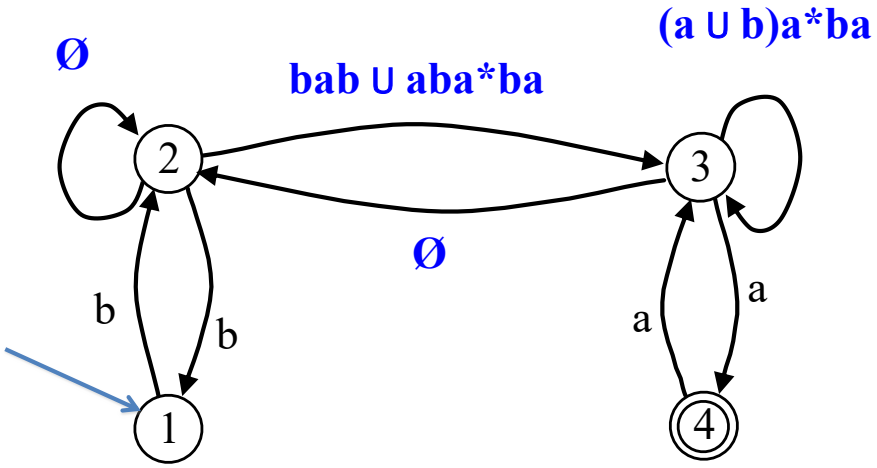
Monash University
Faculty of Information Technology
2^{nd} Semester 2023

FIT2014 Theory of Computation
Solutions for Exercises 4
Kleene's Theorem II and FA State Minimisation

1.



2. [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.

   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. We can convert this "near-FA" to an actual FA using the standard NFA-to-FA conversion algorithm given in lectures. Alternatively, we can create one new *sink state* with four loops (one for each of U,D,L.R) and, for every state and every "missing" outgoing transition from that state, we create a new transition going out from that state, labelled by the

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

"missing" label, and going into the sink state. This ensures that any string of directions that would take you through a wall is rejected.

2. Now use the FA-to-regexp algorithm given in lectures to convert this FA to a regular expression for the same language.

Output: the regular expression.

3. 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	1	*2*	*3*
	2	1	**5**
	3	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 new state types to be found in this table. When this happens, our process of iteratively labelling (or colouring) the states stops, and for each state type, we merge all states of that type into a single state. In this case, we keep state 1 as is, and merge states 2 & 3 together into state 2, and merge states 4 & 5 into state 4.

	state	a	b
Start	1	*2*	*2*
	2	1	**4**
Final	**4**	*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.

4.

The successive stages are:

	state	a	b
Start	1	2	4
	2	2	6
	3	3	4
Final	4	5	3
Final	5	5	5
Final	6	5	1

	state	a	b
Start	*1*	*2*	**4**
	2	*2*	**6**
	3	*3*	**4**
Final	**4**	**5**	*3*
Final	**5**	**5**	**5**
Final	**6**	**5**	*1*

	state	a	b
Start	*1*	*2*	**4**
	2	*2*	**6**
	3	*3*	**4**
Final	**4**	5	*3*
Final	5	5	5
Final	**6**	5	*1*

The colouring process then stops.

We merge states 1,2,3 into state 1, states 4 & 6 into state 4, and keep state 5 as-is.

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

5.

(i) No simplification possible.

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

(iii) Merge the two Final States (first and fourth rows).

(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}.

6.

We just do (ii) 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 $\cup$ ab $\cup$ ba $\cup$ bb)*. This is equivalent to the original regular expression ((a $\cup$ b)(a $\cup$ b))*, and describes the language of all strings of even length.

Supplementary exercises

7.

	.	-	[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

8.

They all have the following minimum state finite automaton.

	a	b
Start/Final 1	1	1