Monash University
Faculty of Information Technology
2^{nd} Semester 2023

FIT2014 Theory of Computation
Solutions for Exercises 3
Regular Languages, Inductive Definitions, Finite Automata, Kleene's Theorem I

Although you may not need to do all the many exercises in this Exercise Sheet, it is still important that you attempt all the main questions and a selection of the Supplementary Exercises.

Even for those Supplementary Exercises that you do not attempt seriously, you should still give some thought to how to do them before reading the solutions.

1. **abab**, **baab**, **abaaab**, **abbbab**, **baaaab**, **babbab**, **abaaaaab**, **abaabbab**, **abbbaaab**, **abbbbbab**, **baaaaaab**, **baaabbab**, **babbaaab**, **babbbbab**

2.

(i) $(\mathbf{a} \cup \mathbf{b})(\mathbf{a} \cup \mathbf{b})$ or $\mathbf{aa} \cup \mathbf{ab} \cup \mathbf{ba} \cup \mathbf{bb}$

(ii) $\mathbf{a}^*\mathbf{ba}^*\mathbf{ba}^* \cup \mathbf{a}^*\mathbf{ba}^*\mathbf{ba}^*\mathbf{ba}^*$

(iii) $(\mathbf{a} \cup \mathbf{b})^*(\mathbf{aa} \cup \mathbf{bb})$

(iv) $(\mathbf{a} \cup \mathbf{b})^*(\mathbf{ab} \cup \mathbf{ba}) \cup \mathbf{a} \cup \mathbf{b} \cup \epsilon$

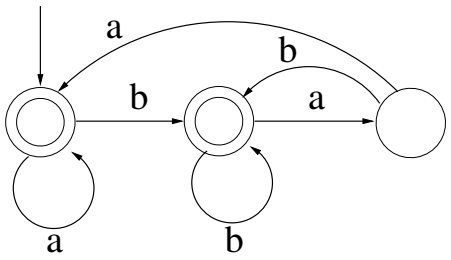
(v) $(\mathbf{b} \cup \epsilon)(\mathbf{ab})^*\mathbf{aa}(\mathbf{ba})^*(\mathbf{b} \cup \epsilon) \cup (\mathbf{a} \cup \epsilon)(\mathbf{ba})^*\mathbf{bb}(\mathbf{ab})^*(\mathbf{a} \cup \epsilon)$

(vi) $(\mathbf{aa} \cup \mathbf{ab} \cup \mathbf{ba} \cup \mathbf{bb})^*$

(vii) $\mathbf{a}^*((\mathbf{b} \cup \mathbf{bb})\mathbf{aa}^*)^*(\mathbf{b} \cup \mathbf{bb} \cup \epsilon)$

3.

4.



5.
(a) states 1,3,4,5,6 (i.e., any state except 2).

(b) states 3, 6.

(c) Yes, the string `aaab` is accepted by the NFA because at least one of its computations ends in a Final State (state 6).

(d) If the string $(\texttt{aaab})^n$ is accepted by the NFA, then (by definition of acceptance) there is some computation path for $(\texttt{aaab})^n$ that goes from state 1 to state 6.

We can use this to create an accepting computation for $(\texttt{aaab})^{n+1}$, in either of two possible ways:

(i) We can start with a path $1 \to 1$ for $\texttt{aaab}$ (which we know is possible by part (a), since the list of states includes 1) and follow it with a path $1 \to 6$ for $(\texttt{aaab})^n$. Putting these together gives a path $1 \to 1 \to 6$ for $(\texttt{aaab})^{n+1}$, which is an accepting computation for $(\texttt{aaab})^{n+1}$.

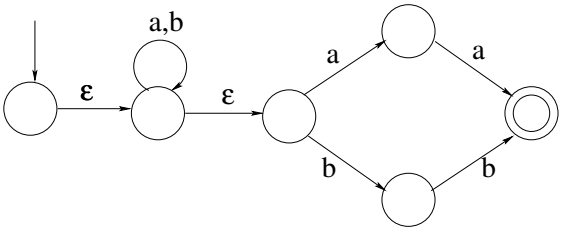
(i) We can start with a path $1 \to 6$ for $(\texttt{aaab})^n$ and follow it with a path $6 \to 6$ for $\texttt{aaab}$ (which we know is possible by part (b), since the list of states there includes 6). Putting these together gives a path $1 \to 6 \to 6$ for $(\texttt{aaab})^{n+1}$, which is an accepting computation for $(\texttt{aaab})^{n+1}$.

(e)

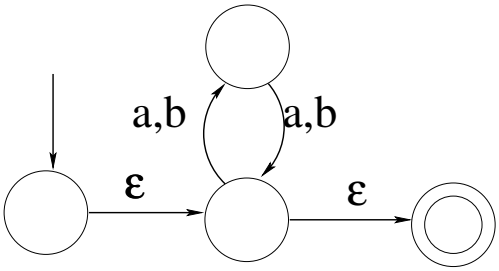
Part (a) gives the inductive basis.
Then (with $n \geq 1$) the assumption that "the string $(\texttt{aaab})^n$ is accepted by the NFA" is the Inductive Hypothesis.
The step from n to $n+1$ can be done in one of two ways, either by (d)(i) or (d)(ii).
Then we finish by deducing that the claim holds for all n, by Mathematical Induction.

6.

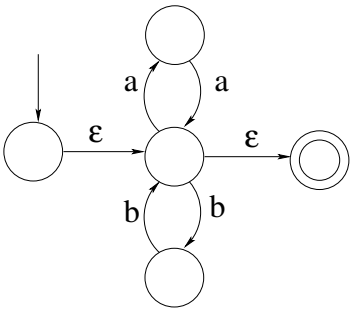
i)



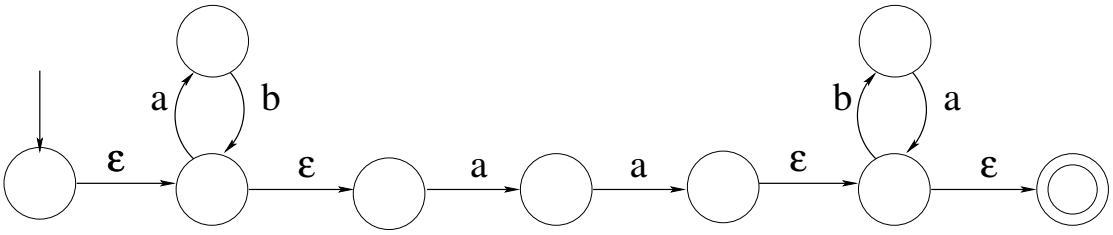
ii)



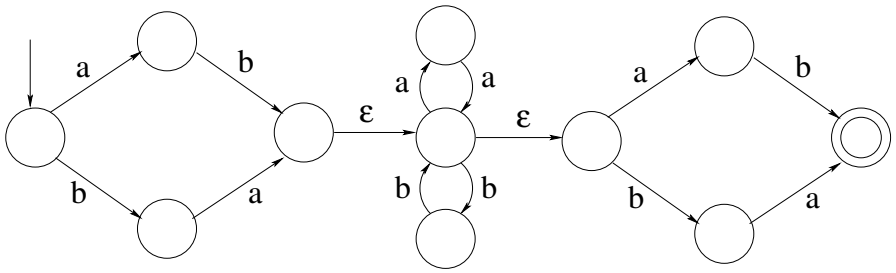
iii)



iv)

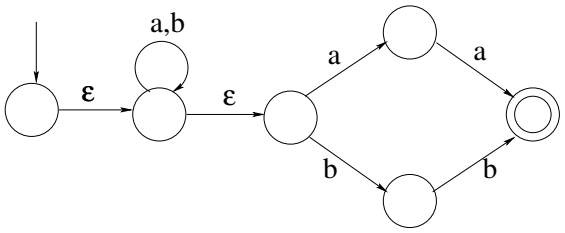


v)



7.

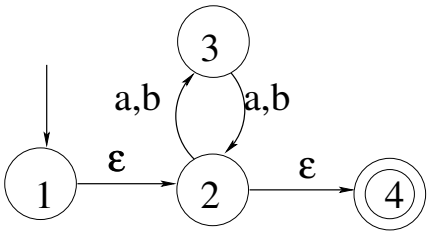
i) NFA



FA

	a	**b**
Start {1,2,3}	{2,3,4}	{2,5,3}
{2,3,4}	{2,3,4,6}	{2,3,5}
{2,3,5}	{2,3,4}	{2,3,5,6}
Final {2,3,4,6}	{2,3,4,6}	{2,3,5}
Final {2,3,5,6}	{2,3,4}	{2,3,5,6}

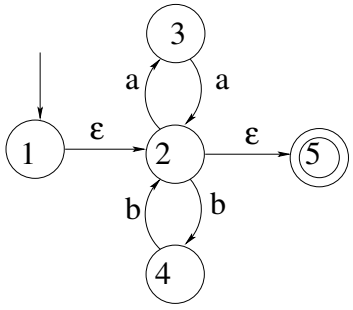
ii) NFA



FA

	a	**b**
Start, Final {1,2,4}	{3}	{3}
{3}	{2,4}	{2,4}
Final {2,4}	{3}	{3}

iii) NFA



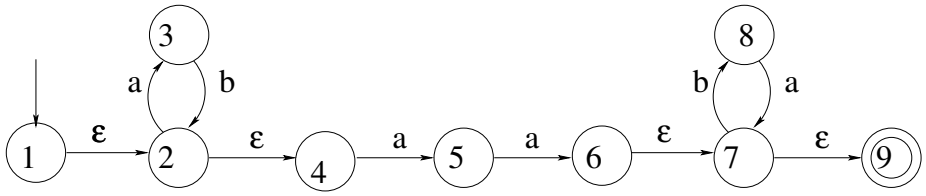
[1]

FA

	a	**b**
Start, Final {1,2,5}	{3}	{4}
{3}	{2,5}	ϕ
{4}	ϕ	{2,5}
Final {2,5}	{3}	{4}
ϕ	ϕ	ϕ

[1] Thanks to FIT2014 student Jingyan Lou and tutor Harald Bögeholz for spotting an error in a previous version of the solution to (iii).

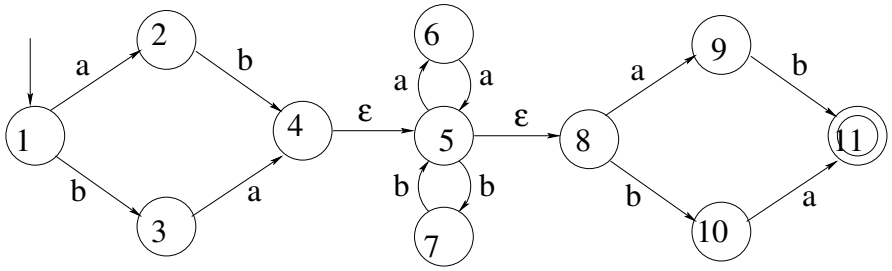
iv) NFA



FA

	a	**b**
Start {1,2,4}	{3,5}	ϕ
{3,5}	{6,7,9}	{2,4}
Final {6,7,9}	ϕ	{8}
{2,4}	{3,5}	ϕ
{8}	{7,9}	ϕ
Final {7,9}	ϕ	{8}
ϕ	ϕ	ϕ

v) NFA



FA

	a	**b**
Start {1}	{2}	{3}
{2}	ϕ	{4,5,8}
{3}	{4,5,8}	ϕ
{4,5,8}	{6,9}	{7,10}
{6,9}	{5,8}	{11}
{7,10}	{11}	{5,8}
{5,8}	{6,9}	{7,10}
Final {11}	ϕ	ϕ
ϕ	ϕ	ϕ

Supplementary exercises

8. **aabaaa**, **aababb**, **aaabaa**, **aaabbb**, **bbbaaa**, **bbbabb**, **bbabaa**, **bbabbb**

9. **aaaaaa**, **aaaabb**, **aabbaa**, **aabbbb**, **bbaaaa**, **bbaabb**, **bbbbaa**, **bbbbbb**

10. A valid date not described is 5/3/2002, and invalid date described is 99/02/2002.

11. $\mathbf{H} : \mathbf{M} : \mathbf{S}$, where $H = [0-9]|[01][0-9]|2[0-3]$, $M = [0-9]|[0-5][0-9]$, and $S = [0-9]|[0-5][0-9]$.

12. (- | +)?(N| N.| N.N| .N)((e| E)(- | +)?N)?, where $N = [0-9]^+$

13.

If we drop 3(i),(ii), then no grouping or concatenation is possible.

So our regular expressions can be constructed from letters (and empty strings) just using alternatives and Kleene $*$. Because no grouping is possible, the Kleene $*$ can only be applied to a single letter. (Note that, if R is any regular expression, then $R^{**} = R^*$.) So in this case the only regular expressions we get are unions of expressions of the form x^* and y, where x, y are single letters from our alphabet. For example, we could have a* ∪ b. The only languages that match such expressions are those in which, firstly, no string has a mix of letters, and secondly, for any letter, we either have all strings consisting just of repetitions of that letter, or just the one-letter string with that letter alone.

For the Challenge: it would be a big task to investigate all these possibilities. We just give one, as an illustration.

Suppose we drop just 3(ii). So we forbid concatenation but allow grouping, alternatives and Kleene $*$.

Firstly, observe that, if R and S are regular expressions, then the regular expression $(R^* \cup S)^*$ may be simplified to $(R \cup S)^*$.

Proof:

($\Leftarrow$) This implication is clear, since R is a special case of R^*.

($\Rightarrow$) Let w be a string that matches $(R^* \cup S)^*$. We must show that it also matches $(R \cup S)^*$. If $w = \varepsilon$, then we are done, since the empty string matches $(R \cup S)^*$ using zero repetitions for the Kleene $*$. So assume $w \neq \varepsilon$. Then w may be partitioned into consecutive substrings, $w = w_1 \cdots w_k$, such that each w_i matches $R^* \cup S$. If w_i matches R^*, then it may be partitioned into consecutive substrings, $w_i = w_{i1} \cdots w_{il_i}$, such that each w_{ij} matches R. (Here, l_i is just the number of such substrings into which w_i is partitioned.) If, on the other hand, w_i matches S, then put $w_{11} = w$ and $l_i = 1$. Then, in any case, we see that we can partition w into strings w_{ij},

$$w = w_{11} \cdots w_{1l_1} w_{21} \cdots w_{2l_2} \ \cdots\cdots \ w_{k1} \cdots w_{kl_k},$$

such that each w_{ij} matches $R \cup S$. This establishes that w matches $(R \cup S)^*$.

So any string that matches $(R^* \cup S)^*$ must also match $(R \cup S)^*$.

We have proved the implication in both directions. So, a string matches $(R^* \cup S)^*$ if and only if it matches $(R \cup S)^*$. Hence $(R^* \cup S)^*$ may be simplified to $(R \cup S)^*$. Q.E.D.

If the last operation applied in the expression is a Kleene $*$, then it has the form $(R_1 \cup \cdots \cup R_k)^*$. By the above observation, we may assume that none of the R_i has the Kleene $*$ as its last operation. We may also assume that none of them has grouping or $\cup$ as its last operation, since then we could just have listed the alternatives in the sequence $R_1, \ldots, R_k$. So the R_i must just be single letters or the empty string. Let L be the set of single letters that appear in the R_i. Then the expression is matched by any string which consists solely of letters from L, and by the empty string.

If the last operation applied in the expression is alternatives, say we have $R_1 \cup \cdots \cup R_k$, then we may suppose that each R_i is either the empty string or a single letter, or its last operation is the $*$. If the latter, then we are back in the situation of the previous paragraph.

In conclusion, the regular expressions that don't need concatenation in their construction are those formed from alternatives which are each either single letters or arbitrary repetitions of letters from some subset of the alphabet. (These subsets may be different for the different alternatives.)

14.

Preamble (which you can skip: to do so, go to 'Solutions' below):

The solutions given here use a particular file `/usr/share/dict/words` with 234,936 words, each on its own line. The file you use will quite likely be from a different source so the numbers may be somewhat different.

I decided to eliminate proper nouns first, for convenience, and put the result in `wordsFile`:

```
$ egrep '^[a-z]' /usr/share/dict/words > wordsFile
```

Here, the line starts with a prompt, which we have represented as `$` . The text in blue is entered by the user. The regular expression is between the two forward quotes (apostrophes). This regular expression matches text at the beginning of the line consisting of any lower-case letter of the alphabet. The `egrep` command picks any *line* containing a match for the regular expression and outputs all these lines. The " `> wordsFile`" causes all these lines to be put into the file `wordsFile`.

This file now has 210,679 words, one per line:

```
$  wc wordsFile
210679  210679 2249128 wordsFile
```

In my `/usr/share/dict/words`, there are no words with nonalphabetic characters, and the only upper-case characters occurred at the starts of words. So all the words in `wordsFile` now consist only of lower-case letters.

You may find that things are a bit different on your system. The words file may have words with apostrophes, like "don't", or hyphens. You can filter these out using `egrep` too, if you wish.

Solution:

Assume that your list of words is in a file called `wordsFile`, with each line consisting of one word (and nothing else).

Regular expression to match any vowel: [aeiou]

Regular expression to match any consonant: [^aeiou] , or [b-df-hj-np-tv-z]

Regular expression to match any word with no vowel: ^[^aeiou]+$

This uses the fact that, in `wordsFile`, each word goes from the start of the line (matched by ^) to the end of the line (matched by $). (In the more typical situation where words in a file are delimited by spaces, you could use [^aeiou]+ with a space before and after it.)

Words with no vowel:

```
$  egrep '^[^aeiou]+$' wordsFile
```

gives 132 such words.

Words with no vowel or 'y':

```
$  egrep '^[^aeiouy]+$' wordsFile
```

gives 30 such words, but some of these are single letters which are always counted as words, in their own right, in this list. We can exclude such, to find that there are ten such words in our file:

```
$  egrep '^[^aeiouy][^aeiouy]+$' wordsFile
cwm
grr
nth
pst
sh
st
tch
tck
th
tst
```

Words with no consonants:

```
$ egrep '^[aeiou]+$' wordsFile
a
aa
ae
ai
e
ea
eu
euouae
i
iao
ie
io
o
oe
oii
u
```

So there are 16 such words in this `wordsFile`.

Consonant-vowel alternations:

```
$ egrep '^[aeiou]?([^aeiou][aeiou])*[^aeiou]?$' wordsFile > consVowelAlt
$ wc consVowelAlt
13219   13219  103647 consVowelAlt
```

So there are 13,219 words in this file with an alternating consonant-vowel pattern. The fraction of words with this property, using this `wordsFile`, is 13219 / 210679 $\simeq$ 0.063, or 6.3%.

Longest run of consonants — and let's make it more interesting by treating 'y' as a vowel (though the solution is easily modified to treat it as a consonant):

```
$ egrep '[^aeiouy][^aeiouy][^aeiouy][^aeiouy][^aeiouy][^aeiouy]' wordsFile
archchronicler
bergschrund
fruchtschiefer
latchstring
lengthsman
postphthisic
veldtschoen
```

Adding an extra [^aeiouy] to the regular expression gives no solutions, so these are the words with the longest run of consonants with 'y' treated as a vowel (six).

Longest run of vowels (now treating 'y' as a consonant, again to focus on the extreme cases):

```
$ egrep '[aeiou][aeiou][aeiou][aeiou][aeiou][aeiou]' wordsFile
euouae
```

Again, the maximum is 6.

Further queries give a runner-up, `cadiueio`, and a 112-way tie for third place.

List of palindromes:

If your regular expression tool allows the use of *back-references*, then this should be possible. For example:

```
$ egrep '^([a-z]?)([a-z]?)([a-z]?)([a-z]?)([a-z]?)([a-z]?)([a-z]?)([a-z]?)([a-z]?)([a-z]?)\9\8\7\6\5\4\3\2\1$'
wordsFile > palindromes
```

(all on a single line) gives a file, `palindromes`, of 135 palindromes. The longest have seven letters (e.g., `rotator`).

The expression used with `egrep` here was chosen to be long enough to cover any word that might be a palindrome in English, and so it turned out to be, for this word list. But the $\backslash n$ construct is usually only applicable when n is a nonzero *digit*, so $n \leq 9$. This means that this expression can be used to detect palindromes of up to $2 \times 9 + 1 = 19$ letters. If a word list had palindromes that were longer than this, then this approach would not detect them.

As we show in Lecture 11, the set of palindromes is *not* regular. Back-references are forbidden in regular expressions; tools that use them go beyond the class of regular expressions.

15. (a) $\ell_{B,1} = 0$, $\ell_{W,1} = 0$, $\ell_{U,1} = 1$, $a_{B,1} = 1$, $a_{W,1} = 1$.
(b)

$$\begin{aligned}
\ell_{B,n+1} &= \ell_{B,n} + \ell_{U,n} \\
\ell_{W,n+1} &= \ell_{W,n} + \ell_{U,n} \\
\ell_{U,n+1} &= \ell_{B,n} + \ell_{W,n} + \ell_{U,n} + a_{B,n} + a_{W,n} \\
a_{B,n+1} &= \ell_{W,n} + a_{B,n} \\
a_{W,n+1} &= \ell_{B,n} + a_{W,n}
\end{aligned}$$

(c) Observe that, by symmetry, $\ell_{B,n} = \ell_{W,n}$ and $a_{B,n} = a_{W,n}$. So, if you want to work out the total number of legal positions on a path graph of some size, then it's enough to work out three quantities for each n from 1 up to the desired size: $\ell_{B,n}$ (equivalently, $\ell_{W,n}$); $\ell_{U,n}$; and $a_{B,n}$ (equivalently, $a_{W,n}$).
(d) Once you reach the desired value of n, the total number of legal positions is just $\ell_{B,n}+\ell_{W,n}+\ell_{U,n}$, which equals $2\ell_{B,n} + \ell_{U,n}$.

Note that $a_{B,n}$ and $a_{W,n}$ aren't part of this total. But you still need to use them, for smaller values of n, in the calculations leading up to the total.

16. The answers for (a) and (b) are not unique.

(a) $((B^* \cup W^*)UU^*(B^* \cup W^*))^*$

(b) $((B^* \cup W^*)UU^*(B^* \cup W^*))^*(B^* \cup W^*)UU^*((BB^*WW^*) \cup (WW^*BB^*))$

(c) Yes. We saw in (c) that there is a Finite Automaton to recognise this language, so by Kleene's Theorem there must be a regular expression for it as well.

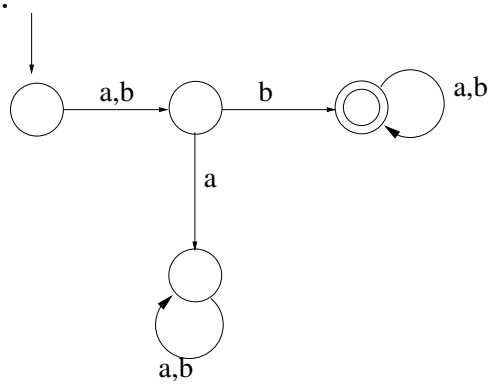
17.

1. All integers are arithmetic expressions.
2. If A and B are arithmetic expressions then so are: (A), A + B, A - B, A/B, and A*B.

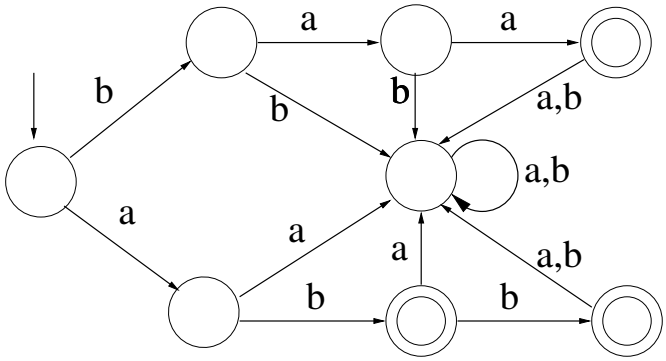
18.

1. The strings ϵ, **a**, and **b** are words in **PALINDROME**.
2. If S is a word in **PALINDROME** then so are: **a**S**a** and **b**S**b**.

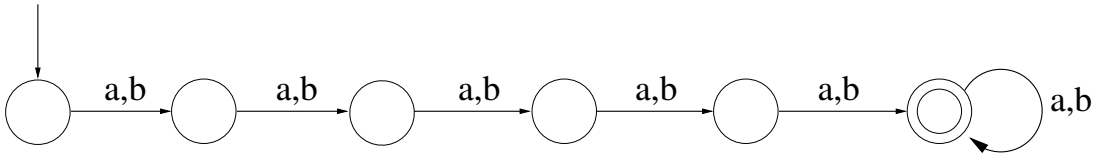
19.



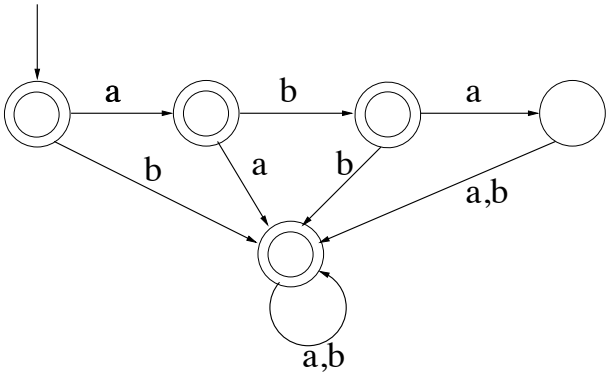
20.



21.

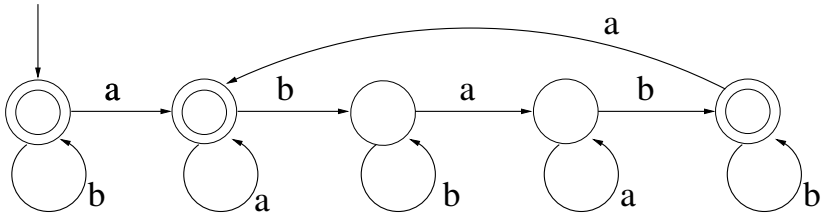


22. [2]



23.

[2]Thanks to FIT2014 tutor Han Duy Phan for advising a correction to this solution.



24.

This uses the observation that a decimal number is divisible by 3 if and only if the sum of its digits is divisible by 3.[3] Furthermore, we may take each digit mod 3 when we do this.

If we ignored the ban on leading 0s, the following FA would do the job. In the table, State 0 serves as both Start State and the sole Final State.

state	transitions		
	0,3,6,9	1,4,7	2,5,8
0	0	1	2
1	1	2	0
2	2	0	1

How do we deal with leading 0s? Since they are forbidden, every string starting with a 0 should be rejected, regardless of what the subsequent letters are. Apart from this, 0 is treated like any other multiple of 3 (as in the above table). So we modify the above table to obtain the FA in the following table. This time, the Start State is denoted by $-$. The sole Final State is still the one labelled 0. The new state labelled 'x' is one from which you can never escape. It represents the unrecoverable error that occurs if a string starts with a 0.

state	transitions			
	0	3,6,9	1,4,7	2,5,8
$-$	x	0	1	2
x	x	x	x	x
0	0	0	1	2
1	1	1	2	0
2	2	2	0	1

25.

FIRST ATTEMPT:

Base case ($n = 1$):
The only input string of length 1 in which `a` appears an odd number of times is "`a`", and this leads immediately to the Final State, so is accepted.

Inductive step:

Suppose $n \geq 2$. Assume that any string of length $n-1$ with an odd number of `a`s is accepted (the Inductive Hypothesis).

Let w be any input string of length n in which `a` occurs an odd number of times. Let v be the string of length $n-1$ obtained from w by removing the last letter.

We're aiming to apply the Inductive Hypothesis to v ... BUT v does not necessarily have an odd number of `a`s. What can we do?

Hmm ...

Let's *start again*, with a *stronger* Inductive Hypothesis.

[3] This trick does not work for divisibility testing in general. But it does work for 9 as well as for 3.

SECOND ATTEMPT:[4]

We prove by induction on n that:

> the FA accepts every string with an *odd* number of a**s**, AND it rejects every string with an *even* number of a**s**.

Base case ($n = 0$, as our argument now needs to cover the empty string): When the input string is empty, the computation by the FA finishes immediately at the Start State, which is *not* a Final State, so the empty string is rejected. So the claim holds in this case.

Inductive step:

Suppose $n \geq 1$. Assume that any string of length $n-1$ with an odd number of a**s** is accepted, AND that any string of length $n-1$ with an even number of a**s** is rejected (the Inductive Hypothesis).

Let w be an input string of length n.

Let v be the string of length $n-1$ obtained from w by removing the last letter.

We consider two cases, according to whether v has an *odd* or *even* number of a**s**.

If v has an odd number of a**s**, then it leads to a Final state, since it would have been accepted if it had been the entire input string (by the Inductive Hypothesis). Therefore, v leads to state 2.

- If the next letter is `a`, then w has an *even* number of a**s**. But reading that last letter `a` would take the FA from state 2 to state 1, which is not a Final state, so the string w is rejected.
- If the next letter is `b`, then w has an *odd* number of a**s**. But reading that last letter `b` would keep the FA in state 2, which is a Final state, so the string w is accepted.

If v has an even number of a**s**, then it leads to a Non-Final state, since it would have been rejected if it had been the entire input string (by the Inductive Hypothesis). Therefore, v leads to state 1.

- If the next letter is `a`, then w has an *odd* number of a**s**. But reading that last letter `a` would take the FA from state 1 to state 2, which is a Final state, so the string w is accepted.
- If the next letter is `b`, then w has an *even* number of a**s**. But reading that last letter `b` would keep the FA in state 1, which is a Non-Final state, so the string w is rejected.

So the claim holds, regardless of the parity of the number of a**s** in v.

The result follows, by Mathematical Induction.

The statement we have proved is *stronger* than the one in the question. That's ok, as it *implies* the statement in the question.

26.

(a) states 2 and 4.

(b)

Base case ($n = 1$): We saw in (a) that `abba` can end up in State 4, which is the Final state. Therefore `abba` is accepted.

Inductive step: suppose $n \geq 2$, and that $(\texttt{abba})^{n-1}$ is accepted.

Since $(\texttt{abba})^{n-1}$ is accepted (by Inductive Hypothesis), there is some path it can take through the NFA that ends at the Final State. We also see that, if we are in the Final State, and we then read `abba`, then we can reach the Final State again, by following the path $4 \to 3 \to 2 \to 1 \to 4$. We can put these two paths together, to give a path for the string $(\texttt{abba})^{n-1}\texttt{abba}$ that goes from the Initial State to the Final State. Therefore this string is accepted by our NFA. But this string is just $(\texttt{abba})^n$. So $(\texttt{abba})^n$ is accepted.

[4]Thanks to FIT2014 tutors Raphael Morris, Rebecca Robinson and Nathan Companez for feedback on earlier versions of this proof.

Therefore, by the Principle of Mathematical Induction, the string $(\texttt{abba})^n$ is accepted for all $n \geq 1$.