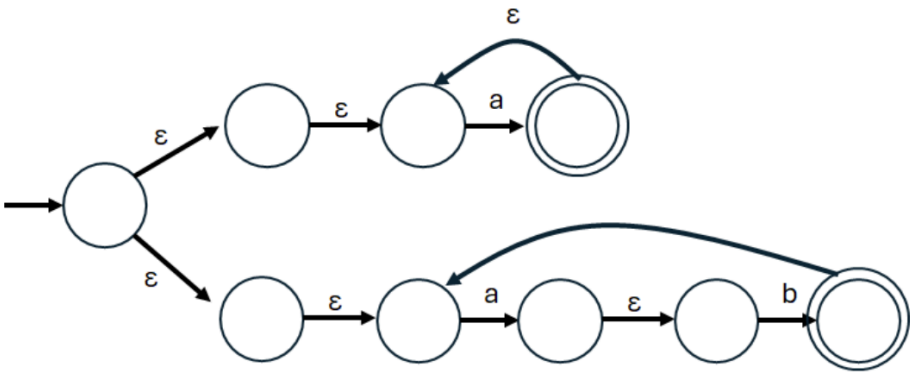
Theory of Computing Modules 0-2 Crash Course

REGEX → NFA

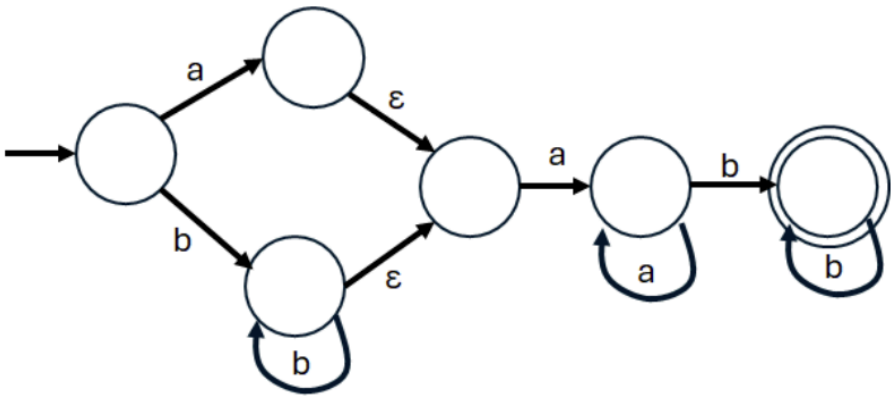
- HW5A #s 1 and 2

Book 1.28 (b) a+ U (ab)+ use the procedure *exactly* as described in Lemma 1.55 and performed in example 1.58 (you don't need to name the states)



Clearly some epsilons that could be skipped

Book 1.28 (c) (a Ub⁺)a⁺b⁺ Use the procedure described in Lemma 1.55 and performed in example 1.58 (you don't need to name the states), however you need not have all the excess epsilon branches that the exact procedure inserts. Remember that a^+ is aa*

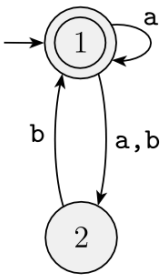


NFA → DFA

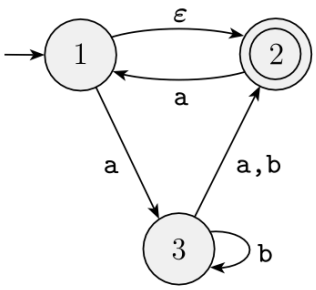
- PAGE 57 IN PAPER COPY OF TEXTBOOK

- HW4B # 6

Use the construction given in Theorem 1.39 to convert the following two nondeterministic finite automata to equivalent deterministic finite automata.



(a)

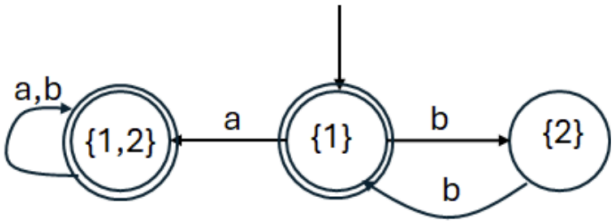


(b)

Book Problem 1.16. Either a state diagram or a transition table is ok (transition table may be simpler but make sure you id start and final states).

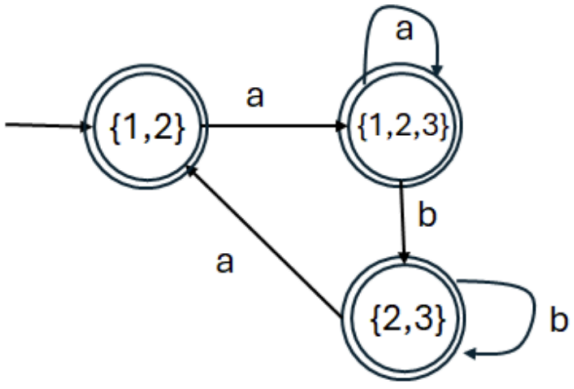
1.16 (a). DFA start = {1}, F = {{1}, {1,2}}

Current State	Input symbol	Next Start
{1}	a	{1,2}
{1}	b	{2}
{1,2}	a	{1,2}
{1,2}	b	{1,2}
{2}	a	Trap - ignore
{2}	b	{1}



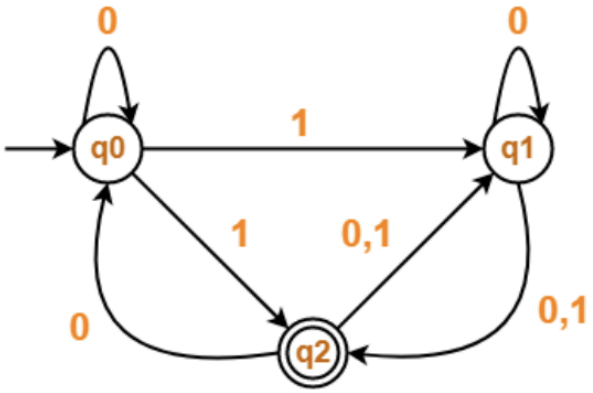
1.16 (b) DFA start = {1,2}, F ={{1,2},{1,2,3},{2,3}}

Current State	Input symbol	Next Start
{1.2}	a	{1,2,3}
{1,2}	b	Trap
{1,2,3}	a	{1,2,3}
{1,2,3}	b	{2,3}
{2,3}	a	{1,2}
{2,3}	b	{2,3}



- HW4B #8

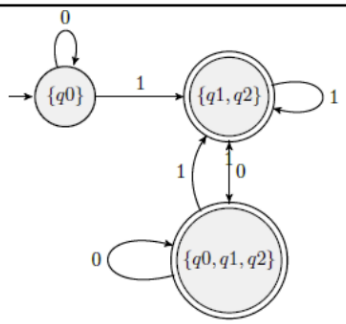
Consider the following NFA.



(c) What is the transition function for the equivalent DFA? Either a state diagram or a transition table is ok (transition table may be simpler).

Start = {q0}, Final = {{q1,q2}, {q0,q1,q2}}

Current State	Current Input	Next State
{q0}	0	{q0}
{q0}	1	{q1,q2}
{q1,q2}	0	{q0,q1,q2}
{q1,q2}	1	{q1,q2}
{q0,q1,q2}	0	{q0,q1,q2}
{q0,q1,q2}	1	{q1,q2}

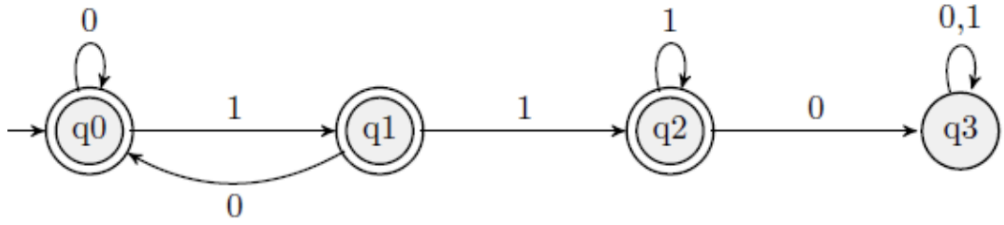


- Unioning NFAs - HW4A #3

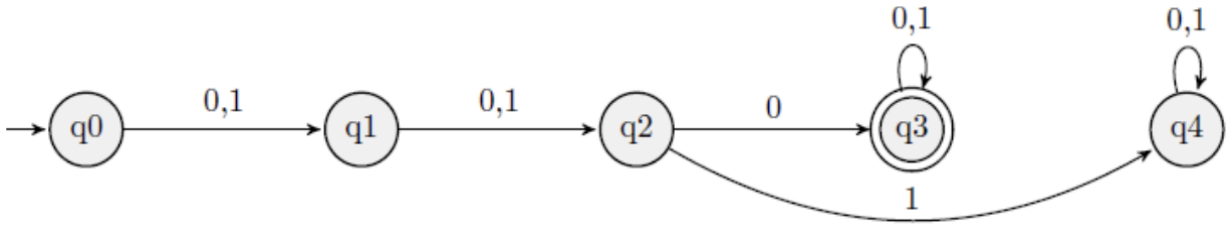
English language → DFA

- HW3A #2

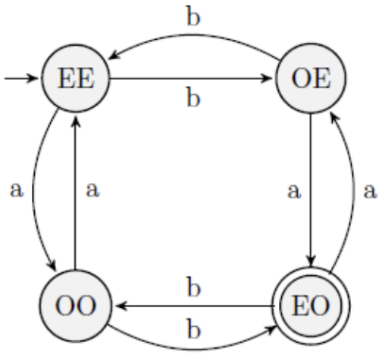
(f) {w| w doesn't contain the substring 110}



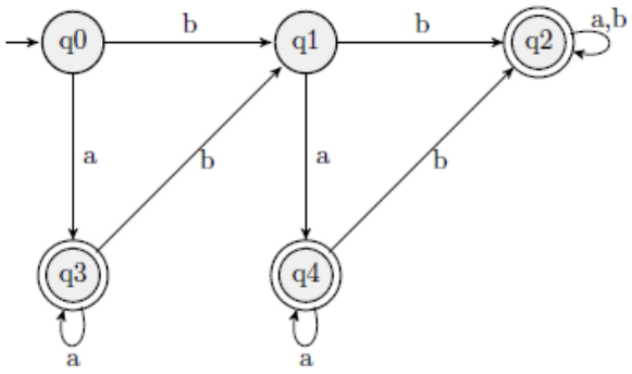
{w| w has length at least 3 and its third symbol is a 0}



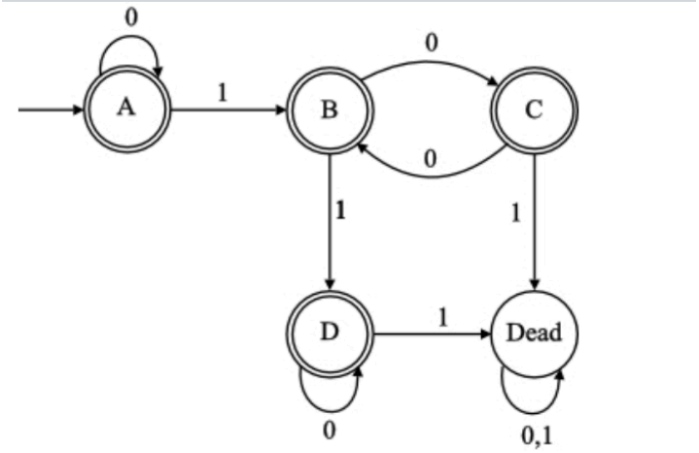
- HW3A #3 ... Construction of a DFA from the intersection of smaller ones. Typically such intersection DFAs are a DFA where the set of states is the cross-product of the two sub machines, and the transitions track what happens to both machines with the same input.
  { w | w has even length and an odd number of a's }



- HW3A #5 : At least 2 bs, with any number of as before and after the first b and any string after the second b.

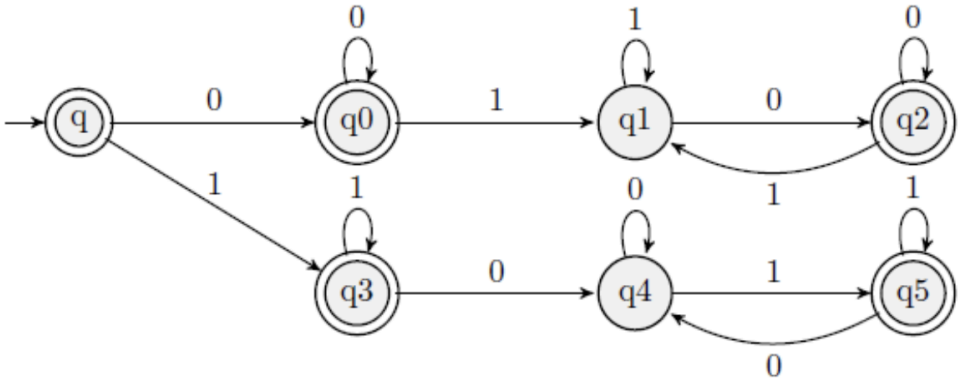


- HW3B #6



- HW5A #3

Book Problem V3:1.48;IE:1.53. Show D (with equal # of 01 and 10) is regular.



The empty set is in D so start is accepting.
If first character is a 0, we are definitely accepting (q0) as long as we don't see a 1. If we do, we have seen a 01 and must wait for a 0 to give us the matching 10 (q2). Continue to accept as long as just a 0. If we see a 1, we have a 01 and its back to q1 where we must wait non-accepting until a matching 0.
If the first character is a 1, a similar path is taken.

LOGIC

Treasure chest (hw3b)

A cave contains three chests, one with treasure, and two with fatal traps. Each chest has a label. Chest one says “the treasure is in chest two.” Chest two says “this chest contains no treasure.” Chest three says “this chest contains no treasure,” Exactly one label is true. Exactly one chest has treasure. Which chest has the treasure? Prove it.
(a) (3pt) Define a set of propositional variables, and for each indicate what they stand for. (Hint: you need 6: 2 for each chest). Solution: Six propositional letters: for i=1,2,3 (1) Ti: Chest i has treasure (2) Li: label on chest i is true
(b) (2pt) Create a wff that is true only if at least one chest has treasure. Solution: (1) (T1 ∨ T2 ∨ T3) - At least one chest must have treasure
(c) (2pt) Create a wff that indicates at least one label is true. Solution: (2) (L1 ∨ L2 ∨ L3) - At least one label must be true
(d) (2pt) For each chest write a wff that indicates what else must be true if that chest has the treasure. Solution: (3) (T1 ⇒ (¬T2 ∧ ¬T3 ∧ ¬L1 ∧ L2 ∧ L3)) If chest 1 has the treasure then chests 2 and 3 cannot have treasure and label 1 must be false and label 2 is true and label 3 must be true (4) (T2 ⇒ (¬T1 ∧ ¬T3 ∧ L1 ∧ ¬L2 ∧ L3)) If chest 2 has the treasure then chests 1 and 3 cannot have treasure and label 1 must be true and label 2 is false and label 3 must be true

(5) (T3 ⇒ (¬T1 ∧ ¬T2 ∧ ¬L1 ∧ L2 ∧ ¬L3)) If chest 3 has the treasure then chests 1 and 2 cannot have treasure and label 1 must be false and label 2 is true and label 3 must be false

(e) (2pt) For each chest write a wff that indicates what else must be true if that chest's label is true.
Solution:
(6) (L1 ⇒ (¬L2 ∧ L3 ∧ ¬T 1 ∧ T 2 ∧ ¬T3)) If chest 1's label is true then label 2 must be false and label 3 must be true and the treasure is in chest 2 and not in chest 1 or 3
(7) (L2 ⇒ (¬L1 ∧ ¬T 2)) If chest 2's label is true then the treasure is not in chest 2 and the label on 1 must be false
(8) (L3 ⇒ ¬T3) If chest 3's label is true then the treasure is not in chest 3

(f) (4pt) Determine from this which chest has the treasure. (Hint: AND your wffs together and use a truth table). Note feel free to construct the truth table using Excel and its logic functions. Note also in doing a truth table you can lump together multiple rows that have the same outcome - Use a * for a logical variable value to indicate this row applies when the variable is either T or F - i.e. a "don't care".
Solution: Note you should add:
(9) (¬T1V¬T2) ∧ (¬T1v¬T3) ∧ (¬T2v¬T3)) no two chests can contain treasure
(10) (¬(L1v¬L2) ∧ (¬L1v¬L3) ∧ (¬L2v¬L3)) no two labels can be true

Interpretation						Wffs - all must be true									
T1	T2	T3	L1	L2	L3	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
Cases where either no chests have treasures or two chests have treasures															
0	0	0	*	*	*	0									
1	1	*	*	*	*			0						0	
1	*	1	*	*	*			0						0	
*	1	1	*	*	*				0					0	
Cases where either no labels are true or two labels are true															
*	*	*	0	0	0		0								
*	*	*	1	1	*										0
*	*	*	1	*	1										0
*	*	*	0	1	1										0
Cases where exactly one label is true and exactly one chest has treasure															
1	0	0	1	0	0			0							
1	0	0	0	1	0			0							
1	0	0	0	0	1			0							
0	1	0	1	0	0				0						
0	1	0	0	1	0				0						
0	1	0	0	0	1				0						
0	0	1	1	0	0						0				
0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1
0	0	1	0	0	1						0				

Only T3 true and L2 true, and all else false satisfy everything. Treasure is in Chest 3.

Tom fired problem (HW2B)

Now consider the following statements, all of which are to be assumed true.

1. Either Tom attended the meeting or Tom was not invited.
2. If the boss wanted Tom at the meeting, then Tom was invited.
3. If the boss did not want Tom there and the boss did not invite Tom, then he is going to be fired.
4. Tom did not attend the meeting.

Define some logical variables as follows - with their meaning to us as humans as follows:

- A is true if Tom attended the meeting
- I is true if Tom was invited
- W is true if the boss wanted Tom at the meeting, and false if he did not.
- N is true if the boss did not want Tom there
- X is true if Tom is going to be fired, and false if he is not.

Now for example the first statement translates to (A +-I). Likewise the third sentence is equivalent to $((-W & -I)=> X). Each of the other statements can be converted similarly, and the wff representing the whole problem is the logical AND of that for each statement (it is permissible to have a clause of one literal).

Problem 6. (10pt) Convert all the sentences into a single wff (AND them together). Don't simplify at this point..

(A + -I) & (W => I) & ((-W & -I) => X) & (-A)

$(A \vee \neg I) \wedge (W \rightarrow I) \wedge ((\neg W \wedge \neg I) \rightarrow X) \wedge (\neg A)$ * $\neg W$ could also be N

Problem 7. (10pt) Convert that wff into CNF form. How many clauses are there?

Steps:
(A + -I) & (W => I) & ((-W & -I) => X) & (-A)
(A + -I) & (-W + I) & (-(W + I) => X) & (-A)
(A + -I) & (-W + I) & (--(W + I) + X) & (-A)
(A + -I) & (-W + I) & ((W + I) + X) & (-A)
(A + -I) & (-W + I) & (W + I + X) & (-A)

Answer:1
(A + -I) & (-W + I) & (W + I + X) & (-A)
4 clauses

Problem 8. (20pt) Assuming all the sentences are true, see if you can determine if Tom will be fired or not, i.e. are the only assignments of True/False to the variables where all sentences are true have the variable X is not True. Show your reasoning.
If you use a truth table where you compute the overall veracity of the wff, it may be useful to have a column that tracks each clause, along with a column for the entire wff. Feel free to use "*" as boolean variable values to signify that given the values for the other variables, the value of that variable doesn't affect the outcome for this row. That allows you to reduce the number of rows in the table.

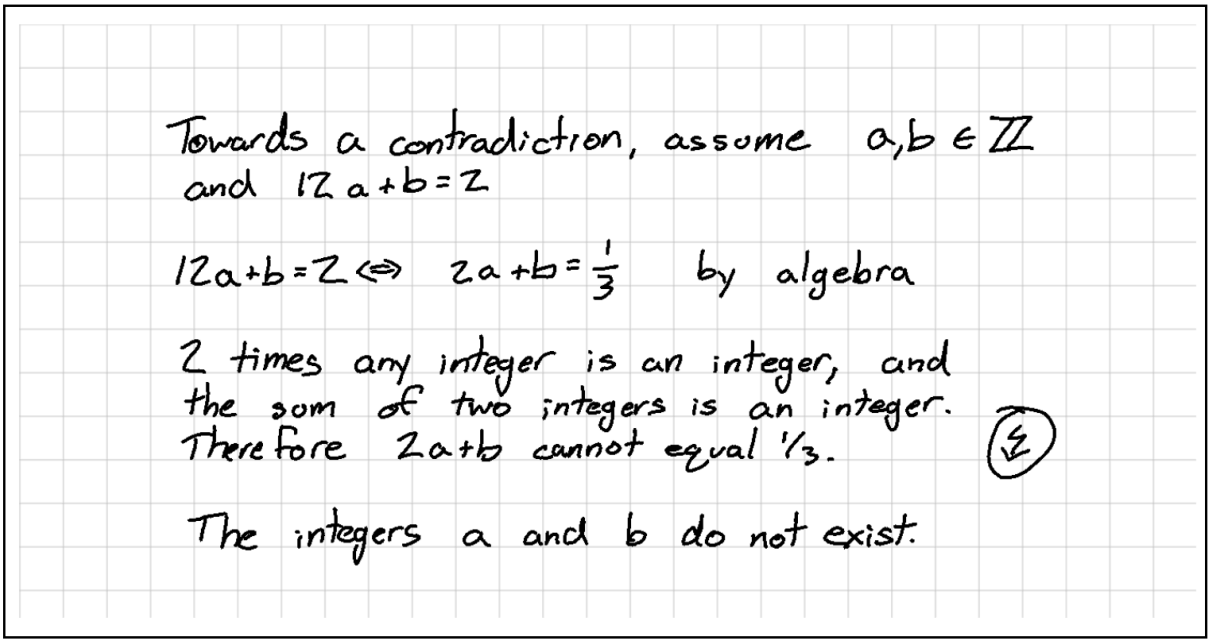
The CNF is only true when t. So Tom will be fired.

A	I	W	X	A + -I	-W + I	W + I + X	-A	Overall wff
0	0	0	1	1	1	1	1	1
0	1	*	*	0	1	*	1	0
0	0	1	*	1	0	*	1	0
1	*	*	*	1	*	*	0	0

Proof by contradiction

- HW1B # 10

Problem 10. (10pt) Prove by contradiction that there do not exist two integers a and b such that 12a+6b=2.

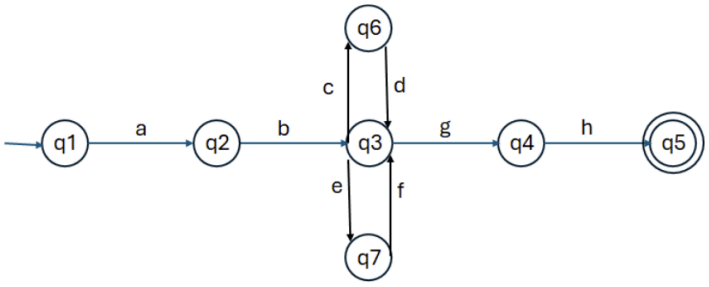


NOTES ON PROOF BY CONTRADICTION

Pumping a string that IS from a regular language

Example in HW6: Consider the regex ab(cd U ef)*gh. Consider also the string w=abcdefgh that is part of this language.

(a) (2pt) How many states are there in the simplest DFA you can think of that accepts the language. Draw the DFA.



Solution: 7 states

(b) (1pt) True/False: is any string of length 4 pumpable? Solution: No - only string of length 4 is abgh

---

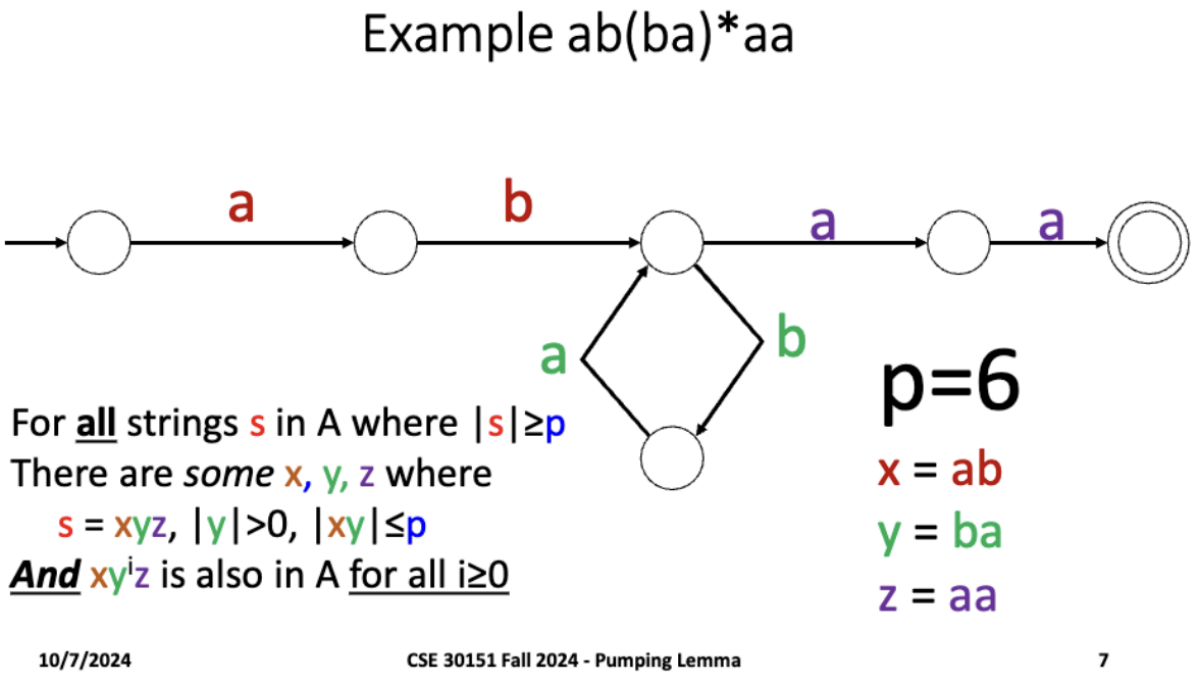
(c) (2pt) Come up with your best estimate of what p might be for this language and describe your choice. Think about the number of transitions between states rather than just states. Remember that p is a minimal string length where for <u>any</u> string in the language that is at or beyond p in length, there is some substring in it that that can be pumped.

Solution: There are 8 transitions so if the string is greater than 8 there must have been a repeat. Better answer: At length 5 (actually 6 as there are no strings in language of length 5), we have taken one of the up/down states and could have repeated them.

(d) (15pt) For the string abcdefgh list all the possible substrings that could take the role of y in the pumping lemma. (Hint: there are more than 2) For each possibility, what are the matching x and z? Also for each choice, what is the string that results from pumping up once (i=2) and down (i=0)? Fill in the table below.

x	y	z	xy^2z	xy^0z
ab	cd	efgh	abcdcdefgh	abefgh
abcd	ef	gh	abcdefefgh	abcdgh
ab	cdef	gh	abcdefcdefgh	abgh

Notes from class: Given a regular expression, pick out a string that is pumpable, show which parts are xyz, and guess a reasonable p.



Determining p:

- P is no larger than the number of states
    - But can be smaller
- Pick the longest path that goes through the loop at least once
- For example: aUdb*eUc+
    - P is 5
- ab(ba)*a → pumping length is 5

Prove a language is NOT regular

The Pumping Lemma states that...

If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, s = xyz, satisfying the following conditions:

1. **For each i >= 0, $xy^iz \in A$,**
2. **|y| > 0, and**
3. **|xy| <= p.**

General Proof Structure:

1. Assume towards a contradiction that the language L is regular.

2. Let p be the pumping length given by the pumping lemma.
3. Choose s.
4. Because s is a member of L and s has length > p, the pumping lemma guarantees that s can be split into 3 pieces, s = xyz, where for any i >= 0, the string xy^iz is in L.
5. Locate the contradiction ...

PUMPING UP

(a) (5pt) Book 1-29b. Use the pumping lemma to show not regular. Note this language has 3 identical copies of some string w concatenated together. Hint: consider the string $0^p10^p10^p1$.
A = {www | w ∈ {a,b}* }

Solution: Assume it is regular with pumping length p. Consider the string $0^p10^p10^p1$. since |xy| must be no greater than p, the y must be some part of the leading p 0s. Since |y| ≥ 1 this means that y is 1 or more 0s, say k of them. Pumping these 0s' to the first part of the string makes the string $0^{p+k}10^p10^p$. But now the first third of the string is different from the latter two, so this does not pump. Thus the language must not be regular.

Problem 6. (10pt) Pumping lemma to show non-regular

Book 1.46c; IE 1.51c. Show non regular. Remember that if a language is regular, then so is its complement.
{w | w ∈ {0,1}* is not a palindrome}

Solution: A language is a palindrome if it ends the same forwards and back. Also If a language is regular then so is its complement, and vice versa. So let's look at {w|w is a palindrome }. Try the string 0^p10^p. Once again y must be all 0s (and at least one 0). Pumping changes the first half in a way that is not replicated in reverse in the second half. Thus it is not regular.
Since this is not regular, then neither can be its complement.

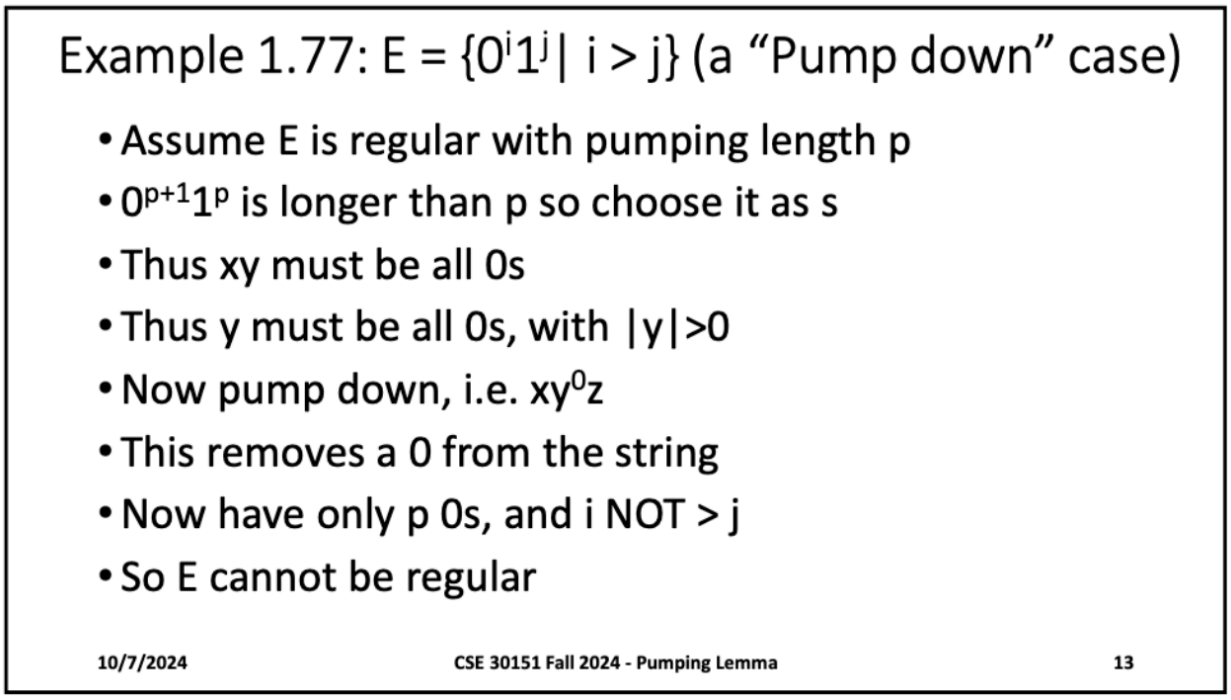
Example 1.76: F = $\{1^{n^2} \mid n \geq 0\}$

- Assume F is regular with pumping length p
- 1^{p^2} is longer than p so choose it as s
- xy must be all 1s, and $|xy| \leq p$, so $y = 1^k$ for some $k \leq p$
- So if we pump once (xy^2z) 1^{p^2+k} must be in F, with $k \leq p$
- But then p^2+k must also be a perfect square
- But $p^2+k < p^2+2p+1 = (p+1)^2$
- Thus 1^{p^2+k} is NOT in F
- F is not regular

PUMPING DOWN

(b) (5pt) Book 1.49b; IE 1.71b. Show non regular. Note this is different than Book 1.49a; IE 1.71a because here the second part can have AT MOST the same number of 1s as the first part. (Also the first part is a bit of a "trick" question)

Solution: Assume it is regular. Consider the string 1^p01^p. The second part has the same numbers of 1s as the first and thus is in the language, so it must be pumpable to be regular. y must be all as' of length between 1 and p. If we pump down, i.e. y^0, we remove a 1's from the first half, and now the second half has more 1s. Thus not regular.



Create a CFG from an English description of a CFL

A context-free grammar is a 4-tuple (V, Σ, R, S), where

1. **V is a finite set called the variables,**
2. **Σ is a finite set, disjoint from V, called the terminals,**
3. **R is a finite set of rules, with each rule being a variable and a string of variables and terminals, and**
4. **4. S ∈ V is the start variable.**

HW7 CFG for 3SAT

Problem 7. (10 pt) CFG for 3SAT

Consider the following language of strings that represent well formed formulae from 3SAT:

- An ***identifier*** is a unique string, as in x or y42.
- A ***literal*** is an identifier or a negated identifier, as in x or $\neg x$
- A ***clause*** is a list of 1, 2, or 3 literals separated by "+" and surrounded by "()", as in "(x+y+¬z)"
- A ***wff*** is an arbitrarily long string of clauses separated by "&" as in "(x+¬y+z) & (¬x+w+t)& (t+¬s+z)..."

Assume the following:

- For nonterminals use I for identifiers, L for literals, C for clauses, and W for wffs.
- Assume that you are given rules that say how to form strings that represent variables, so you need not define any rules like I→ ..., but you can use I in other productions.

Define formally (i.e. all parts) the full CFG grammar; include defining production rules for L, then for C, and then for W. For Σ ignore whatever characters are used in the rules for I.

V = { I, L, C, W }
Σ = {(,), +, ¬, &, ...}
Start is W
L → I | ¬I
C → (L) | (L + L) | (L + L + L)
W → C | W & C | ε

2.4 Give context-free grammars that generate the following languages. In all parts, the alphabet Σ is {0,1}.

[A]a. $\{w \mid w$ contains at least three 1s$\}$

(a) $S \rightarrow R1R1R1R$
$R \rightarrow 0R \mid 1R \mid \varepsilon$

b. $\{w \mid w$ starts and ends with the same symbol$\}$

Solution:
S–> 0A0 | 1A1
A–> 0A | 1A | ε

c. $\{w \mid$ the length of w is odd$\}$

Book 2.4 c (5pt) => {w | the length of w is odd}
Solution: P –> 0P0 | 1P1 | 1P0 | 0P1 | 0 | 1

[A]**d.** $\{w|$ the length of w is odd and its middle symbol is a 0$\}$

(d) $S \rightarrow 0 \mid 0S0 \mid 0S1 \mid 1S0 \mid 1S1$

e. $\{w|\ w = w^{\mathcal{R}}$, that is, w is a palindrome$\}$

$S \rightarrow 0 \mid 1 \mid 0S0 \mid 1S1 \mid \varepsilon$

f. The empty set

S → S

(a) (6 pt) Design a CFG for the language L = {$a^{2n}b^{3m}c^{m}d^{n}$| n,m>0} for example aaaabbbcdd. Include V, Σ, and the starting nonterminal for your language
Solution:
L -> aaLd | aaBd (Note: n must be >0)
B -> bbbBc | bbbc

Note: an L->ϵ is wrong because n and m must be at least 1
V = {L,B}, $\Sigma = \{a, b, c, d\}$, Starting non-terminal = L

2.6 Give context-free grammars generating the following languages.

[A]**a.** The set of strings over the alphabet {a,b} with more a's than b's

b. The complement of the language $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$

[A]**c.** $\{w\#x|\ w^{\mathcal{R}}$ is a substring of x for $w, x \in \{0,1\}^*\}$

d. $\{x_1\#x_2\#\cdots\#x_k|\ k \geq 1$, each $x_i \in \{\mathtt{a}, \mathtt{b}\}^*$, and for some i and j, $x_i = x_j^{\mathcal{R}}\}$

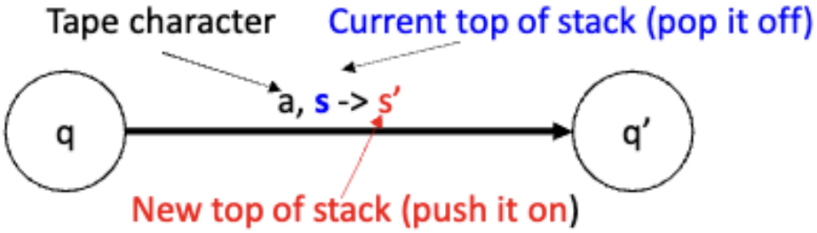
A & C answers:

2.6 **(a)** $S \rightarrow T\mathtt{a}T$
$T \rightarrow TT \mid \mathtt{a}T\mathtt{b} \mid \mathtt{b}T\mathtt{a} \mid \mathtt{a} \mid \varepsilon$
T generates all strings with at least as many a's as b's, and S forces an extra a.

(c) $S \rightarrow TX$
$T \rightarrow 0T0 \mid 1T1 \mid \#X$
$X \rightarrow 0X \mid 1X \mid \varepsilon$

Construct a PDA from a description of a language

PDA transitions

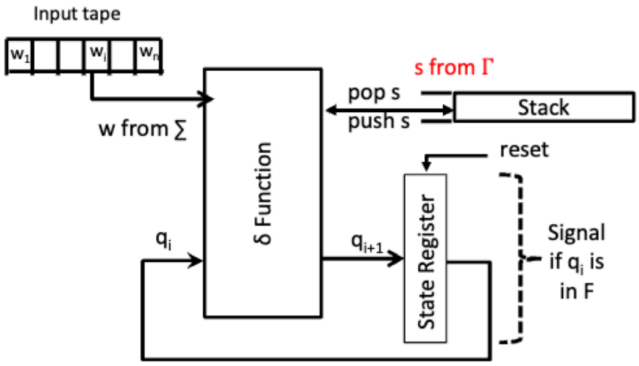


PDA formal structure

Formal PDA Model

6-tuple: $(Q, \Sigma, \delta, q_0, F, \Gamma)$:

- Q: set of states
- Σ: set of input symbols
- δ: $Q x \Sigma_\epsilon x \Gamma_\epsilon \rightarrow P(Q x \Gamma_\epsilon)$:
- q_0: initial state
- F: set of final states
- Γ: Stack alphabet



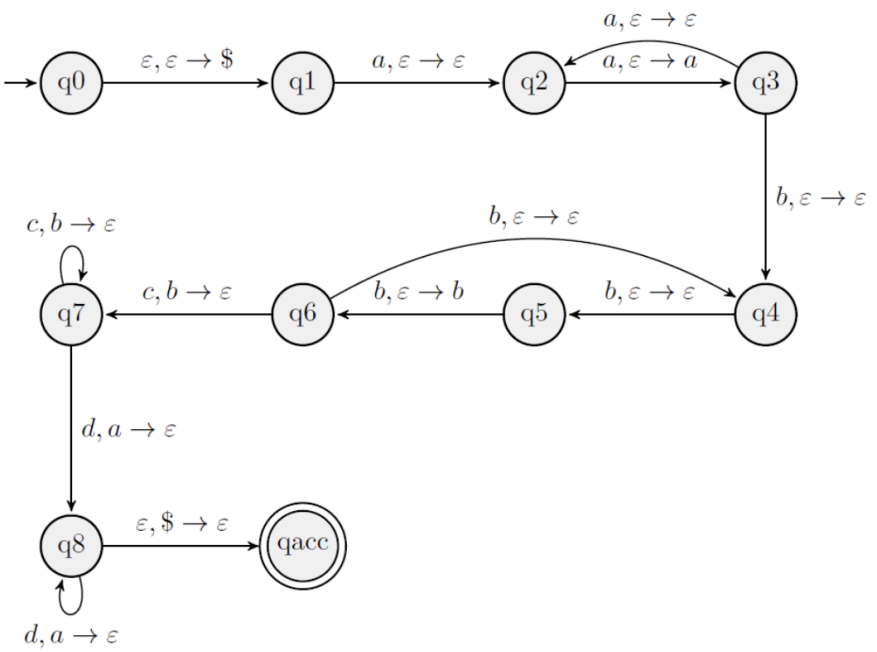
$\Sigma_\epsilon = \Sigma \cup \epsilon$

$\Gamma_\epsilon = \Gamma \cup \epsilon$

ϵ is "no symbol" or "don't care"

Example from homework:

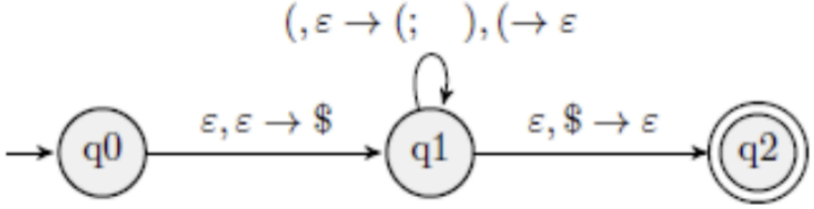
Design a PDA for the language L = {a2nb3mcmdn| n,m>0} for example aaaabbbcdd, (do not use Lemma 2.21).



(b) (5pt) What are Q, Σ, Γ, q0, and F for your design

Q = { q0, q1, q2, q3, q4, q5, q6, q7, q8, qacc }
Σ = { a, b, c, d }
Γ = { a, b, $ }
q0 = q0
F = { qacc }

Another example from the homework... matching parentheses:
P is the language of matching parenthesis, i.e. P={w|w=w1...wi...w2n| n >= 0, ={(,)}, (theΣ empty string is included), where in any prefix substring w1...wk$\ there are never more ``)'' than ``('',ε and in the whole string the numbers of ``('' and ``)'' are equal (n each).}. Examples include ''((()(()))'' but not '')()(``.

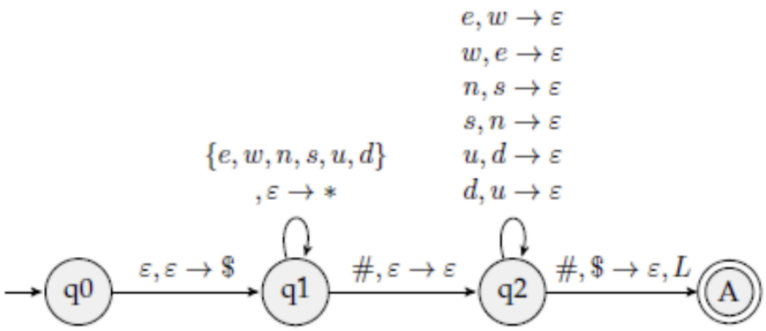


The bird example from the homework

Congratulations on becoming Chief Architect of Brainy Birds, Inc, Stack Division. Your new product, the CP-1, is to be a GPS-add-on for young carrier pigeons that tracks the path of the pigeon from its roost to its destination and back, and rewards it for reversing its path exactly on the return. The CP-1 is not given the destination or path in advance.

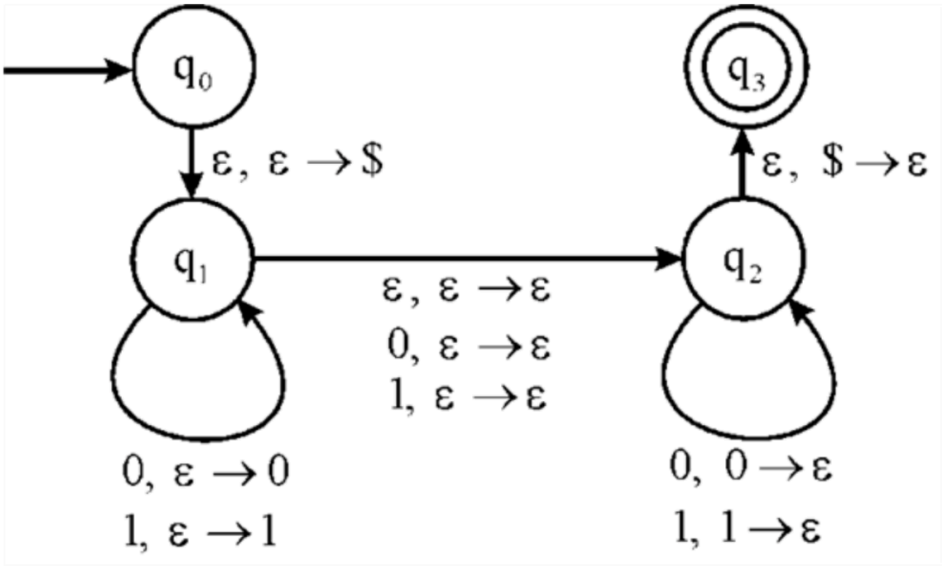
- At each second the bird's GPS sends the CP-1 a symbol from {e,w,n,s,u,d} that signals the bird's current flight path. The signals e,w,n,s are compass directions, u means the pigeon is flying straight up, and d means it is flying straight down.
- When the bird arrives at its destination, it signals arrival by pecking the # key on the CP-1 and immediately takes off to return. This tells the CP-1 that the return trip has started.
- Again the CP-1 tracks the bird's movements second by second, but now when the bird arrives home and hits # again, the CP-1 lights up Green (``accept'') (which releases a food pellet) if, and only if, the bird has followed the exact opposite path in reverse of the out-going path. For example if the outbound was "sueeendw" the return should have been "euswwwdn".
- You may assume that when the bird prepares to take off for the return trip, it always rotates itself exactly 180°. Thus if it landed pointing in the west direction, it takes off going east for the return flight.
- If at any point on the return path, the bird is not following the exact opposite path, the CP-1 should go into a state where the bird cannot receive a pellet when it lands.

Solution: Σ = {e, w, n, s, u, d, #} Don't forget #, Γ = {e, w, n, s, u, d}



Solution: State q0 pushes an initial $ to the stack. State q1 pushes GPS symbols to the stack until the first #. Then the stack is popped but must match in opposite directions. The last # must be matched to the $.

From the textbook ... {w | w^R, that is, w is a palindrome}



Problem 4. (15pt) PDA

Produce a PDA to recognise the language of strings containing more a's than b's. Describe briefly how it works. You don't need to worry about transitions to trap or reject.

Solution 1:

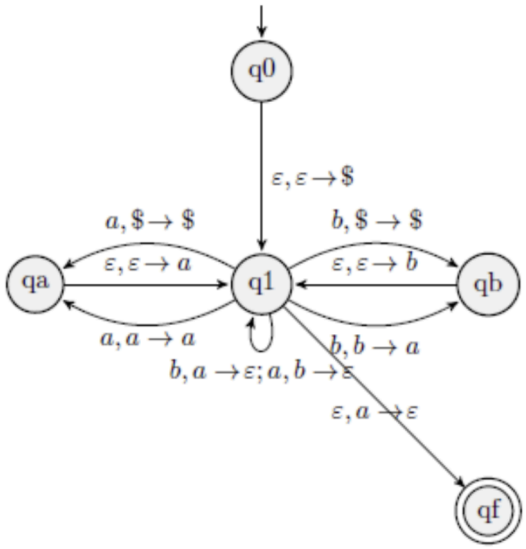
(a) Start by pushing $ to stack

(b) For either an a or b, if the top of stack is $, replace the $ and then push the input

(c) If the next input matches he stack, push both back to the stack

(d) if the next input mismatches the stack (and the stack is not $) delete both

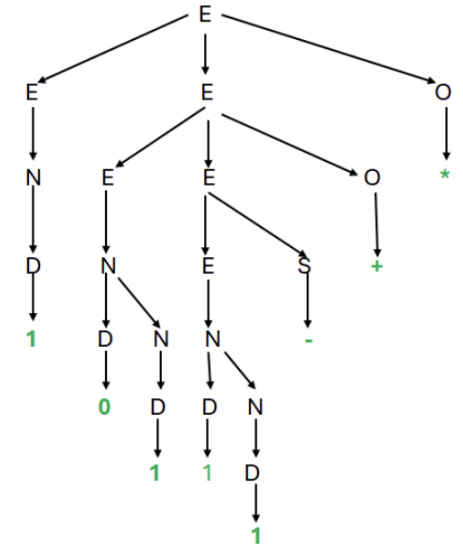
(e) If there are no more inputs and there's an a on the stack, accept



Extract a CFG from a parse tree

From the homework...

Consider the following parse tree.



(a) (1pt) What is the terminal string parsed by it? 1 0 1 1 1 - + *

(b) (1pt) What is Σ (at least that visible here)? {0,1,-,+,*}

(c) (1pt) What is V (at least that visible here)? {E,N,D,S,O}

(d) (1pt) What is the starting symbol? E

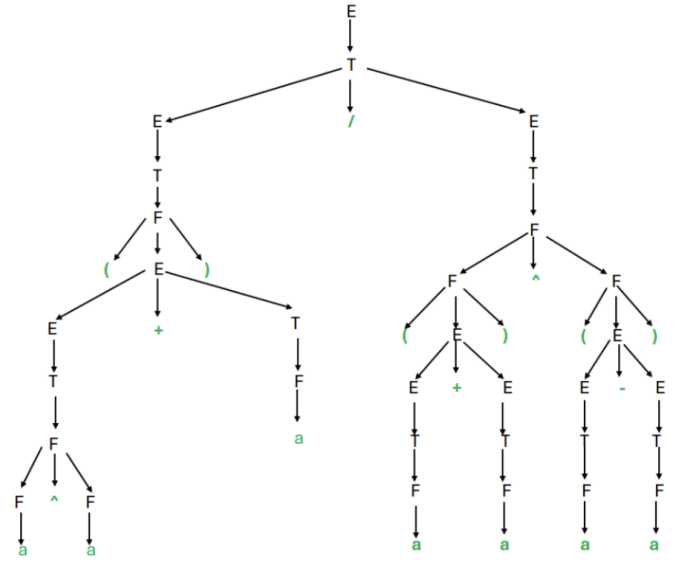
(e) (10pt) Extract all the rules you can from the parse tree.
E -> EEO | ES | N
N -> DN | D
D -> 0 | 1
O -> + | *
S -> -

From a CFG, show a parse tree for some string

Parse tree from HW6 Teamable

(a) (6 pt) Consider the CFG of Example 2.4 in the book. Add rules for divide (“/”), subtract (“-”), and exponentiation (“^”). Ensure you are obeying the normal rules of arithmetic precedence (^ first, then */, then +-). Thus in a^a +a*a we do the ^ first, then the *, and then the +. The parse tree should reflect that. Feel free to add your new rules to end of the following rules:
<EXPR> -> <EXPR>+<TERM> | <TERM> | <EXPR>-<TERM>
<TERM> -> <TERM>*<FACTOR> | <FACTOR> | <TERM>/<FACTOR>
<FACTOR> -> (<EXPR>) | a | <FACTOR>^<FACTOR>

(b) (9pt) Show a parse tree for (a^a + a) / (a+a)^(a-a). To simplify the drawing feel free to use E, T, and F for the nonterminals.



Example from the textbook:

Consider grammar $G_4 = (V, \Sigma, R, \langle \text{EXPR} \rangle)$.

V is $\{\langle \text{EXPR} \rangle, \langle \text{TERM} \rangle, \langle \text{FACTOR} \rangle\}$ and Σ is $\{\texttt{a}, \texttt{+}, \texttt{x}, \texttt{(}, \texttt{)}\}$. The rules are

$$\begin{aligned} \langle \text{EXPR} \rangle &\to \langle \text{EXPR} \rangle \texttt{+} \langle \text{TERM} \rangle \mid \langle \text{TERM} \rangle \\ \langle \text{TERM} \rangle &\to \langle \text{TERM} \rangle \texttt{x} \langle \text{FACTOR} \rangle \mid \langle \text{FACTOR} \rangle \\ \langle \text{FACTOR} \rangle &\to \texttt{(} \langle \text{EXPR} \rangle \texttt{)} \mid \texttt{a} \end{aligned}$$

The two strings `a+axa` and `(a+a)xa` can be generated with grammar G_4. The parse trees are shown in the following figure.

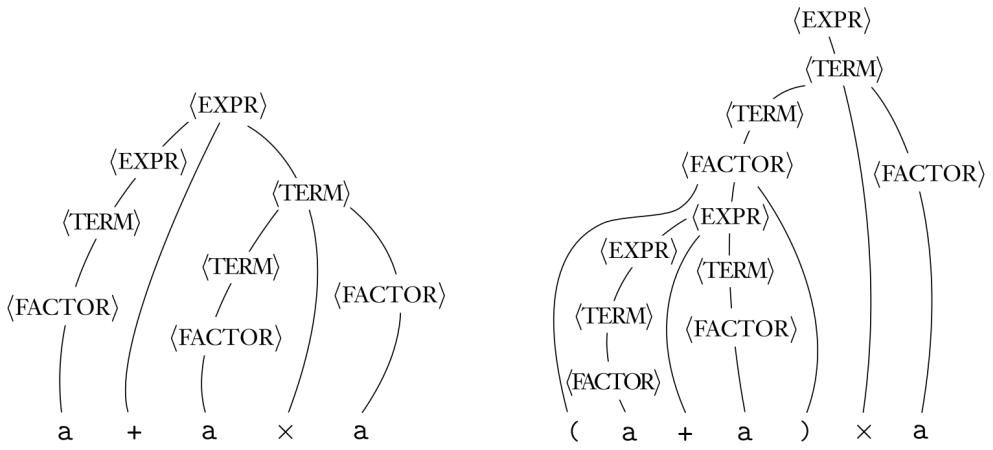


FIGURE 2.5
Parse trees for the strings `a+axa` and `(a+a)xa`

Construct a PDA from a CFG using Lemma 2.21

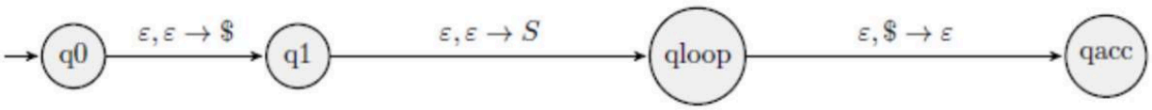
Lemma 2.21: If a language is context free, then some pushdown automaton recognizes it.

CFG to PDA Algorithm

- Stack maintains list of next characters to see in input - in reverse order.
  - A terminal on the top of the stack means that the next input on the tape sold be that character.

* A nonterminal on top of stack imples that what is expected next on input is some string from the set of strings associated with that nonterminal.

– Declare success if when the last character on the input has been processed, the stack is empty.
– Standard 4-state unpersonalized PDA for any CFG:



* Transition from q0 to q1 pushes a $ onto stack to signal bottom
* Transition from q1 to qloop pushes the starting nonterminal to the stack - whatever the input string is, it must be in that set if it is in the language
* Transition from qloop to qacc occurs nondeterministically only if the stack is empty and all characters from the tape have been processed

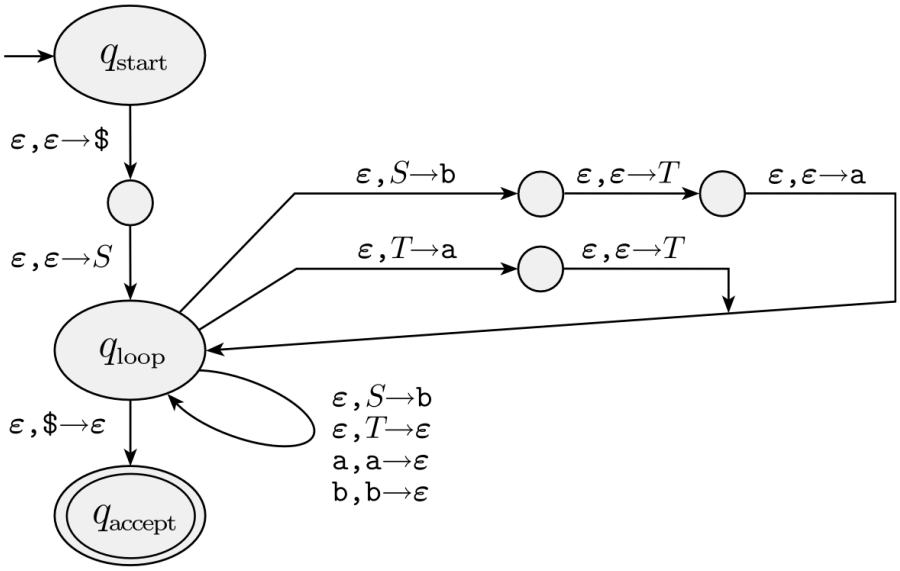
– Grammar-specific modifications to the above skeleton
* Self-loop around qloop for every character $a \in \Sigma$ as in $a, a \to \epsilon$.
* For each rule $< LHS > \to w_1 w_2 ... w_k$ where each w_i is in either V or Σ, a loop around qloop using k-1 "new" states:
  · From qloop a transition $\epsilon, < LHS > \to w_k$
  · Then a transition $\epsilon, \epsilon \to w_{k-1}$
  · ... More of the same for each w_i, with i counting down
  · The last transition $\epsilon, \epsilon \to w_1$ should be back to qloop
  · This replaces a nonterminal on the top of the stack with the righthand side of the "correct" rule, as determined by the crystal ball looking forward. The new top of stack is the leftmost character from the righthand side. Note that no characters are used from the input string.

EXAMPLE 2.25

We use the procedure developed in Lemma 2.21 to construct a PDA P_1 from the following CFG G.

$$S \to \mathtt{a}T\mathtt{b} \mid \mathtt{b}$$
$$T \to T\mathtt{a} \mid \varepsilon$$

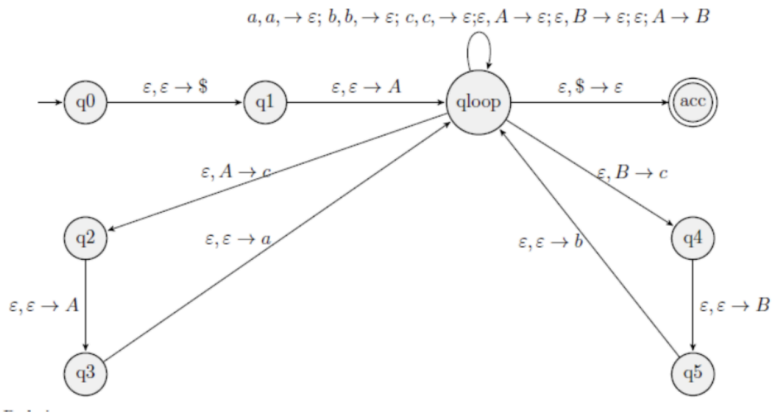
The transition function is shown in the following diagram.



Construct a PDA that accepts the language defined by the following grammar, where V={A,B}, Σ ={a,b,c} and A is the start nonterminal. Use the formal rules from the book to convert from CFGs to PDAs, as in Theorem 2.21 page 117 and demonstrated in Fig. 2.25 page 120. You need not show any trap states, nor do you have to name the states you add. Make sure you identify Γ.

A -> aAc | B | ϵ

B -> bBc | ϵ



$\Gamma = \{A, B, a, b, c, \$\}$

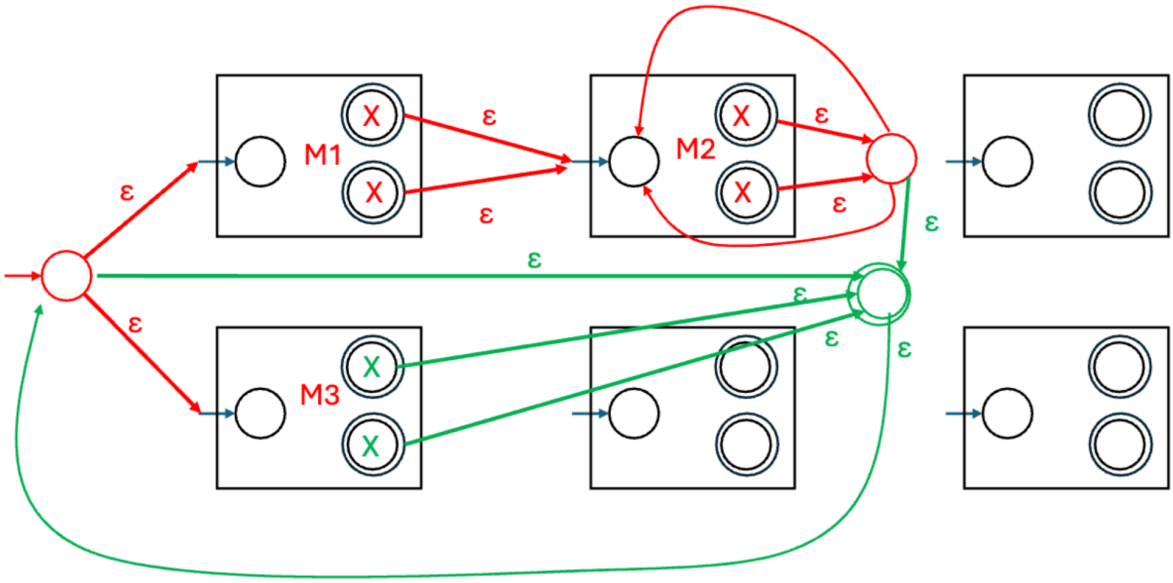
Given a regex, construct an NFA that accepts it.

From exam 1:

5. (15pt) Assume languages R_1, R_2, and R_3 are all regular, and have DFA acceptors M_1, M_2, and M_3, each with a single start state and two final states. Show how to construct an NFA out of these machines that accepts the language $(R_1 R_2^+ \cup R_3)^*$, where there is exactly one final state. Remember that R^+ is the same as RR^*.

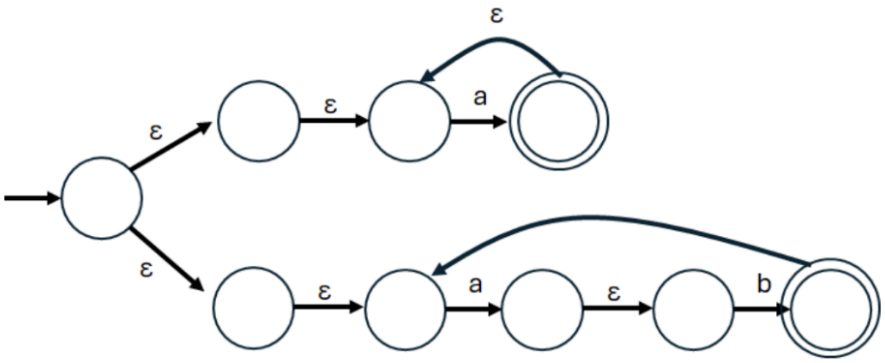
   Each of the following boxes can represent a copy of any of the three machines (just indicate in the box which machine is which). Feel free to use multiple copies of any one machine. You do not need to use all the boxes. You can also add states outside the boxes. You may "simplify" by removing obviously redundant ε transitions, but remember that the goal of this question is to see if you understand the construction process from regex to NFAs, so redundant ε transitions don't hurt.

   To take the "final" marking off of a final state in a box, put an "X" in the circle.



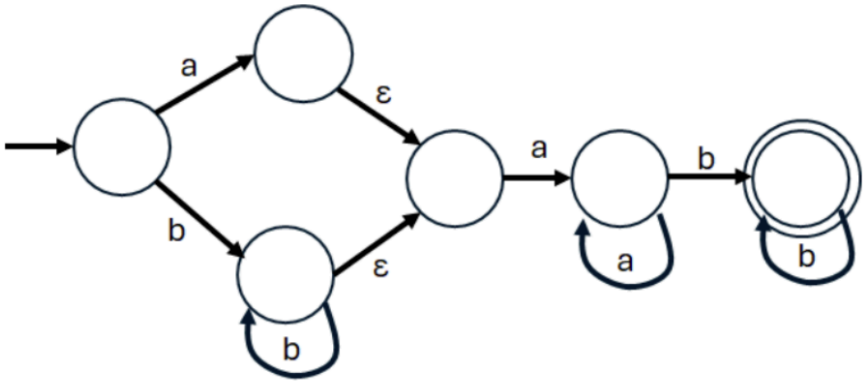
The red handles the R1R2+ U R3. The green handles the outer Kleene*.

Book 1.28 (b) a+ U (ab)+ use the procedure *exactly* as described in Lemma 1.55 and performed in example 1.58 (you don't need to name the states)



Clearly some epsilons that could be skipped

Book 1.28 (c) (a Ub$^+$)a$^+$b$^+$ Use the procedure described in Lemma 1.55 and performed in example 1.58 (you don't need to name the states), however you need not have all the excess epsilon branches that the exact procedure inserts. Remember that a$^+$ is aa*

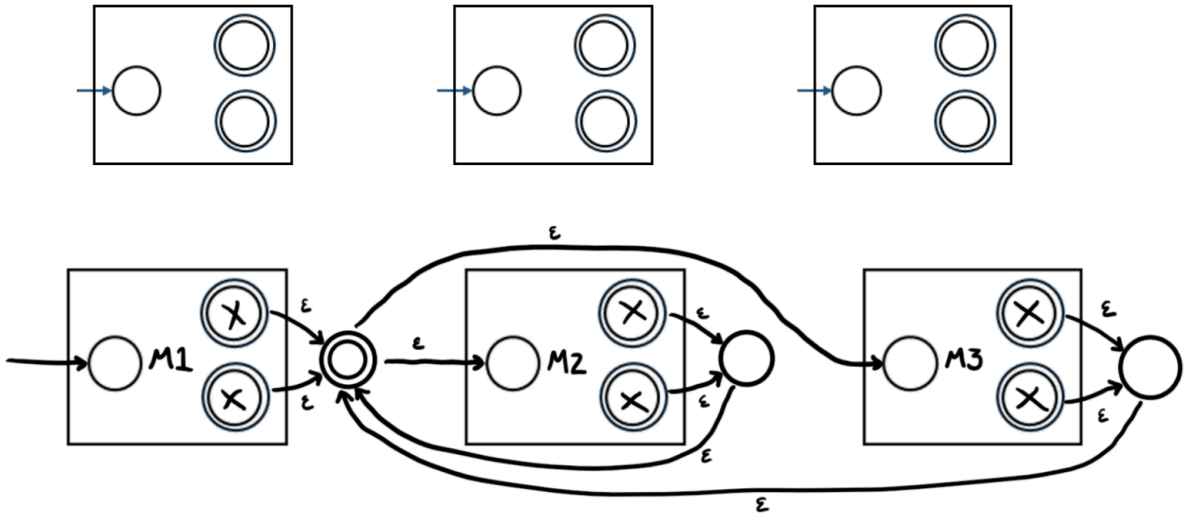


Assume languages R1, R2, and R3 are all regular, and have DFA acceptors M1, M2, and M3, each with a single start state and two final states. Show how to construct an NFA out of these machines that accepts the language R1(R2 U R3)*, with ***exactly one final state***. The details inside the Ms are irrelevant, other than they are all NFAs or DFAs.

Each of the following boxes can represent a copy of any of the three machines (just write the name of the machine inside the box). Feel free to use multiple copies of any one machine. You do not need to use all the boxes. You can (and need to) also add states outside the boxes. You may "simplify" by removing obviously redundant ϵ transitions, but remember that the goal of this question is to see if you understand the construction process from regex to NFAs, so redundant ϵs transitions don't hurt.

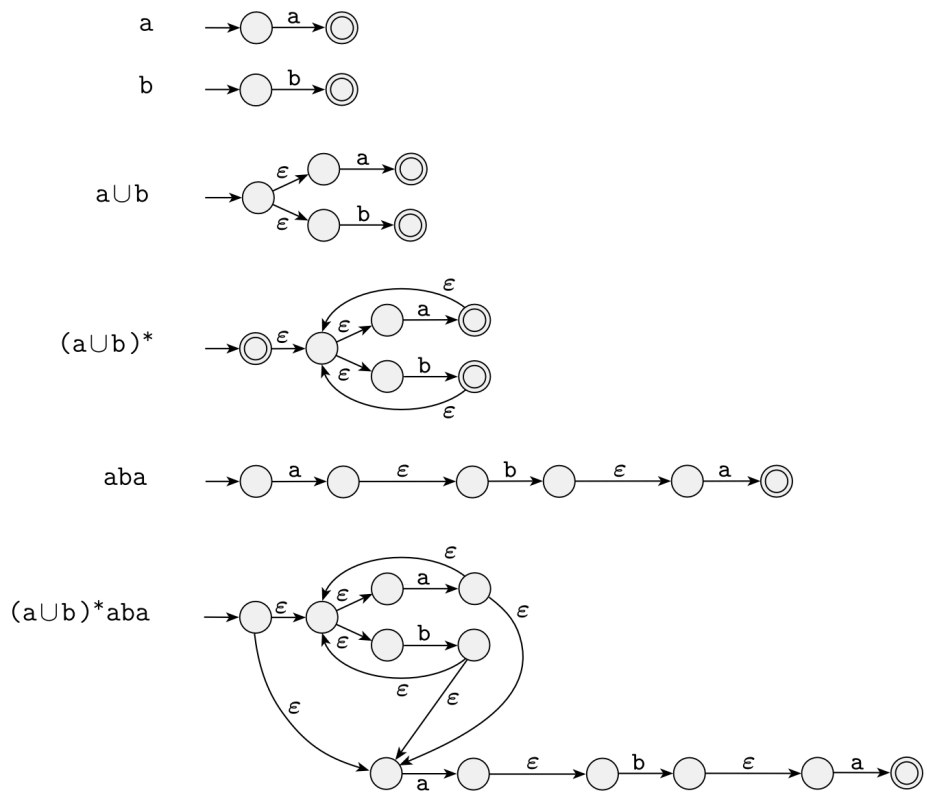
A Google slide with a copy of this is available to copy and markup.

To make a final state in a boxed machine non-final for the bigger answer, put an "X" in the circle.



EXAMPLE 1.58

In Figure 1.59, we convert the regular expression $(a \cup b)^*aba$ to an NFA. A few of the minor steps are not shown.



THE PROCESS WITH THE BOXES

UNIONING:

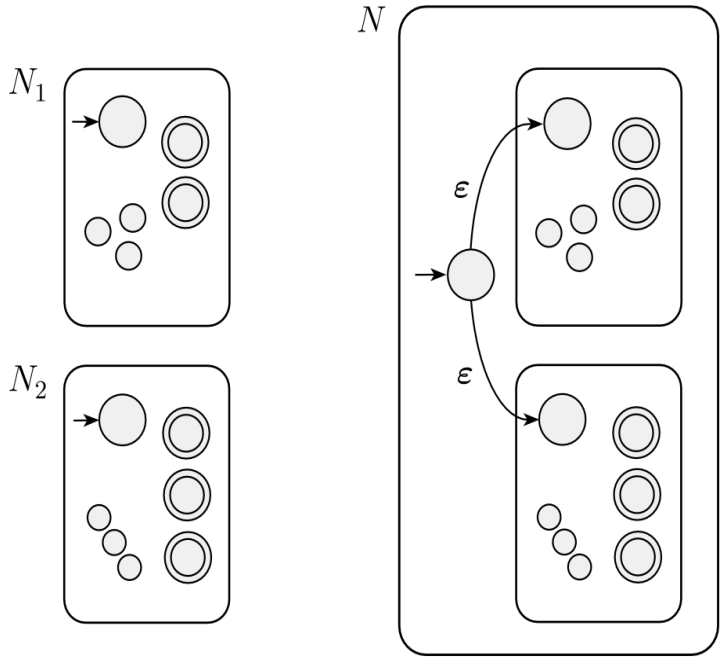


FIGURE 1.46
Construction of an NFA N to recognize $A_1 \cup A_2$

CONCATENATION:

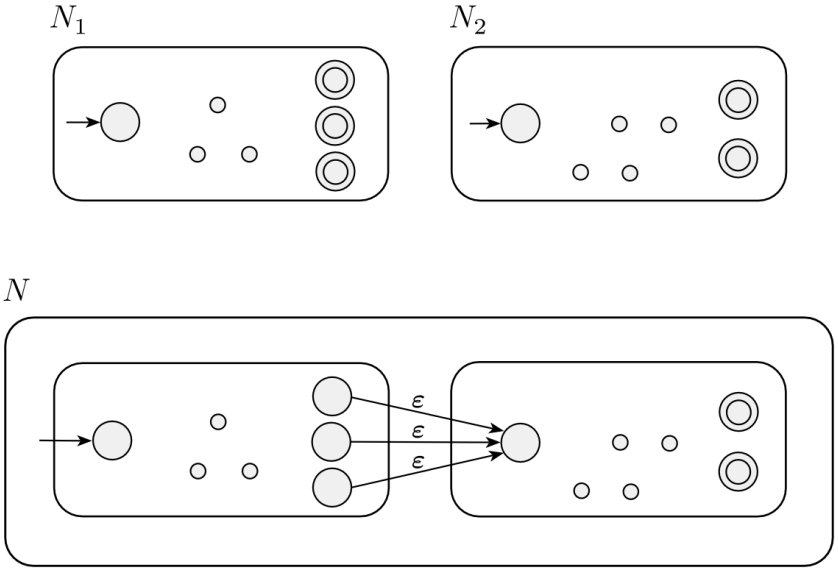


FIGURE 1.48
Construction of N to recognize $A_1 \circ A_2$

KLEENE STAR:

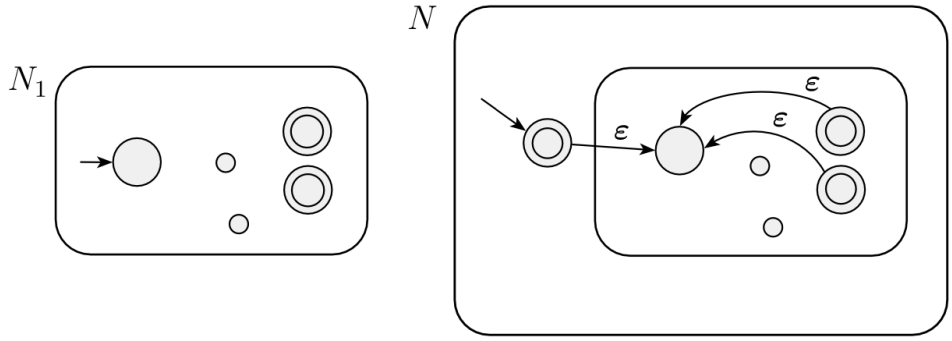


FIGURE 1.50
Construction of N to recognize A^*

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

The following is a second example of a context-free grammar, called G_2, which describes a fragment of the English language.

$$
\begin{aligned}
\langle \text{SENTENCE} \rangle &\to \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle \\
\langle \text{NOUN-PHRASE} \rangle &\to \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{VERB-PHRASE} \rangle &\to \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHRASE} \rangle \\
\langle \text{PREP-PHRASE} \rangle &\to \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle \\
\langle \text{CMPLX-NOUN} \rangle &\to \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle \\
\langle \text{CMPLX-VERB} \rangle &\to \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle \\
\langle \text{ARTICLE} \rangle &\to \texttt{a} \mid \texttt{the} \\
\langle \text{NOUN} \rangle &\to \texttt{boy} \mid \texttt{girl} \mid \texttt{flower} \\
\langle \text{VERB} \rangle &\to \texttt{touches} \mid \texttt{likes} \mid \texttt{sees} \\
\langle \text{PREP} \rangle &\to \texttt{with}
\end{aligned}
$$

Grammar G_2 has 10 variables (the capitalized grammatical terms written inside brackets); 27 terminals (the standard English alphabet plus a space character); and 18 rules. Strings in $L(G_2)$ include:

```
a boy sees
the boy sees a flower
a girl with a flower likes the boy
```

Each of these strings has a derivation in grammar G_2. The following is a derivation of the first string on this list.