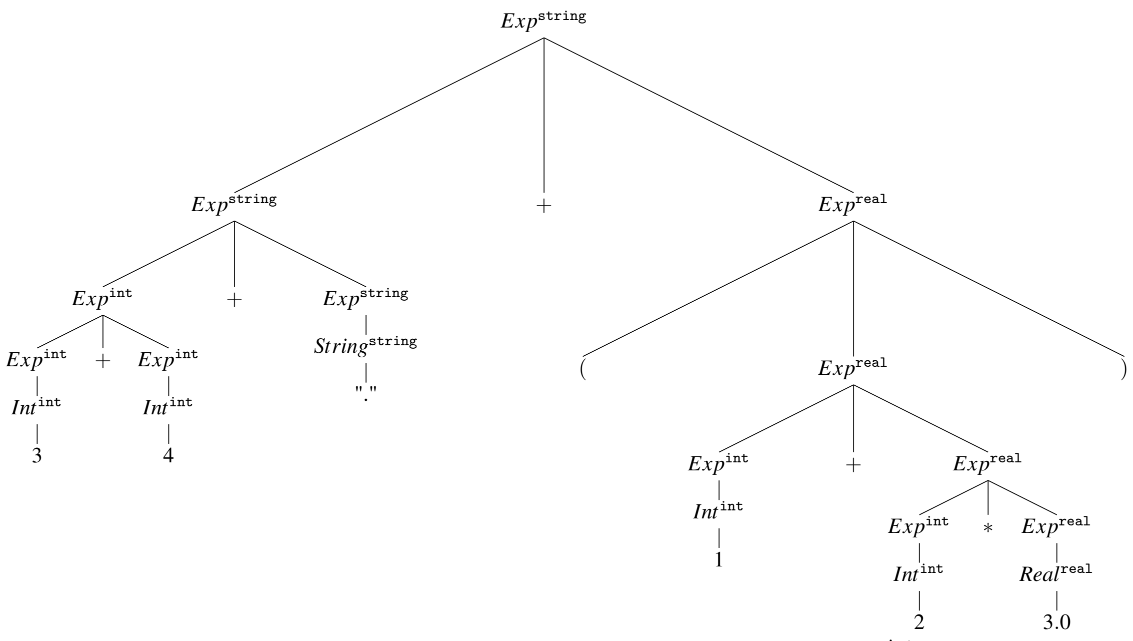
Introduction to Language Theory and Compilation Solutions

Session 9: Semantic Analysis

Solutions

Ex. 1. Write the expression tree decorated with the type of each node.

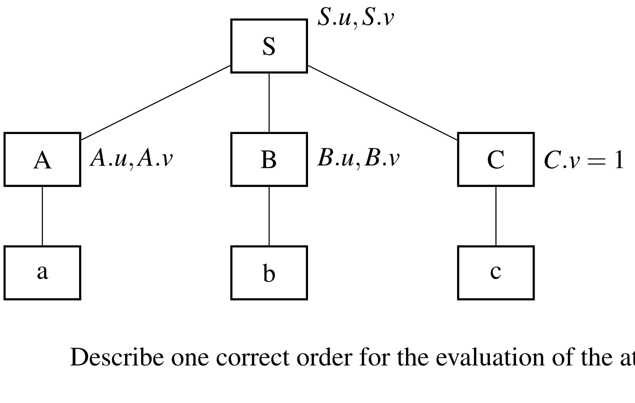


where types are written in typewriter font in superscript (*e.g.* $Int^{\texttt{int}}$)

Ex. 2.

			Grammar rules	Semantic Rules
	<S>	$\rightarrow$	<A> <B> <C>	$B.u = S.u \mid A.u = B.v + C.v \mid S.v = A.v$
	<A>	$\rightarrow$	a	$A.v = 2 * A.u$
	<B>	$\rightarrow$	b	$B.v = B.u$
	<C>	$\rightarrow$	c	$C.v = 1$

1. Draw the parse tree for the input *abc*, (the only string for the language), and show the dependency graph for the associated attributes.



$$\begin{tikzcd}
S.u \arrow[d, "="'] & A.u \arrow[r, "2\times"] & A.v \arrow[r, "="] & S.v \\
B.u \arrow[r, "="] & B.v \arrow[u, "+"] & & C.v(=1) \arrow[ull, "+"']
\end{tikzcd}$$

Describe one correct order for the evaluation of the attributes: $S.u, B.u, B.v, C.v, A.u, A.v, S.v$.

2. Assume that $S.u$ is assigned the value of 3 before starting attribute evaluation. What will be the value of $S.v$ when evaluation has terminated?

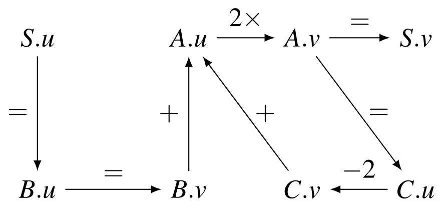
   $S.u = 3 \Rightarrow B.u = S.u = 3 \Rightarrow B.v = B.u = 3 \Rightarrow A.u = B.V + C.v = 3 + 1 = 4 \Rightarrow A.v = 2 \times A.u = 2 \times 4 = 8 \Rightarrow S.v = A.v = 8$

3. Consider now the same grammar with different semantic rules :

		Grammar rules	Semantic Rules
<S>	$\rightarrow$	<A> <B> <C>	$B.u = S.u \mid A.u = B.v + C.v \mid S.v = A.v \mid C.u = A.v$
<A>	$\rightarrow$	a	$A.v = 2 * A.u$
<B>	$\rightarrow$	b	$B.v = B.u$
<C>	$\rightarrow$	c	$C.v = C.u - 2$

   Can you evaluate $S.v$?

   No, there is a circular dependency pattern :



$$\begin{tikzcd}
S.u \arrow[d, "="'] & A.u \arrow[r, "2\times"] & A.v \arrow[r, "="] \arrow[dr, "="] & S.v \\
B.u \arrow[r, "="] & B.v \arrow[u, "+"] & C.v \arrow[ul, "+"] & C.u \arrow[l, "-2"']
\end{tikzcd}$$

Ex. 3. 1. Rewrite the following grammar in order to account for operator precedence and associativity:

<E>	$\rightarrow$	<E> <op> <E> | (<E>) | int
<op>	$\rightarrow$	$+ \mid - \mid * \mid /$

<E>	$\rightarrow$	<E> + <T> | <E> − <T> | <T>
<T>	$\rightarrow$	<T> ∗ <F> | <T> / <F> | <F>
<F>	$\rightarrow$	(<E>) | *int*

2. Associate the rules and attributes necessary to compute the value of an expression E.

<E>	$\rightarrow$	<E>: e_1 + <T>	$\{E.val \leftarrow e_1.val + T.val\}$
	$\rightarrow$	<E>: e_1 − <T>	$\{E.val \leftarrow e_1.val - T.val\}$
	$\rightarrow$	<T>	$\{E.val \leftarrow T.val\}$
<T>	$\rightarrow$	<T>: t_1 ∗ <F>	$\{T.val \leftarrow t_1.val * F.val\}$
	$\rightarrow$	<T>: t_1 / <F>	$\{T.val \leftarrow t_1.val / F.val\}$
	$\rightarrow$	<F>	$\{T.val \leftarrow F.val\}$
<F>	$\rightarrow$	(<E>)	$\{F.val \leftarrow E.val\}$
	$\rightarrow$	*int*	$\{F.val \leftarrow int.val\}$

Finally, remove left recursion from the grammar:

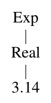
<E>	$\rightarrow$	<T>	$\{E'.h \leftarrow T.val\}$
		<E'>	$\{E.val \leftarrow E'.s\}$
<E'>	$\rightarrow$	+	
		<T>	$\{e_1.h \leftarrow E'.h + T.val\}$
		<E'>: e_1	$\{E'.s \leftarrow e_1.s\}$
	$\rightarrow$	-	
		<T>	$\{e_1.h \leftarrow E'.h - T.val\}$
		<E'>: e_1	$\{E'.s \leftarrow e_1.s\}$
	$\rightarrow$	ε	$\{E'.s \leftarrow E'.h\}$
<T>	$\rightarrow$	<F>	$\{T'.h \leftarrow F.val\}$
		<T'>	$\{T.val \leftarrow T'.s\}$
<T'>	$\rightarrow$	*	
		<F>	$\{t_1.h \leftarrow T'.h * F.val\}$
		<T'>: t_1	$\{T'.s \leftarrow t_1.s\}$
<T'>	$\rightarrow$	/	
		<F>	$\{t_1.h \leftarrow T'.h / F.val\}$
		<T'>: t_1	$\{T'.s \leftarrow t_1.s\}$
	$\rightarrow$	ε	$\{T'.s \leftarrow T'.h\}$
<F>	$\rightarrow$	(<E>)	$\{F.val \leftarrow E.val\}$
	$\rightarrow$	*int*	$\{F.val \leftarrow int.val\}$

Where *V.h* denotes an *inherited* attribute, while *V.s* denotes a *synthesized* attribute. Be careful here: the right-hand sides can carry over several lines. For instance the first rule is the following : $< E > \rightarrow < T > \ \{E'.h \leftarrow T.val\} \ < E' > \ \{E.val \leftarrow E'.s\}$

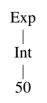
Ex. 4. • Give the table of symbols (ToS)

UID	Name	Context	Initialization	Type
1	PI	Exercise3 class	3.141592653589793	*double*
2	diameter	Exercise3 class	/	*double*
3	args	main function class	parameter	*String[]*
4	perimeter	main function class	50	*double*
5	diameter	main function	ToS(**2**)	*int*

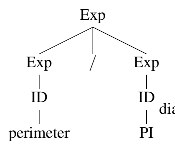
- Give the parse tree of each numerical expression



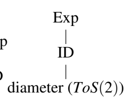
(a) PI



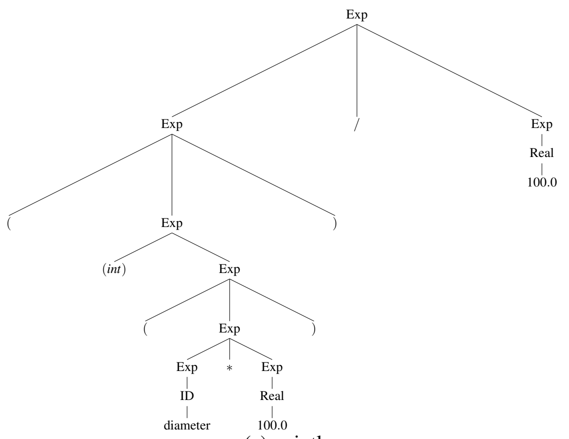
(b) perimeter



(c) diameter



(d) diameter



(e) println

- Annotate the parse trees with changes of the table of symbols

  We add these operations in the root of the trees:

  `PI (a)` $ToS(1) \leftarrow result(Exp)$

  `perimeter (b)` $ToS(4) \leftarrow result(Exp)$

  `diameter (c)` $ToS(2) \leftarrow result(Exp)$

  `diameter (d)` $ToS(5) \leftarrow result(Exp)$

  `println (e) /`

- Report any semantic error.

  Line 7: the global `diameter` is a double and the local `diameter` is an integer. A cast operator is required. This error occurs at the root of the tree (d).