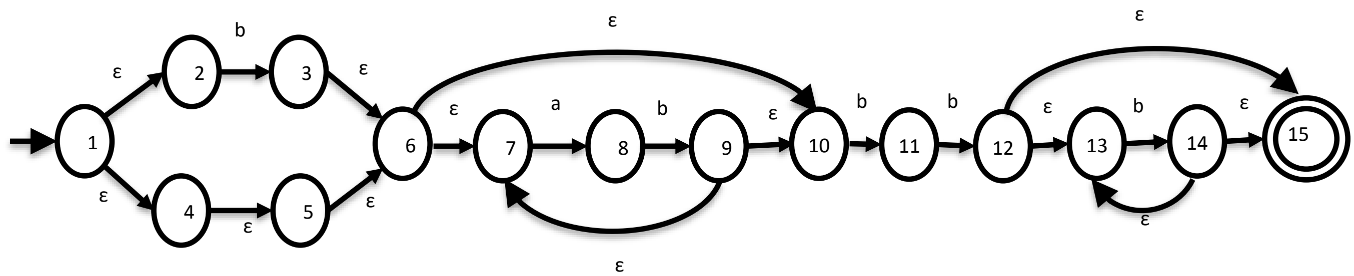
CS 1622 – Homework 2 - Answers

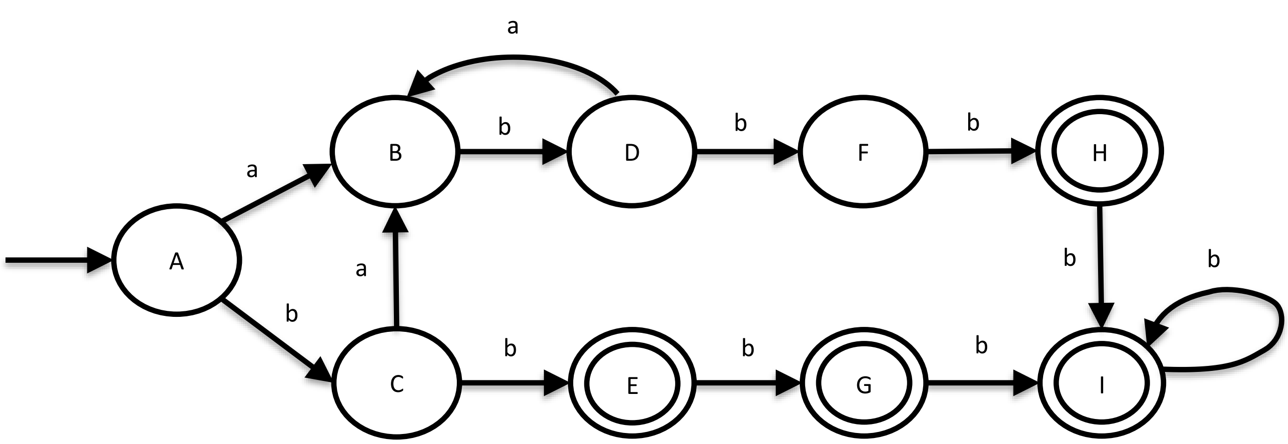
Please submit a typewritten document. I'd prefer you draw your DFA on the computer, but if this is a challenge, you may hand draw them neatly on the paper by hand.

1.) Using the algorithm from lecture, convert the following NFA to a DFA (alphabet is {a,b}):

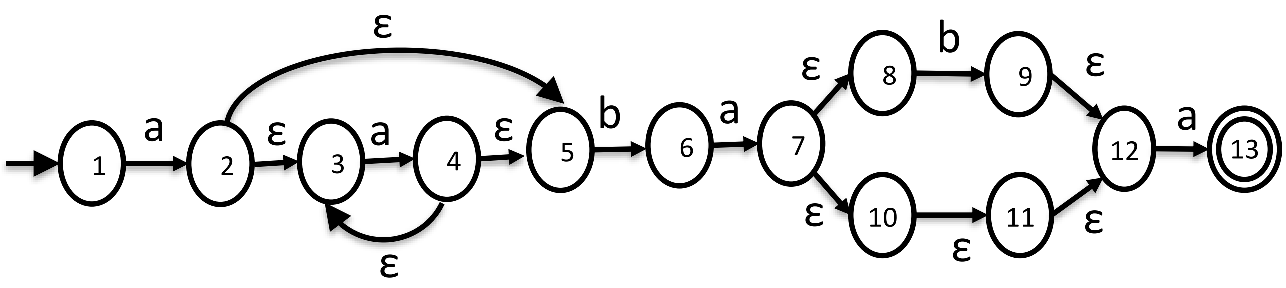


DFA Start State = ε-closure(1) = {1,2,4,5,6,7,10} = *A*

In state:	**See an a**	**See a b**
A = {1,2,3,4,5,6,7,10}	{8} = B	{3, 6, 7, 10, 11} = C
B = {8}		{7, 9, 10} = D
C = {3, 6, 7, 10, 11}	{8} = B	{11, 12, 13, 15} = E
D = {7, 9, 10}	{8} = B	{11} = F
E = {11, 12, 13, 15}		{12, 13, 14, 15} = G
F = {11}		{12, 13, 15} = H
G = {12, 13, 14, 15}		{13, 14, 15} = I
H = {12, 13, 15}		{13, 14, 15} = I
I = {13, 14, 15}		{13, 14, 15} = I

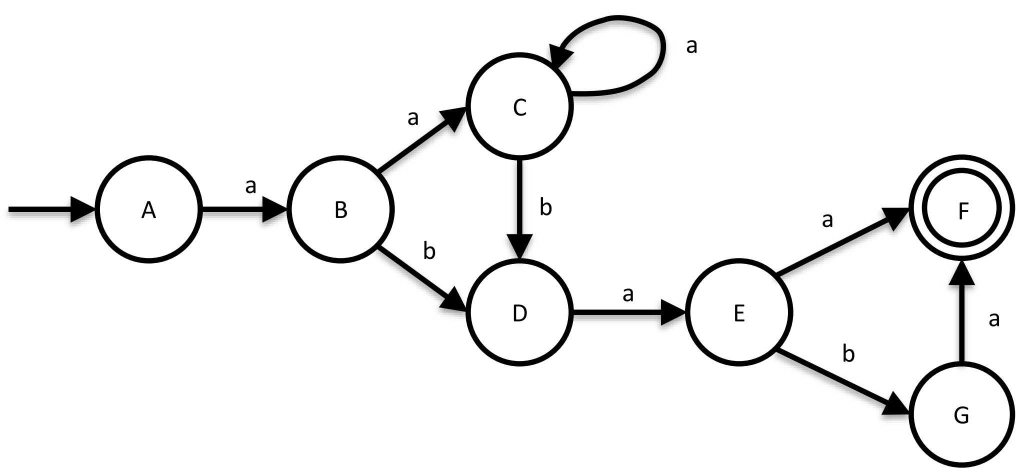


2.) Using the algorithm from lecture, convert the following NFA to a DFA (alphabet is {a,b}):

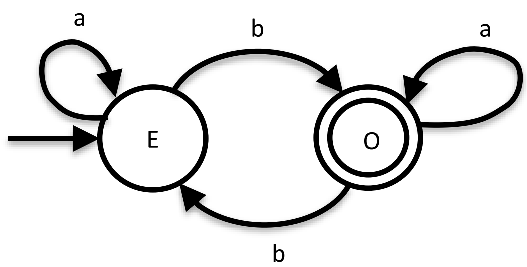


DFA Start State = ε-closure(1) = {1} = *A*

In state:	See an a	See a b
A = {1}	{2,3,5} = B	
B = {2,3,5}	{3,4,5} = C	{6} = D
C = {3,4,5}	{3,4,5} = C	{6} = D
D = {6}	{7,8,10,11,12} = E	
E = {7,8,10,11,12}	{13} = F	{9, 12}=G
F = {13}		
G = {9,12}	{13} = F	



3.) Convert the following DFA into a Regular Grammar:



E -> aE | bO

O -> bE | aO | ε

Each state becomes a nonterminal and each transition follows the pattern “character next_state”. The accepting states produce epsilon. If we wanted to avoid the erase rule, we could rewrite the grammar to do so, but this is most straightforward.

4.) Write a grammar (do not worry about associativity or precedence) for the language of Boolean expressions. Your terminals are the set {T, F, ^, v, !} (for true, false, and, or, not).

Since we have the convention of capital letters being nonterminals, I’ve spelled the terminals that represent tokens T and F as true and false:

E -> E ^ E | E v E | ! E | true | false

5.) Write a grammar for the language of bash input in Linux. Support commands that run programs with or without flags and parameters, and support pipe (|), and redirection (input and output: < >) operators.

Using | as the pipe and not for alternation of productions:

```
S → program_name OPTIONS REDIRECTION
S → program_name OPTIONS REDIRECTION | S
OPTIONS → ε
OPTIONS → parameter OPTIONS
REDIRECTION → < file_name
REDIRECTION → > file_name
REDIRECTION → < file_name > file_name
REDIRECTION → > file_name < file_name
REDIRECTION → ε
```

Notice the difference between a recursive production that can produce at least one terminal, versus something that is optional and can produce zero terminals.