CS335

Compiler Design
Indian Institute of Technology, Kanpur

Naman Singla (200619)

Assignment III

Submission Deadline:
April 02, 2023, 23:55hrs

Question 1

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.

Solution

a) The following CFG can be considered for the desired form:

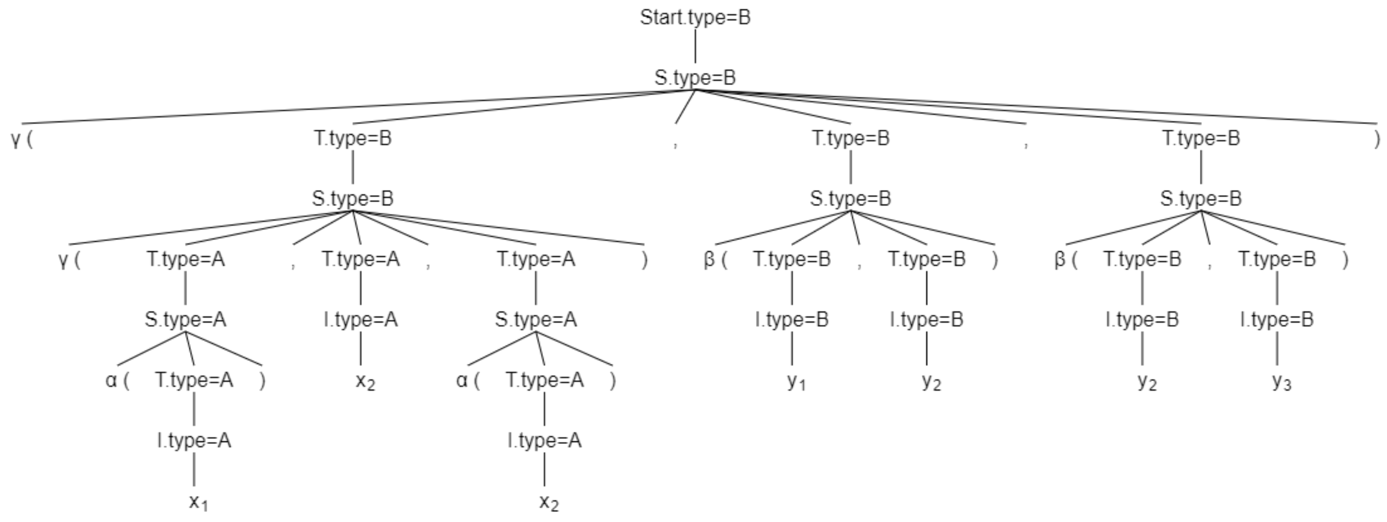
$$\begin{gathered} Start \longrightarrow S \\ S \longrightarrow \alpha(T) \mid \beta(T,T) \mid \gamma(T,T,T) \\ T \longrightarrow S \mid I \\ I \longrightarrow x \mid y \end{gathered}$$

NOTE: The CFG should allow generating incorrect expressions with wrong types.

b) Following is the required SDT :

Production	**Semantic Actions**
$Start \longrightarrow S$	$Start.type = S.type$
$S \longrightarrow \alpha(T)$	if ($T.type! = A$) error("expected A, got %s",$T.type$); else $S.type = A$
$S \longrightarrow \beta(T_1, T_2)$	if ($T_1.type! = B$ || $T_2.type! = B$){ error("expected (B,B), got (%s,%s,)",$T_1.type$,$T_2.type$); } else{$S.type = B$}
$S \longrightarrow \gamma(T_1, T_2, T_3)$	if ($T_1.type! = T_2.type$ || $T_2.type! = T_3.type$ || $T_3.type! = T_1.type$) { error("All must be of same type"); } else{$S.type = B$}
$T \longrightarrow S$	$T.type = S.type$
$T \longrightarrow I$	$T.type = I.type$
$I \longrightarrow x$	$I.type = A$
$I \longrightarrow y$	$I.type = B$

c) Following if the parse tree using the above SDT of given expression :



Question 2

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.

Solution

i) Consider the following CFG for the given problem:

$$\begin{aligned} P &\longrightarrow stms \\ stms &\longrightarrow stms\,;stm \mid stm\,;stm \\ stm &\longrightarrow x = E \\ E &\longrightarrow E + A \mid A \\ A &\longrightarrow A * M \mid M \\ M &\longrightarrow 1 \mid x \end{aligned}$$

NOTE:- Above grammar is left associative and in accordance to the precedence of +,* operators.

ii) We can use following SDT for the given problem : Following is the required SDT :

Production	**Semantic Actions**
$P \longrightarrow stms$	$stms.inh = 0;$ $Start.val = stms.val;$
$stms_1 \longrightarrow stms_2 \,;\, stm$	$stms_2.inh = stms_1.inh;$ $stm.inh = stms_2.val;$ $stms_1.val = stm.val;$
$stms \longrightarrow stm_1 \,;\, stm_2$	$stm_1.inh = stms.inh;$ $stm_2.inh = stm_1.val;$ $stms.val = stm_2.val;$
$stm \longrightarrow x = E$	$E.inh = stm.inh;$ $stm.val = E.val$
$E_1 \longrightarrow E_2 + A$	$E_2.inh = E_1.inh;$ $A.inh = E_1.inh;$ $E_1.val = E_2.val + A.val$
$E \longrightarrow A$	$A.inh = E.inh;$ $E.val = A.val$
$A_1 \longrightarrow A_2 * M$	$A_2.inh = A_1.inh;$ $M.inh = A_1.inh;$ $A_1.val = A_2.val * M.val$
$A \longrightarrow M$	$M.inh = A.inh;$ $A.val = M.val$
$M \longrightarrow x$	$M.val = M.inh$
$M \longrightarrow 1$	$M.val = 1$

iii) Different types of attributes are:

- inherited : $stms.inh$, $stm.inh$, $E.inh$, $A.inh$, $M.inh$
- synthesized : $P.val$, $stms.val$, $stm.val$, $E.val$, $A.val$, $M.val$

Question 3

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 $S2$.

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

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.

Solution

Consider the following SDT:

Production	**Semantic Actions**
$stmt \longrightarrow var = expr$	if (expr.refd != expr.refd ∩ stmt.indefs) error("A variable may be undefined"); else stmt.outdefs = stmt.indefs ∪ {var.name};
$stmt_1 \longrightarrow stmt_2$; $stmt_3$	$stmt_2$.indefs=$stmt_1$.indefs; $stmt_3$.indefs=$stmt_2$.outdefs; $stmt_1$.outdefs=$stmt_3$.outdefs;
$stmt_1 \longrightarrow$ if $expr$ then $stmt_2$ else $stmt_3$ fi	if (expr.refd != expr.refd ∩ $stmt_1$.indefs) error("A variable may be undefined"); else { $stmt_2$.indefs = $stmt_1$.indefs; $stmt_3$.indefs = $stmt_1$.indefs; $stmt_1$.outdefs = $stmt_2$.outdefs ∩ $stmt_3$.outdefs; }
$expr_1 \longrightarrow expr_2 + expr_3$	$expr_1$.refd = $expr_2$.refd ∪ $expr_3$.refd
$expr_1 \longrightarrow expr_2 < expr_3$	$expr_1$.refd = $expr_2$.refd ∪ $expr_3$.refd
$expr \longrightarrow var$	$expr$.refd = { $var.name$ }
$expr \longrightarrow$ int_const	$expr$.refd = { }

Note:- union and intersections are dedonted by symbols, curly-braces denote sets.

Question 4

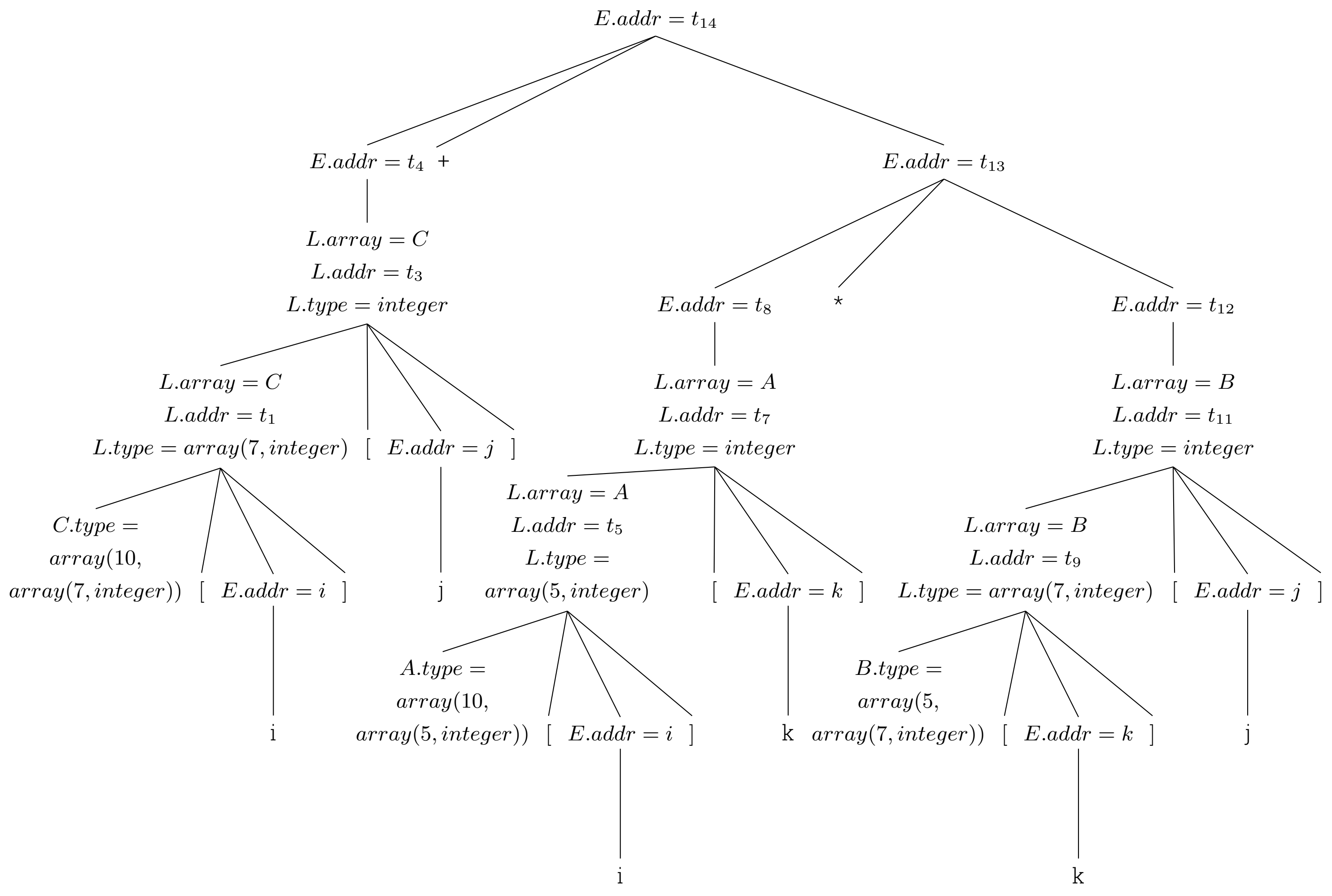
We have discussed generating 3AC for array accesses using semantic translations. Consider the following extended grammar with semantic translation.

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

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.

Solution

Given below is the required parse tree(with attributes) :



From above parse tree 3AC code can be genrated as follows:

- $t_1 = i * 28$
- $t_2 = j * 4$
- $t_3 = t_1 + t_2$
- $t_4 = C[t_3]$
- $t_5 = i * 20$
- $t_6 = k * 4$
- $t_7 = t_5 + t_6$
- $t_8 = A[t_7]$
- $t_9 = k * 28$
- $t_{10} = j * 4$
- $t_{11} = t_9 + t_{10}$
- $t_{12} = B[t_{11}]$
- $t_{13} = t_8 * t_{12}$
- $t_{14} = t_4 + t_{13}$