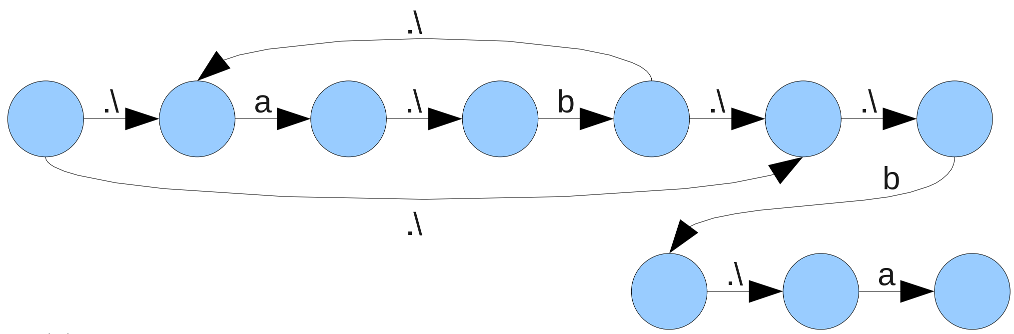
Languages and Machines

Chap 6: (p. 217): 1, 2, 5 (p 218): 7, 14, 24, 27 8,11bc (Use pumping lemma), 20.

1.



2. (a)

beginning table:

	a	b	.\
q0	{q1}	{q0}	{qf}
q1	{q2}	{}	{qf}
q2	{}	{q2,q1}	{}
qf	{}	{}	{}

ending table:

	a	b	.\	aa	a U .\	ba U b	aa(baUb)*b	(a U .\) U aa(baUb)*b
q0	{}	{q0}	{}	{}	{}	{}	{}	{qf}
qf	{}	{}	{}	{}	{}	{}	{}	{}

start = q0, final = qf
choose: q1
for:
j=0, k=0
j=0, k=2 add aa from q0 to q2
j=0, k=f add (a) from q0 to qf ... replace with arc (a U .\) from q0 to qf
j= 2, k=0
j=2, k=2 add (ba) from q2 to q2 ... replace with arc (ba U b)
j=2, k=f add (b) from q2 to qf
j=f, k=0; j=f, k=2; j=f, k=f
remove q1 and all its arcs
choose: q2
j=0, k=0
j=0, k=f add (aa(baUb)*b) from q0 to qf ... replace with arc ((a U .\) U aab)
j=f, k=0
j=f, k=f
remove q2 and all its arcs
final expression: b*(a U .\) U aa(baUb)*b

(I did the rest by hand to save time)
(b)
(((bb*a)Ua)(a*a)*b)*((bb*a)Ua)(ab*a)*
(c)
((aa*b)Uaa)(ba)*
(d)
(abba U (aa U b))((ab(abUb)*b) U (a U .\))

5.
(a)
0 -> a1 | .\
1 -> b0 | a1 | a2
2 -> b1 | .\
(b)
((aa*a(ba*a)*ba*b) U aa*b)*((aa*a(ba*a)*) U .\)

7.
(a)
A regular language concatenated with another regular language is regular (theorem 6.4.1)

(b)
By theorem 6.4.1, the union of two regular languages is regular. The union of the language L without a's with the language L with a's is regular.

(c)
By theorem 6.4.2, the compliment of L (L1) is regular. The language without a's is some subset of the compliment of L (L2), which is regular. The intersection of L1 and L2 is regular (theorem 6.4.3)

(d)
u is regular. V is an element of the compliment of L, which is regular (6.4.2). the concatenation of two regular languages is regular (6.4.1)

14.
(a)
Suppose L is regular. Then there is a DFA with k states which represents L and a decomposition of every string uvw such that uv^iw is in the language.
Since the pumping lemma applies to every string in the language, as long as it is potentially infinite, let us choose a palindromic string of the form b^kab^k, where k is the number of states in the DFA
Palindromes are strings where $w = w^R$. Therefore, a decomposition uvw must equal wvu.
The only possible decomposition:

u	v	w
b^{k-x}	b^x	ab^k

On pumping v i times, we find that the string becomes $b^{k+i}ab^k$, which does not equal its reversal, b^kab^{k+i}, when i >0

(b)

Let our exposition of the proof be the same as above. Let our string in this case be $a^k b^{k+1}$
The only possible decomposition:

u	v	w
a^{k-x}	a^x	b^{k+1}

On pumping v i times, we find that the number of a's will eventually exceed the number of b's, which goes against the rules of the language stating that number of b's must be greater than the number of a's

(c)
Let us choose the string $a^x b^{k-x} c^{2(k-x)}$
The only possible decomposition:

u	v	w
$a^x b^{k-x-y}$	b^y	$c^{2(k-x)}$

On pumping v i times, b will eventually exceed the number of c's, which goes against the rules of the language stating that the number of c's must be double that of the b's

(d)
Let us choose the string $a^k a^k$ which is in the language where $w = a^k$
The only possible decomposition:

u	v	w
a^{k-x}	a^x	a^k

Pumping v i times would produce the string $a^{k+i} a^k$. In the case where i is 1, then it cannot possibly be the string $a^z a^z$, which would be in the language, but could only be $a^z aa^z$, which is not in the language.

(e)
Let us choose the initial sequence ab where a = k and b is any length.
Our decomposition:

u	v	w
p^{k-x}	q^x	b

p and q are a decomposition of a, and we do not know exactly what they consist of, so we must examine the cases:

- q consists of one b. Pumping that b will produce multiple adjacent b's, which is not in the language
- q consists of some number of a's, call it n. Pumping q once will produce of a sequence of a's greater than n. Since the next sequence after the next b is n+1, we will have created a sequence of a's out of order
- q consists of n a's followed by a b. Pumping it once will produce another repetition of n a's and a b, which goes against the pattern, because every next sequence of a's must be n+1
- q consists of a b followed by n a's. Pumping it once will not be in the language for the same reason as above.

(f)
Let us choose the arbitrary string $a^{k^\wedge 3}$ belonging to L and greater than k
We assume there is a decomposition uvw, etc. If we pump v twice, we get uv^3w, which by the assumption is still in L. We now proceed similarly to the proof in example 6.6.1:
length(uv^3w) = length(uvw) + length(v) + length(v) <= $k^3 + k + k < k^3 + 3k^2 + 3k + 1 = (k+1)^3$
Therefore, the length of string uv^3w is in between the cubes of k and k+1, so it must not be a cube itself

and is not in the language

24.
.\
a+
a+b+
b(aUb)* U a+b+a(aUb)*

27.
Let i be an arbitrary number and j be less than i. $a^{2^\wedge i}$ and $a^{2^\wedge j}$ are distinguishable since for the concatenation of the string $a^{2^\wedge i}$, one is in the language and one is not. $a^{2^\wedge i}a^{2^\wedge i} = a^{2^\wedge i+1}$ is in the language. However, $a^{2^\wedge i}a^{2^\wedge j}$ is not, because j is less than i and so is in between the square of the two natural numbers i and i+1 and thus is not a perfect square.

8.
Set difference can be defined as taking the intersection of L_1 and L_2, which we may call L_i, and then the complement of L_i with respect to L_1. Since intersection and complement preserve regularity, difference then also preserves regularity.

11.
I think that the pumping lemma cannot be used to prove the regularity of these languages, because we have no assurance that the words in L are potentially infinite, which is required for the language to be guaranteed to have cycles.
(b)
By the definition of reversal (2.1.5), the only modifying process used in the definition is concatenation, which is by definition a process used to create regular languages.
(c)
Again, concatenation by definition creates regular languages. We do not have any specifications about the contents of the suffix, so I assume it is either the union of all elements of the alphabet starred, or it is a fixed list of words.

20.
Theorem 6.3.1 shows that any grammar of the form X -> aY | b (right recursive, a regular grammar) can be translated into a DFA. A language is regular if it can be accepted by a DFA (p. 200 and elsewhere)