CS 335 Semester 2022–2023-II: Assignment 3

17th March 2023

Due Your assignment is due by Apr 2 2023 11:59 PM IST.

General Policies

- You should do this assignment ALONE.
- Do not plagiarize or turn in solutions from other sources. You will be PENALIZED if caught.

Submission

- Submission will be through Canvas.
- Upload a zip file named "⟨`roll`⟩`-assign3.zip`". The zipped file should contain a directory ⟨`roll`⟩`-assign3` with the following file:
  - A `PDF` file "⟨`roll`⟩`-assign3.pdf`" that contains the solutions for the pen-paper problems.
- We encourage you to use the LaTeX typesetting system for generating the PDF file.
- You will get up to TWO LATE days to submit your assignment, with a 25% penalty for each day.

Question 1 **[50 marks]**

Consider a computation with three operators: α, β, and γ. The inputs can be of two types: A and B.

Operator	**# Inputs**	**Input Types**	**# Outputs**	**Output Types**
α	1	A	1	A
β	2	B,B	1	B
γ	3	A,A,A or B,B,B	1	B

(a) Propose a context-free grammar (CFG) to generate expressions of the desired form. Given *type*$(x) = A$ and *type*$(y) = B$, $\beta(\gamma(\alpha(x), x, x), y)$ is an example of a type-correct expression. The CFG should allow generating incorrect expressions with wrong types.

(b) Define an SDT based on your grammar from part (a) for type checking expressions. Include the wrong type information in the error message.

(c) Given *type*$(x_i) = A$ *type*$(y_i) = B$, draw the annotated parse tree for the expression:

$$\gamma(\gamma(\alpha(x_1), x_2, \alpha(x_2)), \beta(y_1, y_2), \beta(y_2, y_3))$$

Neither x_1 or y_1 by themselves are valid expressions. An expression must have an operator.

Question 2 **[50 marks]**

A program P is a sequence of two or more statements separated by semicolons. A semicolon is not required for the last statement in P. Each statement assigns the value of an expression E to the variable x. An expression is either the sum of two expressions, a multiplication of two expressions, the constant 1, or the current value of x.

Statements are evaluated in left-to-right order.

- For the i^{th} statement $x = E_i$, the value of references to x inside E_i is the value assigned to x in the previous statement $x = E_{i-1}$.
- For the first statement $x = E_1$, the value of references to x in E_1 is 0.
- The value of a program is the value assigned to x by the last statement.

Answer the following:

(i) Propose a CFG to represent programs generated by the above specification,

(ii) Propose an SDT to compute the value of the program generated by P. Your solution should assign attribute $P.val$ the value of the program generated by P.

(iii) Indicate for each attribute whether it is inherited or synthesized.

Your solution should not use any global state. You can also assume that P cannot be empty.

Question 3 **[50 marks]**

Consider a programming language where reading from a variable *before* it has been assigned is an error. You are required to design an "undefined variable" checker for the language. Do not modify the grammar.

Your SDT should support the following requirements:

- If a statement S contains an expression E, and E references a variable that maybe undefined before reaching S, print the error message "A variable may be undefined". You need not print which variable (or variables) is undefined. It is okay to exit the program after encountering an error.
- If v is defined before a statement S, then v is also defined after S.
- Variable v is defined after the statement $v = E$.
- A variable defined inside an `if` is defined after the `if` when it is defined in BOTH branches.
- In a statement sequence $S1; S2$, variables defined after $S1$ are defined before $S2$.

$$
\begin{aligned}
&\mathit{stmt} \rightarrow \mathit{var} = \mathit{expr} \\
&\mathit{stmt} \rightarrow \mathit{stmt}\ ;\ \mathit{stmt} \\
&\mathit{stmt} \rightarrow \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stmt}\ \textbf{else}\ \mathit{stmt}\ \textbf{fi} \\
&\mathit{expr} \rightarrow \mathit{expr} + \mathit{expr} \\
&\mathit{expr} \rightarrow \mathit{expr} < \mathit{expr} \\
&\mathit{expr} \rightarrow \mathit{var} \\
&\mathit{expr} \rightarrow \textbf{int\_const}
\end{aligned}
$$

Your solution should include the following attributes. You can assume the sets start empty.

`var.name` is a string containing the variable's name. This is defined by the lexer, so you do not need to compute it, you can just use it.

`expr.refd` is the set of variables referenced inside the expression.

`stmt.indefs` is the set of variables defined at the beginning of the statement.

`stmt.outdefs` is the set of variables defined at the end of the statement.

You can invoke common set operations like unions and intersections on the set attributes. You are also allowed to use temporaries for intermediate computations.

Question 4 **[50 marks]**

We have discussed generating 3AC for array accesses using semantic translations. Consider the following extended grammar with semantic translation.

$$
\begin{array}{ll}
S \rightarrow \textbf{id} = E & \{gen(symtop.get(\textbf{id}.lexeme)``="E.addr)\} \\
S \rightarrow L = E & \{gen(L.array.base``["L.addr``]"``="E.addr)\} \\
E \rightarrow E_1 + E_2 & \{E.addr = newTemp(); gen(E.addr``="E_1.addr``+"E_2.addr)\} \\
E \rightarrow E_1 * E_2 & \{E.addr = newTemp(); gen(E.addr``="E_1.addr``*"E_2.addr)\} \\
E \rightarrow \textbf{id} & \{E.addr = symtop.get(\textbf{id}.lexeme)\} \\
E \rightarrow L & \{E.addr = newTemp(); gen(E.addr``="L.array.base``["L.addr``]"))\} \\
L \rightarrow \textbf{id}[E] & \{L.array = symtop.get(\textbf{id}.lexeme); L.type = L.array.type.elem; \\
 & L.addr = newTemp(); gen(L.addr``="E.addr``*"L.type.width)\} \\
L \rightarrow L_1[E] & \{L.array = L_1.array; L.type = L_1.type.elem; t = newTemp(); \\
 & gen(t``="E.addr``*"L.type.width); gen(L.addr``="L_1.addr``+"t);\}
\end{array}
$$

Assume the size of integers to be four bytes, and that the arrays are zero-indexed. Let A, B, and C be integer arrays of dimensions 10 × 5, 5 × 7, and 10 × 7 respectively. Construct an annotated parse tree for the expression $C[i][j] + A[i][k] \times B[k][j]$ and show the 3AC code sequence generated for the expression.