Compiler Construction Problem Set 1

Magnus Brevik

January 2024

1 Regular languages, NFAs and DFAs

1.1

$\mathcal{L}$(b*a((a|b|c)*cb*a)?(a|b)*)

1.2

Not including a graph as this should be sufficient to replicate one.
State 0 is the starting state.
State 5 is the accepting state.

State	**Letter**	**Result State**
0 :	a	1
	b	0
	ϵ	0
1 :	a	2
	b	2
	c	2
	ϵ	5
2 :	a	2
	b	2
	c	2
	ϵ	3
3 :	c	4
	ϵ	3
4 :	a	5
	b	4
	ϵ	4
5 :	a	5
	b	5
	ϵ	5

1.3

DFA State 0 is the start state.
Accepting DFA States are {0, 1, 2, 3, 4}
State 5 is a dead end.

DFA State	**NFA States**
0	$\{0, 1, 2, 5\}$
1	$\{1, 2, 5\}$
2	$\{2, 3, 4, 5\}$
3	$\{2, 4, 5\}$
4	$\{2, 5\}$
5	$\{\}$

Rules:

State	**Letter**	**Result State**
0 :	a	1
	b	2
1 :	a	4
	b	2
2 :	a	3
	b	2
3 :	a	4
	b	2
4 :	a	5
	b	2
5 :	$a\|b$	5

1.4

I forgot my crayons at home, so instead of colours I will use a letter with a subscript number for the intermediate states during minimisation.
Accepting States (a_0): {0, 1, 2, 3, 4}
Non-accepting States (a_1): {5}

Non-minimal State:	0	1	2	3	4	5
$a:$	a_0	a_0	a_0	a_0	a_1	a_1
$b:$	a_0	a_0	a_0	a_0	a_0	a_1

States {0, 1, 2, 3} are identical within group a_0, while 4 is different.
Next step uses the groups

$$\begin{aligned} b_0 &: \{0,1,2,3\} \\ b_1 &: \{4\} \\ b_2 &: \{5\} \end{aligned}$$

Non-minimal State:	0	1	2	3	4	5
$a:$	b_0	b_1	b_0	b_1	b_2	b_2
$b:$	b_0	b_0	b_0	b_0	b_0	b_1

States {0, 2} and {1, 3} are identical within group b_0.
Next step uses the groups

$$\begin{aligned} c_0 &: \{0,2\} \\ c_1 &: \{1,3\} \\ c_2 &: \{4\} \\ c_3 &: \{5\} \end{aligned}$$

Non-minimal State:	0	1	2	3	4	5
$a:$	c_1	c_2	c_1	c_2	c_3	c_3
$b:$	c_0	c_0	c_0	c_0	c_0	c_3

Now all groups consist of only identical NFA states, therefore we can rename them to just the number from the intermediate state names, and get

$$\begin{aligned} 0 &: \{0,2\} \\ 1 &: \{1,3\} \\ 2 &: \{4\} \\ 3 &: \{5\} \end{aligned}$$

with the rules

State	Letter	Result State
0 :	a	1
	b	0
1 :	a	2
	b	0
2 :	a	3
	b	0
3 :	$a\|b$	3

where state 0 is the starting state, states $\{0, 1, 2\}$ are accepting states, and state 5 is a dead end.

1.5

To be honest I cannot find a good way to negate DFA without just going through the same steps of making a regular expression and following the same steps. I might just be missing something in the book or online about it, but did not find anything interesting when searching for "negate deterministic finite automaton". Instead I made the regular expression (a|b)*aaa(a|b)* and made an automaton through NFA and NFA to DFA algorithms.
State 0 is the starting state, state 3 is the accepting state, and state 4 is a dead end.

State	Letter	Result State
0 :	a	1
	b	0
1 :	a	2
	b	4
2 :	a	3
	b	4
3 :	$a\|b$	3
4 :	$a\|b$	4

I found inverting the regular expression easier, but this was also a very simple expression to just create without the original, which is what I actually did. There is probably some algorithm to invert or negate a DFA which I did not find which makes it easier than negating a regular expression.

2 DFA for a small language

2.1

In modern regex one can usually use the identifier \d to denote a digit, i.e. [0-9].
(d(x|y)=-?\d+)|(go)\n

2.2

State 9 is the invalid state, so any invalid characters point there.

2.3

Code implementation