Compilers (CS30003)

Lecture 14

Pralay Mitra

Autumn 2023-24

Syntax Directed Translation

PRODUCTION	SEMANTIC RULE
$E \rightarrow E_1 + T$	$E.code=E_1.code \parallel T.code \parallel$ '+'

Syntax Directed Definition: A CFG together with attributes and rules. Attributes are associated with grammar symbols and rules are associated with productions.

Synthesized Attribute: For a non-terminal *A* at a parse tree node *N* synthesized attribute is defined by a semantic rule associated with the production at *N*.

Inherited Attribute: For a non-terminal *B* at a parse tree node *N* inherited attribute is defined by a semantic rule associated with the production at the parent of *N*.

Syntax Directed Definition

Sl No	PRODUCTION	SEMANTIC RULES
1	L → E $	L.val=E.val
2	E → E_1 + T	E.val=E_1.val + T.val
3	E → T	E.val=T.val
4	T → T_1 * F	T.val=T_1.val × F.val
5	T → F	T.val=F.val
6	F → (E)	F.val=E.val
7	F → digit	F.val=digit.lexval

Draw an annotated parse tree for 3 * 5 + 4 $

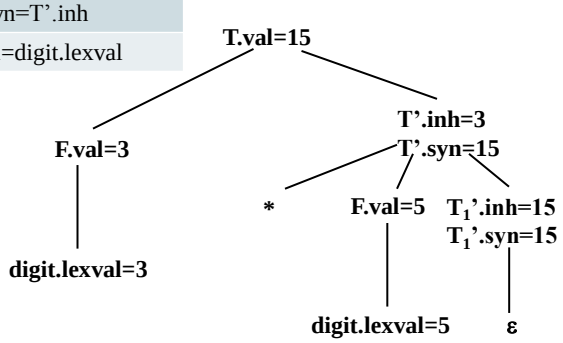
Homework

- Using previous SDD draw the annotated parse tree for

(3+4)*(5+6)$

SDD for top-down parsing

	PRODUCTION	SEMNATIC RULE
1	T → FT’	T’.inh=F.val
		T.val=T’.syn
2	T’ → *FT_1’	T_1’.inh=T’.inh × F.val
		T’.syn=T_1’.syn
3	T’ → ε	T’.syn=T’.inh
4	F → digit	F.val=digit.lexval



Annotated parse tree for 3 * 5

Evaluation order for SDD

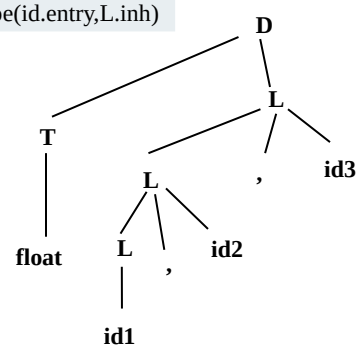
- ***Dependency Graph***
- ***S-Attributed Definitions:*** An SDD is S-attributed if every attribute is synthesized.
- ***L-Attributed Definitions:*** Each attribute must be either (i) Synthesized or (ii) Inherited with the limited rules.

	PRODUCTION	SEMNATIC RULE
1	T → FT’	T’.inh=F.val
		T.val=T’syn
2	T’ → *FT_1’	T_1’.inh=T’.inh × F.val
		T’.syn=T_1’.syn
3	T’ → ε	T’.syn=T’.inh
4	F → digit	F.val=digit.lexval

Semantic Rules with controlled side effects

	PRODUCTION	SEMANTIC RULES
1	D → T L	L.inh=T.type
2	T → int	T.type=integer
3	T → float	T.type=float
4	L → L_1, id	L_1.inh=L.inh
		addtype(id.entry,L.inh)
5	L → id	addtype(id.entry,L.inh)

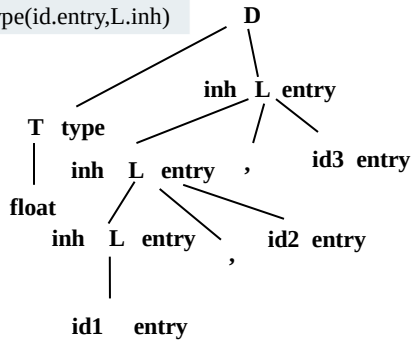
float id1, id2, id3



Semantic Rules with controlled side effects

	PRODUCTION	SEMANTIC RULES
1	D → T L	L.inh=T.type
2	T → int	T.type=integer
3	T → float	T.type=float
4	L → L_1, id	L_1.inh=L.inh
		addtype(id.entry,L.inh)
5	L → id	addtype(id.entry,L.inh)

float id1, id2, id3



AST???

Syntax Directed Translation

	PRODUCTION	SEMANTIC RULES
1	E → E_1 + T	E.node=new Node('+',E_1.node,T.node)
2	E → E_1 – T	E.node=new Node('-',E_1.node,T.node)
3	E → T	E.node=T.node
4	T → (E)	T.node=E.node
5	T → id	T.node= new Leaf(id, id.entry)
6	T → num	T.node= new Leaf(num, num.val)

Homework

Draw the dependency graph and report if there is a cycle.

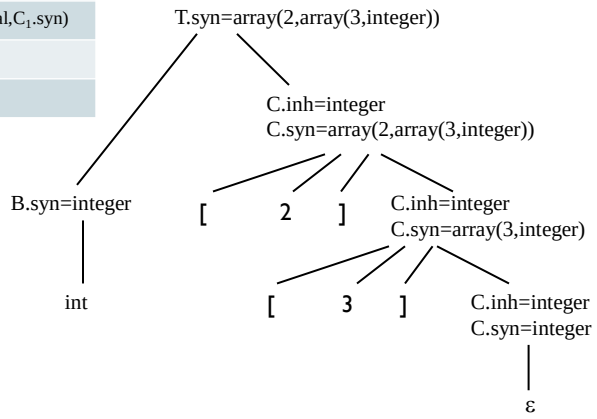
Syntax tree construction for a-4+c

1. p1=new Leaf(id, entry-a);
2. p2=new Leaf(num,4);
3. p3=new Node('-',p1,p2);
4. p4=new Leaf(id,entry-c);
5. p5=new Node('+',p3,p4);

An example

	PRODUCTION	SEMANTIC RULES
1	T → B C	T.syn=C.syn
		C.inh=B.syn
2	B → int	B.syn=integer
3	B → float	B.syn=float
4	C → [num] C_1	C.syn=array(num.val,C_1.syn)
		C_1.inh=C.inh
5	C → ε	C_1.syn=C.inh

int [2][3]



Homework

Give an SDD to translate infix expression with + and × into equivalent expression without redundant parentheses.

((a×(b+(c)))×(d)) → a×(b+c)×d

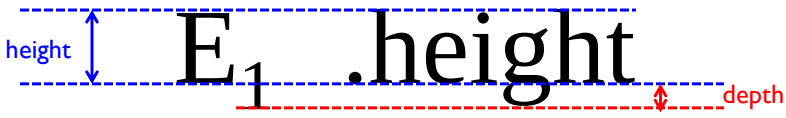
Encode that × is with higher precedence than +.

Syntax Directed Translation

L → E $	{ print (E.val);}
E → E_1 + T	{ E.val=E_1.val+T.val;}
E → T	{ E.val=T.val;}
T → T_1 * F	{ T.val=T_1.val × F.val;}
T → F	{ T.val=F.val;}
F → (E)	{ F.val=E.val;}
F →digit	{ F.val=digit.lexval;}

PRODUCTION	ACTIONS
L → E $	{ print(stack[top-1].val); top=top-1; }
E → E_1 + T	{ stack[top-2].val=stack[top-2].val + stack[top].val; top=top-2; }
E → T	
T → T_1 * F	{ stack[top-2].val=stack[top-2].val × stack[top].val; top=top-2; }
T → F	
F → (E)	{ stack[top-2].val=stack[top-1].val; top=top-2; }
F →digit	

Type setting example



Homework