Compilers

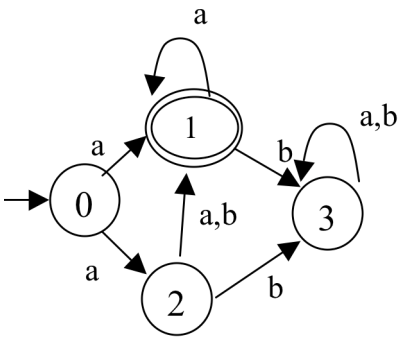
Spring 2008

Solution to Midterm Exam

Problem 1: From NFA to a DFA [30 points]

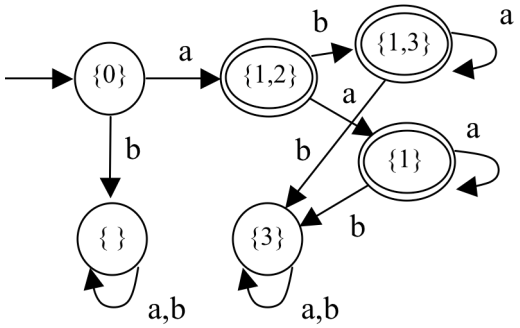
Given the NFA below answer the following questions:

(a) [10] Convert this NFA to a DFA using the closure computation;
(b) [10] Minimize the resulting DFA
(c) [10] Describe the set of strings accepted by the minimal DFA and verify if the strings "a" and "aaa" are in the language accepted bythe DFA.

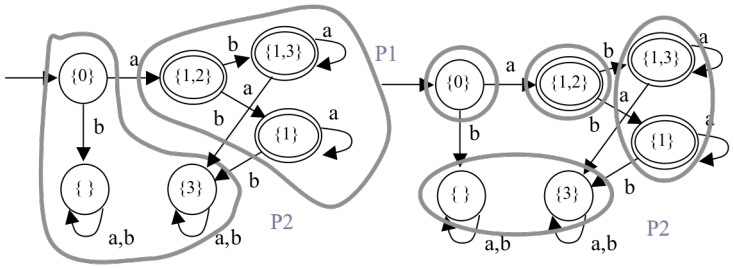


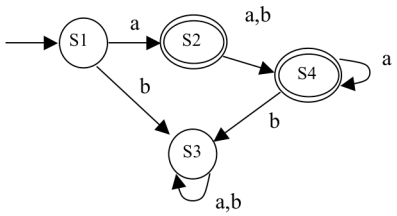
Solution:

(a) Using the subset construction (denoted here by showing the composition of each new state as a set), we obtained the DFA shown below and not yet minimized.



(b) Using the iterative refinement of the various states, we get the sequence of partitions below.





Minimized DFA

(c) This DFA clearly recognizes the language denoted by the RE = a(b?)a* over the alphabet {a, b}.

Problem 2 [50 points]: Table-based LL(1) Predictive Top-Down Parsing

Given the following CFG grammar G = ({SL}, S, {a, "(","),",", P) with P:

S → (L) | a
L → L, S | S

a) Is this grammar suitable to be parsed using the recursive descendent parsing method? Justify and modify the grammar if needed.
b) Compute the FIRST and FOLLOW set of non-terminal sysmbols of the grammar resulting from your answer in a)
c) Construct the corresponding parsing table using the predictive parsing LL method.
d) Show the stack contents, the input and the rules used during parsing for the input string w = (a,a)

Solution:

a) No because it is left-recursive. You can expand L using a production with L as the left-most symbol without consuming any of the input terminal symbols. To eliminate this left recursion we add another non-terminal symbol, L' and productions as follows:

S → (L) | a
L → S L'
L' → , S L' | ε

b) FIRST(S) = { (, a }, FIRST(L) = { (, a } and FIRST(L') = { , , ε}
FOLLOW(L) = {) }, FOLLOW(S) = { , ,) , $ }, FOLLOW(L') = {) }

c) The parsing table is as shown below:

	(	)	a	,	$
S	S→(L)		S → a		
L	L→S L'		L→ S L'		
L'		L'→ε		L'→ , S L'	

d) The stack and input are as shown below using the predictive, table-drven parsing algorithm:

STACK	INPUT	RULE/OUTPUT
$S	(a,a)$	
$) L (	(a,a)$	S → (L)
$) L	a,a)$	
$) L' S	a,a)$	L → S L'
$) L' a	a,a)$	S → a
$) L'	,a)$	
$) L' S ,	,a)$	L' → , S L'
$) L' S	a)$	
$) L' a	a)$	S → a
$) L'	)$	
$)	)$	L' → ε
$	$	

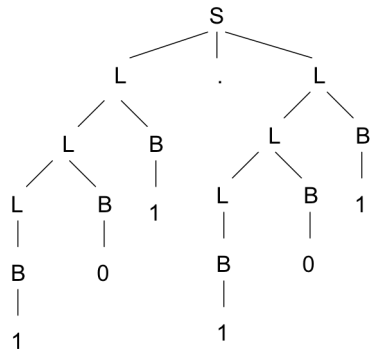
Problem 3: Attributive Grammar and Syntax-Directed Definitions [25 points]

For the grammar below design an L-attributed SDD to compute S.val, the decimal value of an input string in binary. For example the translation of the string `101.101` (whose parse tree is also shonw below) should be the decimal number 5.625. Hint: Use an inherited attribute L.side that tells which side of the decimal point a given bit is on. For all symbols specific their attribute, if they are inherited or synthesized and the various semantic rules. Also show your work for the parse tree example given by indication the values of the various attributes for each grammar symbol.

Grammar

S → L . L | L
L → L B | B
B → 0 | 1

Parse Tree



Solution: One possible solution is to define 4 attributes, 2 inherited and 2 synthesized, respectively, *side* and *pos* and the two synthesized attributes as *depth* and *value*. The *depth* attribute is to allow the RHS of the binary string to compute the length. When computing the final value the value on the RHS of the decimal point is simply shift to the right (i.e., divided by $2^{-(depth+1)}$) of the length of the RHS string of the binary representation. The LHS is done by computing at each instance of L the value of the B symbol by using the value of position. The postion is thus incremented at each stage downwards. The semantic rules for each production are given below along with the annotated parse tree for the input string "101.101".

Productions	Sematic Rules
$S \rightarrow L_1 . L_2$	$L_1.side = 0$; $L_2.side = -1$; $L_1.pos = 0$; $L_2.pos = 0$; $S.val = L_1.val + (L_2.val * 2^{-(L2.depth+1)})$;
$S \rightarrow L$	$L_1.side = 0$; $L_1.pos = 0$; $S.val = L_1.val$;
$L \rightarrow L_1 B$	$L_1.side = L.side$; $L.pos = L_1.pos + 1$; $L_0.depth = L_1.depth + 1$; $B.pos = L.pos$; $L.val = L_1.val + B.val$;
$L \rightarrow B$	$L.depth = L.pos$; $B.pos = L.pos$; $L.val = B.val$;
$B \rightarrow 0$	$B.val = 0$;
$B \rightarrow 1$	$B.val = 1*2^{B.pos}$;