

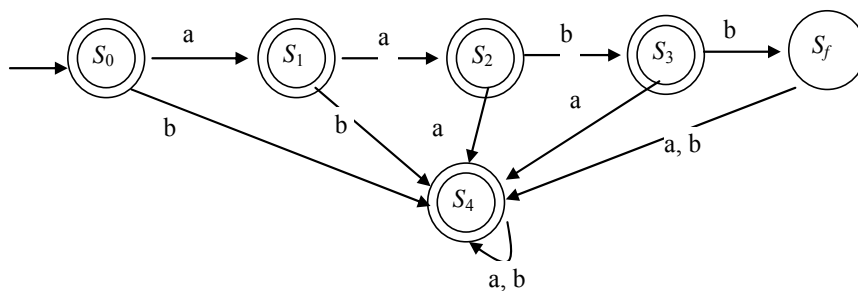
- Write regular expressions for the following informally described languages:
 - All strings of 0's and 1's with the subsequence 011.

Intuitive solution: $(0|1)^* 0 (0|1)^* 1 (0|1)^* 1 (0|1)^*$

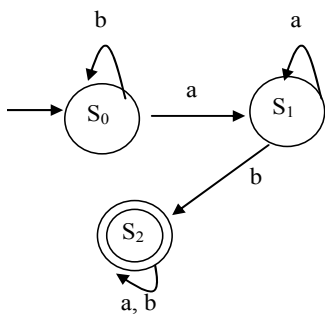
- All strings of 0's and 1's with the substring 00^*1 .

Intuitive solution: $(0|1)^* 00^* 1 (0|1)^*$

- Consider $\Sigma = \{a, b\}$. Answer the following DFA related questions. When constructing DFA, there is no need to show your construction steps, but you need to informally state how you get the DFAs.
 - Construct a DFA that accepts $(a|b)^*$ except for $aabb$.



- Construct a DFA that accepts $(a|b)^*$ except for b^*a^* .



- Based on the techniques you use in (a) and (b), can you come up with a DFA construction algorithm for the “except for” type of languages?

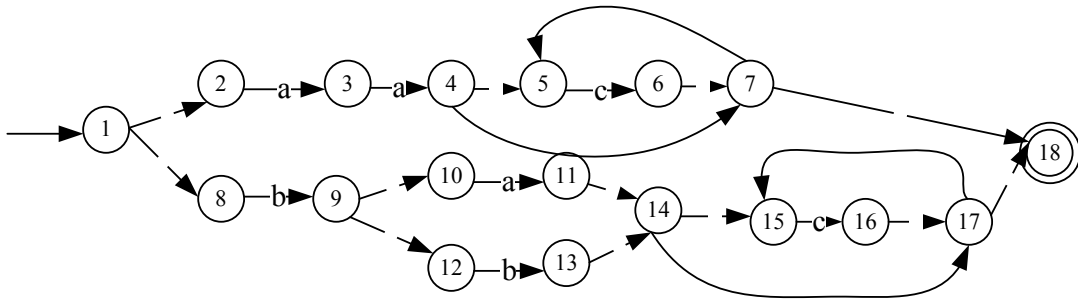
Step 1: (Optional) Construct NFA accepting the language corresponding to the “except for” part. For example, in (a), construct NFA accepting “aabb”.

Step 2: Convert NFA to DFA

Step 3: Reverse states, i.e., changing original “accept” states to “reject” states and original “reject” states to “accept” states.

- Consider the regular expression $aac^* | b(a|b)c^*$ defined on $\Sigma = \{a, b, c\}$.

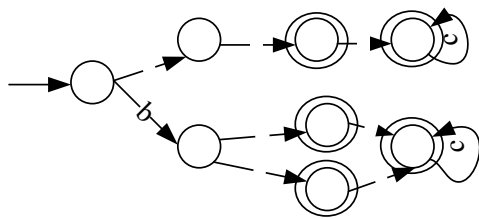
- Construct the NFA for the regular expression. You can directly draw the NFA without going through the RE-to-NFA steps.



b. Convert the NFA to DFA. You need to show the conversion steps.

| | a | b | c | |
|----|----|----|----|---|
| S0 | S1 | S2 | - | |
| S1 | S3 | - | - | |
| S2 | S4 | S5 | - | |
| S3 | - | - | S6 | F |
| S4 | - | - | S7 | F |
| S5 | - | - | S7 | F |
| S6 | - | - | S6 | F |
| S7 | - | - | S7 | F |

- S0 = {1, 2, 8}
- S1 = {3}
- S2 = {9, 10, 12}
- S3 = {4, 5, 7, 18}
- S4 = {11, 14, 15, 17, 18}
- S5 = {13, 14, 15, 17, 18}
- S6 = {5, 6, 7, 18}
- S7 = {15, 16, 17, 18}



c. Minimize the DFA. You need to show the minimization steps.

First, divide the states into two groups, final, non-final

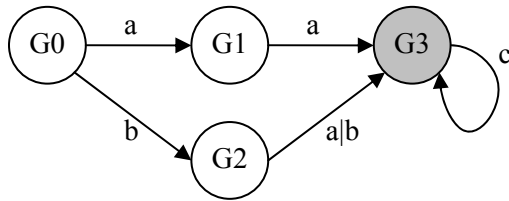
| | a | b | c | | |
|----|----|----|----|---|----|
| S0 | S1 | S2 | - | | G0 |
| S1 | S3 | - | - | | |
| S2 | S4 | S5 | - | | |
| S3 | - | - | S6 | F | G3 |
| S4 | - | - | S7 | F | |
| S5 | - | - | S7 | F | |
| S6 | - | - | S6 | F | |
| S7 | - | - | S7 | F | |

S0's transitions on a,b are all to G0
 S1's transitions on a is to G3, on b to Null
 S2's transition on a,b are all to G3
 ⇒ divide S0 | S1 | S2 (they are all different)

In G3, all transitions on c are back to G3
 for a,b are all Null
 ⇒ no division

| | a | b | c | | |
|----|----|----|----|---|----|
| S0 | S1 | S2 | - | | G0 |
| S1 | S3 | - | - | | G1 |
| S2 | S4 | S5 | - | | G2 |
| S3 | - | - | S6 | F | G3 |
| S4 | - | - | S7 | F | |
| S5 | - | - | S7 | F | |
| S6 | - | - | S6 | F | |
| S7 | - | - | S7 | F | |

No further division needed



4. Consider $\Sigma = \{a, b, c\}$.

a. Construct a minimized DFA for token recognition and the tokens are defined as follows.

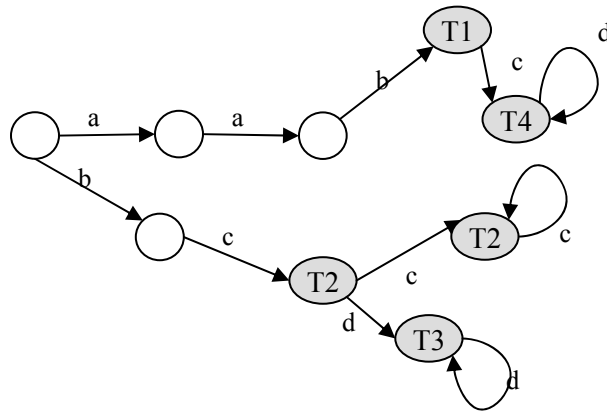
T1 = aab

T2 = bcc*

T3 = bcd*

T4 = aabcd*

The DFA should specify the specific token names (T1, T2, ...) it accepts at the corresponding final states. You do not need to show the steps for the construction if you can draw the DFA directly. Note that the longest matching and first matching rules for ambiguity resolution should be observed. Also, the backtracking mark should be given.



b. Execute your DFA to process the following string to identify tokens. List every token string and its token name. Also, describe all the backtracking actions taken during the process.

aabaabcbdddbcbccccaabbcbddd

| | |
|---------------------------------------|----------------------------|
| aab aabcbdddbcbccccaabbcbddd | -- got token aab(T1) |
| aab aabcbddd bcbccccaabbcbddd | -- got token aabcbddd (T4) |
| aab aabcbddd bc bccccaabbcbddd | -- got token bc(T2) |
| aab aabcbddd bc bcccc aabbcbddd | -- got token bcccc(T2) |
| aab aabcbddd bc bcccc aab bcbddd | -- got token aab(T1) |
| aab aabcbddd bc bcccc aab bcbddd | -- got token bcbddd (T3) |

c. Create the lex definition file for the token recognizer. You need to give proper lookahead strings. Use lex to generate the scanner program from the definition file. After compiling the scanner, generate input strings to test it. Attach the print out for the definition file, the input string, and the output of the scanner.

aab doesn't have any problems with bcc*, bcd*. Because of longest match, aab won't have any problem with aabcb* either.

Because of longest match, aabcb* won't have any issues with bcc* and bcd*.

With first match, bcc* will not have problems with bcd* with regard to bc.

5. Create a lex definition file for recognizing the following sets of tokens. After compiling the scanners, generate input strings to test them. Attach the print out for the definition files, the input strings, and the output of the scanner.

a. Consider $\Sigma = \{a, b, c, d\}$.

T1 = aab /ccc

T2 = aab /ddd

T3 = c*

T4 = d*

T5 = all other strings

You need to define the lookahead strings as specified.

lex/flex supports first match and longest match. It supports look-ahead also although the results of lookahead are often not what we would like to see.

b. Consider $\Sigma = \{a, b, c\}$.

T1 = abcc*

T2 = ca

T3 = cb*

Please also state in your answer what you have learnt from the results.

lex/flex doesn't deal with backtracking. It often returns results that are not what we have expected.

6. Consider the following grammar defined over $\Sigma = \{0, 1\}$.

$S \rightarrow 0S11$

$S \rightarrow S1$

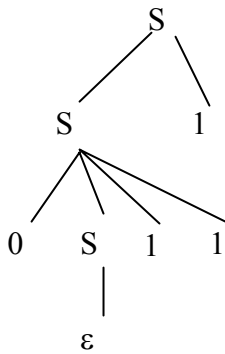
$S \rightarrow \epsilon$

(a) Briefly describe the language generated by this grammar.

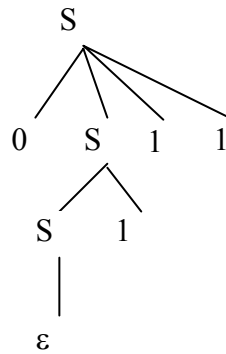
$0^m 1^n$ ($n \geq 2m \geq 0$)

(b) Show that this grammar is ambiguous by giving a string that can be parsed in two different ways and showing the two corresponding parse trees.

For example, the string "0111" can be parsed in the following two ways



(a)



(b)

(c) Rewrite the grammar to eliminate the ambiguity.

$S \rightarrow 0S11 \mid T$

$T \rightarrow T1$

$T \rightarrow \epsilon$

7. Let L be a language defined over $\Sigma = \{a, b\}$ and L consists of all strings with the same number of a's and b's, Give a context free grammar for L.

$S \rightarrow aSbS$

$S \rightarrow bSaS$

$S \rightarrow \epsilon$

8. Construct a regular grammar for the language L, where L accepts $(a|b)^*$ except for b^*a^* (check your answer for 2 for hints).

$S \rightarrow bS \mid aA$
 $A \rightarrow aA \mid bB$
 $B \rightarrow aB \mid bB \mid a \mid b \mid \varepsilon$

9. Construct a context sensitive grammar for the language $\delta c \delta$, where δ can be any string of a's and b's. Use some examples to illustrate how your grammar would work.

Basic concept: generate $\delta c \delta^r$, which is context free and easy to do.

Then, switch δ^r to δ .

Use R to mark the reverse move, F the forward move, and E the end of string

$S \rightarrow X R E$
 $X \rightarrow a X A \mid b X B \mid c$

First move the last nonterminal backward, use R to mark it

$A A R \rightarrow A R A$
 $A B R \rightarrow B R A$
 $B A R \rightarrow A R B$
 $B B R \rightarrow B R B$

Till the last nonterminal reaches a terminal, now it is at the correct position for the nonterminal, change it to a terminal, change R to F so that F can later be moved forward

$c A R \rightarrow c a F$
 $c B R \rightarrow c b F$
 $a A R \rightarrow a a F$
 $a B R \rightarrow a b F$
 $b A R \rightarrow b a F$
 $b B R \rightarrow b b F$

F needs to be moved to the end, the order of the nonterminals do not change

$F A A \rightarrow A F A$
 $F A B \rightarrow A F B$
 $F B A \rightarrow B F A$
 $F B B \rightarrow B F B$

When F moves to the end, before the last terminal, switch and change it to R

$F A E \rightarrow A R E$
 $F B E \rightarrow B R E$

Termination, should be done when only one terminal left

$a A R E \rightarrow a a$ -- could also terminate on y F Y E, but won't be symmetric with c Y R E
 $a B R E \rightarrow a b$
 $b A R E \rightarrow b a$
 $b B R E \rightarrow b b$

$c A R E \rightarrow c a$ - when w is of length 1

$c B R E \rightarrow c b$

$c R E \rightarrow c$ - when w is an empty string

First example

$X R E$
 $y_1 X Y_1 R E$
 $y_1 y_2 X Y_1 Y_2 R E$
 ...
 $y_1 y_2 y_3 y_4 c Y_4 Y_3 Y_2 Y_1 R E$
 $y_1 y_2 y_3 y_4 c Y_4 Y_3 Y_1 R Y_2 E$
 $y_1 y_2 y_3 y_4 c Y_4 Y_1 R Y_3 Y_2 E$
 $y_1 y_2 y_3 y_4 c Y_1 R Y_4 Y_3 Y_2 E$
 $y_1 y_2 y_3 y_4 c y_1 F Y_4 Y_3 Y_2 E$
 $y_1 y_2 y_3 y_4 c y_1 Y_4 F Y_3 Y_2 E$
 $y_1 y_2 y_3 y_4 c y_1 Y_4 Y_3 F Y_2 E$
 $y_1 y_2 y_3 y_4 c y_1 Y_4 Y_3 Y_2 R E$
 ...
 $y_1 y_2 y_3 y_4 c y_1 Y_2 R Y_4 Y_3 E$
 $y_1 y_2 y_3 y_4 c y_1 y_2 F Y_4 Y_3 E$
 $y_1 y_2 y_3 y_4 c y_1 y_2 Y_4 F Y_3 E$
 $y_1 y_2 y_3 y_4 c y_1 y_2 Y_4 Y_3 R E$
 $y_1 y_2 y_3 y_4 c y_1 y_2 Y_3 R Y_4 E$
 $y_1 y_2 y_3 y_4 c y_1 y_2 y_3 F Y_4 E$
 $y_1 y_2 y_3 y_4 c y_1 y_2 y_3 Y_4 R E$
 $y_1 y_2 y_3 y_4 c y_1 y_2 y_3 y_4$

-- could also terminate on $y F Y E$

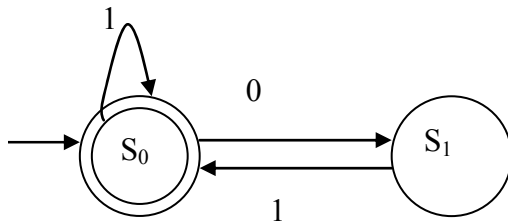
Second example

$y_1 c Y_1 R E$
 $y_1 c y_1$

Third example

$c R E$
 c

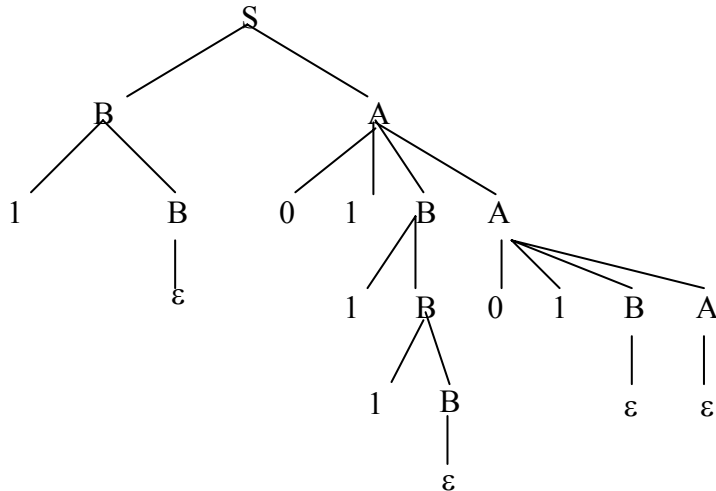
10. Consider the language: All strings of 0's and 1's such that every 0 is immediately followed by at least one 1.
- (a) Construct a DFA for the language and convert it to a grammar (following the conversion algorithm).



$S_0 \rightarrow 1S_0 \mid 0S_1 \mid \epsilon$
 $S_1 \rightarrow 1S_0$

(b) As we discussed in class, the grammar you constructed in (a) is a regular grammar (type 3 grammar). The following grammar is also for the same language. Show a parse tree for the string 1011101.

$S \rightarrow BA$
 $A \rightarrow 01BA \mid \epsilon$
 $B \rightarrow 1B \mid \epsilon$



(c) Give the leftmost derivation for (b).

$S \Rightarrow BA \Rightarrow 1BA \Rightarrow 1A \Rightarrow 101BA \Rightarrow 1011BA \Rightarrow 10111BA \Rightarrow 10111A \Rightarrow 1011101BA \Rightarrow 1011101A \Rightarrow 1011101$

(d) Give the rightmost derivation for (b).

$S \Rightarrow BA \Rightarrow B01BA \Rightarrow B01B01BA \Rightarrow B01B01B \Rightarrow B01B01 \Rightarrow B011B01 \Rightarrow B0111B01 \Rightarrow B011101 \Rightarrow 1011101$

(e) How many different derivations can the string 1011101 has? Briefly justify your answer.

$S \Rightarrow \underset{a}{AB} (\Rightarrow \underset{b}{01BA} (\Rightarrow \underset{c}{011B} \Rightarrow \underset{d}{0111B} \Rightarrow \underset{e}{0111})) \parallel (\Rightarrow \underset{f}{01BA} (\Rightarrow \underset{g}{01} \parallel \Rightarrow \underset{h}{\epsilon})) \parallel (\Rightarrow \underset{i}{1B} \Rightarrow \underset{j}{1})$

Equivalent to execution of: $a \rightarrow \{ b \rightarrow \{ (c \rightarrow d \rightarrow e) \parallel (f \rightarrow (g \parallel h)) \} \} \parallel (i \rightarrow j)$

Simplified to: $a \rightarrow \{ b \rightarrow \{ (cde) \parallel (f \rightarrow (g \parallel h)) \} \} \parallel (ij)$

Number of different orders in executing $(g \parallel h) = C(2,1) = 2$

-- e.g., hg or gh

Number of different orders in executing $f \rightarrow (g \parallel h) = 2$

-- sequential execution has no alternative

Number of different orders in executing $(cde \parallel fgh) = C(6,3) = 20$

-- e.g., cfdegh, cdefgh, fgdche, ...

Number of different orders in executing $(cde \parallel f \rightarrow (g \parallel h)) = C(6,3) = 20 * 2 = 40$

Let $X = b \rightarrow \{ (c \rightarrow d \rightarrow e) \parallel (f \rightarrow (g \parallel h)) \}$ and X_k is one way to execute X

Number of different orders in executing $X_k \parallel ij = C(9,2) = 36$

Number of different orders in executing $X \parallel ij = 36 * 40 = 1440$

-- X has 40 execution orders

Number of different orders in executing $a \rightarrow \{ X \parallel ij \}$ is still 1440

So, the answer is 1440