# Compilers

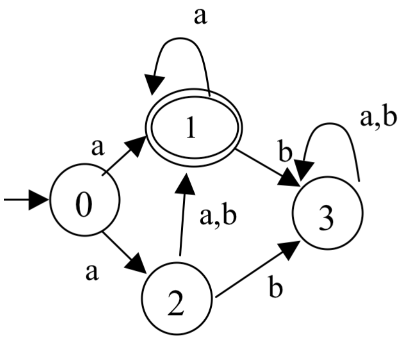
Spring 2009

# Solution to the Midterm Exam

---

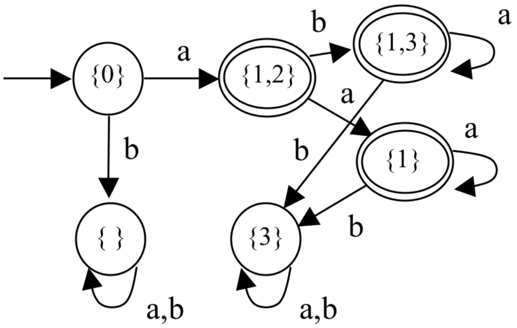
**Problem 1: From NFA to a DFA [20 points]**
Given the NFA below answer the following questions:

(a) [05 points] Convert this NFA to a DFA using the closure computation;
**(b)** [10 points] Minimize the resulting DFA
**(c)** [05 points] Describe the set of strings accepted by the minimal DFA and verify if the strings "a" and "aaa" are in the language accepted by the DFA.

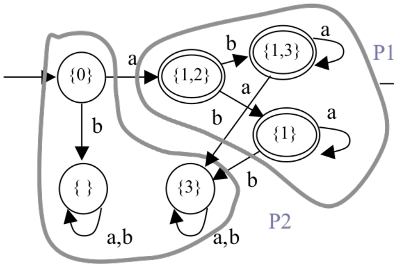


**Answer:**

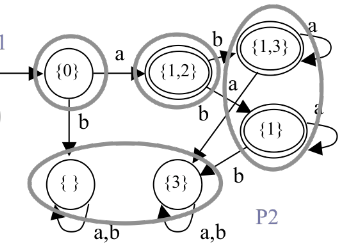
**(a)** Using the subset construction (denoted here by showing the composition of each new state as a set), we obtained the DFA shown below and not yet minimized.



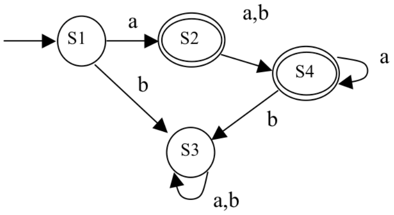
**(b)** Using the iterative refinement of the various states, we get the sequence of partitions below.



refinement final states



refinement on b



Minimized DFA

**(c)** This DFA clearly recognizes the language denoted by the RE = a(b?)a* over the alphabet {a, b} of which the two strings “a” and “aaa” are part of.

## Problem 2: LR Top-Down Parsing [50 points]

Given the following CFG grammar G = ({SAB}, S, {a, b, x), P) with P:

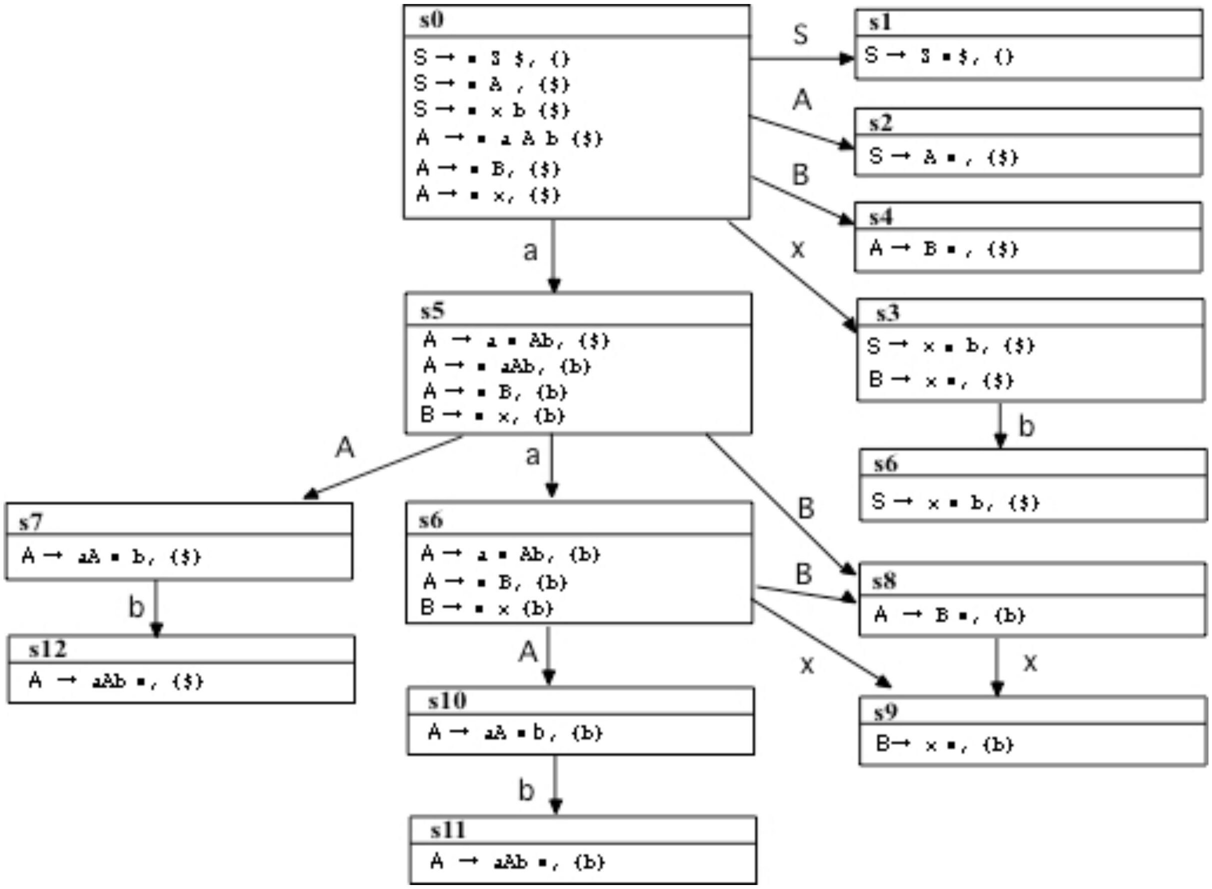
(1) S → A
(2) S → xb
(3) A → aAb
(4) A → B
(5) B → x

For this grammar

a) Compute the set of LR(1) items for this grammar and the corresponding DFA. Do not forget to augment the grammar with the default initial production S' → S$ as the production (0).
b) Construct the corresponding LR parsing table.
c) Show the stack contents, the input and the rules used during parsing for the input string w = axb$

## Answer:

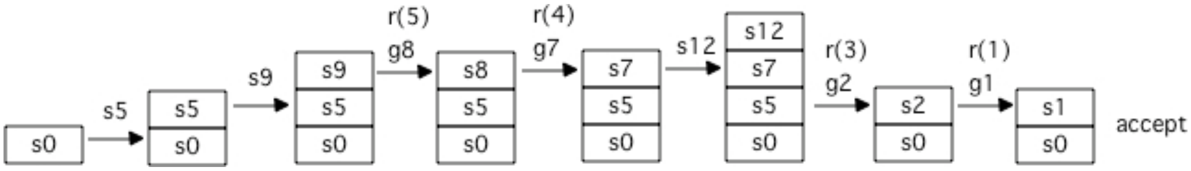
a) The set of LR(1) items and the corresponding DFA are shown below:



b) The parsing table is as shown below:

| State | Terminals | | | | Goto | | |
|---|---|---|---|---|---|---|---|
| | a | b | x | $ | S | A | B |
| 0 | s5 | | s3 | | g1 | g2 | g4 |
| 1 | | | | acc | | | |
| 2 | | | | r(1) | | | |
| 3 | | s6 | | r(5) | | | |
| 4 | | | | r(4) | | | |
| 5 | s6 | | s9 | | | g7 | g8 |
| 6 | | | | r(2) | | | |
| 7 | | s12 | | | | | |
| 8 | | r(4) | | | | | |
| 9 | | r(5) | | | | | |
| 10 | | s11 | | | | g10 | |
| 11 | | r(3) | | | | | |
| 12 | | | | r(3) | | | |

c) The stack contents for the input string w = axb$.



## Problem 3: Syntax-Directed Translation [30 points]

Consider the following syntax-directed semantic specification over the grammar defined by G = (S, {S, A, Sign}, {‘,’, ‘-‘, ‘+’, ‘n’}, P) with P the set of production and the corresponding semantic rules depicted below. There is a special terminal symbol “n” that is lexically matched by any string of one or more digits and whose value is the numeric value of its decimal representation. For the non-terminal symbols in G we have defined two attributes, *sign* and *value*. The non-terminal A has these two attributes whereas S only has the value attribute and Sign only has the *sign* attribute.

```
S     → A Sign       ||   S.val = A.val; A.sign = Sign.sign; print(A.val);
Sign → +             ||   Sign.sign = 1
Sign → -             ||   Sign.sign = 0
A     → n            ||   A.val = value(n)
A     → $A_1$ , n    ||   $A_1$.sign = A.sign;
                            if(A.sign = 1) then
                              A.val = min ($A_1$.val,value(n));
                            else
                              A.val = max($A_1$.val,value(n));
```

For this Syntax-Directed Definition answer to the following questions:

a) [05 points] Explain the overall operation of this syntax-directed definition and explain which of the attributes, if any, are either synthesized or inherited.

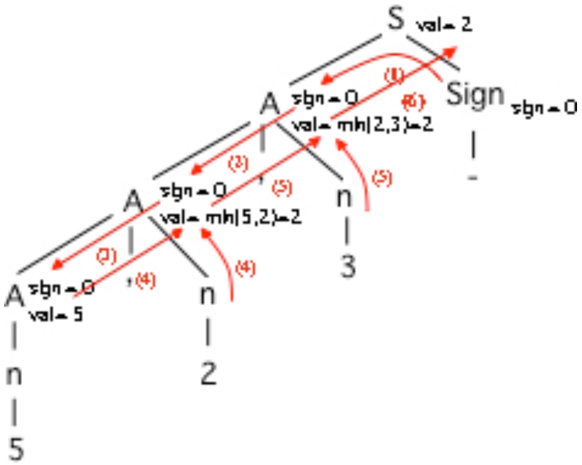
b) [10 points] Give an attributed parse tree for the source string “5,2,3-“ evaluate the attributes in the attributed parse tree depicting the order in which the attributes are evaluated.

c) [15 points] Suggest a modified grammar and possibly attributes using only synthesized attributes.

### Answers:

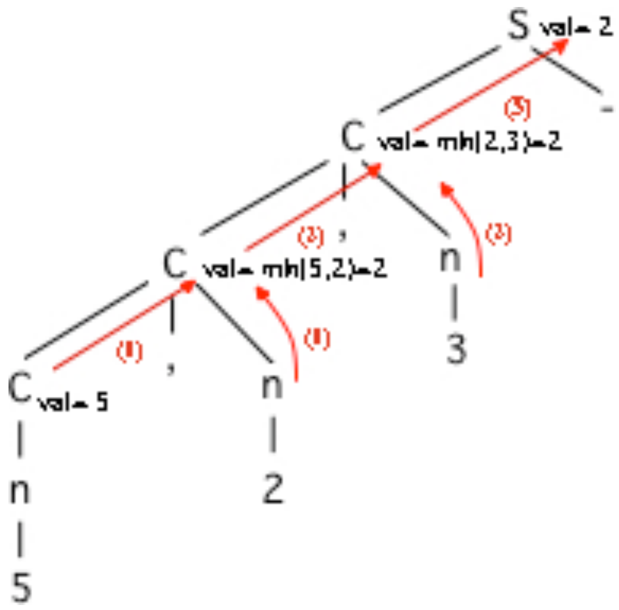
a) This syntax-directed definition computes the minimum or maximum of a sequence of integers depending on the suffix ‘-‘ or ‘+’ respectively. As to the attributes, clearly the sign attribute for the Sign non-terminal is synthesized. The *val* attribute is also synthesized but the *sign* attribute for A is inherited as it flows from “right-to-left” (thus from top to bottom in the parse tree) for a production of A.

b) The parse tree and the corresponding attribute values an evaluation order is as shown below.



c) The idea is to decouple at the top the two situations, computing a minimum or maximum based on the value of the Sign non-terminal. To accomplish this we have two very similar non-terminal symbols B and C as shown below, which replace the A terminal symbol. Note also the presence of two productions for S for the two computation functions, min and max and the absence of the symbol Sign. All attributes are now synthesized as shown in the figure below (right hand side).

| | | | |
|---|---|---|---|
| S | → B + | ll | S.val = B.val; print(B.val) |
| S | → C - | ll | S.val = C.val; print(C.val) |
| B | → n | ll | B.val = value(n) |
| B | → $B_1$ , n | ll | B.val = max($B_1$.val,value(n)) |
| C | → n | ll | C.val = value(n) |
| C | → $C_1$ , n | ll | C.val = min($C_1$.val,value(n)) |


S val= 2
(3)
-
C val= min{2,3}=2
(2)
,
n
(2)
3
C val= min{5,2}=2
(1)
,
n
(1)
C val= 5
n
2
5