University of Pretoria ■ EBIT Faculty ■ Department of Computer Science ■ Academic Year 2023

COS341 Compiler Construction ■ Semester Test

STUDENT	RESULT
First [redacted] [redacted]	

This questio[redacted]st be filled in and returned together with the answer-booklet.

Course Lecturer: Prof. ***S.Gruner***
Quality Control: Dr. ***L.Marshall***

INSTRUCTIONS:
Read each question carefully before writing your answers *neatly* into the answer-booklet provided. Please **avoid** (as far as possible) to fill the answer-booklet with irrelevant rough-scribbles: instead, use your own rough-scribble pad for such purposes.
Wait until the invigilators give the start-signal before you begin to work.

FORBIDDEN:

- Any kind of *academic dishonesty* (as defined by the rule book of the University of Pretoria);
- Any *electronic or computational devices* such as portable computers, "smart" cell-phones, "smart" wrist-watches, e-book display devices (e.g.: "Kindle"), and the like;
- The *textbook as a whole*;
- *Large-quantity bulk-prints* of 3rd-party digital materials which you have not authored, also including: the E-book version of our textbook, the PDF lecture slides, any internet wikipedia web-pages, any scientific papers from GoogleScholar, etc.;
- *Direct prints of any JPG/GIF/photos* taken with cameras from the lecturer's chalk-board.
- Writing with *pencil that can be erased* with a rubber-gum.

ALLOWED:

- *Student's <u>own</u> self-made* (self-authored) crib-notes in unlimited quanity (typed or written);
- *Manually re-written scribbles* of the JPG/GIF/photos which were taken with cameras from the lecturer's chalk-board;
- *Manually re-written* sections or paragraphs of the Textbook which are difficult to remember (including mathematical formulae with cumbersome symbols, important definitions, important explanations and illustrative examples, and the like);
- *Manually re-written* sections or paragraphs from the PDF Lecture Slides (again with special focus on anything that is difficult to remember, such as cumbersome mathematical formulae, and the like).

Time Advice (applicable to "regular" students without time concession letter)	MARKING
Question **A**: *30 minutes*	[12 Points]
Question **B**: *20 minutes*	[7 Points]
Question **C**: *20 minutes*	[8 Points]
Question **D**: *20 minutes*	[8 Points]
Total Time: *90 minutes*	**[35 Points]**

Question A.

SCENARIO: Two undergraduate students, Natasha Naidoo and her study-buddy Vanessa Venter, have constructed the following two context-free grammars for describing Branching-Programs (with *PROG* as the start *Non-Terminal*). Now the students are quarreling which grammar is "better":

Natasha's CFG for Branching Programs	**Vanessa's CFG for Branching Programs**
1. *PROG* → *INSTR* ; *SEQ*	1. *PROG* → *INSTR* ; *PROG*
2. *SEQ* → *PROG*	2. *PROG* →
3. *SEQ* →	3. *INSTR* → command
4. *INSTR* → command	4. *INSTR* → *BRANCH*
5. *INSTR* → *BRANCH*	5. *BRANCH* → if(bool) { *PROG* } *ELSE*
6. *BRANCH* → if (bool) { *PROG* }	6. *ELSE* → else { *PROG* }
7. *BRANCH* → if (bool) { *SEQ* } else { *PROG* }	7. *ELSE* →

The *languages* produced by these two grammars are obviously very similar, though *not the same:* Vanessa allows for programs to be completely empty, whereas Natasha insists that programs must contain at least one instruction. Whilst this difference is perhaps merely "a matter of taste", the two students also want to find out whether any of their two grammars is "easier" for Parser-construction.

A.1
[3 Points]
Analyse by way of the Look-Ahead-Set method (and the additional special rule *S* → *PROG* $) *whether* **Natasha**'s grammar is in ***LL1***.

A.2
[7 Points]
Analyse by way of the Look-Ahead-Set method (and the additional special rule *S* → *PROG* $) *whether* **Vanessa**'s grammar is in ***LL1***.

For the next two sub-questions:
Given is now the *sentence*

command ; if(bool) { } else { command ; } ;

A.3
[1 Point]
Draw the complete (concrete) syntax tree of the given *sentence* w.r.t. **Natasha**'s grammar.

A.4
[1 Point]
Draw the complete (concrete) syntax tree of the given *sentence* w.r.t. **Vanessa**'s grammar.

--------------------------------*--------------------------------

Question B.

[7 Points]
SCENARIO: Mbali Motsepe (another one of Natasha's study buddies) now wants to use **Natasha's** grammar (from Question **A**) for the construction of an *SLR* parser, but she is not exactly sure about how to begin. *Construct* the *Control-NFA* (*with* **ε** transitions) for Mbali from Natasha's grammar.

Turn the page →

In the meantime, student Vanessa Venter has designed the following CFG for *Looping-Programs*:

1. *PROG*	→	*INSTR* ; *PROG*
2. *PROG*	→	
3. *INSTR*	→	variable := *EXPR*
4. *INSTR*	→	while (*BOOL*) { *PROG* }
5. *EXPR*	→	variable | number | true | false
6. *EXPR*	→	numOp(*EXPR* , *EXPR*)
7. *EXPR*	→	boolOp(*EXPR* , *EXPR*)
8. *BOOL*	→	*EXPR*

Question C.
[8 Points]
Help Vanessa with the *semantic attribution* of her grammar for the purpose of ***Type Analysis***, such that 1 suitable Type Analysis Rule is "attached" to each 1 of the given grammar's 8 syntactic rules. The semantic rules must be presented in a "functional" style (similar to the style shown in textbook) and must also include the printing of *error messages* for any Looping-Program which (albeit correct in its syntax) happens to be found to be semantically inconsistent.

Question D.
[8 Points]
Help Vanessa with the *semantic attribution* of her grammar for the purpose of ***Intermediate Code Generation***, such that 1 suitable Code Generator Rule is "attached" to each 1 of her grammar's 8 syntactic rules. The semantic rules must be presented in a "functional" style (similar to the style in our textbook).

Additional Advice:

- For the 2nd Syntax Rule of the grammar given above, you may assume that the intermediate language also offers a **NOP** command which means "**no operation**" (*do nothing*).
- For grammar lines 5-8 you must define *TransExp*. For grammar lines 1-4 you must define *TransStat*. In grammar lines 3-4 the *TransStat* function must *also use* help from *TransExp*.

- End of Paper: There are no further questions -

AD DESTINATUM PERSEQUOR

Follow (SEQ ⊆
F   F(B) ⊆ F

A.1) Rule	Nullable	First	Follow
S	~~×~~ NO	First(PROG)	
PROG	No	First(INSTR) = {Command, if}	Follow(SEQ) ∪ {}} ∪ {$} = {}, $}
SEQ	Yes	First(PROG) = {command, if}	Follow(PROG) ∪ {}} = {}, $}
INSTR	No	First(BRANCH) ∪ {command}	{;} ✓
BRANCH	~~Ye~~ No	{if}	Follow (INSTR) = {;}

R0) LA(S → PROG$) = {Command, if}

R1) LA (PROG → INSTR; SEQ) = {Command, if}

R2) LA (SEQ → PROG) = {command, if}

R3) LA (SEQ → ε) = {}, $}

R4) LA(INSTR → command) = {command}

R5) LA(INSTR → BRANCH = {if}

R6) LA(BRANCH → if(bool){PROG}) = {if}

R7) LA (BRANCH → if(bool) {SEQ} else {PROG} = {if}

Since the LA sets of ~~B~~ rule 6 and rule 7 are not disjoint and both use the same BRANCH non-terminal on the LHS, this grammar is not LL1 ✓

3

A2) Nullable(PROG) = Nullable(INSTR ; PROG) ∨ Nullable(ε)

= Nullable(INSTR ; PROG) ∨ True

= True

Nullable(INSTR) = Nullable(command) ∨ Nullable(BRANCH)

= False ∨ False

= False

Nullable(BRANCH) = Nullable(if(bool) {PROG} ELSE)

= False

Nullable(ELSE) = Nullable(else {PROG}) ∨ Nullable(ε)

= True

First(PROG) = First(INSTR) = {command, if}

First(INSTR) = First(command) ∪ First(BRANCH)

= {command, if}

First(BRANCH) = First(if(bool) {PROG} ELSE) = {if}

First(ELSE) = First(else {PROG}) = {else}

Introduce rule 0 S → PROG $

✓

Rule	Constraints
0	{$} ⊆ Follow(PROG)
1	{;} ⊆ Follow (INSTR)
2	
3	
4	Follow(INSTR) ⊆ Follow(BRANCH)
5	{}} ⊆ Follow(PROG) & Follow(BRANCH) ⊆ Follow(ELSE)

Rule	Constraints
6	$\{\$\} \subset \text{Follow}(PROG)$
7	

Rule NT	Follow set iteration 1	Iteration 2
~~$~~ PROG	$\{\$, \cancel{\$}, \}\}$	$\{\$, \}\}$
INSTR	$\{;\}$	$\{;\}$
BRANCH		$\{;\}$
ELSE		$\{;\}$

$LA(S \rightarrow PROG\,\$) = \{\text{commad}, \text{if}, \$, \}\}$

$LA(PROG\cancel{\}} \rightarrow INSTR\,;\,PROG) = \{\text{command}, \text{if}\}$

$LA(PROG \rightarrow \varepsilon) = \{\$, \}\}$

$LA(INSTR \rightarrow \text{command}) = \{\text{command}\}$

$LA(INSTR \rightarrow BRANCH) = \{\text{if}\}$

$LA(BRNCH \rightarrow \text{if} \ldots) = \{\text{if}\}$

~~LA~~

$LA(ELSE \rightarrow \text{else}\,\{PROG\}) = \{\underline{\text{else}}\}$

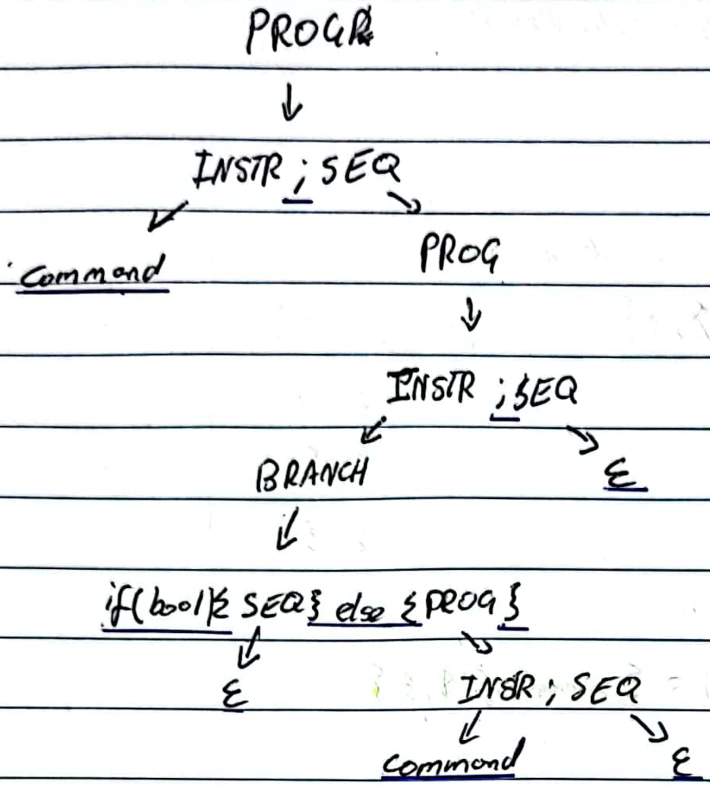
$LA(ELSE \rightarrow \varepsilon) = \{;\}$

~~Notice, that are~~ no

Notice that all LA sets for rules with the same Non-terminal are disjoint ∴ This grammar ✓ <u>is</u> LL1

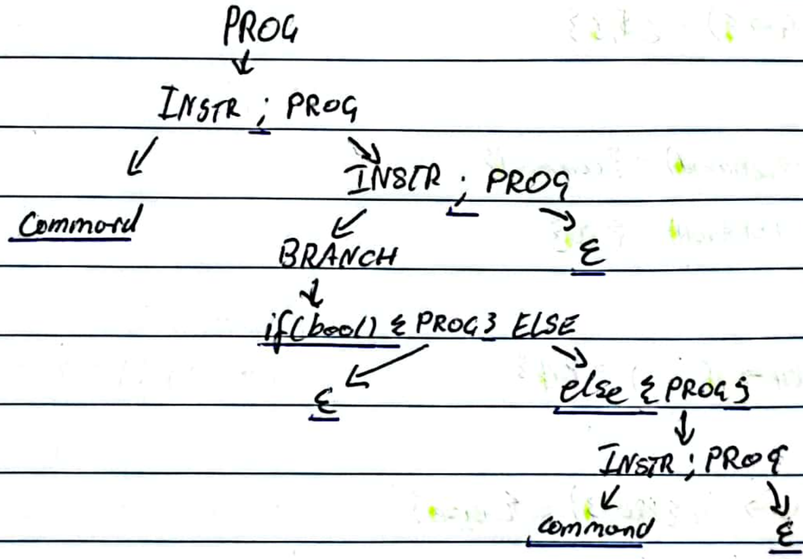
7

A3


PROG
INSTR ; SEQ
Command
PROG
INSTR ; SEQ
BRANCH
ε
if(bool){ SEQ} else {PROG}
ε
INSR ; SEQ
command
ε

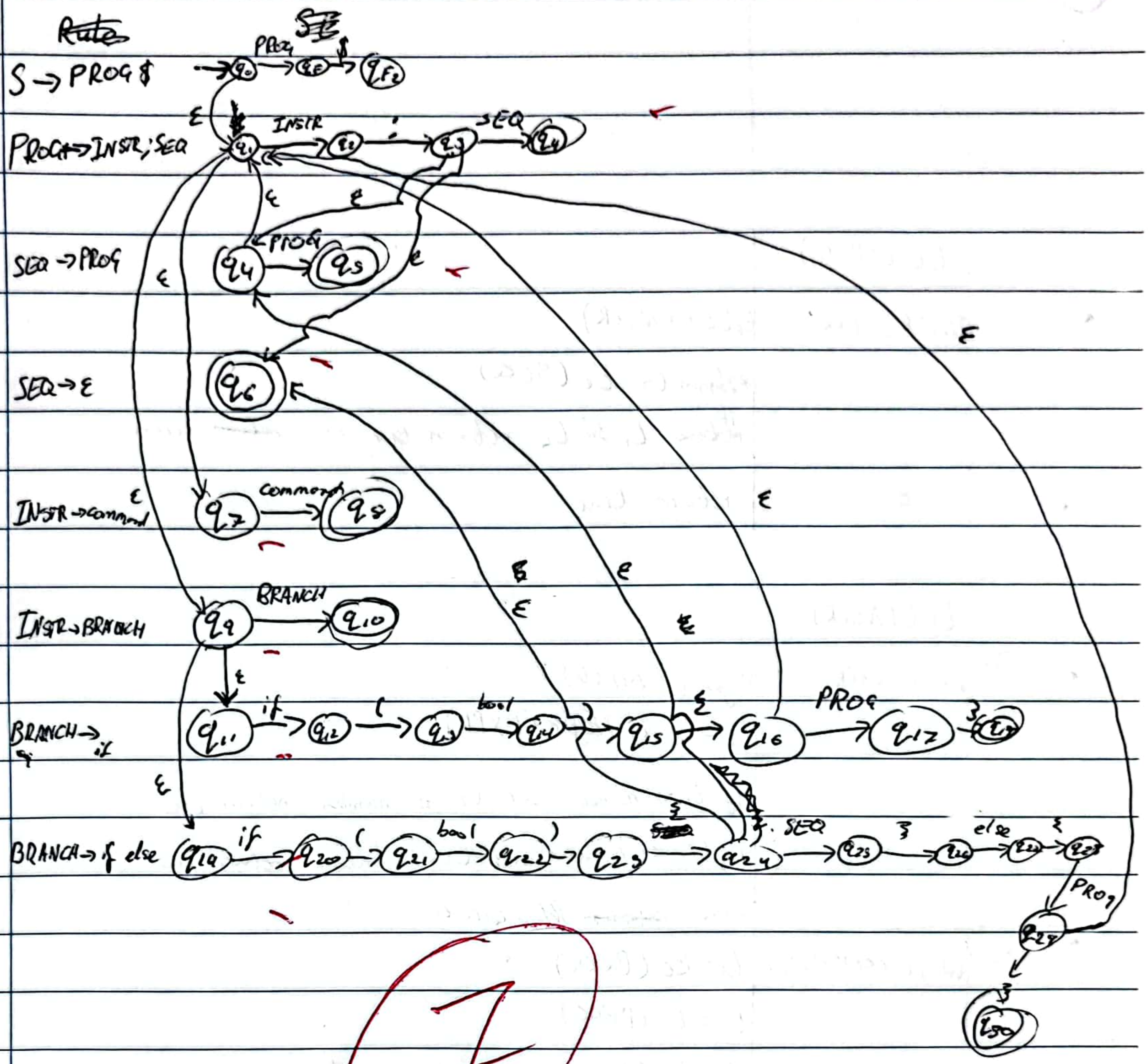
1

A4


PROG
INSTR ; PROG
Command
INSTR ; PROG
BRANCH
ε
if(bool) {PROG} ELSE
ε
else {PROG}
INSTR ; PROG
command
ε

1

B)



~~Rules~~
~~SE~~
S → PROG $
PROG → INSTR ; SEQ
SEQ → PROG
SEQ → ε
INSTR → command
INSTR → BRANCH
BRANCH → if
BRANCH → if else
q0
PROG
qE
$
qF2
ε
q1
INSTR
q2
;
q3
SEQ
q4
PROG
q5
q6
q7
command
q8
q9
BRANCH
q10
q11
if
q12
(
q13
bool
q14
)
q15
q16
PROG
q17
}
q18
q19
if
q20
(
q21
bool
q22
)
q23
q24
SEQ
q25
}
q26
else
q27
{
q28
PROG
q29
}
q30
ε

7

8 c) ~~typeCheck(PROG)~~

	tc (PROG)	
1	INSTR ; SEQ	t_1=tc(INSTR) ~~return~~ t_2 = tc(SEQ) if ~~return~~ t_1 && t_2 return true else ~~return~~ error()
2	ε	return true

	tc(INSTR)	
3	Variable :=EXPR	t_1 typeOf(variable) t_2 = typeof ~~(Expr~~ (EXPR) if t_1 is number and t_2 is number return true If t_1 is boolean and t_2 is boolean return true else ~~return false~~ error()
4	while(BOOL){PROG}	t_1 = tc (BOOL) t_2 = tc (PROG) if t_1 && t_2 return true else error()

	typeof ~~tc~~ (EXPR)	
6	numOp(EXPR$_1$, EXPR$_2$)	t_1 = typeOf(EXPR$_1$) t_2 = typeOf(EXPR$_2$) if t_1 is number and t_2 is number return number else error()
7	boolOp(EXPR$_1$, EXPR$_2$)	t_1 = typeof(EXPR$_1$) t_2 = typeof(EXPR$_2$) if t_1 is boolean and t_2 is boolean return boolean else error()

~~tc~~ typeof(EXPR) contined

5	varieble	~~return vtable, gettype(id)~~ return getType(vtable, id)
5	number	return ~~nu~~ number
5	true	return boolean
5	false	return boolean

	tc(Bool)	
8	EXPR	~~if~~ t_1 = typeOf(EXPR) if t_1 is boolean return true else error()

8

D) Assume vtable global

	Trans ~~PROG~~ Stat(Stat)	
R1	INSTR; PROG	$code_1$ = Trans~~S~~Stat(INSTR) $code_2$ = Trans ~~PROG~~ Stat (PROG) return [$code_1$] @ [$code_2$]
R2	ε	NoP

Same Table

	~~Trans~~ Stat ~~(INSTR)~~	

R 3	Variable := EXPR	place = newVar()
		x = lookup (vtable, getname (id))
		$code_1$ = Trans EXPR (EXPR) place]
		return (code$_1$] @ [x := place]

R4	while (Bool) {PROG}	$lable_1$ = newlable() // check_while
		$lable_2$ = newlable() // while_body
		$lable_3$: newlable() // end_while
		$code_1$ = Trans EXPR (BOOL)
		$code_2$ = Trans ~~PROGR~~ Stat (PROGR)
		return Lable $lable_1$ @ $code_1$ @ Lable $lable_2$ @ $code_2$ @ Lable $lable_3$

See Trans EXPR ON Next page

	Trans EXPR(Expr,	place)
	variable	~~return~~ ~~place = newVar()~~
RS.1	variable	x := lookup (vtable, getname(x1))
		return [place := x]
RS.2	number	v = getvalue (number
		return [place := v]
RS.3	true	v=getvalue(true)
		return [place := v]

RS.4	false	v=getvalue(false)
		return [place := v]

R6	numOp(EXPR1, EXPR2)	$place_1$ = newVar()
		$place_2$ = newVar()
		$code_1$ = TransEXPR ($EXPR_1$, $place_1$)
		$code_2$ = TransEXPR($EXPR_2$, $place_2$)
		op = transop(getopname($\overset{numop}{op}$))
		return (code1] @ code2 @ [place = place1 op place2]
R7	boolop(Expr1, ExPR2)	$place_1$ = newVar()
		$place_2$ = new Var()
		$code_1$ = TransEXPR(EXPR1, $place_1$)
		$code_2$ = Trans EXPR($EXPR_2$, $place_2$)
		op = transop(getopname(boolop))
		return (code1]@ (code2]@ [place = place1 op place2]
R8	Bool	return TransEXPR (BooL)