Assignment 3

Ayush Kumar (170195)

8 March 2020

I pledge on my honor that I have not given or received any unauthorized assistance.

1. (a) To generate expressions for the computation described in the problem, we have the following CFG,

$$S \to \alpha(E) \mid \beta(E, E) \mid \gamma(E, E, E)$$
$$E \to \alpha(E) \mid \beta(E, E) \mid \gamma(E, E, E) \mid ID$$

where ID is the literal token returned by the lexer for an Identifier. Assume that ID has attributes *type* and *name* provided by the lexer, containing the type and the name of the Identifier respectively.

(b) Using the CFG defined in (a) and assuming that $ID.type$ can be one of 'A', 'B', we propose the following SDT (shown in **Table 1**) for generating and checking types of the expressions (Note that **raise Error**(*name*) returns an error object with the specified name and exits with a non-zero code),

		Production Rules	Actions

S	$\rightarrow$	$\alpha(E)$	{**if** ($E.type$ == 'A'):
			    $S.type$ = 'A'
			  else:
			    raise Error("Expected type A but got type " + **str**($E.type$))}

S	$\rightarrow$	$\beta(E_1, E_2)$	{**if** ($E_1.type$ == 'B' **and** $E_2.type$ == 'B'):
			    $S.type$ = 'B'
			  else:
			    raise Error("Expected types (B, B) but got types "+
			str(($E_1.type, E_2.type$)))}

S	$\rightarrow$	$\gamma(E_1, E_2, E_3)$	{**if** (($E_1.type$ == 'A' **and** $E_2.type$ == 'A' **and** $E_3.type$ == 'A') **or**
			      ($E_1.type$ == 'B' **and** $E_2.type$ == 'B' **and** $E_3.type$ == 'B')):
			    $S.type$ = 'B'
			  else:
			    raise Error("Expected types (A, A, A) or (B, B, B) but got types "+
			str(($E_1.type, E_2.type, E_3.type$)))}

E_1	$\rightarrow$	$\alpha(E_2)$	{**if** ($E_2.type$ == 'A'):
			    $E_1.type$ = 'A'
			  else:
			    raise Error("Expected type A but got type " + **str**($E_2.type$))}

E_1	$\rightarrow$	$\beta(E_2, E_3)$	{**if** ($E_2.type$ == 'B' **and** $E_3.type$ == 'B'):
			    $E_1.type$ = 'B'
			  else:
			    raise Error("Expected types (B, B) but got types "+
			str(($B_2.type, B_3.type$)))}

E_1	$\rightarrow$	$\gamma(E_2, E_3, E_4)$	{**if** (($E_2.type$ == 'A' **and** $E_3.type$ == 'A' **and** $E_4.type$ == 'A') **or**
			      ($E_2.type$ == 'B' **and** $E_3.type$ == 'B' **and** $E_4.type$ == 'B')):
			    $E_1.type$ = 'B'
			  else:
			    raise Error("Expected types (A, A, A) or (B, B, B) but got types "+
			str(($E_2.type, E_3.type, E_4.type$)))}

E	$\rightarrow$	ID	{$E.type = ID.type$}

Table 1: SDT for type checking expressions

(c)

$$\gamma(\gamma(\alpha(x1), x2, \alpha(x2)), \beta(y1, y2), \beta(y2, y3))$$

The annotated parse tree of the given expression is as shown in **Figure 1**,

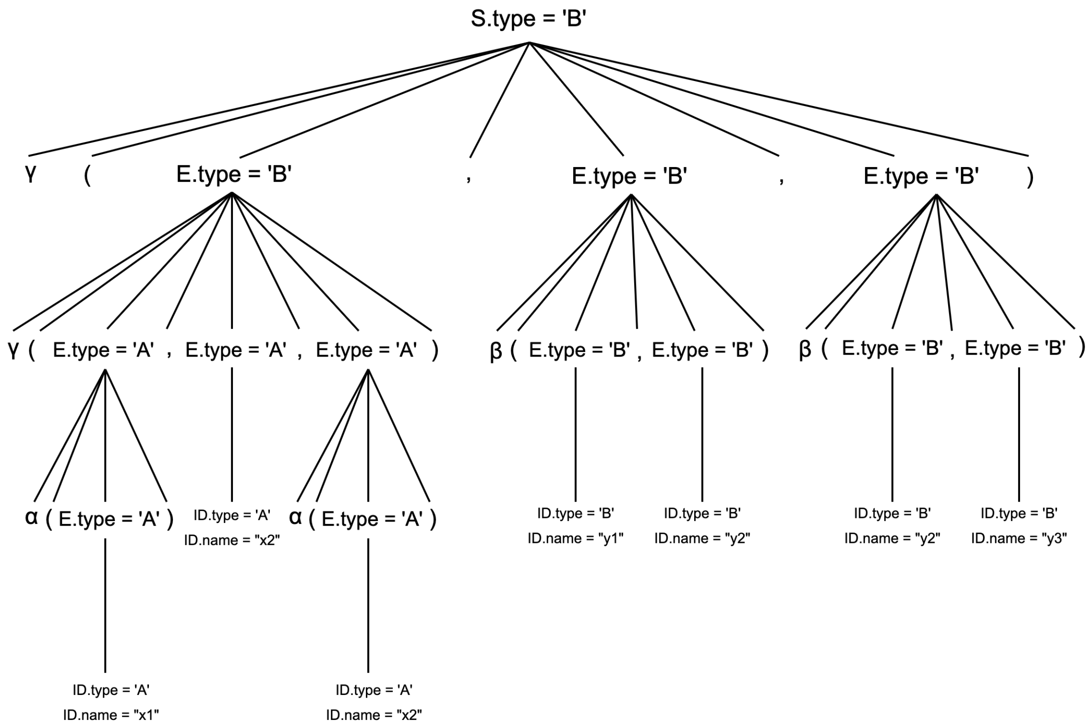

Figure 1: Annotated Parse Tree

2. The given CFG for generating Roman Numerals is as shown,

$$\begin{aligned} rnum &\to thousand\ hundred\ ten\ digit \\ thousand &\to M \mid MM \mid MMM \mid \epsilon \\ hundred &\to smallHundred \mid CD \mid D\ smallHundred \mid CM \\ smallHundred &\to C \mid CC \mid CCC \mid \epsilon \\ ten &\to smallTen \mid XL \mid L\ smallTen \mid XC \\ smallTen &\to X \mid XX \mid XXX \mid \epsilon \\ digit &\to smallDigit \mid IV \mid V\ smallDigit \mid IX \\ smallDigit &\to I \mid II \mid III \mid \epsilon \end{aligned}$$

The proposed SDT scheme for converting Roman Numerals to decimal integers is shown in **Table 2**,

		Production Rules	Actions
$rnum$	$\rightarrow$	$thousand\ hundred\ ten\ digit$	$\{rnum.val = thousand.val + hundred.val + ten.val + digit.val\}$
$thousand$	$\rightarrow$	M	$\{thousand.val = 1000\}$
$thousand$	$\rightarrow$	MM	$\{thousand.val = 2000\}$
$thousand$	$\rightarrow$	MMM	$\{thousand.val = 3000\}$
$thousand$	$\rightarrow$	ϵ	$\{thousand.val = 0\}$
$hundred$	$\rightarrow$	$smallHundred$	$\{hundred.val = smallHundred.val\}$
$hundred$	$\rightarrow$	CD	$\{hundred.val = 400\}$
$hundred$	$\rightarrow$	$D\ smallHundred$	$\{hundred.val = 500 + smallHundred.val\}$
$hundred$	$\rightarrow$	CM	$\{hundred.val = 900\}$
$smallHundred$	$\rightarrow$	C	$\{smallHundred.val = 100\}$
$smallHundred$	$\rightarrow$	CC	$\{smallHundred.val = 200\}$
$smallHundred$	$\rightarrow$	CCC	$\{smallHundred.val = 300\}$
$smallHundred$	$\rightarrow$	ϵ	$\{smallHundred.val = 0\}$
ten	$\rightarrow$	$smallTen$	$\{ten.val = smallTen.val\}$
ten	$\rightarrow$	XL	$\{ten.val = 40\}$
ten	$\rightarrow$	$L\ smallTen$	$\{ten.val = 50 + smallTen.val\}$
ten	$\rightarrow$	XC	$\{ten.val = 90\}$
$smallTen$	$\rightarrow$	X	$\{smallTen.val = 10\}$
$smallTen$	$\rightarrow$	XX	$\{smallTen.val = 20\}$
$smallTen$	$\rightarrow$	XXX	$\{smallTen.val = 30\}$
$smallTen$	$\rightarrow$	ϵ	$\{smallTen.val = 0\}$
$digit$	$\rightarrow$	$smallDigit$	$\{digit.val = smallDigit.val\}$
$digit$	$\rightarrow$	IV	$\{digit.val = 4\}$
$digit$	$\rightarrow$	$V\ smallDigit$	$\{digit.val = 5 + smallDigit.val\}$
$digit$	$\rightarrow$	IX	$\{digit.val = 9\}$
$smallDigit$	$\rightarrow$	I	$\{smallDigit.val = 1\}$
$smallDigit$	$\rightarrow$	II	$\{smallDigit.val = 2\}$
$smallDigit$	$\rightarrow$	III	$\{smallDigit.val = 3\}$
$smallDigit$	$\rightarrow$	ϵ	$\{smallDigit.val = 0\}$

Table 2: SDT for converting Roman Numerals to Decimal Integers

3. (a) The CFG for recognizing the program P as described in the problem is shown below,

$$\begin{aligned} P &\to S \\ S &\to S \ ; \ Stmt \mid Stmt \\ Stmt &\to x = E \\ E &\to E + F \mid F \\ F &\to F * T \mid T \\ T &\to 1 \mid x \end{aligned}$$

(b) An SDT to compute the value of the program P is presented in **Table 3**,

		Production Rules & Actions	
P	$\rightarrow$	S	$\{P.val = S.val\}$
S_1	$\rightarrow$	S_2 ; $\{Stmt.xval = S_2.val\}$ $Stmt$	$\{S_1.val = Stmt.val\}$
S	$\rightarrow$	$\{Stmt.xval = 0\}$ $Stmt$	$\{S.val = Stmt.val\}$
$Stmt$	$\rightarrow$	$x = \{E.xval = Stmt.xval\}$ E	$\{Stmt.val = E.val\}$
E_1	$\rightarrow$	$\{E_2.xval = E_1.xval\}$ E_2 + $\{F.xval = E_1.xval\}$ F	$\{E_1.val = E_2.val + F.val\}$
E	$\rightarrow$	$\{F.xval = E.xval\}$ F	$\{E.val = F.val\}$
F_1	$\rightarrow$	$\{F_2.xval = F_1.xval\}$ F_2 $*$ $\{T.xval = F_1.xval\}$ T	$\{F_1.val = F_2.val * T.val\}$
F	$\rightarrow$	$\{T.xval = F.xval\}$ T	$\{F.val = T.val\}$
T	$\rightarrow$	1	$\{T.val = 1\}$
T	$\rightarrow$	x	$\{T.val = T.xval\}$

Table 3: SDT for computing the value of program P

(c) The type and purpose of each attribute for each non-terminal is indicated in **Table 4**,

Attribute	Type	Purpose
$P.val$	synthesized	stores the value of the program
$S.val$	synthesized	stores the value of x computed by the last statement of S
$Stmt.xval$	inherited	stores the value to be used for all references of x inside $Stmt$
$Stmt.val$	synthesized	stores the new value assigned to x after evaluating $Stmt$
$E.xval$	inherited	stores the value to be used for all references of x inside E
$F.xval$	inherited	stores the value to be used for all references of x inside F
$T.xval$	inherited	stores the value to be used for all references of x inside T
$E.val$	synthesized	stores the value computed by the expression E
$F.val$	synthesized	stores the value computed by the expression F
$E.val$	syntehsized	stores the value computed by the expression T

Table 4: Attribute Types

$Stmt.xval$, $E.xval$, $F.xval$ and $T.xval$ are inherited attributes since $Stmt.xval$ depends on the sibling attribute $S.val$ while $E.xval$, $F.xval$ and $T.xval$ depend on parent attributes $Stmt.xval$, $E.xval$ and $F.xval$ respectively. All the remaining attributes are synthesized since they only depend on the attributes of their children or themselves. Also, non-terminals F and T have been introduced to remove ambiguity and define precedence order in the grammar.

4. We have the following CFG given for the "undefined variable" checker,

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

Using the CFG given above and the attributes $var.name$, $expr.refd$, $stmt.indefs$ and $stmt.outdefs$ we have the following SDT (shown in **Table 5**) for "undefined variable" checker (Note that **raise Error**($name$) returns an error object with the specified name and exits with a non-zero code),

		Production Rules & Actions	
$stmt_1$	$\rightarrow$	$\{stmt_2.indefs = stmt_1.indefs\}$ $stmt_2$; $\{stmt_3.indefs = stmt_2.outdefs\}$ $stmt_3$	$\{stmt_1.outdefs = stmt_3.outdefs\}$

$stmt_1$	$\rightarrow$	**if** $expr$ **then** $\{stmt_2.indefs = stmt_1.indefs\}$ $stmt_2$ **else** $\{stmt_3.indefs = stmt_1.indefs\}$ $stmt_3$ **fi**	$\{$**if** (**not isSubsetOf**($expr.refd$, $stmt_1.indefs$)):
			    raise Error("A variable may be undefined")
			  else:
			    $stmt_1.outdefs$ = **intersection**($stmt_2.outdefs$, $stmt_3.outdefs$)$\}$

$stmt$	$\rightarrow$	$var = expr$	$\{$**if** (**not isSubsetOf**($expr.refd$, $stmt.indefs$)):
			    raise Error("A varible may be undefined")$\}$
			  else:
			    $stmt.outdefs$ = **union**($stmt.indefs$, $\{var.name\}$)$\}$

$expr_1$	$\rightarrow$	$expr_2 + expr_3$	$\{expr_1.refd$ = **union**($expr_2.refd$, $expr_3.refd$)$\}$
$expr_1$	$\rightarrow$	$expr_2 < expr_3$	$\{expr_1.refd$ = **union**($expr_2.refd$, $expr_3.refd$)$\}$
$expr$	$\rightarrow$	var	$\{expr.refd = \{var.name\}\}$
$expr$	$\rightarrow$	**int_const**	$\{expr.refd = \{\}\}$

Table 5: SDT for computing the value of program P

where **union**(A, B), **intersection**(A, B) and **isSubsetOf**(A, B) are three subroutines for performing the following functions,

- **union**(A, B): returns the union of the sets A and B.
- **intersection**(A, B): returns the intersection of the sets A and B.
- **isSubsetOf**(A, B): returns **True** if A is a subset of B, otherwise returns **False**.

References

[1] Alfred Aho, Jeffrey Ullman, Ravi Sethi, Monica S. Lam. 1986. Compilers: Principles, Techniques, and Tools.