

Compilers (221)

Exercises - Lexical Analysis

Check your answers with the tutorial helpers during tutorials and with each other on Piazza.

PART 1 - Regular Expressions.

Tips. (1) When devising a regular expression also devise good test cases e.g. shortest strings it should match, strings it shouldn't match. (2) To aid thinking and for clarity use spaces to separate parts of a complicated expression, (3) A good approach for developing regular expressions that involve regular expression meta-characters is to re-cast the question changing the meta-character to standard characters that are easier to work with, developing a solution and then changing the meta-characters back, escaping them when necessary.

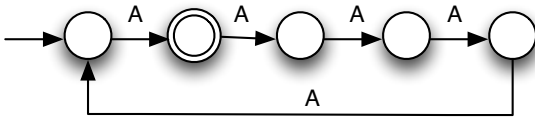
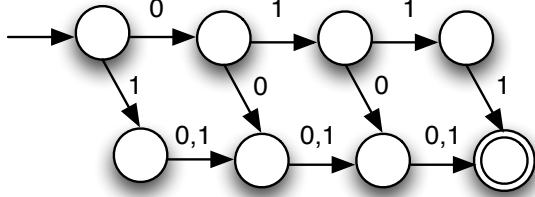
Suggested order to do the questions: 1, 2, 3, 12, 14, 15, 16, 20, 23, 25 and then the remaining questions. Let me know if find a mistake or have a better answer.

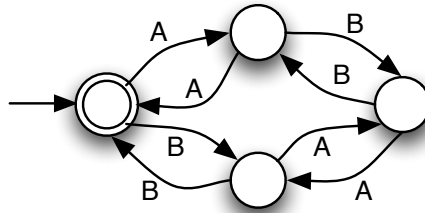
1.	Rewrite each of the following expressions using only the basic regular expression operators *, , concatenation, grouping () and ε: (i) [ABCD] (ii) M? (iii) M+	L1
	(i) A B C D (ii) M ε (iii) M M*	
2.	Give a regular expression for optionally signed floating points numbers . You can assume that the forms 3. (no fractional part) and .3 (not integer part) are not permitted. Allow for an optional exponent part also. Use the notation \X to escape Regex special characters like dot.	L2
	Here's one solution: <code>[+-]? [0-9]+ (\.[0-9]+)? ([Ee][+-]?[0-9]+)?</code>	
3.	For the alphabet {A,B,C} give a regular expression for strings that contain exactly one B .	L1
	<code>(A C)* B (A C)*</code>	
4.	For the alphabet {A,B,C} give a regular expression for strings that contain at most one B .	L1
	<code>(A C)* B? (A C)*</code> we could use <code>(B ε)</code> instead of <code>B?</code>	
5.	For the alphabet {A,B} give a regular expression for all strings containing exactly two B's .	L1
	<code>A* B A* B A*</code>	
6.	For the alphabet {A,B,C} give a regular expression for strings that do not contain two consecutive B's , i.e. between any two B's there must be at least one A or one C.	L2
	Here's one solution: <code>(A C BA BC)* B?</code>	
7.	For the alphabet {A,B} give a regular expression for all strings in which every A is immediately followed by at least two B's .	L1
	<code>(ABB B)*</code>	
8.	For the alphabet {A,B} give a regular expression for all strings containing two consecutive A's or two consecutive B's .	L1
	<code>[AB]* (AA BB) [AB]*</code>	
9.	For the alphabet {A,B} give a regular expression for all strings containing an even number of A's and an even number of B's .	L4
	<code>(AA BB (AB BA) (AA BB)* (AB BA))*</code>	
10.	Devise a regular expression for all binary strings of 0's and 1's with at most one pair of consecutive 1's .	L3
	Solution1: <code>(0 10)* 1? 1? (0 01)*</code> Solution2: <code>(1? 0+)* (1 11)? (0+ 1?)*</code>	

11.	Devise a regular expression for all strings of A's, B's and C's where no letter can appear more than once .	L4
	<p>Solution1: $A?B?C? \mid A?C?B? \mid B?A?C? \mid B?C?A? \mid C?A?B? \mid C?B?A?$</p> <p>Solution2: $A? (B? C? \mid C? B?)$ Factorised version $\mid B? (A? C? \mid C? A?)$ $\mid C? (A? B? \mid B? A?)$</p> <p>Could also enumerate all 16 alternates including empty string.</p>	
12.	Give a regular expression for Pascal-style comments of the form <code>{ XXX }</code> where XXX is any sequence of characters except <code>}</code> , i.e. where the comment is enclosed in braces. You can use the shortcut <code>[^A]</code> to denote any character except the character A, <code>[^AB]</code> for any character except the characters A and B, and so on.	L2
	$\{ [^\}]^* \}$	
13.	Give a regular expression for C-style comments of the form <code>/* comment */</code> Heed tip 3 above.	L5
	<p>Heeding tip 3 we'll build a regex for <code>AB XXXX BA</code> where A is <code>/</code>, B is <code>*</code> and X is any character other than A or B.</p> <p>I think the regex $AB (X \mid A \mid B+ X)^* B+A$ should work.</p> <p>We can change this back by using <code>/</code> for A, <code>'*'</code> for B and <code>[^/*]</code> for any character other than A or B.</p> <p>Giving the following solution $/ ' * ' (' * ' + [^/*] \mid [^/*] \mid /)^* ' * ' + /$</p> <p>I think the regex $AB (B+ X \mid Y)^* B^* BA$ where Y is any character other than <code>*</code></p>	
14.	Give a regular expression for the set of strings in the set $\{0^n 1^n : n \geq 1\}$. That is, the regular expression should accept n zeroes followed by n ones. Do not spend more than 5 minutes on this question!	L2
	Sorry this was a trick question. We can't count the number of 0's for arbitrary n using a regular expression. We need a more powerful grammar, e.g. a context free grammar.	

PART 2 - DFAs.

Tip. When reasoning about DFAs, it can be very useful to list out matching strings, starting with the shortest ones.

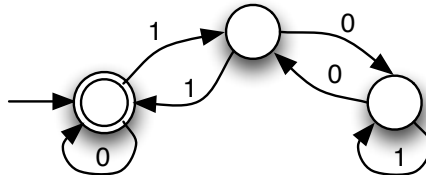
15.	Give a short (one sentence) English description for the following DFA.	L2
	 <p>All strings of A's of length $5n+1$ where $n \geq 0$</p>	
16.	Give a short English description for the following DFA (has a small gotcha).	L2
	 <p>All strings of 4-bit binary numbers except 0110 (decimal 6).</p>	
17.	Give a short English description for the following DFA.	L4



All strings of A's and B's with an even number of A's and an even number of B's, where even includes 0.

18. Give a short English description for the following DFA.

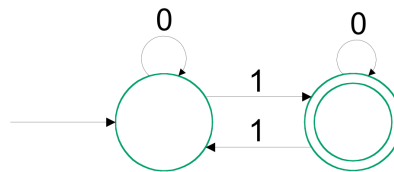
L5



All strings of binary numbers whose decimal value is $3n$ where $n \geq 0$ plus the empty string.

19. Give a regular expression for the following DFA and state what set of strings it accepts:

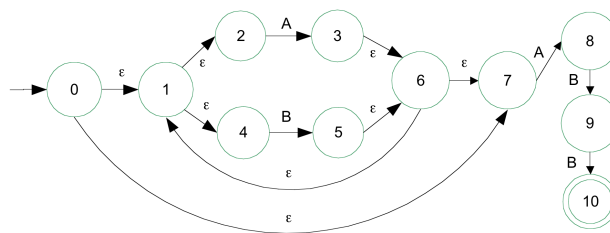
L4



$0^*10^*(10^*10^*)^*$ is one possibility. This DFA only accepts strings of 0's and 1's containing an odd number of 1's i.e. it checks for odd parity.

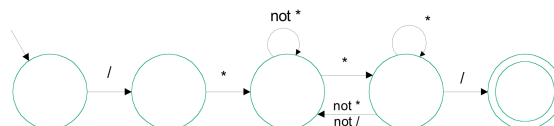
20. Derive the NFA for the regular expression $(A|B)^*ABB$. Number your states from 0. You should have 11 states (0 to 10).

L2



21. Give a DFA for C-style comments i.e. $/*$ comment with no appearances of $*/$ followed by $*/$. Use the label **not X** for any char except X, e.g. **not ***, **not /**

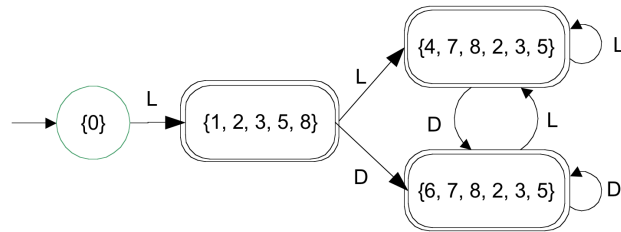
L4



22. Use the subset construction to derive the DFA for the regular expression $L(L|D)^*$ from its NFA (see slide with NFA for an identifier in lecture slides). You should have 4 states.

L3

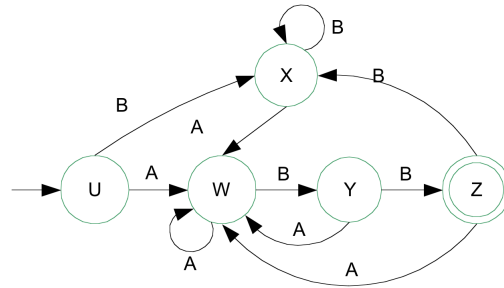
+++++



23. Using the subset construction derive the DFA for the regular expression $(A|B)^*ABB$ from its NFA (using your solution from question above). You should have 5 states.

L4

+++++

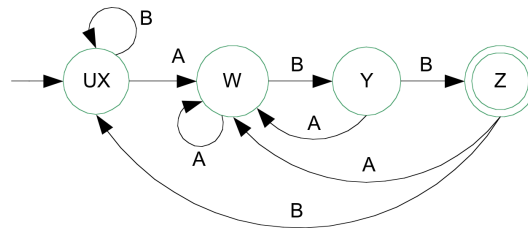


$U=\{0,1,2,4,7\}$, $W=\{3,6,7,1,2,4,8\}$, $X=\{5, 6, 7, 1, 2, 4\}$, $Y=\{1,2,4,5,6,7,9\}$, $Z=\{1,2,4,5,6,7,10\}$

24. Derive the minimal DFA for the regular expression $(A|B)^*ABB$ from your DFA above. You should have 4 states. (OPTIONAL MATERIAL)

L4

+++++



In the first pass we have $\text{Accept} = \{Z\}$, $\text{Non-Accept} = \{U, W, X, Y\}$. Since the Accept set has only one element we include it in the minimal DFA. For the Non-Accept set, U, W, X have transitions A and B within the Non-Accept Set. Y has a transition A within the non-Accept set and a transition B to the Accept set.

Since Y differs from the other states and is a single state we include it in the minimal DFA. Continuing with the set of remaining states $\{U, W, X\}$, U and X have transitions A and B within the partition $\{U, W, X\}$, while W has a transition A within the partition $\{U, W, X\}$ and a transition B within the partition $\{Y\}$.

Since W differs from U and X and is a single state we include it in the minimal DFA. Continuing with $\{U, X\}$ its transitions are identical. Since the transitions for U and X can't be split we include UX as a combined state in the minimal DFA.

25. For the regular expression:

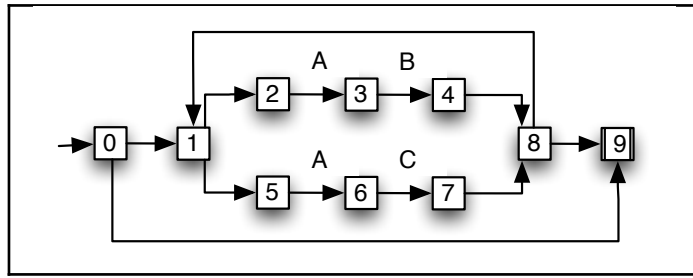
$(AB|AC)^*$

L3

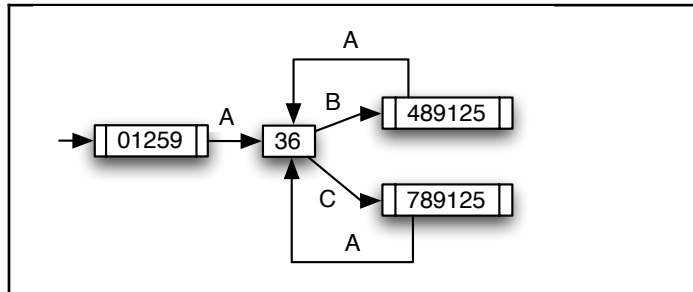
- Convert the regular expression to a NFA using Thompson's construction (10 states)
- Convert the NFA to a DFA using the Subset construction (4 states)
- Minimise the DFA (2 states) OPTIONAL

+++++

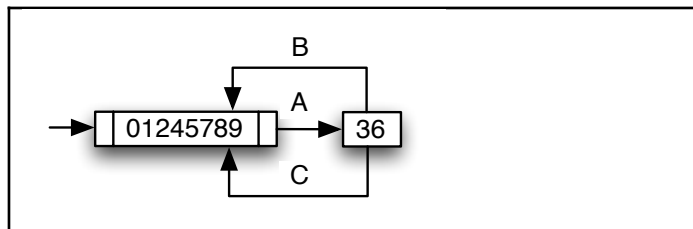
NFA Unlabelled transitions are epsilon transitions.



DFA

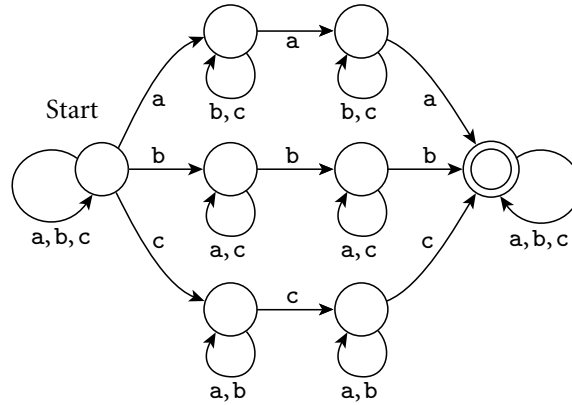


Minimal DFA



26. What strings does the following NFA recognise?

L5



DFA for all strings in $(a|b|c)^*$ for which one letter appears at least three times! If you have a better explanation let me know.