Compilers (L.EIC026)

Spring 2022

Faculdade de Engenharia da Universidade do Porto
Departamento de Engenharia Informática

Midterm Exam (Make-Up Grade Exam)
With Solutions

July 06, 2022 from 09:00 to 10:30

Please label all pages you turn in with your name and student number.

Name: _____ Number: _____

Grade:

Problem 1 [10 points]:
Problem 2 [20 points]:
Problem 3 [20 points]:

Total:

INSTRUCTIONS:

1. This is an open-book exam but you may bring one A4 sheet with notes either typed or handwritten for your own personal use during the exam.

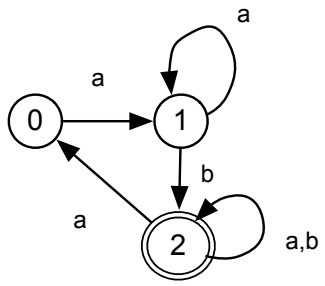
2. Please identify all the pages where you have answers that you wish to be graded. Also make sure to label each of the additional pages with the problem you are answering.

3. Use black or blue ink. No pencil answers allowed.

4. Staple or bind additional answer sheets together with this document to avoid being misplaced or worse, lost. Make sure this cover page is stapled at the front.

5. Please avoid laconic answers so that we can understand you understood the concepts being asked.

Problem 1 [10 points] Consider the Non-Deterministic Finite Automaton depicted below with 3 states and with transitions over the alphabet Σ = {a, b}. For this automaton answer the following questions:



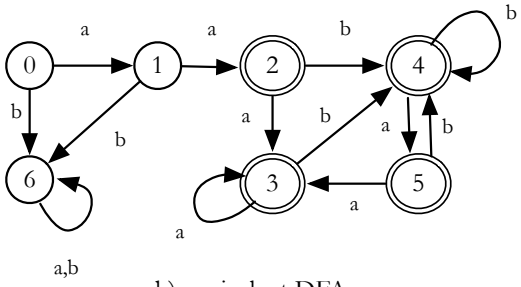
a) Why is this an NFA?
b) Convert the NFA to an equivalent DFA using the subset construction described in class.
c) Minimize the resulting DFA.

Answers:

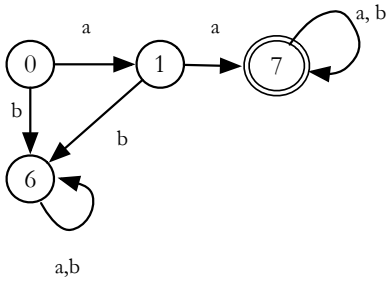
a) Although there are no ε-transitions, for state 2 there are two out-going transitions labelled "a" making this FA non-deterministic.
b) The equivalent DFA below was obtained using the subset construction. Also shown is the mapping between NFA and DFA states.
c) See minimal DFA below.

DFA	NFA
0	{ 0 }
1	{ 1 }
2	{ 1, 2 }
3	{ 0, 1, 2 }
4	{ 2 }
5	{ 0, 2 }
6	{ }

a) NFA to DFA state mapping



b) equivalent DFA



c) minimal DFA

Problem 2 [20 points] Consider the CFG grammar G with the productions listed below,

$S \rightarrow a\ A\ B$
$A \rightarrow b$
$A \rightarrow \varepsilon$
$B \rightarrow b$

where 'a', 'b' are terminals, and 'S' is the start symbol. For this grammar, determine the following:

a) Does this G does have the LL(1) property, i.e., it it not parseable using the LL parsing algorithm described in class with a single lookahead token. Why or why not?
b) Build the LL parsing table for the by explicitly computing the FIRST, FOLLOW and NULLABLE sets for the various non-terminals and productions and confirm that G is either LL(1) or not.

Answers:

a) Although the productions for every non-terminal have disjoint FIRST sets, this grammar still doesn't satisfy the LL(1) condition. Since A is nullable, we must consider the FOLLOW set as well when checking for conflicts -- and because c is in the FIRST set for one production of A and the FOLLOW set for another, nullable production, this grammar isn't LL(1).

b) The following are the FIRST and FOLLOW sets:

FIRST(S) = {a}	FOLLOW(S) = {$}	Nullable = {A}
FIRST(A) = {b}	FOLLOW(A) = {b}	
FIRST(B) = {b}	FOLLOW(B) = {$}	

The LL(1) table, as shown below, would yield a conflict as for the input c there is ambiguity in selecting between the two A productions.

	a	b
S	$S \rightarrow a\ A\ B$	
A		**$A \rightarrow b$ / $A \rightarrow \varepsilon$**
B		$B \rightarrow b$

Problem 3 [20 points] Consider the simple arithmetic expression grammar (which is ambiguous) described in class and depicted below where id and num stand for terminal symbols denoting respectively identifiers and numbers and * and + stand for the common addition and multiplication operators.

$E \rightarrow E + E \mid E * E \mid (E) \mid \text{id} \mid \text{num}$

Given this grammar, where the parenthesis (terminal symbols) are used to enforce priority of arithmetic operators, you are asked to devised a syntax-direct translation (SDT) scheme that detects the priority violations after parsing assuming that the evaluation of the arithmetic value of an expression is done by carrying out an in-order depth-first traversal of the parse tree. To make this example clearer, start by drawing the parse tree that results from the left-most derivation of the following sequence of input tokens where the subscripts are used to differentiate between multiple instances of a symbol.

$(\text{id}_1 + \text{num}_1) + \text{id}_2 * \text{id}_3$

Your SDT scheme should only use synthesized attributes and detect when an un-parenthetic arithmetic expression is giving priority to the + operator with respect to the * operator. In these cases, it should simply report a violation (through a synthesized attribute) propagated all the way up to the top of the parse tree. The use of parenthesis voids possible priority violation in the enclosing subtree as it explicitly enforces the order of application of the arithmetic operators.

Answer:
As discussed in class, this grammar is indeed ambiguous as there are multiple parse trees for the same input. In this case we can detect arithmetic "violations" by defining two Boolean synthesized attributes, namely, "violation" and "plus". The parenthesis production effectively nullifies the plus attribute as it enforces the priorities. At each terminal symbol these attributes are initialized to "false".

$E_1 \rightarrow E_2 + E_3$	{ E_1.violation = E_2.violation OR E_3.violation;
	E_1.plus = true;
	if (E_2.plus = true) OR (E_3.plus = true) E_1.violation = true;
	}

$E_1 \rightarrow E_2 * E_2$	{ E_1.violation = E_2.violation OR E_3.violation;
	E_1.plus = false;
	if (E_2.plus = true) OR (E_3.plus = true) E_1.violation = true;
	}

$E_1 \rightarrow (E_2)$	{ E_1.plus = false; E_1.violation = E_2.violation; }
$E \rightarrow$ id	{ E.plus = false; E.violation = false; }
$E \rightarrow$ num	{ E.plus = false; E.violation = false; }

The figure below, using the input provided illustrates the use of this SDT and the corresponding attributes values.