Monash University
Faculty of Information Technology
2^{nd} Semester 2022

...ory of Computation
...s for Tutorial 3
Regular Langu... Definitions, Finite Automata, Kleene's ...Theorem

Although you may ... any exercises in this Tutorial 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.

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 $\mathrm{a}^* \cup \mathrm{b}$. 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 [illegible] since the empty string matches $(R \cup S)^*$ using zero repetitions [illegible]ssume $w \neq \varepsilon$. Then w may be partitioned into consecutive subs[illegible]uch that each w_i matches $R^* \cup S$. If w_i matches R^*, then it may [illegible]nsecutive substrings, $w_i = w_{i1} \cdots w_{il_i}$, such that each w_{ij} match[illegible] the number of such substrings into which w_i is partitioned.) If, [illegible] matches S, then put $w_{11} = w$ and $l_i = 1$. Then, in any case, we [illegible]n w into strings w_{ij},

$$[\text{illegible}] \cdots w_{2l_2} \quad \cdots\cdots \quad w_{k1} \cdots w_{kl_k},$$

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 $(R \cup S)^*$.

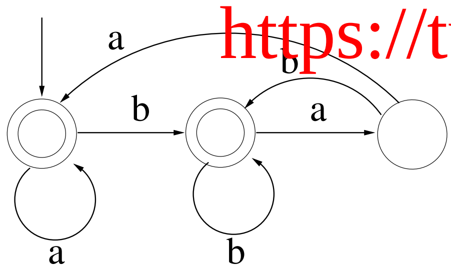
We have proved the implication in both directions. So, a string matches $(R^* \cup S)^*$ if and only if it matches $(R \cup S)^*$. Hence $(R^* \cup S)^*$ may be simplified to $(R \cup S)^*$. Q.E.D.

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.)

5.



6. [1]

Input: a maze.

1. Construct an automaton A from the maze as follows.

   (a) For each cell in the grid, create a state.

   (b) For every two adjacent cells *that have no wall between them*,
   add transitions in both directions between their corresponding states, with each transition labelled by the direction (U or D or L or R) you have to go to move between the cells in that direction.

   (c) Label the state corresponding to the start cell as the Start State.

   (d) Label the state corresponding to the destination cell as the sole Final State.

[1]Thanks to FIT2014 tutor Nathan Companez for this question.

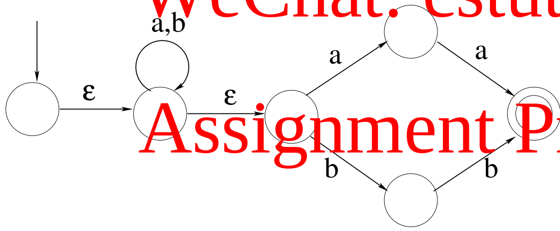
Observe that this automaton is almost an FA. It has no ambiguity (i.e., no *real* nondeterminism), in the sense that there is no state which has two identically-labelled outgoing transitions. But, because of the walls in the maze, in general some states will have no outgoing transition for some label. W[illegible]ar-FA" to an actual FA using the standard NFA-to-FA conversion algor[illegible] Alternatively, we can create one new *sink state* with four loops (one [illegible]d, for every state and every "missing" outgoing transition from that [illegible] transition going out from that state, labelled by the "missing" label, [illegible] state. This ensures that any string of directions that would take you [illegible]ed.

2. Now use the FA-[illegible]en in lectures to convert this FA to a regular expression for the same lan[illegible]

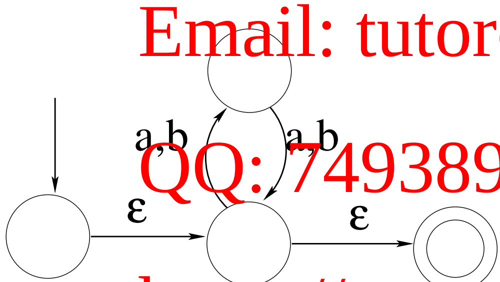
Output: the regular expression.

7.

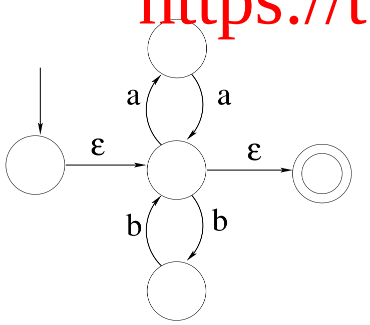
i)



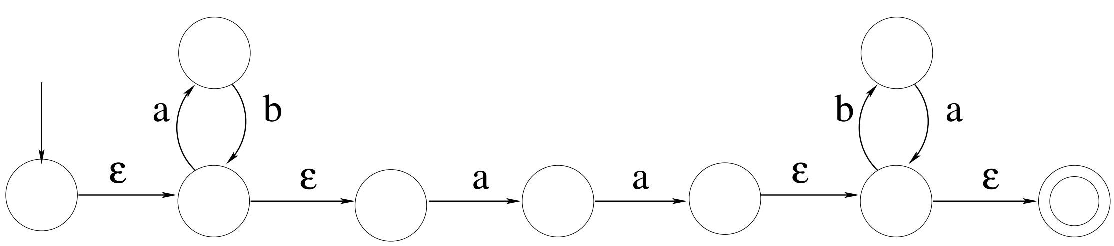
ii)

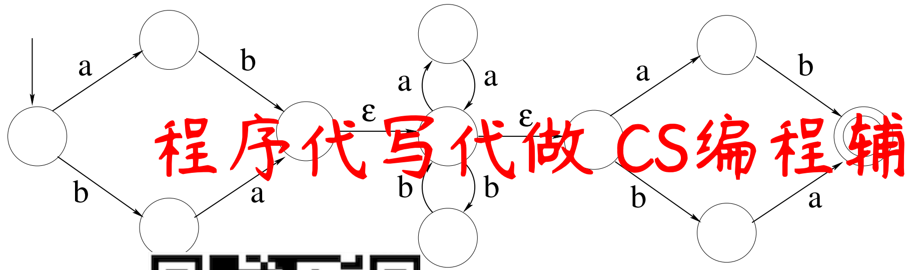


iii)

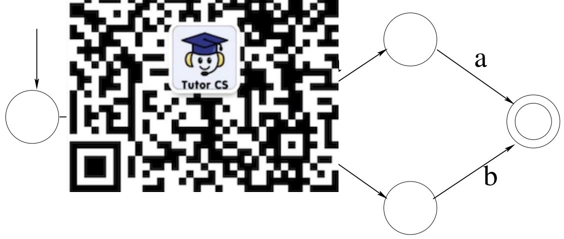


iv)



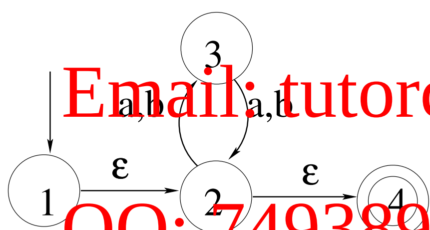
v) 

8.

i) NFA 

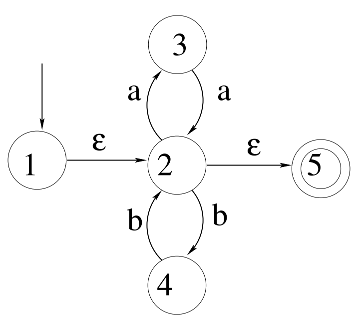
FA

	a	b
Start {1,2,3}	{2,3,4}	{2,3,5}
{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 

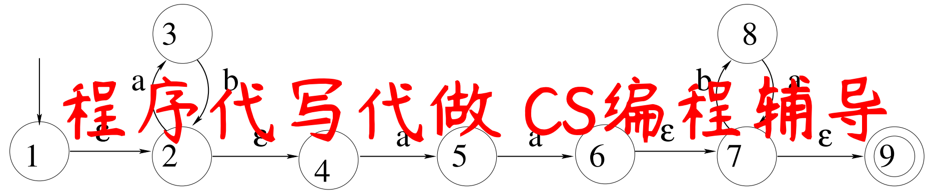
[2]

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

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

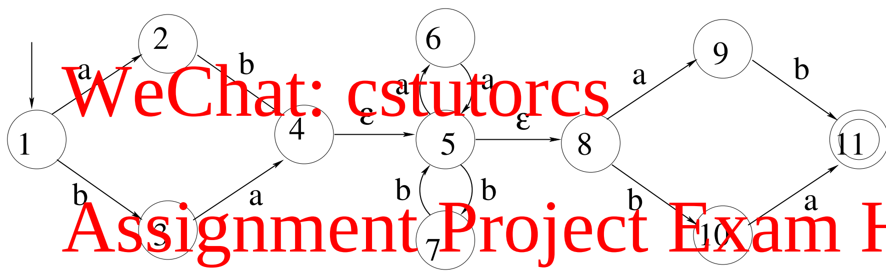
iv) NFA



FA

	a	b
	{3,5}	ϕ
	{6,7,9}	{2,4}
	ϕ	{8}
	{3,5}	ϕ
	{7,9}	ϕ
	ϕ	{8}
	ϕ	ϕ

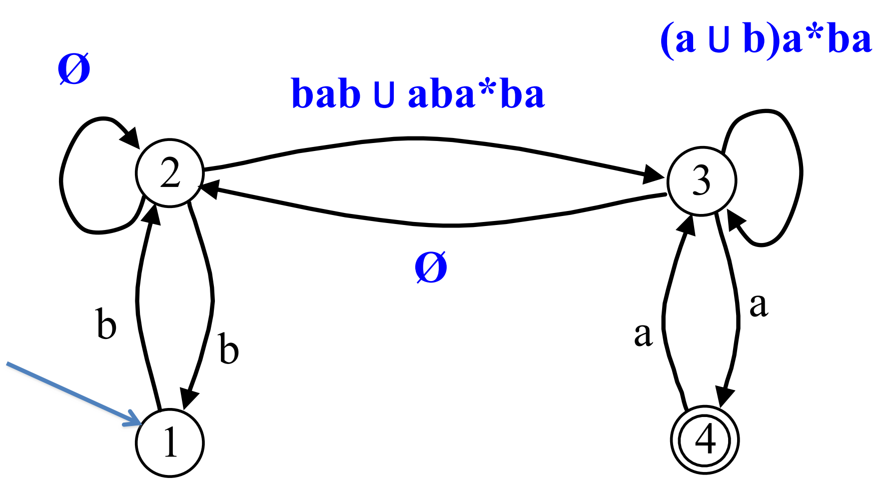
v) NFA



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

9.



10. First, we just distinguish between two *types* of state: Final states, and Non-Final states. Let's indicate the Final states (4 and 5) by bold text (and colour them blue), and the Non-Final states (1,2,3) by italic text (and colour them green). Let's first do it in the left-hand column (where all the states are listed):

	state	a	b
Start	*1*	2	3
	2	1	5
	3	1	4
Final	**4**	2	5
Final	**5**	3	5

…and then apply that same colour scheme throughout the table:

	state	a	b
Start	*1*	*2*	*3*
	2	*1*	**5**
	3	*1*	**4**
Final	**4**	*2*	**5**
Final	**5**	*3*	**5**

The underlying principles are:

- states of a *different* type (i.e., different colour) *cannot* be equivalent.
  - Initially, this is clear, because if a string finishes in a Final state, it is accepted, whereas a string finishing in a Non-Final state is rejected. So the behaviours of these two state types are fundamentally different, as far as membership of the language is concerned.
- States of the *same* type *may or may not* be equivalent — we don't yet know whether they are equivalent or not.

That concludes the initial iteration.

Now we do the next iteration. We must identify any *state type* (i.e., colour) whose rows have *different patterns* (meaning different patterns of *state type*, i.e., of *colour*).

Consider first the **bold**/blue state type, states 4 & 5. This type corresponds to two rows in the table, and these rows have the same pattern. So, no different patterns there. (Although these two rows have different sequences of *states*, they have the same sequence of state *types*, as indicated by the identical sequences of colours along the two rows.)

Now consider the *italic*/green state type, states 1,2 & 3. We have three rows, and we see they are *not* all of the same type: the first row has a different pattern to the second and third rows; two patterns, instead of one.

This difference in pattern tells us that we need to subdivide this state type into two types. We'll use a new text style, underlining (also a new colour, red), for a new state type consisting just of state 1. The state type consisting of states 2 & 3 will still be denoted by italic text (and colour green). So we need to change the way we indicate state 1 *throughout the table*:

	state	a	b
Start	$\underline{1}$	*2*	*3*
	2	$\underline{1}$	**5**
	3	$\underline{1}$	**4**
Final	**4**	*2*	**5**
Final	**5**	*3*	**5**

That concludes the second iteration.

Now we come to the third iteration. Once again, we look at each state type, and study its rows to see if they all have the same pattern. This time, we find that, within each state type, all the rows *do* have the same pattern. You can check that the rows for states 2 & 3 have the same colour pattern, and the rows for states 4 & 5 have the same colour pattern. (State 1 is in a state type by itself, which can't be split any further, so doesn't need to be checked.)

So there are no ne[illegible] und in this table. When this happens, our process of iteratively labelling ([illegible] stops, and for each state type, we merge all states of that type into a singl[illegible] e keep state 1 as is, and merge states 2 & 3 together into state 2, and merg[illegible] e 4.

[illegible]e	a	b
[illegible]	2	2
[illegible]	1	4
[illegible]	2	4

The algorithm now stops, and it is guaranteed by the theory of this algorithm that we have found an FA that is equivalent to the original FA and has the minimum possible number of states. So, from our original five-state FA, we have constructed an equivalent three-state FA, and this is the best possible.

Supplementary exercises

11.

(i) Inductive basis: $H_1 = 1 = \log_e e > \log_e 2$, using the fact that $e > 2$.

(ii) Our inductive hypothesis is that $H_n \geq \log_e(n+1)$ is true for n. We need to use this to show that n can be replaced by $n+1$ in this inequality, i.e., $H_{n+1} \geq \log_e((n+1)+1)$.

$$H_{n+1} = 1 + \frac{1}{2} + \cdots + \frac{1}{n+1}$$ First, we need to relate this to H_n.

$$= \left(1 + \frac{1}{2} + \cdots + \frac{1}{n}\right) + \frac{1}{n+1}$$

$= H$ [illegible] 1 [illegible] We've now expressed H_{n+1} in terms of H_n. So we can apply the Inductive Hypothesis.

$\geq$ l [illegible] by the Inductive Hypothesis

$\geq$ l [illegible] $\frac{1}{n+1}\Big)$ using $\log_e(1+x) \leq x$, with $x = \frac{1}{n+1}$

$=$ l [illegible] $\frac{\ 2}{\ 1}\Big)$

$$= \log_e\left((n+1)\cdot\frac{n+2}{n+1}\right)$$

$$= \log_e(n+2)$$

$$= \log_e((n+1)+1).$$

So, we've shown that, if the claimed inequality holds for n, then it holds for $n+1$.

(iii) By the Principle of Mathematical Induction, the claimed inequality must hold for all n.

Further properties of the harmonic numbers, and their applications in computer science, may be found in:

- Donald E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms. Third Edition.* Addison-Wesley, Reading, Ma., USA, 1997. See Section 1.2.7, pp. 75–79.

Comments:

The n-th harmonic number H_n is close to $\log_e n$, in fact they differ by < 1. The amount by which they differ is denoted by γ and is known as *Euler's constant*. Its value is $0.57721566\ldots$. It is not yet known whether or not this number is rational.

To see why H_n should be so closely related to $\log_e n$, we compare the *sum* H_n to the *integral* $\int_1^{n+1} \frac{1}{x}\,dx$. This is the area under the curve $y = \frac{1}{x}$ between the vertical lines $x = 1$ and $x = n+1$. Roughly speaking,

$$\sum_{i=1}^{n} \frac{1}{i} \;\approx\; \int_1^{n+1} \frac{1}{x}\,dx.$$

Now, the derivative of $1/x$ is $\log_e(x)$, so the integral is

$$\int_1^{n+1} \frac{1}{x}\,dx = \log_e(n+1) - \log_e 1 = \log_e(n+1).$$

So

$$\sum_{i=1}^{n} \frac{1}{i} \;\approx\; \log_e(n+1).$$

12.

Inductive basis:

When $n = 1$, the formula gives

$$F_1 = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)\right)$$

$$= \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}-1+\sqrt{5}}{2}\right)$$

$$= \frac{1}{\sqrt{5}} \cdot \frac{2\sqrt{5}}{2}$$
$$= 1.$$

When $n = 2$, the form [illegible]

$$\begin{aligned}
F_2 &= \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^2 - \left(\frac{1-\sqrt{5}}{2} \right)^2 \right) \\
&= \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} + 1 \right) - \left(\frac{1-\sqrt{5}}{2} + 1 \right) \right)
\end{aligned}$$

(using the fact that, for $x = (1 \pm \sqrt{5})/2$, we know $x^2 - x - 1 = 0$, i.e., $x^2 = x + 1$)

$$\begin{aligned}
&= \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2} + 1 - 1 \right) \\
&= \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}-1+\sqrt{5}}{2} \right) \\
&= \frac{1}{\sqrt{5}} \cdot \frac{2\sqrt{5}}{2} \\
&= 1.
\end{aligned}$$

Since we have dealt with the cases $n \le 2$, we may now assume $n \ge 3$.

Inductive step:

Suppose that, *for all* $m < n$, the formula holds for F_m. This is our inductive hypothesis. (It is a stronger type of inductive hypothesis than just assuming the formula holds for F_m when $m = n - 1$. But that's ok.)

For convenience, write

$$x = \frac{1+\sqrt{5}}{2}, \qquad y = \frac{1-\sqrt{5}}{2}.$$

Now consider F_n.

$$\begin{aligned}
F_n &= F_{n-1} + F_{n-2} \\
&\quad \text{(using the definition of } F_n \text{ for } n \ge 3) \\
&= \frac{1}{\sqrt{5}} \left(x^{n-1} - y^{n-1} \right) + \frac{1}{\sqrt{5}} \left(x^{n-2} - y^{n-2} \right) \\
&\quad \text{(by our inductive hypothesis, applied twice: once with } m = n-1 \text{, and once with } m = n-2) \\
&= \frac{1}{\sqrt{5}} \left(x^{n-1} - y^{n-1} + x^{n-2} - y^{n-2} \right) \\
&= \frac{1}{\sqrt{5}} \left(x^{n-1} + x^{n-2} - (y^{n-1} + y^{n-2}) \right) \\
&= \frac{1}{\sqrt{5}} \left(x^{n-2}(x+1) - y^{n-2}(y+1) \right) \\
&= \frac{1}{\sqrt{5}} \left(x^{n-2}x^2 - y^{n-2}y^2 \right) \\
&\quad \text{(using } x^2 = x+1 \text{ and } y^2 = y+1) \\
&= \frac{1}{\sqrt{5}} \left(x^n - y^n \right).
\end{aligned}$$

So the formula holds for F_n as well.

In summary of the inductive step: we showed that, if the formula for F_m holds when $m < n$, then it holds for $m = n$.

Conclusion: by the Principle of Mathematical Induction, it follows that the formula is true for all values of n.

This exercise illust[illegible] may be hard to discover the correct formula for some quantity (…surely, th[illegible]nes as a surprise when you first meet it, and you may wonder how anyone c[illegible] *you have the formula*, induction is a powerful tool for proving it. In research[illegible]ct" by an unpredictable process of creative exploration, and then *prove* it by [illegible] often, by induction.

This exercise has a[illegible]ection between the famous Fibonacci numbers and the equally famous "Golden Ratio", $\varphi = (1+\sqrt{5})/2$.

Extra exercise to give more insight into this connection, for the curious or mathematically inclined: use the above result to prove that the ratio between two successive Fibonacci numbers tends to φ as $n \to \infty$.

Among the many applications of Fibonacci numbers in computer science is the fact that the Euclidean algorithm, for computing GCD, takes longest when the two input numbers are successive Fibonacci numbers.

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

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

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

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

17. $(\text{-}\mid +)?(N\mid N.\mid N.N\mid .N)((e\mid E)(\text{-}\mid +)?N)?$, where $N = [0-9]^+$

18.

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 [illegible]nt: [^aeiou] , or [b-df-hj-np-tv-z]

Regular expression [illegible]th no vowel: ^[^aeiou]+$

This uses the fact [illegible]ch word goes from the start of the line (matched by ^) to the end of the li[illegible] the more typical situation where words in a file are delimited by spaces, y[illegible] with a space before and after it.)

Words with no vow[illegible]

```
$ egrep '^[^aei[illegible]
```

gives 132 such wor[illegible]

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 [illegible] expression gives no solutions, so these are the words with the longest run [illegible] treated as a vowel (six).

Longest run of vow[illegible] 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 \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.

19. (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.

20. 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 b[illegible] for it as well.

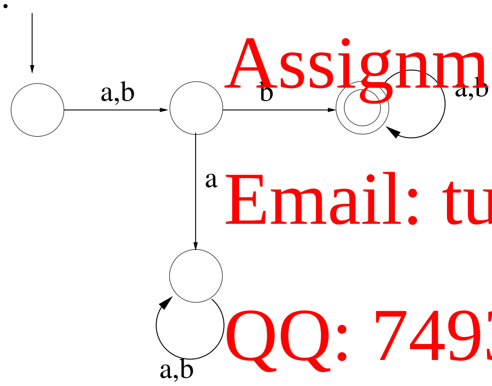
21.

1. All integers are [illegible]
2. If A and B are a[illegible]hen so are: (A), A + B, A - B, A/B, and A*B.

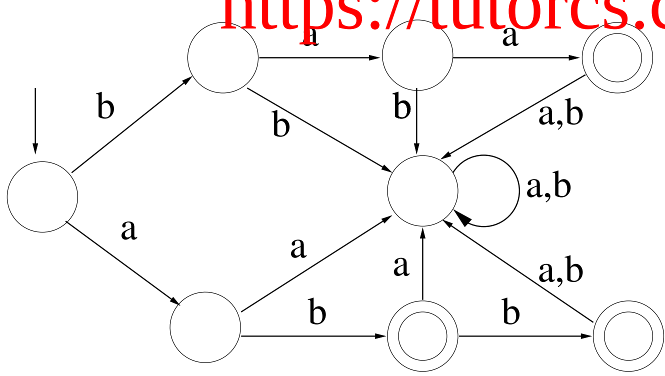
22.

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**.

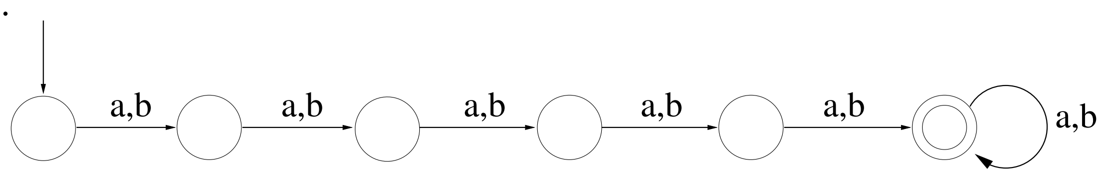
23.



24.



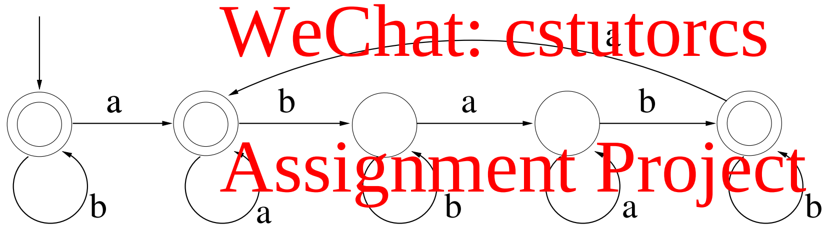
25.



26. [3]

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

27.



28.

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.[4] 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

29.

FIRST ATTEMPT:

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

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 $n \geq 2$. A[illegible]of length $n-1$ with an odd number of `as` is accepted (the Inductive Hypoth[illegible]

Let w be any inpu[illegible]which `a` occurs an odd number of times. Let v be the string of length $n-1$ [illegible]noving the last letter.

We're aiming to a[illegible]othesis to v ... BUT v does not necessarily have an odd number of `as`. W[illegible]

Hmm ...

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

SECOND ATTEMPT:[5]

We prove by induction on n that:

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

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 `as` is accepted, AND that any string of length $n-1$ with an even number of `as` 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 `as`.

If v has an odd number of `as`, 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 `as`. 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 `as`. 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 `as`, 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 `as`. 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 `as`. 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 `as` 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.

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

30.

(a) states 2 and 4.

(b)

Base case ($n = 1$) [illegible] abba can end up in State 4, which is the Final state. Therefore abba is acc[illegible]

Inductive step: su[illegible] (abba)$^{n-1}$ is accepted.
Since (abba)$^{n-1}$ is ac[illegible]ypothesis), there is some path it can take through the NFA that ends at the [illegible]e that, if we are in the Final State, and we then read abba, then we can re[illegible]in, by following the path $4 \to 3 \to 2 \to 1 \to 4$. We can put these two pa[illegible]path for the string (abba)$^{n-1}$abba that goes from the Initial State to the Fi[illegible] string is accepted by our NFA. But this string is just (abba)^n. So (abba)^n is accepted.

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

31.

	.	-	[0-9]
Start 1	2	3	4
2	6	6	5
3	2	6	4
Final 4	5	6	4
Final 5	6	6	5
6	6	6	6

32.

They all have the following minimum state finite automaton.

	a	b
Start/Final 1	1	1

33.

(i) No simplification possible.

(ii) Merge the two Final States (first and third rows).

(iii) No simplification possible.

(iv) Merge the two states {1,2,4} and {2,4}, and merge the two states {6,7,9} and {7,9}.

(v) Merge the two states {4,5,8} and {5,8}.

34.

We just do (iii) as the others involve long computations.

First, turn it into a GNFA by adding a new Final State 3 and an ε-transition from the Start State to it. The Start State is no longer a Final State.

Similarly, a new Start State is added, with a new empty string transition from it to the old Start State.

Then, applying the algorithm, we obtain (aa ∪ ab ∪ ba ∪ bb)*. This is equivalent to the original regular expression ((a ∪ b)(a ∪ b))*, and describes the language of all strings of even length.