Introduction to Automata Theory, Languages, and Computation

Solutions for Chapter 4

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.

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.

Exercise 4.1.4(a)

We cannot pick *w* from the empty language.

Exercise 4.1.4(b)

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

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.

Solutions for Section 4.2

Exercise 4.2.1(a)

aabbaa.

Exercise 4.2.1(c)

The language of regular expression **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}*.

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 *delta(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*.

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:

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)*.

Exercise 4.2.5(e)

L may have no string that begins with 0.

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.

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)].

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*.

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:

- *q* is the state *A* would be in after reading whatever input *B* has read so far.
- *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.

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*.

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

- 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.

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 O*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.

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*.

Solutions for Section 4.3

Exercise 4.3.1

Let *n* be the pumping-lemma constant. Test all strings of length between *n* and 2*n*-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*.

Suppose, however, that there are no strings in *L* whose length is in the range *n* to 2*n*-1. We claim there are no strings in *L* of length 2*n* or more, and thus there are only a finite number of strings in *L*. In proof, suppose *w* is a string in *L* of length at least 2*n*, and *w* is as short as any string in *L* that has length at least 2*n*. 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 2*n*, because it is shorter than *w*, and *w* is as short as any string in *L* of length 2*n* or more. *n*, because *xz* is at most *n* shorter than *w*. Thus, *xz* is of length between *n* and 2*n*-1, which is a contradiction, since we assumed there were no strings in *L* with a length in that range.

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.