果能够找到一个串， 则 L 是无穷的。理由是把它作为“泵”得到了无穷的结果串，这些串都在 L 中。

Suppose, however, that there are no strings in L whose length is in the range n to 2n-1. We claim there are no strings in L of length 2n or more, and thus there are only a finite number of strings in L.

假设 L 中不存在这样的串，其长度在 n 到 2n-1 之间。我们称 L 中不存在长度大于等于 2n 的串，因此 L 的串都是有穷的。

In proof, suppose w is a string in L of length at least 2n, and w is as short as any string in L that has length at least 2n. Then the pumping lemma applies to w, and we can write w = xyz, where xz is also in L. How long could xz be? It can't be as long as 2n, because it is shorter than w, and w is as short as any string in L of length 2n or more. n, because xz is at most n shorter than w. Thus, xz is of length between n and 2n-1, which is a contradiction, since we assumed there were no strings in L with a length in that range.

假设 w 是 L 中一个长度至少为 2n 的串， 并且 w 和 L 中长度至少为 2n 的任意串的长度一样短。 然后 w 作为“泵”， w = xyz, 其中 xz 在 L 中。xz 的长度不可能超过 2n, 因为它比 w 短而 w 和 L 中长度为 2n 或者大于 2n 的串的长度相等,因为 xz 的长度最多比 w 短 n，因此 xz 的长度在 n 和 2n-1 之间, 这与假设 L 中不存在这样的串矛盾。

Solutions for Section 4.4

Exercise 4.4.1

Revised 10/23/01.

```
    B|x
    C|x x
    D|x x x
    E|x x     x
    F|x     x x x
    G|   x x x x x
    H|x x x x x x x
     ---------------
       A B C D E F G
```

	0	1
->AG	BF	AG
BF	AG	CE
CE	D	BF
*D	D	AG
H	AG	D

Note, however, that state H is inaccessible, so it should be removed, leaving the first four states as the minimum-state DFA.

Note, however, 状态 H 是从初始状态出发不可达的， 所以它应该被去掉, 剩下前面四个状态为最小状态 DFA.

Solutions for Section 5.1

Exercise 5.1.1(a)

S -> 0S1 | 01

Exercise 5.1.1(b)

S -> AB | CD
A -> aA | ε
B -> bBc | E | cD
C -> aCb | E | aA
D -> cD | ε
E -> bE | b

To understand how this grammar works, observe the following:

- A generates zero or more a's.
- D generates zero or more c's.
- E generates one or more b's.
- B first generates an equal number of b's and c's, then produces either one or more b's (via E) or one or more c's (via cD). That is, B generates strings in b*c* with an unequal number of b's and c's.
- Similarly, C generates unequal numbers of a's then b's.
- Thus, AB generates strings in a*b*c* with an unequal numbers of b's and c's, while CD generates strings in a*b*c* with an unequal number of a's and b's.

Exercise 5.1.1(b)

S -> AB | CD
A -> aA | ε
B -> bBc | E | cD
C -> aCb | E | aA
D -> cD | ε
E -> bE | b

意思如下:

- A 产生 0 或者一个或多个 a。.
- D 产生 0 或者一个或多个 c。
- E 产生一个或多个 b。
- B 产生式中 b 和 c 的个数一样多，然后产生一个或多个 b (通过 E) 或 一个或多个 c 通过 cD)。即， B 产生这样的字符串 b*c* ，其中 b 和 c 的个数不相等。
- 同样， C 产生 a 和 b 个数不相等的字符串。
- 因此，AB 产生字符串 a*b*c* ，并且 b 和 c 的数目不同； CD 产生字符串 a*b*c*并且 a 和 b 的数目不同。

Exercise 5.1.2(a)

Leftmost: S => A1B => 0A1B => 00A1B => 001B => 0010B => 00101B => 00101

Rightmost: S => A1B => A10B => A101B => A101 => 0A101 => 00A101 => 00101

Exercise 5.1.2(a)

最左推导：S => A1B => 0A1B => 00A1B => 001B => 0010B => 00101B => 00101

最有推导: S => A1B => A10B => A101B => A101 => 0A101 => 00A101 => 00101

Exercise 5.1.5

S -> S+S | SS | S* | (S) | 0 | 1 | phi | e

The idea is that these productions for S allow any expression to be, respectively, the sum (union) of two expressions, the concatenation of two expressions, the star of an expression, a parenthesized expression, or one of the four basis cases of expressions: 0, 1, phi, and ε.

Exercise 5.1.5

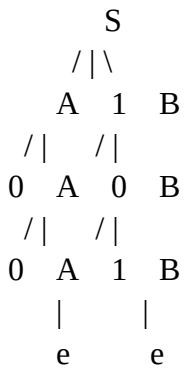
S -> S+S | SS | S* | (S) | 0 | 1 | phi | e

上式表示 S 可以表示任何的表达式，分别是，两个表达式的求和、串联，一个表达式的星，括号表达式，或者是表达式的四条基础规则: 0, 1, phi，和 ε。

Return to Top

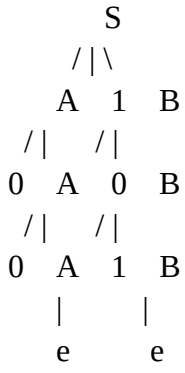
Solutions for Section 5.2

Exercise 5.2.1(a)



In the above tree, e stands for ε.

Exercise 5.2.1(a){ Exercise 5.1.1(a)的语法分析树}



In the above tree, e stands for ε.

Return to Top

Solutions for Section 5.3

Exercise 5.3.2

B -> BB | (B) | [B] | ε

Exercise 5.3.2

B -> BB | (B) | [B] | ε

Exercise 5.3.4(a)

Change production (5) to:
ListItem -> <LI> Doc </LI>

Exercise 5.3.4(a)

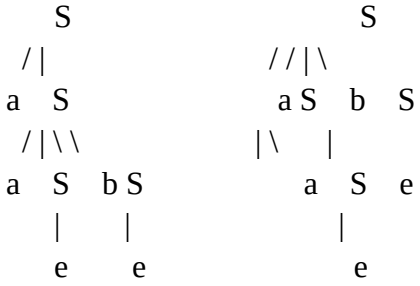
Change production (5) to:
ListItem（列表项） -> <LI> Doc </LI>

Return to Top

Solutions for Section 5.4

Exercise 5.4.1

Here are the parse trees:

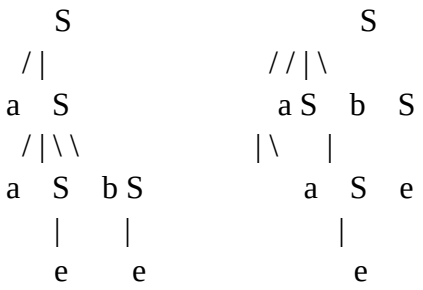


The two leftmost derivations are: S => aS => aaSbS => aabS => aab and S => aSbS => aaSbS => aabS => aab.

The two rightmost derivations are: S => aS => aaSbS => aaSb => aab and S => aSbS => aSb => aaSb => aab.

Exercise 5.4.1

语法分析树：



最左推导: S => aS => aaSbS => aabS => aab 和 S => aSbS => aaSbS => aabS => aab.

最右推导: S => aS => aaSbS => aaSb => aab 和 S => aSbS => aSb => aaSb => aab.

Exercise 5.4.3

The idea is to introduce another nonterminal T that cannot generate an unbalanced a. That strategy corresponds to the usual rule in programming languages that an ``else'' is associated with the closest previous, unmatched ``then.'' Here, we force a b to match the previous unmatched a. The grammar:

S -> aS | aTbS | ε
T -> aTbT | ε

Exercise 5.4.3

引入另一个非终结符 T that cannot generate an unbalanced a. That strategy corresponds to the usual rule in programming languages that an ``else'' is associated with the closest previous, unmatched ``then.'' Here, we force a b to match the previous unmatched a。无歧义文法如下：

S -> aS | aTbS | ε
T -> aTbT | ε

Exercise 5.4.6

Alas, it is not. We need to have three nonterminals, corresponding to the three possible ``strengths'' of expressions:

1. A factor cannot be broken by any operator. These are the basis expressions, parenthesized expressions, and these expressions followed by one or more *'s.
2. A term can be broken only by a *. For example, consider 01, where the 0 and 1 are concatenated, but if we follow it by a *, it becomes 0(1*), and the concatenation has been ``broken'' by the *.
3. An expression can be broken by concatenation or *, but not by +. An example is the expression 0+1. Note that if we concatenate (say) 1 or follow by a *, we

parse the expression 0+(11) or 0+(1*), and in either case the union has been broken.

The grammar:

E -> E+T | T
T -> TF | F
F -> F* | (E) | 0 | 1 | phi | e

Exercise 5.4.6

不是无歧义文法。我们需要三个不同的变元，每个变元代表拥有同样级别的“黏结强度”的那些表达式： (P144)

1. 因子是不能被相邻的运算符打断的表达式。基本的表达式有：括号表达式，这些表达式可以跟一个或多个*。
2. 项是不能被相邻的*运算打断的表达式。例如 01 是可以被打断的，只要采用右结合的规则并且把 a *放到它的右边，也就是它被结合成 0(1*)，因此 01 就被*打断了。
3. 表达式是指被相邻的*打断的表达式，但不能被+打断的表达式。例如表达式 0+1。采用右结合的规则并且把 a 或者 a *放到它的右边，我们得到表达式 0+(11) 或者 0+(1*),都被和运算打断了。

语法:

E -> E+T | T
T -> TF | F
F -> F* | (E) | 0 | 1 | phi | e

Solutions for Section 6.1

Exercise 6.1.1(a)

$(q,01,Z_0) \vdash (q,1,XZ_0) \vdash (q,\varepsilon,XZ_0) \vdash (p,\varepsilon,Z_0)$
$\vdash (p,1,Z_0) \vdash (p,\varepsilon,\varepsilon)$

Exercise 6.1.1(a)

可能的 ID 的序列如下：
$(q,01,Z_0) \vdash (q,1,XZ_0) \vdash (q,\varepsilon,XZ_0) \vdash (p,\varepsilon,Z_0)$
$(q,01,Z_0) \vdash (q,1,XZ_0) \vdash (p,1,Z_0) \vdash (p,\varepsilon,\varepsilon)$

Return to Top

Solutions for Section 6.2

Exercise 6.2.1(a)

We shall accept by empty stack. Symbol X will be used to count the 0's on the input. In state q, the start state, where we have seen no 1's, we add an X to the stack for each 0 seen. The first X replaces Z_0, the start symbol. When we see a 1, we go to state p, and then only pop the stack, one X for each input 1. Formally, the PDA is $(\{q,p\},\{0,1\},\{X,Z_0\},\delta,q,Z_0)$. The rules:

1. $\delta(q,0,Z_0) = \{(q,X)\}$
2. $\delta(q,0,X) = \{(q,XX)\}$
3. $\delta(q,1,X) = \{(p,\varepsilon)\}$
4. $\delta(p,1,X) = \{(p,\varepsilon)\}$

Exercise 6.2.1(a)

以空栈方式接受。符号 X 表示符号 0。初始状态 q 中不包含 1，我们每碰到一个 0 就把符号 X 压入栈中。 第一个符号 X 代替了初始符号 Z_0。当碰到 1 时，我们就进入状态 p，并且出栈，每输入一个 1 就有一个符号 X 出栈。下推自动机 PDA 是$(\{q,p\},\{0,1\},\{X,Z_0\},\delta,q,Z_0)$。规则如下：

5. $\delta(q,0,Z_0) = \{(q,X)\}$
6. $\delta(q,0,X) = \{(q,XX)\}$
7. $\delta(q,1,X) = \{(p,\varepsilon)\}$
8. $\delta(p,1,X) = \{(p,\varepsilon)\}$

Exercise 6.2.2(a)

Revised 6/20/02.

Begin in start state q0, with start symbol Z_0, and immediately guess whether to check for:

1. i=j=0 (state q1).
2. i=j>0 (state q2).
3. j=k (state q3).

We shall accept by final state; as seen below, the accepting states are q1 and q3. The rules, and their explanations:

- $\delta(q0,\varepsilon,Z_0) = \{(q1,Z_0), (q2,Z_0), (q3,Z_0)\}$, the initial guess.
- $\delta(q1,c,Z_0) = \{(q1,Z_0)\}$. In case (1), we assume there are no a's or b's, and we consume all c's. State q1 will be one of our accepting states.
- $\delta(q2,a,Z_0) = \{(q2,XZ_0)\}$, and $\delta(q2,a,X) = \{(q2,XX)\}$. These rules begin case (2). We use X to count the number of a's read from the input, staying in state q2.

- $\delta(q2,b,X) = \delta(q4,b,X) = \{(q4,\varepsilon)\}$. When b's are seen, we go to state q4 and pop X's against the b's.
- $\delta(q4,\varepsilon,Z_0) = \{(q1,Z_0)\}$. If we reach the bottom-of-stack marker in state q4, we have seen an equal number of a's and b's. We go spontaneously to state q1, which will accept and consume all c's, while continuing to accept.
- $\delta(q3,a,Z_0) = \{(q3,Z_0)\}$. This rule begins case (3). We consume all a's from the input. Since j=k=0 is possible, state q3 must be an accepting state.
- $\delta(q3,b,Z_0) = \{(q5,XZ_0)\}$. When b's arrive, we start counting them and go to state q5, which is not an accepting state.
- $\delta(q5,b,X) = \{(q5,XX)\}$. We continue counting b's.
- $\delta(q5,c,X) = \delta(q6,c,X) = \{(q6,\varepsilon)\}$. When c's arrive, we go to state q6 and match the c's against the b's.
- $\delta(q6,\varepsilon,Z_0) = \{(q3,\varepsilon)\}$. When the bottom-of-stack marker is exposed in state q6, we have seen an equal number of b's and c's. We spontaneously accept in state q3, but we pop the stack so we cannot accept after reading more a's.

Exercise 6.2.2(a)

Revised 6/20/02.

q0 表示初始状态, Z_0 表示初始符号， 猜测是否满足下列式子：

4. i=j=0 (状态 q1).
5. i=j>0 (状态 q2).
6. j=k (状态 q3).

接受状态是 q1 和 q3， 规则如下： （和 p161 类似）

- $\delta(q0,\varepsilon,Z_0) = \{(q1,Z_0), (q2,Z_0), (q3,Z_0)\}$,最初的猜测 。
- $\delta(q1,c,Z_0) = \{(q1,Z_0)\}$. 条件(1), 我们假设没有 a 或者 b，并且接受所有的 c。状态 q1 是我们接受状态的一部分。
- $\delta(q2,a,Z_0) = \{(q2,XZ_0)\}$, 并且 $\delta(q2,a,X) = \{(q2,XX)\}$. 根据条件 (2)。这条规则是说在看到一个 a 时就压入一个 X。
- $\delta(q2,b,X) = \delta(q4,b,X) = \{(q4,\varepsilon)\}$。当输入 b，我们进入状态 q4 并且所有的 X 出栈。
- $\delta(q4,\varepsilon,Z_0) = \{(q1,Z_0)\}$。 如果我们进入状态 q4, 就得到了符号 a 和符号 b 数目相等。 于是自然就进入状态 q1， 其中接受了所有符号 c。
- $\delta(q3,a,Z_0) = \{(q3,Z_0)\}$。 根据条件(3)，这条规则是在说看见 a 时，栈顶元素不变，状态也不变。因为 j=k=0，状态 q3 就是接受状态。
- $\delta(q3,b,Z_0) = \{(q5,XZ_0)\}$。这条规则是在说看见 b 时就压入一个 X，并且进入非终结状态 q5。
- $\delta(q5,b,X) = \{(q5,XX)\}$。这条规则是说在看到一个 b 时就压入一个 X。
- $\delta(q5,c,X) = \delta(q6,c,X) = \{(q6,\varepsilon)\}$。当输入 c，进入状态 q6 并开始匹配 c。
- $\delta(q6,\varepsilon,Z_0) = \{(q3,\varepsilon)\}$。 当栈空时有在状态 q6，则符号 b 和 c 的数目相等。于是接受状态 q3，并且栈清空。这时如果再读入 a，则不能再接受。

Exercise 6.2.4

Introduce a new state q, which becomes the initial state. On input ε and the start symbol of P, the new PDA has a choice of popping the stack (thus accepting ε), or going to the start state of P.

Exercise 6.2.4

引入一个新的状态 q，使其为初始化状态。当输入 ε 和初始化符号 P，新的 PDA 选择出栈 (thus accepting ε)，或者进入状态 P。

Exercise 6.2.5(a)

Revised 6/6/06.

$(q0,bab,Z_0) \vdash (q2,ab,BZ_0) \vdash (q3,b,Z_0) \vdash (q1,b,AZ_0) \vdash (q1,\varepsilon,Z_0) \vdash (q0,\varepsilon,Z_0) \vdash (f,\varepsilon,\varepsilon)$

Exercise 6.2.5(a)

Revised 6/6/06.

$(q0,bab,Z_0) \vdash (q2,ab,BZ_0) \vdash (q3,b,Z_0) \vdash (q1,b,AZ_0) \vdash (q1,\varepsilon,Z_0) \vdash (q0,\varepsilon,Z_0) \vdash (f,\varepsilon,\varepsilon)$

Exercise 6.2.8

Suppose that there is a rule that $(p,X_1X_2...X_k)$ is a choice in $\delta(q,a,Z)$. We create k-2 new states $r_1,r_2,...,r_{k-2}$ that simulate this rule but do so by adding one symbol at a time to the stack. That is, replace $(p,X_1X_2...X_k)$ in the rule by $(r_{k-2},X_{k-1}X_k$. Then create new rules $\delta(r_{k-2},\varepsilon,X_{k-1}) = \{(r_{k-3},X_{k-2}X_{k-1})\}$, and so on, down to $\delta(r_2,\varepsilon,X_3) = \{(r_1,X_2X_3)\}$ and $\delta(r_1,X_2) = \{(p,X_1X_2)\}$.

Exercise 6.2.8

假设在 $\delta(q,a,Z)$中，存在规则 $(p,X_1X_2...X_k)$。我们创建 k-2 个新的状态 $r_1,r_2,...,r_{k-2}$ 来模仿这个规则，每次只有一个符号进栈。即，$(r_{k-2},X_{k-1}X_k)$代替$(p,X_1X_2...X_k)$。创建新的规则 $\delta(r_{k-2},\varepsilon,X_{k-1}) = \{(r_{k-3},X_{k-2}X_{k-1})\}$，…，$\delta(r_2,\varepsilon,X_3) = \{(r_1,X_2X_3)\}$ ，$\delta(r_1,X_2) = \{(p,X_1X_2)\}$。

Return to Top

Solutions for Section 6.3

Exercise 6.3.1

$(\{q\},\{0,1),\{0,1,A,S\},\delta,q,S)$ where δ is defined by:

1. δ(q,ε,S) = {(q,0S1), (q,A)}
2. δ(q,ε,A) = {(q,1A0), (q,S), (q,ε)}
3. δ(q,0,0) = {(q,ε)}
4. δ(q,1,1) = {(q,ε)}

Exercise 6.3.1

({q},{0,1),{0,1,A,S},δ,q,S) 其中 δ 定义为：

5. δ(q,ε,S) = {(q,0S1), (q,A)}
6. δ(q,ε,A) = {(q,1A0), (q,S), (q,ε)}
7. δ(q,0,0) = {(q,ε)}
8. δ(q,1,1) = {(q,ε)}

Exercise 6.3.3

In the following, S is the start symbol, e stands for the empty string, and Z is used in place of Z_0.

1. S -> [qZq] | [qZp]

   The following four productions come from rule (1).

2. [qZq] -> 1[qXq][qZq]
3. [qZq] -> 1[qXp][pZq]
4. [qZp] -> 1[qXq][qZp]
5. [qZp] -> 1[qXp][pZp]

   The following four productions come from rule (2).

6. [qXq] -> 1[qXq][qXq]
7. [qXq] -> 1[qXp][pXq]
8. [qXp] -> 1[qXq][qXp]
9. [qXp] -> 1[qXp][pXp]

   The following two productions come from rule (3).

10. [qXq] -> 0[pXq]
11. [qXp] -> 0[pXp]

    The following production comes from rule (4).

12. [qXq] -> e

    The following production comes from rule (5).

13. [pXp] -> 1

    The following two productions come from rule (6).

14. [pZq] -> 0[qZq]
15. [pZp] -> 0[qZp]

Exercise 6.3.3

S 表示初始符号, e 表示空串, Z 代替 Z_0。

16. S -> [qZq] | [qZp]

    根据规则(1)得到的 4 个产生式。

17. [qZq] -> 1[qXq][qZq]
18. [qZq] -> 1[qXp][pZq]
19. [qZp] -> 1[qXq][qZp]
20. [qZp] -> 1[qXp][pZp]

    根据规则(2)得到的 4 个产生式.

21. [qXq] -> 1[qXq][qXq]
22. [qXq] -> 1[qXp][pXq]
23. [qXp] -> 1[qXq][qXp]
24. [qXp] -> 1[qXp][pXp]

    根据规则(3)得到的 2 个产生式.

25. [qXq] -> 0[pXq]
26. [qXp] -> 0[pXp]

    根据规则(4)得到的 1 个产生式.

27. [qXq] -> e

    根据规则(5)得到的 1 个产生式.

28. [pXp] -> 1

    根据规则(6)得到的 2 个产生式.

29. [pZq] -> 0[qZq]
30. [pZp] -> 0[qZp]

Exercise 6.3.6

Convert P to a CFG, and then convert the CFG to a PDA, using the two constructions given in Section 6.3. The result is a one-state PDA equivalent to P.

Exercise 6.3.6

把下推自动机 P 转变成一个 CFG，然后再把 CFG 转变成 PDA。利用两个构造方法（ in Section 6.3 p165），从 PDA 到文法的定理 6.13 和从文法到 PDA 的定理 6.14， 最后得到的结果是一个 PDA 对应于一个 P，即等价性 $N(P_1)=N(P)$。

Return to Top

Solutions for Section 6.4

Exercise 6.4.1(b)

Not a DPDA. For example, rules (3) and (4) give a choice, when in state q, with 1 as the next input symbol, and with X on top of the stack, of either using the 1 (making no other change) or making a move on ε input that pops the stack and going to state p.

Exercise 6.4.1(b)

不是 DPDA。例如， 根据规则 (3) 和规则 (4)（P157 把这两个规则抄一下），当在 q 状态时，输入符号是 1，并且 X 在栈顶，最后得到的结果是： 变成 1 (其他的不变)或者当是输入是 ε 则出栈并进入状态 p。

Exercise 6.4.3(a)

Suppose a DPDA P accepts both w and wx by empty stack, where x is not ε (i.e., N(P) does not have the prefix property). Then $(q_0,wxZ_0) \vdash^* (q,x,\varepsilon)$ for some state q, where q_0 and Z_0 are the start state and symbol of P. It is not possible that $(q,x,\varepsilon) \vdash^* (p,\varepsilon,\varepsilon)$ for some state p, because we know x is not ε, and a PDA cannot have a move with an empty stack. This observation contradicts the assumption that wx is in N(P).

Exercise 6.4.3(a)

假设一个 DPDA P 通过空栈能够接受 w 和 wx， 其中 x 不是 ε (即，N(P) 没有前缀性质)。 则存在一个状态 q 使得$(q_0,wxZ_0) \vdash^* (q,x,\varepsilon)$，其中 q_0 和 Z_0 分别是 P 的初始状态和初始符号。因为 x 不是 ε 并且一个 PDA 在空栈的情况下也是不能转变的，所以不存在状态 p 使得$(q,x,\varepsilon) \vdash^* (p,\varepsilon,\varepsilon)$ 成立。 与假设 wx 在 N(P) 中矛盾。

Exercise 6.4.3(c)

Modify P' in the following ways to create DPDA P:

1. Add a new start state and a new start symbol. P, with this state and symbol, pushes the start symbol of P' on top of the stack and goes to the start state of P'. The purpose of the new start symbol is to make sure P doesn't accidentally accept by empty stack.
2. Add a new ``popping state" to P. In this state, P pops every symbol it sees on the stack, using ε input.
3. If P' enters an accepting state, P enters the popping state instead.

As long as L(P') has the prefix property, then any string that P' accepts by final state, P will accept by empty stack.

Exercise 6.4.3(c)

根据下列改变 P'最后创建 DPDA P:

4. 增加一个新的初始状态和初始符号。通过这个初始状态和初始符号，P 使 P'中在栈顶的初始符号出栈并且进入 P'的初始状态。建新的初始符号目的是防止因为空栈 P 没有被接受。
5. Add a new ``popping state" to P. 在这个状态，P 看到的所有栈中的符号都出栈， ε 进栈。
6. 若 P' 进入了接受状态，则 P 进入 the popping state instead.

只要 L(P') 有前缀属性, 则 P 通过空栈接受任何符号串，其中这些符号串被 P' 通过最终状态接受的。

Solutions for Section 7.1

Exercise 7.1.1

A and C are clearly generating, since they have productions with terminal bodies. Then we can discover S is generating because of the production S->CA, whose body consists of only symbols that are generating. However, B is not generating. Eliminating B, leaves the grammar

S -> CA
A -> a
C -> b

Since S, A, and C are each reachable from S, all the remaining symbols are useful, and the above grammar is the answer to the question.

A 和 C 显然是产生的，因为它们都有体中只有终结符的产生式。S 也是产生的，因为有产生式 S->CA，其产生式体中都由产生组成。B 不是产生的，故消去。最后剩下 S -> CA

A -> a

C -> b

因为 S，A 和 C 均从 S 可达，所以剩下的符号都是有用的。上述文法即是所求的等价文法。

Exercise 7.1.2

Revised 6/27/02.

a)

Only S is nullable, so we must choose, at each point where S occurs in a body, to eliminate it or not. Since there is no body that consists only of S's, we do not have to invoke the rule about not eliminating an entire body. The resulting grammar:

S -> ASB | AB
A -> aAS | aA | a
B -> SbS | bS | Sb | b | A | bb

S 是可空的，所以任意生产式体中的 S 可以消去，也可以保留。因为没有体中只有 S 的产生式，所以不必引入部分消去的规则。所求结果为

S -> ASB | AB
A -> aAS | aA | a
B -> SbS | bS | Sb | b | A | bb

b)

The only unit production is B -> A. Thus, it suffices to replace this body A by the bodies of all the A-productions. The result:

S -> ASB | AB
A -> aAS | aA | a
B -> SbS | bS | Sb | b | aAS | aA | a | bb

B->A 是仅有的单位产生式。于是用 A 的产生式的体替代该产生式的体 A 即得

S -> ASB | AB
A -> aAS | aA | a
B -> SbS | bS | Sb | b | aAS | aA | a | bb

c)

Observe that A and B each derive terminal strings, and therefore so does S. Thus, there are no useless symbols.

观测可得 A 和 B 都能推导出终结符号串，S 也是，所以没有无用符号。

d)

Introduce variables and productions C -> a and D -> b, and use the new variables in all bodies that are not a single terminal:

S -> ASB | AB
A -> CAS | CA | a
B -> SDS | DS | SD | b | CAS | CA | a | DD
C -> a
D -> b

Finally, there are bodies of length 3; one, CAS, appears twice. Introduce new variables E, F, and G to split these bodies, yielding the CNF grammar:

S -> AE | AB
A -> CF | CA | a
B -> SG | DS | SD | b | CF | CA | a | DD
C -> a
D -> b
E -> SB
F -> AS
G -> DS

引入新变元和新产生式 C->a 和 D->b，将新变元替代产生式中的终结符得

S -> ASB | AB
A -> CAS | CA | a
B -> SDS | DS | SD | b | CAS | CA | a | DD
C -> a
D -> b

最后处理长度为 3 的几个产生式，比如 CAS 出现了两次。引入新变元 E、F 和 G 来拆分这些产生式的体部分，于是得到 CNF 文法如下：

S -> AE | AB
A -> CF | CA | a
B -> SG | DS | SD | b | CF | CA | a | DD
C -> a
D -> b
E -> SB
F -> AS
G -> DS

Exercise 7.1.10

It's not possible. The reason is that an easy induction on the number of steps in a derivation shows that every sentential form has odd length. Thus, it is not possible to find such a grammar for a language as simple as {00}.

这是不可能的。理由是通过对推导步数的简单归纳即可知每个句型的长度都是奇数的。因此比如语言{00}，不可能找到这样的文法。

To see why, suppose we begin with start symbol S and try to pick a first production. If we pick a production with a single terminal as body, we derive a string of length 1 and are done. If we pick a body with three variables, then, since there is no way for a variable to derive epsilon, we are forced to derive a string of length 3 or more.

为了说明原因，我们假设从开始符号 S 出发，选择第一个产生式。如果该产生式的体中只包含单个终结符，那么我们可推导出长度为 1 的串。如果该产生式的体中只包含 3 个变元，那么，由于无法从一个变元推导出 epsilon，只能推导出长度大于等于 3 的串。

【没讲】Exercise 7.1.11(b)

The statement of the entire construction may be a bit tricky, since you need to use the construction of part (c) in (b), although we are not publishing the solution to (c). The construction for (b) is by induction on i, but it needs to be of the stronger statement that if an A_i-production has a body beginning with A_j, then j > i (i.e., we use part (c) to eliminate the possibility that i=j).

Basis: For i = 1 we simply apply the construction of (c) for i = 1.

Induction: If there is any production of the form A_i -> A_1..., use the construction of (a) to replace A_1. That gives us a situation where all A_i production bodies begin with at least A_2 or a terminal. Similarly, replace initial A_2's using (a), to make A_3 the lowest possible variable beginning an A_i-production. In this manner, we eventually guarantee that the body of each A_i-production either begins with a terminal or with A_j, for some j >= i. A use of the construction from (c) eliminates the possibility that i = j.

【没讲】Exercise 7.1.11(d)

As per the hint, we do a backwards induction on i, that the bodies of A_i productions can be made to begin with terminals.

Basis: For i = k, there is nothing to do, since there are no variables with index higher than k to begin the body.

Induction: Assume the statement for indexes greater than i. If an A_i-production begins with a variable, it must be A_j for some j > i. By the induction hypothesis, the A_j-productions all have bodies beginning with terminals now. Thus, we may use the construction (a) to replace the initial A_j, yielding only A_i-productions whose bodies begin with terminals.

After fixing all the A_i-productions for all i, it is time to work on the B_i-productions. Since these have bodies that begin with either terminals or A_j for some j, and the latter variables have only bodies that begin with terminals, application of construction (a) fixes the B_j's.

Return to Top

Solutions for Section 7.2

Exercise 7.2.1(a)

Let n be the pumping-lemma constant and consider string z = $a^n b^{n+1} c^{n+2}$. We may write z = uvwxy, where v and x, may be ``pumped,'' and |vwx| <= n. If vwx does not have c's, then uv^3wx^3y has at least n+2 a's or b's, and thus could not be in the language.

令 n 为 bang 引理得到的常数，并取 z = $a^n b^{n+1} c^{n+2}$ ，我们可以把 z 打断为 z = uvwxy，其中 v 和 x 是被 bang 的两段，且|vwx| <= n 。如果 vwx 不包含 c，那么 uv^3wx^3y 至多有 n+2 个 a 或者 b，因此不在该语言中。

If vwx has a c, then it could not have an a, because its length is limited to n. Thus, uwy has n a's, but no more than 2n+2 b's and c's in total. Thus, it is not possible that uwy has more b's than a's and also has more c's than b's. We conclude that uwy is not in the language, and now have a contradiction no matter how z is broken into uvwxy.

如果 vwx 包含 a、c 那么就不可能包含 a，因为其长度受限于 n。于是 uwy 包含 n 个 a，但总共至多包含 2n+2 个 b 和 c。因此，uwy 拥有的 b 不可能比 a 多，同样 c 也不可能比 b 多。我们可得出结论：uwy 不在该语言中。于是得到矛盾不管 Z 是怎样被打断的。

Exercise 7.2.1(d)

Let n be the pumping-lemma constant and consider z = $0^n 1^{n2}$. We break Z = uvwxy according to the pumping lemma. If vwx consists only of 0's, then uwy has n^2 1's and fewer than n 0's; it is not in the language. If vwx has only 1's, then we derive a contradiction similarly. If either v or x has both 0's and 1's, then uv^2wx^2y is not in 0*1*, and thus could not be in the language.

令 n 为 bang 引理得到的常数，并取 z = $0^n 1^{n2}$ 。根据 bang 引理我们可以把 z 打断为 z = uvwxy。如果 vwx 只有 1，那么可以类似地推出矛盾。如果 v 或者 x 拥有 0 和 1，那么 uv^2wx^2y 不在 0*1* ，因此也不在该语言中。

Finally, consider the case where v consists of 0's only, say k 0's, and x consists of m 1's only, where k and m are both positive. Then for all i, $uv^{i+1}wx^{i+1}y$ consists of n + ik 0's and n^2 + im 1's. If the number of 1's is always to be the square of the number of 0's, we must have, for some positive k and m: $(n+ik)^2 = n^2 + im$, or $2ink + i^2k^2 = im$. But the left side grows quadratically in i, while the right side grows linearly, and so this equality for all i is impossible. We conclude that for at least some i, $uv^{i+1}wx^{i+1}y$ is not in the language and have thus derived a contradiction in all cases.

最后考虑 v 只由 0 组成的情况，设有 k 个 0，x 只由 m 个 1 组成，这里 k 和 m 均为正数。对所有的 i，$uv^{i+1}wx^{i+1}y$ 由 n+ik 个 0 和 $n^2 + im$ 个 1 组成。如果 1 的个数总是 0 的个数的平方，则存在正数 k 和 m，使得$(n+ik)^2 = n^2 + im$, or $2ink + i^2k^2 = im$ 。但是左边关于 i 成平方增长，而右边只是线性增长，所以不可能对所以的 i 成立。因此对一些 i，$uv^{i+1}wx^{i+1}y$ 不在该语言中。于是对各种情况都导出矛盾。

Exercise 7.2.2(b)

It could be that, when the adversary breaks z = uvwxy, $v = 0^k$ and $x = 1^k$. Then, for all i, uv^iwx^iy is in the language.

当对手将 z 打断成 z = uvwxy ，且 $v = 0^k$ and $x = 1^k$ 时，对所有的 i，uv^iwx^iy 都在该语言里。

Exercise 7.2.2(c)

The adversary could choose z = uvwxy so that v and x are single symbols, on either side of the center. That is, |u| = |y|, and w is either epsilon (if z is of even length) or the single, middle symbol (if z is of odd length). Since z is a palindrome, v and x will be the same symbol. Then uv^iwx^iy is always a palindrome.

对手将 z 打断成 z = uvwxy ，使得 v 和 x 是单个符号串，也就是说，|u| = |y|，w 或者是 epsilon（当 z 的长度是偶数时），或者是单个的中间的符号（当 z 的长度是奇数时）。由于 z 是回文，所以 v 和 x 是同一个符号。那么 uv^iwx^iy 总是回文。

Exercise 7.2.4

The hint turns out to be a bad one. The easiest way to prove this result starts with a string $z = 0^n1^n0^n1^n$ where the middle two blocks are distinguished. Note that vwx cannot include 1's from the second block and also 1's from the fourth block, because then vwx would have all n distinguished 0's and thus at least n+1 distinguished symbols. Likewise, it cannot have 0's from both blocks of 0's. Thus, when we pump v and x, we must get an imbalance between the blocks of 1's or the blocks of 0's, yielding a string not in the language.
提示方法不怎么样。最简单的方法是取 $z = 0^n1^n0^n1^n$ ，其中中间两块可区分。注意 vwx 分别从第二块和第四块起不含 1，因为 vwx 有 n 个可区分的 0，于是至少有 n+1 个可区分符号。同样的，它从含 0 的两块起都不含 0。因此，当以 v 和 x 为泵，将在 1 的块和 0 的块之间产生不均衡，于是产生有不属于该语言的串。

Return to Top

Solutions for Section 7.3

Exercise 7.3.1(a)

For each variable A of the original grammar G, let A' be a new variable that generates init of what A generates. Thus, if S is the start symbol of G, we make S' the new start symbol.

对于原来文法 G 的每个变元 A，记 A' 为 A 经过 init 运算后得到的新变元。于是，若 S 是 G 的开始符号，则记 S' 为新的开始符号。

If A -> BC is a production of G, then in the new grammar we have A -> BC, A' -> BC', and A' -> B'. If A -> a is a production of G, then the new grammar has A -> a, A' -> a, and A' -> epsilon.

若 A -> BC 是 G 的产生式，则在新的文法中有 A -> BC, A' -> BC', and A' -> B'. 若 A -> a 是 G 的产生式，则在新的文法中有 A -> a, A' -> a, and A' -> epsilon.符合乔姆斯基范式

Exercise 7.3.1(b)

The construction is similar to that of part (a), but now A' must be designed to generate string w if and only if A generates wa. That is, A"s language is the result of applying /a to A's language.

构造过程类似于（a），只是 A' 产生 w 当且仅当 A 产生 wa。也就是说，A' 的语言是由 A 的语言应用/a 得到。

If G has production A -> BC, then the new grammar has A -> BC and A' -> BC'. If G has A -> b for some b != a, then the new grammar has A -> b, but we do not add any production for A'. If G has A -> a, then the new grammar has A -> a and A' -> epsilon.

若 A -> BC 是 G 的产生式，则在新的文法中有 A -> BC and A' -> BC'. 若对某些 b != a，A -> b 是 G 的产生式，则在新的文法中有 A -> b，但是我们不能把任何一个产生式加入到 A' 中。若 A -> a 是 G 的产生式，则在新的文法中有 A -> a and A' -> epsilon.

Exercise 7.3.3(a)

Consider the language L = $\{a^ib^jc^k \mid 1 <= i \text{ and } 1 <= j \text{ and } (i <= k \text{ or } j <= k)\}$. L is easily seen to be a CFL; you can design a PDA that guesses whether to compare the a's or b's with the c's. However, min(L) = $\{a^ib^jc^k \mid k = \min(i,j)\}$. It is also easy to show, using the pumping lemma, that this language is not a CFL. Let n be the pumping-lemma constant, and consider z = $a^nb^nc^n$.

考虑语言 L = $\{a^ib^jc^k \mid 1 <= i \text{ and } 1 <= j \text{ and } (i <= k \text{ or } j <= k)\}$ 。L 显然是 CFL，则

可以设计一个 PDA 用来测试是否进行 a or b 与 c 的比较。但是，min(L) = $\{a^ib^jc^k \mid k = \min(i,j)\}$ 。用 bang 引理很容易证明该语言不是 CFL。令 n 为根据 bang 引理得到的常数，取 z = $a^nb^nc^n$ 。

Exercise 7.3.4(b)

If we start with a string of the form 0^n1^n and intersperse any number of 0's, we can obtain any string of 0's and 1's that begins with at least as many 0's as there are 1's in the entire string.

若我们从形如 0^n1^n 的串开始随意插入任意个 0，则从 0 和 1 的数目相等的串开始可得 0 和 1 的任意串。

Exercise 7.3.4(c)

Given DFA's for L_1 and L_2, we can construct an NFA for their shuffle by using the product of the two sets of states, that is, all states [p,q] such that p is a state of the automaton for L_1, and q is a state of the automaton for L_2. The start state of the automaton for the shuffle consists of the start states of the two automata, and its accepting states consist of pairs of accepting states, one from each DFA.

给定 L1 和 L2 的 DFP，则通过两组状态集的产生式可构造出一个它们重组的 NFA，也就是，构造状态[p,q]，其中 p 是 L1 自动机的一个状态，q 是 L2 自动机的一个状态。重组的初始状态由两个自动机的初始状态组成，接收状态由两个自动机的接收状态组成。

The NFA for the shuffle guesses, at each input, whether it is from L_1 or L_2. More formally, $\delta([p,q],a) = \{[\delta_1(p,a),q], [p,\delta_2(q,a)]\}$, where δ_i is the transition function for the DFA for L_i (i = 1 or 2). It is then an easy induction on the length of w that $\hat{\delta}([p_0,q_0],w)$ contains [p,q] if and only if w is the shuffle of some x and y, where δ_1-hat$(p_0,x) = p$ and δ_2-hat$(q_0,y) = q$.

重组 NFA 对每个来自 L1 或者 L2 的输入进行推测，更形式化的说，

$\delta([p,q],a) = \{[\delta_1(p,a),q], [p,\delta_2(q,a)]\}$

其中 δ_i 是 L_i (i = 1 or 2)的转移函数。接下来是对 w 长度的简单归纳，且满足 $\hat{\delta}([p_0,q_0],w)$包含[p,q]当且仅当 w 是 x 和 y 的重组，其中 δ_1-hat$(p_0,x) = p$ and δ_2-hat$(q_0,y) = q$.

Exercise 7.3.5

a)

Consider the language of regular expression **(0l)***. Its permutations consist of all strings with an equal number of 0's and 1's, which is easily shown not regular. In proof, use the pumping lemma for regular languages, let n be the pumping-lemma constant, and consider string 0^n1^n.

考虑正则表达式**(0l)***的语言。其置换由所有 0 的个数与 1 的个数相等的串组成，这很容易证明是非正则的。证明中对正则语言使用泵引理，记 n 为泵引理常数，考虑串 0^n1^n 。

b)

The language of **(012)*** serves. Its permutations are all strings with an equal number of 0's 1's, and 2's. We can prove this language not to be a CFL by using the pumping lemma on $0^n1^n2^n$, where n is the pumping-lemma constant.

考虑**(012)***的语言，其置换由所有 0，1，2 个数相等的串组成。对 0^n1^n2 使用泵引理可证明该语言不是 CFL，其中 n 为泵引理常数。

c)

Assume the alphabet of regular language L is {0,1}. We can design a PDA P to recognize perm(L), as follows. P simulates the DFA A for L on an input string that it guesses. However, P must also check that its own input is a permutation of the guessed string. Thus, each time P guesses an input for A, it also reads one of its own symbols. P uses its stack to remember whether it has seen more 0's than it has guessed, or seen more 1's than it has guessed. It does so by keeping a stack string with a bottom-of-stack marker and either as many more 0's as it has seen than guessed, or as many more 1's as it has seen than guessed.

假设是字母表{0,1}上的正则语言 L。我们可以设计一个 PDA P 来识别 perm(L)，具体如下：P 模拟 L 的 DFA A 对输入进行猜测。只是 P 必须检测它自己的输入是待推测串的置换。于是，每次 P 推测 A 的一个输入，同时读入它自己的一个符号。P 使用栈来记住是否看到比猜测更多的 0 或 1。It does so by keeping a stack string with a bottom-of-stack marker and either as many more 0's as it has seen than guessed, or as many more 1's as it has seen than guessed.

For instance, if P guesses 0 as an input for A but sees a 1 on its own input, then P:

1. If 0 is the top stack symbol, then push another 0.
2. If 1 is the top stack symbol, then pop the stack.
3. If Z_0, the bottom-of-stack marker is on top, push a 0.

例如：如果 P 推测 0 作为 A 的一个输入，但看到自己输入了一个 1，那么 P

1. 如果 0 是栈顶符号，那么推入另一个 0。
2. 如果 1 是栈顶符号，那么出栈。
3. 如果从栈底到栈顶标记 Z_0 刚好在栈顶，那么 0 出栈。

In addition, if P exposes the bottom-of-stack marker, then it has guessed, as input to A, a permutation of the input P has seen. Thus, if A is in an accepting state, P has a choice of move to pop its stack on epsilon input, thus accepting by empty stack.

另外，如果 P expose 从栈底到栈顶标记，那么它可以猜测，作为 A 的输入，这个输入的置换已经发现。因此，若 A 处于接收状态，则 P 可以将其 epsilon 输入出栈，于是以空栈接收。

Return to Top

Solutions for Section 7.4

Exercise 7.4.1(a)

If there is any string at all that can be ``pumped," then the language is infinite. Thus, let n be the pumping-lemma constant. If there are no strings as long as n, then surely the language is finite. However, how do we tell if there is some string of length n or more? If we had to consider all such strings, we'd never get done, and that would not give us a decision algorithm.

如果任意串可以被泵，那么该语言无限。记 n 为泵引理常数。若没有长度为 n 的串，则该语言肯定是有限的。However, how do we tell if there is some string of length n or more? If we had to consider all such strings, we'd never get done, and that would not give us a decision algorithm.

The trick is to realize that if there is any string of length n or more, then there will be one whose length is in the range n through 2n-1, inclusive. For suppose not. Let z be a string that is as short as possible, subject to the constraint that |z| >= n. If |z| < 2n, we are done; we have found a string in the desired length range. If |z| >= 2n, use the pumping lemma to write z = uvwxy. We know uwy is also in the language, but because |vwx| <= n, we know |z| > |uwy| >= n. That contradicts our assumption that z was as short as possible among strings of length n or more in the language.

这个技巧可以说明若存在长度大于等于 n 的任意串，则有一个长度在 n 到 2n+1 范围内的串。假设不存在。记 z 为长度尽可能短的串，并限制|z| >= n 。若

|z| >= 2n ，使用泵引理将 z 写成 z = uvwxy 。我们知道 uwy 在该语言里，但是由于|vwx| <= n ，可知|z| > |uwy| >= n 。这与假设的 z 是该语言中长度大于等于 n 的串中最短的串相矛盾。

We conclude that |z| < 2n. Thus, our algorithm to test finiteness is to test membership of all strings of length between n and 2n-1. If we find one, the language is infinite, and if not, then the language is finite.

可得|z| < 2n 。因此，测试有限性的算法也就是测试长度在 n 到 2n+1 之间的串的成员性。如果找到一个，那么该语言有限，否则无限。

Exercise 7.4.3(a)

Here is the table:

```
    {S,A,C}
    {B}       {B}
    {B}       {S,C}    {B}
    {S,C}     {S,A}    {S,C}    {S,A}
    {A,C}     {B}      {A,C}    {B}      {A,C}
-----------------------------------------
       a         b         a         b         a
```

Since S appears in the upper-left corner, ababa is in the language.

由于 S 出现在左上角，因此 ababa 在该语言中。

Exercise 7.4.4

The proof is an induction on n that if A =>* w, for any variable A, and |w| = n, then all parse trees with A at the root and yield w have 2n-1 interior nodes.
对 n 进行归纳可证：如果对任意变元 A，A =>* w，and |w| = n。则以 A 为根节点产物为 w 的所有分析树有 2n+1 个内部结点。

Basis: n = 1. The parse tree must have a root with variable A and a leaf with one terminal. This tree has 2n-1 = 1 interior node.

基础：n=1。该分析树一定以变元 A 为根且有一个终结符的叶子。该树有 2n-1=1 的内部结点。

Induction: Assume the statement for strings of length less than n, and let n > 1. Then the parse tree begins with A at the root and two children labeled by variables B and C. Then we can write w = xy, where B =>* x and C =>* y. Also, x and y are each shorter than length n, so the inductive hypothesis applies to them, and we know that the parse trees for these derivations have, respectively, 2|x|-1 and 2|y|-1 interior nodes.

归纳：假设该陈述是对长度小于 n 的串，且 n>1。然后该分析树以 A 为根节点且有两子结点分别标记为 B 和 C。可以记 w = xy ， 其中 B =>* x and C =>* y 。

X 和 y 长度都小于 n，由归纳假设可知对于这些推导的分析树分别有 2|x|-1 and 2|y|-1 个内部结点。

Thus, the parse tree for A =>* w has one (for the root) plus the sum of these two quantities number of interior nodes, or 2(|x|+|y|-1) interior nodes. Since |x|+|y| = |w| = n, we are done; the parse tree for A =>* w has 2n-1 interior nodes.

因此对于 A =>* w 的分析树共有一个根节点外加两个子结点的内部结点，即 2(|x|+|y|-1)个内部结点。由于|x|+|y| = |w| = n ，则对于 A =>* w 的分析树共有 2n+1 个内部结点。

Solutions for Section 8.1

Exercise 8.1.1(a)

We need to take a program P and modify it so it:

1. Never halts unless we explicitly want it to, and
2. Halts whenever it prints hello, world.

构造程序 P 并修改使之：

1．除非有明确要求，否则不停机，且
2．一旦显示 hello,world，则停机。

For (1), we can add a loop such as while(1){x=x;} to the end of main, and also at any point where main returns. That change catches the normal ways a program can halt, although it doesn't address the problem of a program that halts because some exception such as division by 0 or an attempt to read an unavailable device. Technically, we'd have to replace all of the exception handlers in the run-time environment to cause a loop whenever an exception occurred.

对于(1)，我们可以在 main 结尾处或者 main 任意返回点增加一个循环，比如 while(1){x=x;}。这修改可以抓住程序停机的正常方式，虽然它并没有定位程序因一些意外，比如被 0 除或者读一个不可得的装置，而导致的停机。从技术上说，我们没必要代替意外发生时导致循环的运行环境中的所有意外。

For (2), we modify P to record in an array the first 12 characters printed. If we find that they are hello, world., we halt by going to the end of main (past the point where the while-loop has been installed).

对于(2)，修改 P 用来在一个数组中记录前 12 个显示字符。如果是 hello,world，那么继续运行到 main 结尾除停机(忽略先前插入的 while 循环)

Return to Top

Solutions for Section 8.2

Exercise 8.2.1(a)

To make the ID's clearer in HTML, we'll use [q0] for q_0, and similarly for the other states.

[q0]00 |- X[q1]0 |- X0[q1]

The TM halts at the above ID.

为了使在 HTML 中的 ID 更清楚，我们把 q_0 记为[q0] ，其它状态类似

[q0]00 |- X[q1]0 |- X0[q1]

TM 在上述 ID 停机

Exercise 8.2.2(a)

Here is the transition table for the TM:
TM 转移表如下：

state	0	1	B	X	Y
q0	(q2,X,R)	(q1,X,R)	(qf,B,R)	-	(q0,Y,R)
q1	(q3,Y,L)	(q1,1,R)	-	-	(q1,Y,R)
q2	(q2,0,R)	(q3,Y,L)	-	-	(q2,Y,R)
q3	(q3,0,L)	(q3,1,L)	-	(q0,X,R)	(q3,Y,L)
qf	-	-	-	-	-

In explanation, the TM makes repeated excursions back and forth along the tape. The symbols X and Y are used to replace 0's and 1's that have been cancelled one against another. The difference is that an X guarantees that there are no unmatched 0's and 1's to its left (so the head never moves left of an X), while a Y may have 0's or 1's to its left.

说明：TM 在带上来回反复移动。符号 X 和 Y 用来代替相互抵消的 0 和 1。区别是：在 X 左边没有不匹配的 0 和 1(因此带头移动从不会越过 X 的左边)，而在 Y 的左边有 0 或者 1。

Initially in state q0, the TM picks up a 0 or 1, remembering it in its state (q1 = found a 1; q2 = found a 0), and cancels what it found with an X. As an exception, if the TM sees the blank in state q0, then all 0's and 1's have matched, so the input is accepted by going to state qf.

In state q1, the TM moves right, looking for a 0. If it finds it, the 0 is replaced by Y, and the TM enters state q3 to move left an look for an X. Similarly, state q2 looks for a 1 to match against a 0.

开始时处于状态 q0，TM 寻找 0 或者 1，并记住其状态(看到 1 记 q1，看到 0 记 q2)，再将其改为 X。作为一种例外，若 TM 进入状态 q0 时看到空格，则所有的 0 和 1 已经匹配，因此进入状态 qf 输入被接收。处于状态 q1，TM 向右移动，寻找 0。若找到，则用 Y 代替 0，TM 进入状态 q3 并开始向左移动寻找 X。类似地，在状态 q2 寻找 1 匹配 0。

In state q3, the TM moves left until it finds the rightmost X. At that point, it enters state q0 again, moving right over Y's until it finds a 0, 1, or blank, and the cycle begins again.

在状态 q3，TM 向左移动直到找到最右的 X。这时重新进入状态 q0，向右移动越过 Y 直到看到 0，1 或者空格，循环重新开始。

Exercise 8.2.4

These constructions, while they can be carried out using the basic model of a TM are much clearer if we use some of the tricks of Sect. 8.3.

For part (a), given an input [x,y] use a second track to simulate the TM for f on the input x. When the TM halts, compare what it has written with y, to see if y is indeed f(x). Accept if so.

For part (b), given x on the tape, we need to simulate the TM M that recognizes the graph of f. However, since this TM may not halt on some inputs, we cannot simply try all [x,i} to see which value of i leads to acceptance by M. The reason is that, should we work on some value of i for which M does not halt, we'll never advance to the correct value of f(x). Rather, we consider, for various combinations of i and j, whether M accepts [x,i] in j steps. If we consider (i,j) pairs in order of their sum (i.e., (0,1), (1,0), (0,2), (1,1), (2,0), (0,3),...) then eventually we shall simulate M on [x,f(x)] for a sufficient number of steps that M reaches acceptance. We need only wait until we consider pairs whose sum is f(x) plus however many steps it takes M to accept [x,f(x)]. In this manner, we can discover what f(x) is, write it on the tape of the TM that we have designed to compute f(x), and halt.

Now let us consider what happens if f is not defined for some arguments. Part (b) does not change, although the constructed TM will fail to discover f(x) and thus will

continue searching forever. For part (a), if we are given [x,y], and f is not defined on x, then the TM for f will never halt on x. However, there is nothing wrong with that. Since f(x) is undefined, surely y is not f(x). Thus, we do not want the TM for the graph of f to accept [x,y] anyway.

Exercise 8.2.4

These constructions, while they can be carried out using the basic model of a TM, are much clearer if we use some of the tricks of Sect. 8.3. 可用基本图灵机实现，但若用些技巧会使构造更加清楚！

For part (a), given an input [x,y] use a second track to simulate the TM for f on the input x. When the TM halts, compare what it has written with y, to see if y is indeed f(x). Accept if so. (多道).将 y 写在第一道，在第二道上模拟计算 f(x)的图灵机，将第二道上的输出与 y 比较，若相同则接受.

For part (b), given x on the tape, we need to simulate the TM M that recognizes the graph of f. However, since this TM may not halt on some inputs, we cannot simply try all [x,i} to see which value of i leads to acceptance by M.(注意不能简单地穷举以测试[x,i}是否被接受，因为对于某些输入不停机，导致检测不能继续进行，因此考虑(i,j)对，以保证每次测试都停机.若测试通过，则 i 即为 f(x)) The reason is that, should we work on some value of i for which M does not halt, we'll never advance to the correct value of f(x). Rather, we consider, for various combinations of i and j, whether M accepts [x,i] in j steps. If we consider (i,j) pairs in order of their sum (i.e., (0,1), (1,0), (0,2), (1,1), (2,0), (0,3),...) then eventually we shall simulate M on [x,f(x)] for a sufficient number of steps that M reaches acceptance. We need only wait until we consider pairs whose sum is f(x) plus however many steps it takes M to accept [x,f(x)]. In this manner, we can discover what f(x) is, write it on the tape of the TM that we have designed to compute f(x), and halt.

对(b)部分，在带上输入 x，我们需要模拟识别 f 图的 TM(i.e.图灵机)M.然而，因为该 TM 可能在某些输入上不停机，我们不能简单地试验所有的[x,i}以判断哪一个 i 值导致被 M 接受.理由是如果对某个 i 值 M 运行不停止，我们将不能继续前进到达正确的 f(x)值.对不同的 i 和 j 组合，考虑 M 是否在 j 步接受[x,i].如果按(i,j)对和的顺序考虑(i.e. (0,1), (1,0), (0,2), (1,1), (2,0), (0,3),... (这样的二元对既保证能逐次穷举 i，又保证每次都停机))则我们会在[x,f(x)]上模拟 M 足够多步最终使得 M 到达接受态. We need only wait until we consider pairs whose sum(和) is f(x) plus(加) however many steps it takes M to accept [x,f(x)](意思是我们只需要等到这样的序对出现，其序对的和是 f(x)加 M 接受[x,f(x)]所花费的步数).以这样的方式，我们能发现 f(x)是多少(写在设计的计算 f(x)的 TM 的带子上),并停机.

Now let us consider what happens if f is not defined for some arguments. Part (b) does not change, although the constructed TM will fail to discover f(x) and thus will continue searching forever. For part (a), if we are given [x,y], and f is not defined on x, then the TM for f will never halt on x. However, there is nothing wrong with that.

Since f(x) is undefined, surely y is not f(x). Thus, we do not want the TM for the graph of f to accept [x,y] anyway. (起作用，无须修改)现在让我们考虑当对某些参数 f 没定义时会发生什么.(b)部分不改变，尽管构造的 TM 将不能发现 f(x)从而永远扫描下去.对(a)部分，如果给定[x,y]，并且 f 在 x 上无定义，则计算 f 的 TM 在 x 上永远不停机.然而，这是没有问题的.因为 f(x)没定义，y 肯定不等于 f(x).因此，我们无论如何不想让接受 f 的图的 TM 接受[x,y].

Exercise 8.2.5(a)

This TM only moves right on its input. Moreover, it can only move right if it sees alternating 010101... on the input tape. Further, it alternates between states q_0 and q_1 and only accepts if it sees a blank in state q_1. That in turn occurs if it has just seen 0 and moved right, so the input must end in a 0. That is, the language is that of regular expression **(01)*0**.

Exercise 8.2.5(a)

This TM only moves right on its input. Moreover, it can only move right if it sees alternating (交替的)010101... on the input tape. Further, it alternates between states q_0 and q_1 and only accepts if it sees a blank in state q_1. That in turn occurs if it has just seen 0 and moved right, so the input must end in a 0. That is, the language is that of regular expression(正则表达式) **(01)*0**. (在状态 q_1 看到 B 进入接受态，但只当输入的最后一个为 0 时才能到达 q_1, 因此输入以 0 结尾)

Return to Top

Solutions for Section 8.3

Exercise 8.3.3

Here is the subroutine. Note that because of the technical requirements of the subroutine, and the fact that a TM is not allowed to keep its head stationary, when we see a non-0, we must enter state q3, move right, and then come back left in state q4, which is the ending state for the subroutine.

state	0	1	B
q1	(q2,0, R)	-	-
q2	(q2,0, R)	(q3,1, R)	(q3,B, R)
q3	(q4,0, L)	(q4,1, L)	(q4,B, L)

Now, we can use this subroutine in a TM that starts in state q0. If this TM ever sees the blank, it accepts in state qf. However, whenever it is in state q0, it knows only that it has not seen a 1 immediately to its right. If it is scanning a 0, it must check (in state q5) that it does not have a blank immediately to its right; if it does, it accepts. If it sees 0 in state q5, it comes back to the previous 0 and calls the subroutine to skip to the next non-0. If it sees 1 in state q5, then it has seen 01, and uses state q6 to check that it doesn't have another 1 to the right.

In addition, the TM in state q4 (the final state of the subroutine), accepts if it has reached a blank, and if it has reached a 1 enters state q6 to make sure there is a 0 or blank following. Note that states q4 and q5 are really the same, except that in q4 we are certain we are not scanning a 0. They could be combined into one state. Notice also that the subroutine is not a perfect match for what is needed, and there is some unnecessary jumping back and forth on the tape. Here is the remainder of the transition table.

state	0	1	B
q0	(q5,0, R)	(q6,1, R)	(qf,B, R)
q5	(q1,0, L)	(q6,1, R)	(qf,B, R)
q6	(q0,0, R)	-	(qf,B, R)
q4	-	(q6,1, R)	(qf,B, R)

Exercise 8.3.3

Here is the subroutine. Note that because of the technical requirements of the subroutine, and the fact that a TM is not allowed to keep its head stationary, when we see a non-0, we must enter state q3, move right, and then come back left in state q4, which is the ending state for the subroutine.
下面是子程序.注意到因为子程序的技术要求以及 TM 不允许带头保持静止这个事实，当我们看到一个非 0 时，必须进入状态 q3,向右移，然后向回走进入子程序的终止状态 q4.

state	0	1	B
q1	(q2,0, R)	-	-
q2	(q2,0,	(q3,1,	(q3,B,

	R)	R)	R)
q3	(q4,0, L)	(q4,1, L)	(q4,B, L)

Now, we can use this subroutine in a TM that starts in state q0. If this TM ever sees the blank, it accepts in state qf. However, whenever it is in state q0, it knows only that it has not seen a 1 immediately to its right. If it is scanning a 0, it must check (in state q5) that it does not have a blank immediately to its right; if it does, it accepts. If it sees 0 in state q5, it comes back to the previous 0 and calls the subroutine to skip to the next non-0. If it sees 1 in state q5, then it has seen 01, and uses state q6 to check that it doesn't have another 1 to the right.

现在我们可以在以 q0 为起始状态的 TM 里利用这个子程序.如果这个图灵机曾经看到空格，则进入状态 qf 接受.然而，无论何时它处在状态 q0，它仅仅知道它没有在紧挨的右边看到一个 1.

In addition, the TM in state q4 (the final state of the subroutine), accepts if it has reached a blank, and if it has reached a 1 enters state q6 to make sure there is a 0 or blank following. Note that states q4 and q5 are really the same, except that in q4 we are certain we are not scanning a 0. They could be combined into one state. Notice also that the subroutine is not a perfect match for what is needed, and there is some unnecessary jumping back and forth on the tape. Here is the remainder of the transition table.

state	**0**	**1**	**B**
q0	(q5,0, R)	(q6,1, R)	(qf,B, R)
q5	(q1,0, L)	(q6,1, R)	(qf,B, R)
q6	(q0,0, R)	-	(qf,B, R)
q4	-	(q6,1, R)	(qf,B, R)

Return to Top

Solutions for Section 8.4

Exercise 8.4.2(a)

For clarity, we put the state in square brackets below. Notice that in this example, we

never branch. $[q_0]01 \vdash 1[q_0]1 \vdash 10[q_1] \vdash 10B[q_2]$

Exercise 8.4.2(a)

For clarity, we put the state in square brackets below. Notice that in this example, we never branch. $[q_0]01 \vdash 1[q_0]1 \vdash 10[q_1] \vdash 10B[q_2]$ 为了清楚起见，我们将状态放在方括号内.注意在此例中，没有分支产生，i.e.没有发生在状态 q_1 读到 1 的情况.

Exercise 8.4.3(a)

We'll use a second tape, on which the guess x is stored. Scan the input from left yo right, and at each cell, guess whether to stay in the initial state (which does the scanning) or go to a new state that copies the next 100 symbols onto the second tape. The copying is done by a sequence of 100 state, so exactly 100 symbols can be placed on tape 2.

Once the copying is done, retract the head of tape 2 to the left end of the 100 symbols. Then, continue moving right on tape 1, and at each cell guess either to continue moving right or to guess that the second copy of x begins. In the latter case, compare the next 100 symbols on tape 1 with the 100 symbols on tape 2. If they all match, then move right on tape 1 and accept as soon as a blank is seen.

Exercise 8.4.3(a)

We'll use a second tape, on which the guess x is stored. Scan the input from left to right, and at each cell, guess whether to stay in the initial state (which does the scanning) or go to a new state that copies the next 100 symbols onto the second tape. The copying is done by a sequence of 100 state, so exactly 100 symbols can be placed on tape 2. 我们使用两条带，x 被储存在第二条带上.在第一条带上从左至右扫描输入，在每一单元“猜测”是呆在做扫描的初始状态，还是进入把接下来的 100 个字符复制到第二条带的新状态。复制由 100 个状态的序列完成，因此恰好 100 个字符能被放在第二条带上.

Once the copying is done, retract(收回) the head of tape 2 to the left end of the 100 symbols. Then, continue moving right on tape 1, and at each cell guess either to continue moving right or to guess that the second copy of x begins. In the latter case, compare the next 100 symbols on tape 1 with the 100 symbols on tape 2. If they all match, then move right on tape 1 and accept as soon as a blank is seen. 一旦复制完成，将第二条带子的读写头收回到 100 个字符的左端.接着，在第一条带上继续右移，并且在每一个单元“猜测”是继续右移，还是猜测 x 的第二次复制开始.在后一种情况下，将第一条带上接下来的 100 个字符和第二条带上的 100 个字符比较.如果全部匹配成功，则在第一条带上右移，并且看到空格立即接受.

Exercise 8.4.5

For part (a), guess whether to move left or right, entering one of two different states, each responsible for moving in one direction. Each of these states proceeds in its direction, left or right, and if it sees a $ it enters state p. Technically, the head has to move off the $, entering another state, and then move back to the $, entering state p as it does so.

Part (b), doing the same thing deterministically, is trickier, since we might start off in the wrong direction and travel forever, never seeing the $. Thus, we have to oscillate, using left and right endmarkers X and Y, respectively, to mark how far we have traveled on a second track. Start moving one cell left and leave the X. Then, move two cells right and leave the Y. Repeatedly move left to the X, move the X one more cell left, go right to the Y, move it once cell right, and repeat.

Eventually, we shall see the $. At this time, we can move left or right to the other endmarker, erase it, move back to the end where the $ was found, erase the other endmarker, and wind up at the $.

Exercise 8.4.5

For part (a), guess whether to move left or right, entering one of two different states, each responsible for(是...的原由) moving in one direction. Each of these states proceeds in its direction, left or right, and if it sees a $ it enters state p. Technically, the head has to move off (离开) the $, entering another state, and then move back to the $, entering state p as it does so.

对问题(a)，猜测是右移还是左移，进入两个不同状态中的一个，每一个状态引发一个方向上的移动.这些状态的每一个在它的方向上前进，向左或者向右，如果看到$就进入状态 p.在技术上，读写头必须离开$，进入另一个状态，然后返回到$，进入状态 p.

Part (b), doing the same thing deterministically, is trickier, since we might start off in the wrong direction and travel forever, never seeing the $. Thus, we have to oscillate(摆动), using left and right endmarkers X and Y, respectively, to mark how far we have traveled on a second track. Start moving one cell left and leave the X. Then, move two cells right and leave the Y. Repeatedly move left to the X, move the X one more cell left, go right to the Y, move it once cell right, and repeat.

对问题(b),用确定型自动机来做同样的事情，需要些技巧，因为我们可能从错误的方向开始，以致永远搜索下去，不可能看到$.因此，我们必须左右“摆动”，用 X 和 Y(左右端标记)分别在第二道上标记出已经走了多远.一开始左移一个单元并留下标记 X.然后右移两个单元并留下标记 Y.重复执行下面过程：左移到 X，再把 X 左移一个单元，向左移找到 Y，把 Y 右移一个单元〉

Eventually, we shall see the $. At this time, we can move left or right to the other endmarker, erase it, move back to the end where the $ was found, erase the other endmarker, and wind up(停止；结束) at the $.

最终，我们会看到$.此时，我们左移或右移到另外一个端点标记处，擦掉该处标记，然后返回到$被找到的一端，也擦掉该处标记，并在$停止.

注意此题目标是找到带子上仅有的符号$，如果$在当前读写头的左侧，而我们错误地一直朝右搜索，会失去在左边搜索$的机会(带子向右无限延伸)，因此采用每次朝两个方向扫描确定的长度,保证能扫描到所有的单元，上面的论述就是这个想法的细化。问题(a)用非确定型 TM 实现，做法很简单,因为非确定保证向两个方向同时猜测。问题(b)用确定型 TM 实现，就要避免单方向一直搜索下去的错误.

Exercise 8.4.8(a)

There would be 10 tracks. Five of the tracks hold one of the symbols from the tape alphabet, so there are 7^5 ways to select these tracks. The other five tracks hold either X or blank, so these tracks can be selected in 2^5 ways. The total number of symbols is thus $7^5 * 2^5 = 537,824$.

Exercise 8.4.8(a)(?不知所云)

There would be 10 tracks. Five of the tracks hold one of the symbols from the tape alphabet(带字母表), so there are 7^5 ways to select these tracks. The other five tracks hold either X or blank, so these tracks can be selected in 2^5 ways. The total number of symbols is thus $7^5 * 2^5 = 537,824$. 用具有 10 道的单带图灵机模拟 5 带图灵机. 其中的 5 个道拥有带字母表符号里的一个，因此有 7^5 种方式选择这些道.另外 5 个道拥有要么 X 要么空格，因此有 2^5 种方式选择这些道.从而符号总数为 $7^5 * 2^5 = 537,824$.

Exercise 8.4.8(b)

The number of symbols is the same. The five tracks with tape symbols can still be chosen in 7^5 ways. The sixth track has to tell which subset of the five tapes have their head at that position. There are 2^5 possible subsets, and therefore 32 symbols are needed for the 6th track. Again the number of symbols is $7^5 * 2^5$.

Exercise 8.4.8(b)

The number of symbols is the same. The five tracks with tape symbols can still be chosen in 7^5 ways. The sixth track has to tell which subset of the five tapes have their head at that position. There are 2^5 possible subsets, and therefore 32 symbols are needed for the 6th track. Again the number of symbols is $7^5 * 2^5$. 需要的带符号数相同. 有带符号(with tape symbols)的 5 个道仍然有 7^5 种选择方式.第 6 道必须指明 5

带中哪个子集在那个位置有它们的读写头.有 2^5 个可能的子集, 因此第 6 道需要 32 个符号.再一次得到带符号数为 $7^5 * 2^5$

Return to Top

Solutions for Section 8.5

Exercise 8.5.1(c)

In principle, any language that is recursively enumerable can be recognized by a 2-counter machine, but how do we design a comprehensible answer for a particular case? As the a's are read, count them with both counters. Then, when b's enter, compare them with one counter, and accept if they are the same. Continue accepting as long as c's enter. If the numbers of a's and b's differ, then compare the second counter with the number of c's, and accept if they match.

Exercise 8.5.1(c)(计数器机器 P244 页，PPT 中没有，也没讲)

In principle, any language that is recursively enumerable can be recognized by a 2-counter machine, but how do we design a comprehensible answer for a particular case? As the a's are read, count them with both counters. Then, when b's enter, compare them with one counter, and accept if they are the same. Continue accepting as long as c's enter. If the numbers of a's and b's differ, then compare the second counter with the number of c's, and accept if they match.

Solutions for Section 9.1

Exercise 9.1.1(a)

37 in binary is 100101. Remove the leading 1 to get the string 00101, which is thus w_{37}.

Exercise 9.1.3(a)

Suppose this language were accepted by some TM M. We need to find an i such that $M = M_{2i}$. Fortunately, since all the codes for TM's end in a 0, that is not a problem; we just convert the specification for M to a code in the manner described in the section.

We then ask if w_i is accepted by M_{2i}? If so, then w_i is not accepted by M, and therefore not accepted by M_{2i}, which is the same TM. Similarly, if w_i is not accepted by M_{2i}, then w_i is accepted by M, and therefore by M_{2i}. Either way, we reach a contradiction, and conclude that M does not exist.

Return to Top

Solutions for Section 9.2

Exercise 9.2.2(a)

A(2,1) = A(A(1,1),0) [rule 4] = A(A(A(0,1),0),0) [rule 4] = A(A(1,0),0) [rule 1] = A(2,0) [rule 2] = 4 [rule 3].

Exercise 9.2.3(a)

Let's keep i, the integer in unary, whose square we have most recently printed on the output tape, on tape 1, and keep i^2 on tape 2, also in unary. Initially, i = 0. Repeatedly do the following:

1. Add 1 to tape 1; now we have i+1 there.
2. Copy tape 1 to tape 2 twice, and remove one to change i^2 to $i^2 + 2(i+1) - 1 = (i+1)^2$.
3. Copy tape 2 to the output as a block of 0's and append a 1.

Exercise 9.2.4

By symmetry, if we can prove L_1 is recursive, we can prove any of the languages to be recursive. Take TM's M_1, M_2,...,M_k for each of the languages L_1, L_2,...,L_k, respectively. Design a TM M with k tapes that accepts L_1 and always halts. M copies its input to all the tapes and simulates M_I on the ith tape. If M_1 accepts, then M accepts. If any of the other TM's accepts, M halts without accepting. Since exactly one of the M_i's will accept, M is sure to halt.

Exercise 9.2.5

Note that the new language defined in the displayed text should be L'; it is different from the given language L, of course. Also, we'll use -L for the complement of L in what follows.

Suppose L' were RE. Then we could design a TM M for -L as follows. Given input w, M changes its input to 1w and simulates the hypothetical TM for L'. If that TM accepts, then w is in -L, so M should accept. If the TM for L' never accepts, then neither does M. Thus, M would accept exactly -L, which contradicts the fact that -L is not RE. We conclude that L' is not RE.

Exercise 9.2.6(a)

To test whether an input w is in the union of two recursive languages L1 and L2, we design a TM to copy its input w onto a second tape. It then simulates the halting TM for L1 on one tape and the halting TM for L2 on the other. If either accepts, then we accept. If both halt without accepting, we halt without accepting. Thus, the union is accepted by a TM that always halts.

In the case where L1 and L2 are RE, do the same, and accept if either accepts. The resulting TM accepts the union, although it may not halt. We conclude that both the recursive languages and the RE languages are closed under union.

Exercise 9.2.6(e)

Consider the case where L is RE. Design a NTM M for h(L), as follows. Suppose w is the input to M. On a second tape, M guesses some string x over the alphabet of L, checks that h(x) = w, and simulates the TM for L on x, if so. If x is accepted, then M accepts w. We conclude that the RE languages are closed under homomorphism.

However, the recursive languages are not closed under homomorphism. To see why, consider the particular language L consisting of strings of the form (M,w,c^i), where M is a coded Turing machine with binary input alphabet, w is a binary string, and c is a symbol not appearing elsewhere. The string is in L if and only if M accepts w after making at most i moves. Clearly L is recursive; we may simulate M on w for i moves and then decide whether or not to accept. However, if we apply to L the homomorphism that maps the symbols other than c to themselves, and maps c to ε, we find that h(L) is the universal language, which we called L_u. We know that L_u is not recursive.

Return to Top

Solutions for Section 9.3

Exercise 9.3.1

The property of languages ``contains all the palindromes'' is a nontrivial property, since some languages do and others don't. Thus, by Rice's theorem, the question is undecidable.

Exercise 9.3.4(d)

Revised 7/19/05

We shall reduce the problem L_e (does a TM accept the empty language?) to the question at hand: does a TM accept a language that is its own reverse? Given a TM M, we shall construct a nondeterministic TM M', which accepts either the empty language (which is its own reverse), or the language {01} (which is not its own reverse). We shall make sure that if L(M) is empty, then L(M') is its own reverse (the empty language, in particular), and if L(M) is not empty, then L(M') is not its own reverse. M' works as follows:

1. First, check that its input is 01, and reject if not.
2. Guess an input w for M.

3. Simulate M on w. If M accepts, then M' accepts its own input, 01.

Thus, if L(M) is nonempty, M' will guess some string M accepts and therefore accept 01. If L(M) is empty, then all guesses by M' fail to lead to acceptance by M, so M' never accepts 01 or any other string.

Exercise 9.3.6(a)

After making m transitions (not m+1 as suggested by the hint), the TM will have been in m+1 different states. These states cannot all be different. Thus, we can find some repeating state, and the moves of the TM look like $[q_0]$ |-* q |-* q |-* ..., where the central |-* represents at least one move. Note that we assume the tape remains blank; if not then we know the TM eventually prints a nonblank. However, if it enters a loop without printing a nonblank, then it will remain forever in that loop and never print a nonblank. Thus, we can decide whether the TM ever prints a nonblank by simulating it for M moves, and saying ``yes'' if and only if it prints a nonblank during that sequence of moves.

Exercise 9.3.7(a)

We reduce the complement of L_u to this problem, which is the complement of the halting problem for Turing Machines. The crux of the argument is that we can convert any TM M into another TM M', such that M' halts on input w if and only if M accepts w. The construction of M' from M is as follows:

1. Make sure that M' does not halt unless M accepts. Thus, add to the states of M a new state p, in which M' runs right, forever; i.e., $\delta(p,X) = (p,X,R)$ for all tape symbols X. If M would halt without accepting, say $\delta(q,Y)$ is undefined for some nonaccepting state q, then in M', make $\delta(q,Y) = (p,Y,R)$; i.e., enter the right-moving state and make sure M' does not halt.
2. If M accepts, then M' must halt. Thus, if q is an accepting state of M, then in M', $\delta(q,X)$ is made undefined for all tape symbols X.
3. Otherwise, the moves of M' are the same as those of M.

The above construction reduces the complement of L_u to the complement of the halting problem. That is, if M accepts w, then M' halts on w, and if not, then not. Since the complement of L_u is non-RE, so is the complement of the halting problem.

Exercise 9.3.8(a)

We'll show this problem not to be RE by reducing the problem of Exercise 9.3.7(a), the ``nonhalting'' problem to it. Given a pair (M,w), we must construct a TM M', such that M' halts on every input if and only if M does not halt on w. Here is how M' works:

1. Given an input x of length n, M' simulates M on w for n steps.
2. If during that time, M halts, then M' enters a special looping state [as discussed in the solution to Exercise 9.3.7(a)] and M' does not halt on its own input x.
3. However, if M does not halt on w after n steps, then M' halts.

Thus, M' halts on all inputs if and only if M does not halt on w. Since we proved in the solution to Exercise 9.3.7(a) that the problem of telling whether M does not halt on w is non-RE, it follows that the question at hand --- whether a given TM halts on all inputs --- must not be RE either.

Exercise 9.3.8(d)

This language is the complement of the language of Exercise 9.3.8(a), so it is surely not recursive. But is it RE? We can show it isn't by a simple reduction from the nonhalting problem. Given (M,w), construct M' as follows:

1. M' ignores its own input and simulates M on w.
2. If M halts, M' halts on its own input. However, if M never halts on w, then M' will never halt on its own input.

As a result, M' fails to halt on at least one input (in fact, on all inputs) if M fails to halt on w. If M halts on w, then M' halts on all inputs.

Return to Top

Solutions for Section 9.4

Exercise 9.4.1(a)

There is no solution. First, a solution would have to start with pair 1, because that is the only pair where one is a prefix of the other. THus, our partial solution starts:

A: 01
B: 011

Now, we need a pair whose A-string begins with 1, and that can only be pair 3. The partial solution becomes

A: 0110
B: 01100

Now, we need a pair whose A-string begins with 0, and either pair 1 or pair 2 might serve. However, trying to extend the solution with pair 1 gives us:

A: 011001
B: 01100011

while extending by pair 2 yields:

A: 0110001
B: 0110010

In both cases, there is a mismatch, and we conclude no solution exists.

Exercise 9.4.3

The problem is decidable by the following, fairly simple algorithm. First, if all the A-strings are strictly longer than their corresponding B-strings, then there is surely no solution. Neither is there a solution in the opposite case, where all the B-strings are strictly longer than their corresponding A-strings.

We claim that in all other cases, there is a solution. If any corresponding pair of strings are the same length, then they are identical, and so just that pair is a solution. The only possibility remains has at least one pair, say i, with the A-string longer than the B-string, say by m symbols, and another pair, say j, where the B-string is longer than the A-string, say by n symbols. Then $i^n j^m$, i.e., n uses of pair i followed by m uses of pair j is a solution. In proof, it is easy to check that both the A- and B-strings that result have the same length. Since there is only one symbol, these strings are therefore identical.

Return to Top

Solutions for Section 9.5

Exercise 9.5.1

Given an instance (A,B) of PCP, construct the grammar G_A as in the text. Also, construct a grammar G_{BR}, that is essentially G_B, but with the bodies of all productions reversed, so its language is the reverse of the language of G_B. Assume the start symbols of these grammars are A and B, they contain no variables in common, and that c is a terminal that does not appear in these grammars.

Construct a new grammar G with all the productions of G_A and G_{BR}, plus the production S -> AcB. Then a solution to the PCP instance yields a string y such that y is generated by G_A and y^R is generated by G_{BR}. Thus, G generates the palindrome ycy^R. However, any palindrome generated by G must have the c in the middle and thus implies a solution to the PCP instance, that is, a string y that appears in $L(G_A)$ while y^R appears in $L(G_{BR})$ [and therefore y appears in $L(G_B)$].

Exercise 8.1.1(a)
（参考 P219-220 例 8.1，注意归约的方向）
We need to take a program P and modify it so it: **修改程序 P 使得它满足**

3. Never halts unless we explicitly want it to, and 从不停机除非我们明显要求它停机
4. Halts whenever it prints hello, world. 一旦打印 hello, world 则停机

For (1), we can add a loop such as while(1){x=x;} to the end of main, and also at any point where main returns. That change catches the normal ways a program can halt, although it doesn't address the problem of a program that halts because some exception such as division by 0 or an attempt to read an unavailable device. Technically, we'd have to replace all of the exception handlers in the run-time environment(运行时期环境) to cause a loop whenever an exception occurred.

为做到(1),给程序 P 的 main 函数结尾处和任一 main 返回语句处增加死循环，如 while(1){x=x;}。接下来一段话意思是说，这样的改变能捕捉到程序正常停机情况，但不能区分异常停机(如被 0 除或试图读一个不能得到的设备)情况.因此我们必须做些技术处理——对异常停机替换异常处理装置使得一旦发生异常则引发死循环.这样保证只要停机必然打印 hello, world，因为不会有其它原因导致停机了.

For (2), we modify P to record in an array the first 12 characters printed. If we find that they are hello, world. we halt by going to the end of main (past the point where the while-loop has been installed).

为做到（2），修改 P，用一个数组来记录其打印的前 12 个字符。如果检查数组是 hello, world，则跳到 main 的结尾（也就是越过设置的 While 循环）。

注：上面的做法是为了从程序 P 构造程序 R，使得 R 对输入 z 最终停机当且仅当 P 对输入 y 显示 hello, world.记修改后所得程序为 R，如果能判定 R 对输入 z 是否停机，也就知 P 对输入 y（y=z）是否显示 hello, world. 因为已知不存在算法判定 hello-world 问题，而且通过编辑程序代码的程序就能完成从 P 到 R 的构造，所以存在最终停机检验程序的假设是错误的.不存在这样的程序，程序在输入上是否最终停机是不可判定的.具体说明类似 P221 第一段。

Exercise 8.2.1(a)

To make the ID's clearer in HTML, we'll use [q0] for q_0, and similarly for the other states.

[q0]00 |- X[q1]0 |- X0[q1]

The TM halts at the above ID.

利用 P225 图 8-9 的状态转移图.ID(瞬时描述)参见 P223-224

Exercise 8.2.2(a)

(注意与 P224 例 8.2 中 0^n1^n 的区别)
Here is the transition table for the TM: 下面是转移函数表

state	0	1	B	X	Y
q0	(q2,X,R)	(q1,X,R)	(qf,B,R)	-	(q0,Y,R)
q1	(q3,Y,L)	(q1,1,R)	-	-	(q1,Y,R)
q2	(q2,0,R)	(q3,Y,L)	-	-	(q2,Y,R)
q3	(q3,0,L)	(q3,1,L)	-	(q0,X,R)	(q3,Y,L)
qf	-	-	-	-	-

In explanation, the TM makes repeated excursions(短程旅行; 游览) back and forth along the tape. The symbols X and Y are used to replace 0's and 1's that have been cancelled one against another. The difference is that an X guarantees that there are no unmatched 0's and 1's to its left (so the head never moves left of an X), while a Y may have 0's or 1's to its left. 思路是 TM(图灵机)沿着带子不断地向后和向前游历.符号 X 和 Y 用来替代已经相互抵消了的 0 和 1.差别在于，X 保证其左侧没有不能匹配的 0 和 1，因此带头从不移到 X 的左侧。但是 Y 的左侧可能有 0 和 1。对照转移表，下面说明每个转移的含义.

Initially in state q0, the TM picks up a 0 or 1, remembering it in its state (q1 = found a 1; q2 = found a 0), and cancels what it found with an X. As an exception, if the TM sees the blank in state q0, then all 0's and 1's have matched, so the input is accepted by going to state qf.

(状态转移表的 q0 行) 起初在状态 q0，TM 读到一个 0 或 1，将其记在状态里(q1=读到一个 1；q2=读到一个 0)，并将所读到的用一个 X 抵消(也就是用 X 替换 0 或 1).有一个例外，如果 TM 在状态 q0 看到空格，则所有 0 和 1 已匹配，因此进入状态 qf 输入被接受.

In state q1, the TM moves right, looking for a 0. If it finds it, the 0 is replaced by Y, and the TM enters state q3 to move left an look for an X. Similarly, state q2 looks for a 1 to match against a 0.

(状态转移表的 q0 行和 q1 行)在状态 q1，TM 向右移，寻找一个 0.如果找到了，用 Y 代替 0，TM 进入状态 q3 向左移寻找一个 X.类似地，状态 q2 寻找一个 1 来匹配 0.

In state q3, the TM moves left until it finds the rightmost X. At that point, it enters state q0 again, moving right over Y's until it finds a 0, 1, or blank, and the cycle begins again.

(状态转移表的 q3 行)在状态 q3, TM 向左移直至找到最右边的 X.此时，又一次进入状态 q0，向右移穿过 Y 直至找到 0，1，或空格，循环又一次开始.

Exercise 8.2.4

These constructions, while they can be carried out using the basic model of a TM, are much clearer if we use some of the tricks of Sect. 8.3. 可用基本图灵机实现，但若用些技巧会使构造更加清楚！

For part (a), given an input [x,y] use a second track to simulate the TM for f on the input x. When the TM halts, compare what it has written with y, to see if y is indeed f(x). Accept if so. (多道).将 y 写在第一道，在第二道上模拟计算 f(x)的图灵机，将第二道上的输出与 y 比较，若相同则接受.

For part (b), given x on the tape, we need to simulate the TM M that recognizes the graph of f. However, since this TM may not halt on some inputs, we cannot simply try all [x,i} to see which value of i leads to acceptance by M.(注意不能简单地穷举以测试[x,i}是否被接受，因为对于某些输入不停机，导致检测不能继续进行，因此考虑(i,j)对，以保证每次测试都停机.若测试通过，则 i 即为 f(x)) The reason is that, should we work on some value of i for which M does not halt, we'll never advance to the correct value of f(x). Rather, we consider, for various combinations of i and j, whether M accepts [x,i] in j steps. If we consider (i,j) pairs in order of their sum (i.e., (0,1), (1,0), (0,2), (1,1), (2,0), (0,3),...) then eventually we shall simulate M on [x,f(x)] for a sufficient number of steps that M reaches acceptance. We need only wait until we consider pairs whose sum is f(x) plus however many steps it takes M to accept [x,f(x)]. In this manner, we can discover what f(x) is, write it on the tape of the TM that we have designed to compute f(x), and halt.

对(b)部分，在带上输入 x，我们需要模拟识别 f 图的 TM(i.e.图灵机)M.然而，因为该 TM 可能在某些输入上不停机，我们不能简单地试验所有的[x,i]以判断哪一个 i 值导致被 M 接受.理由是如果对某个 i 值 M 运行不停止，我们将不能继续前进到达正确的 f(x)值.对不同的 i 和 j 组合，考虑 M 是否在 j 步接受[x,i].如果按(i,j)对和的顺序考虑(i.e. (0,1), (1,0), (0,2), (1,1), (2,0), (0,3),... (这样的二元对既保证能逐次穷举 i，又保证每次都停机))则我们会在[x,f(x)]上模拟 M 足够多步最终使得 M 到达接受态. We need only wait until we consider pairs whose sum(和) is f(x) plus(加) however many steps it takes M to accept [x,f(x)](意思是我们只需要等到这样的序对出现，其序对的和是 f(x)加 M 接受[x,f(x)]所花费的步数).以这样的方式，我们能发现 f(x)是多少(写在设计的计算 f(x)的 TM 的带子上),并停机.

Now let us consider what happens if f is not defined for some arguments. Part (b) does not change, although the constructed TM will fail to discover f(x) and thus will

continue searching forever. For part (a), if we are given [x,y], and f is not defined on x, then the TM for f will never halt on x. However, there is nothing wrong with that. Since f(x) is undefined, surely y is not f(x). Thus, we do not want the TM for the graph of f to accept [x,y] anyway. (起作用，无须修改)现在让我们考虑当对某些参数 f 没定义时会发生什么.(b)部分不改变，尽管构造的 TM 将不能发现 f(x)从而永远扫描下去.对(a)部分，如果给定[x,y]，并且 f 在 x 上无定义，则计算 f 的 TM 在 x 上永远不停机.然而，这是没有问题的.因为 f(x)没定义，y 肯定不等于 f(x).因此，我们无论如何不想让接受 f 的图的 TM 接受[x,y].

Exercise 8.2.5(a)

This TM only moves right on its input. Moreover, it can only move right if it sees alternating (交替的)010101... on the input tape. Further, it alternates between states q_0 and q_1 and only accepts if it sees a blank in state q_1. That in turn occurs if it has just seen 0 and moved right, so the input must end in a 0. That is, the language is that of regular expression(正则表达式) **(01)*0**. (在状态 q_1 看到 B 进入接受态，但只当输入的最后一个为 0 时才能到达 q_1. 因此输入以 0 结尾)

Exercise 8.3.3

（将状态转移表转成状态转移图，下面文字的含义从图中易见）
Here is the subroutine. Note that because of the technical requirements of the subroutine, and the fact that a TM is not allowed to keep its head stationary, when we see a non-0, we must enter state q3, move right, and then come back left in state q4, which is the ending state for the subroutine.
下面是子程序.注意到因为子程序的技术要求以及 TM 不允许带头保持静止这个事实，当我们看到一个非 0 时，必须进入状态 q3,向右移，然后向回走进入子程序的终止状态 q4.

state	0	1	B
q1	(q2,0, R)	-	-
q2	(q2,0, R)	(q3,1, R)	(q3,B, R)
q3	(q4,0, L)	(q4,1, L)	(q4,B, L)

Now, we can use this subroutine in a TM that starts in state q0. If this TM ever sees the blank, it accepts in state qf. However, whenever it is in state q0, it knows only that it has not seen a 1 immediately to its right. If it is scanning a 0, it must check (in state q5) that it does not have a blank immediately to its right; if it does, it accepts. If it sees 0 in state q5, it comes back to the previous 0 and calls the subroutine to skip to the

next non-0. If it sees 1 in state q5, then it has seen 01, and uses state q6 to check that it doesn't have another 1 to the right.

现在我们可以在以 q0 为起始状态的 TM 里利用这个子程序.如果这个图灵机曾经看到空格，则进入状态 qf 接受.然而，无论何时它处在状态 q0，它仅仅知道它没有在紧挨的右边看到一个 1.如果它正在扫描 0，则必须检查（在状态 q5）在其紧挨的右侧没有空格。若有空格则接受。若在状态 q5 看见 0，则左移至前一个 0，并调用子程序以跳到下一个非 0。若在状态 q5 看见 1，那么就已经看见 01，用状态 q6 检查在在右侧没有另一个 1。

In addition, the TM in state q4 (the final state of the subroutine), accepts if it has reached a blank, and if it has reached a 1 enters state q6 to make sure there is a 0 or blank following. Note that states q4 and q5 are really the same, except that in q4 we are certain we are not scanning a 0. They could be combined into one state. Notice also that the subroutine is not a perfect match for what is needed, and there is some unnecessary jumping back and forth on the tape. Here is the remainder of the transition table.

另外，处在状态 q4（子程序的终态）的 TM 如果已到达一个空格则接受，如果已到达一个 1 则进入状态 q6 以确保接下来有一个 0 或空格。注意状态 q4 和 q5 实际相同，只不过在 q4 我们确定正在扫描的不是 0. q4 和 q5 可以合并为一个状态。也要注意到这个子程序并非与所需要的完全相配，在带上存在一些不必要的向前及向后跳跃。下面是转移表的其余部分.

state	0	1	B
q0	(q5,0, R)	(q6,1, R)	(qf,B, R)
q5	(q1,0, L)	(q6,1, R)	(qf,B, R)
q6	(q0,0, R)	-	(qf,B, R)
q4	-	(q6,1, R)	(qf,B, R)

Exercise 8.4.2(a)

For clarity, we put the state in square brackets below. Notice that in this example, we never branch. $[q_0]01 \vdash 1[q_0]1 \vdash 10[q_1] \vdash 10B[q_2]$ 为了清楚起见，我们将状态放在方括号内.注意在此例中，没有分支产生，i.e.没有发生在状态 q_1 读到 1 的情况.

Exercise 8.4.3(a)

We'll use a second tape, on which the guess x is stored. Scan the input from left to right, and at each cell, guess whether to stay in the initial state (which does the scanning) or go to a new state that copies the next 100 symbols onto the second tape. The copying is done by a sequence of 100 state, so exactly 100 symbols can be placed on tape 2. 我们使用两条带，x 被储存在第二条带上.在第一条带上从左至右扫描输入，在每一单元猜测是呆在初始状态（做扫描），还是进入把接下来的 100 个字符复制到第二条带的新状态。复制由 100 个状态的序列完成，因此恰好 100 个符号能被放在第二条带上.

Once the copying is done, retract(收回) the head of tape 2 to the left end of the 100 symbols. Then, continue moving right on tape 1, and at each cell guess either to continue moving right or to guess that the second copy of x begins. In the latter case, compare the next 100 symbols on tape 1 with the 100 symbols on tape 2. If they all match, then move right on tape 1 and accept as soon as a blank is seen. 一旦完成复制，将第二条带的读写头收回到这 100 个符号的左端.接着，在第一条带上继续右移，并且在每一个单元猜测是继续右移，还是猜测第二个 x 开始.在后一种情况下，将第一条带上接下来的 100 个字符和第二条带上的 100 个字符比较.如果全部匹配成功，则在第一条带上右移，并且看到空格就立即接受.

Exercise 8.4.5

For part (a), guess whether to move left or right, entering one of two different states, each responsible for(是...的原由) moving in one direction. Each of these states proceeds in its direction, left or right, and if it sees a $ it enters state p. Technically, the head has to move off (离开) the $, entering another state, and then move back to the $, entering state p as it does so.

对问题(a)，猜测是右移还是左移，进入两个不同状态中的一个，每一个状态引发一个方向上的移动.这些状态的每一个在它的方向上前进，向左或者向右，如果看到$就进入状态 p.在技术上，读写头必须离开$，进入另一个状态，然后返回到$，进入状态 p.

Part (b), doing the same thing deterministically, is trickier, since we might start off in the wrong direction and travel forever, never seeing the $. Thus, we have to oscillate(摆动), using left and right endmarkers X and Y, respectively, to mark how far we have traveled on a second track. Start moving one cell left and leave the X. Then, move two cells right and leave the Y. Repeatedly move left to the X, move the X one more cell left, go right to the Y, move it once cell right, and repeat.

对问题(b),用确定型自动机来做同样的事情，需要些技巧，因为我们可能从错误的方向开始，以致永远搜索下去，不可能看到$.因此，我们必须左右“摆动”，用 X 和 Y(左右端标记)分别在第二道上标记出已经走了多远.一开始左移一个单元并留下标记 X.然后右移两个单元并留下标记 Y.重复执行下面过程：左移到 X，再把 X 左移一个单元，向右移找到 Y，把 Y 右移一个单元。

Eventually, we shall see the $. At this time, we can move left or right to the other endmarker, erase it, move back to the end where the $ was found, erase the other endmarker, and wind up(停止；结束) at the $.

最终，我们会看到$.此时，我们左移或右移到另外一个端点标记处，擦掉该处标记，然后返回到$被找到的一端，也擦掉该处标记，并在$停止.

注意此题目标是找到带子上仅有的符号$，如果$在当前读写头的左侧，而我们错误地一直朝右搜索，会失去在左边搜索$的机会(带子向右无限延伸)，因此采用每次朝两个方向扫描确定的长度,保证能扫描到所有的单元，上面的论述就是这个想法的细化。问题(a)用非确定型 TM 实现，做法很简单,因为非确定保证向两个方向同时猜测。问题(b)用确定型 TM 实现，就要避免单方向一直搜索下去的错误.

Exercise 8.4.8(a)(?)

There would be 10 tracks. Five of the tracks hold one of the symbols from the tape alphabet(带字母表), so there are 7^5 ways to select these tracks. The other five tracks hold either X or blank, so these tracks can be selected in 2^5 ways. The total number of symbols is thus $7^5 * 2^5 = 537,824$. 用具有 10 道的单带图灵机模拟 5 带图灵机. 其中的 5 个道拥有带字母表符号里的一个，因此有 7^5 种方式选择这些道.另外 5 个道拥有要么 X 要么空格，因此有 2^5 种方式选择这些道.从而符号总数为 $7^5 * 2^5 = 537,824$.

Exercise 8.4.8(b)

The number of symbols is the same. The five tracks with tape symbols can still be chosen in 7^5 ways. The sixth track has to tell which subset of the five tapes have their head at that position. There are 2^5 possible subsets, and therefore 32 symbols are needed for the 6th track. Again the number of symbols is $7^5 * 2^5$. 需要的带符号数相同. 有带符号(with tape symbols)的 5 个道仍然有 7^5 种选择方式.第 6 道必须指明 5 带中哪个子集在那个位置有它们的读写头.有 2^5 个可能的子集, 因此第 6 道需要 32 个符号.再一次得到带符号数为 $7^5 * 2^5$

Exercise 8.5.1(c)(计数器机器 P244 页，PPT 中没有，也没讲)

In principle, any language that is recursively enumerable can be recognized by a 2-counter machine, but how do we design a comprehensible answer for a particular case? As the a's are read, count them with both counters. Then, when b's enter, compare them with one counter, and accept if they are the same. Continue accepting as long as c's enter. If the numbers of a's and b's differ, then compare the second counter with the number of c's, and accept if they match.

Exercise 9.1.1(a)

37 in binary is 100101. Remove the leading 1 to get the string 00101, which is thus w_{37}.
在整数与二进制串之间建立一一对应关系. w_{37} 指第 37 个串.(参考 P256 枚举二进制)
37 的二进制表示是 100101.去掉最高位的 1 得到串 00101，这就是 w_{37}

Exercise 9.1.3(a)

Suppose this language were accepted by some TM M. We need to find an i such that M = M_{2i}. Fortunately, since all the codes for TM's end in a 0, that is not a problem; we just convert the specification for M to a code in the manner described in the section.

We then ask if w_i is accepted by M_{2i}? If so, then w_i is not accepted by M, and therefore not accepted by M_{2i}, which is the same TM. Similarly, if w_i is not accepted by M_{2i}, then w_i is accepted by M, and therefore by M_{2i}. Either way, we reach a contradiction, and conclude that M does not exist.

注意:M_{2i} 指第 i 的 2 倍个图灵机.类似论证参考定理 9.2 及图 9-1。L={w_i ｜w_i 不属于 M_{2i}}。
假设这个语言被某个 TM M 接受.我们需要寻找 i 使得 M = M_{2i}.幸运的是，所有图灵机(TM)的编码以 0 结尾，这不是个问题;我们只需要把 M 的规格转化为本节描述的编码(参见 P257 图灵机的编码),然后在编码前加 1，再转为 10 进制，这个数必然是偶数(因为编码以 0 结尾),i 是这个数的一半.
现在问 w_i 是否被 M_{2i} 接受。如果 w_i 被 M_{2i} 接受，那么 w_i 不被 M 接受(定义)，从而不被 M_{2i} 接受(M = M_{2i})，矛盾.类似地，如果 w_i 不被 M_{2i} 接受，那么按定义 w_i 被 M 接受，从而被 M_{2i} 接受,矛盾.无论哪种，我们都得到了矛盾，因此这样的 M 不存在.
注意:现在我们找到的无穷点序列是(2,1),(4,2),(6,3)…

Exercise 9.2.2(a)

A(2,1) = A(A(1,1),0) [规则 4] = A(A(A(0,1),0),0) [规则 4] = A(A(1,0),0) [规则 1] = A(2,0) [rule 2] = 4 [规则 3].

Exercise 9.2.3(a)

Let's keep i, the integer in unary, whose square we have most recently printed on the output tape, on tape 1, and keep i^2 on tape 2, also in unary. Initially, i = 0. Repeatedly do the following:

4. Add 1 to tape 1; now we have i+1 there.
5. Copy tape 1 to tape 2 twice, and remove one to change i^2 to $i^2 + 2(i+1) - 1 = (i+1)^2$.
6. Copy tape 2 to the output as a block of 0's and append a 1.

在带 1 上保留 i，它的平方已经在最近打印到输出，在带 2 上保留 i^2，i 和 i^2 是以 i 和 i^2 个 0 的方式表示的(此即 in unary 的含义). 开始时 i = 0.重复如下操作:

1. 给带 1 的 i 加 1，即给带 1 添加一个 0；现在带 1 上是 i+1,即 i+1 个 0.
2. 将带 1 复制两次到带 2，再去掉 1 个 0，这就把带 2 上的 i^2 变成了 $i^2 + 2(i+1) - 1 = (i+1)^2$.
3. 将带 2(上面有 $(i+1)^2$ 个 0)复制到输出，并在尾部添加一个 1.

Exercise 9.2.4

By symmetry, if we can prove L_1 is recursive, we can prove any of the languages to be recursive. Take TM's M_1, M_2,...,M_k for each of the languages L_1, L_2,...,L_k, respectively. Design a TM M with k tapes that accepts L_1 and always halts. M copies its input to all the tapes and simulates M_I on the ith tape(此处答案印刷有误，应该是 simulates M_i on the ith tape). If M_1 accepts, then M accepts. If any of the other TM's accepts, M halts without accepting. Since exactly one of the M_i's will accept, M is sure to halt.
由对称性(L_1, L_2,...,L_k 的地位平等),如果我们能够证明 L_1 是递归的，那么就能够证明任一个语言是递归的.设 M_1, M_2,...,M_k 分别是语言 L_1, L_2,...,L_k 的图灵机(TM). 现在设计一个图灵机 M，它有 k 条带，接受语言 L_1 并且总是停机.M 把它的输入复制到所有的带子上，并在第 i 条带子上模拟 M_i. 如果 M_1 接受，则 M 接受.如果其它任何一个图灵机接受，M 停机但不接受.因为正好有 M_1, M_2,...,M_k 的一个接受，M 肯定会停机.

Exercise 9.2.5

Note that the new language defined in the displayed text should be L'; it is different from the given language L, of course. Also, we'll use -L for the complement of L in what follows.
Suppose L' were RE. Then we could design a TM M for -L as follows. Given input w, M changes its input to 1w and simulates the hypothetical TM for L'. If that TM accepts, then w is in -L, so M should accept. If the TM for L' never accepts, then neither does M. Thus, M would accept exactly -L, which contradicts the fact that -L is not RE. We conclude that L' is not RE.

注意在文中新定义的语言应该是 L' ；它同给定的语言 L 当然不同.同时我们在下面用-L 代表 L 的补.
假设 L'是递归可枚举的(RE),那么我们按如下方式设计一个 -L 的图灵机 M.给定输入 w，M 将其输入变为 1w 并模拟假设的 L'的图灵机.如果这个图灵机接受，则 w 属于-L，因此 M 应该接受.如果这个图灵机从不接受，则 M 也不会接受.因此，M 将正好接受-L，这与事实-L 不是递归可枚举的(RE)矛盾(已知条件).我们得到 L'不是递归可枚举的(RE)

Exercise 9.2.6(a)

To test whether an input w is in the union of two recursive languages L1 and L2, we design a TM to copy its input w onto a second tape. It then simulates the halting TM for L1 on one tape and the halting TM for L2 on the other. If either accepts, then we accept. If both halt without accepting, we halt without accepting. Thus, the union is accepted by a TM that always halts.

In the case where L1 and L2 are RE, do the same, and accept if either accepts. The resulting TM accepts the union, although it may not halt. We conclude that both the recursive languages and the RE languages are closed under union.

为了测试一个输入 w 是否属于递归语言 L1 和 L2 的并，我们设计一个图灵机，它把输入 w 复制到第二条带上.然后它在一条带上模拟 L1 的图灵机，在另外一条带上模拟 L2 的图灵机.如果任意一个接受，则我们接受.如果两者都停机但没有接受，那么我们停机但不接受.因此并集被一个总是停机的 TM 接受. 我们得出递归在并运算下封闭.
当 L1 和 L2 都是递归可枚举时，做法相同，如果任意一个接受则接受.这样的图灵机接受并集，尽管可能不会停机.我们得出递归可枚举语言在并运算下封闭.

Exercise 9.2.6(e)

Consider the case where L is RE. Design a NTM M for h(L), as follows. Suppose w is the input to M. On a second tape, M guesses some string x over the alphabet of L, checks that h(x) = w, and simulates the TM for L on x, if so. If x is accepted, then M accepts w. We conclude that the RE languages are closed under homomorphism.

However, the recursive languages are not closed under homomorphism. To see why, consider the particular language L consisting of strings of the form (M,w,c^i), where M is a coded Turing machine with binary input alphabet, w is a binary string, and c is a symbol not appearing elsewhere. The string is in L if and only if M accepts w after making at most i moves. Clearly L is recursive; we may simulate M on w for i moves and then decide whether or not to accept. However, if we apply to L the homomorphism that maps the symbols other than c to themselves, and maps c to ε, we find that h(L) is the universal language, which we called L_u. We know that L_u is not recursive. 同态：用相关串替换每个符号。参见 P95 正则语言的同态.
考虑 L 是 RE（递归可枚举）的情况。按如下方式对 h(L)设计一个 NTM（非确定性图灵机）M。假设 w 是 M 的输入。在第二条带上，M 在字母表 L 中猜测某个串 x，检验 h(x) = w 是否成立。如果成立，则在输入 x 上模拟 L 的图灵机。如果 x 被接受，则 M 接受 w，我们得出 RE 语言在同态下封闭.(定理 8.11)
但是递归语言在同态运算下不封闭。为了看清楚怎么回事，我们考虑如下的特殊语言，它由形如(M,w,c^i)的串组成，其中 M 是具有输入字母表{0,1}的已编码的图灵机，w 是二进制串，c 是一个在别处不出现的符号。串 w 属于 L 当且仅当 M 在做了至多 i 次移动后接受 w.很明显 L 是递归的——我们可以在输入 w 上模拟 M 的 i 次移动，然后决定是否接受。但是如果我们将如下的同态作用到 L 上：将不是 c 的其余符号映射到自身，将 c 映射到 ε，我们发现 h(L)是通用语言（称作 L_u），但已知 L_u 不是递归的（定理 9.6）.