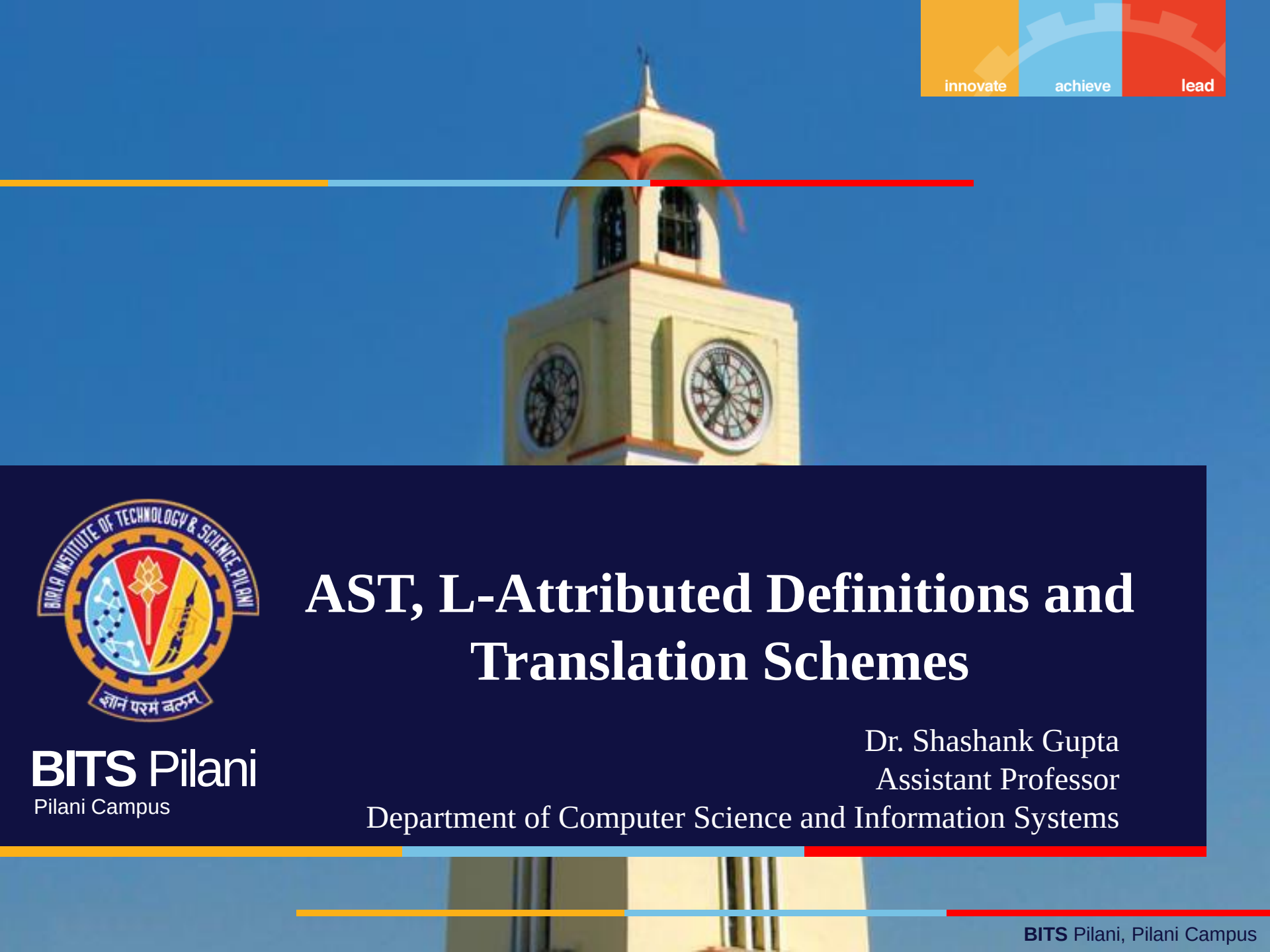


innovate achieve lead

AST, L-Attributed Definitions and Translation Schemes

Dr. Shashank Gupta
Assistant Professor
Department of Computer Science and Information Systems

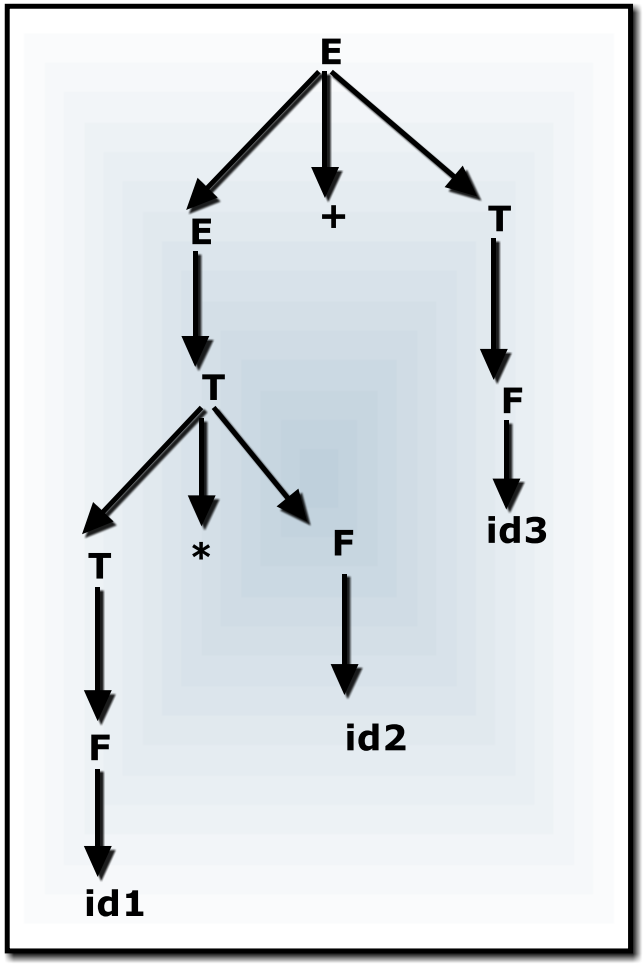
BITS Pilani
Pilani Campus

Abstract Syntax Tree (AST)

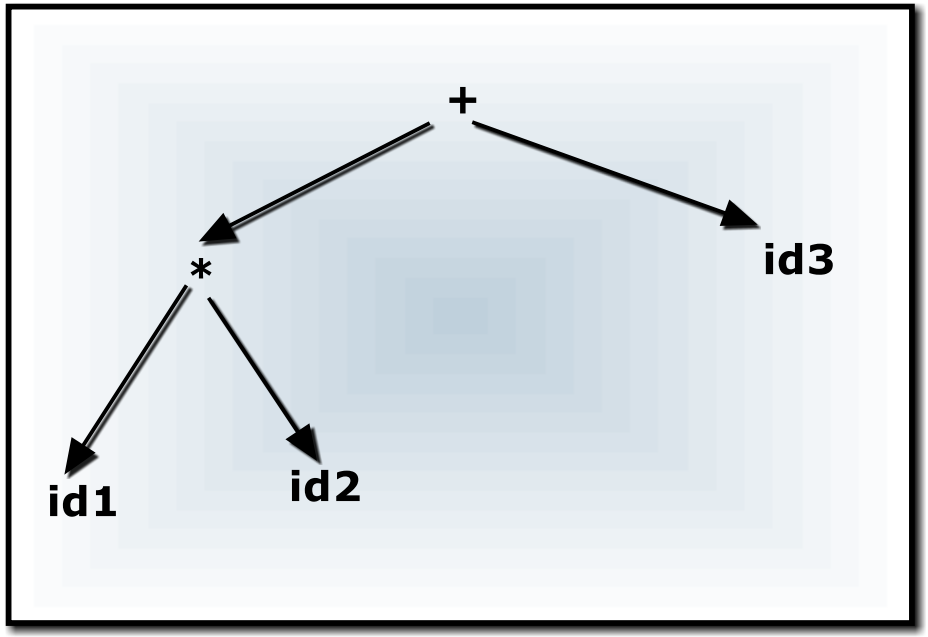
AST is condensed form of parse tree.

Chain of single productions may be collapsed, and operators move to the parent nodes

Abstract Syntax Tree (AST)



SYNTAX TREE



ABSTRACT SYNTAX TREE

Constructing AST for Expression

Each node can be represented as a record

- **Operators**: one field for operator, remaining fields ptrs to operands
- `mknode (op, left, right )`

Identifier: one field with label id and another ptr to symbol table

`mkleaf(id,entry)`

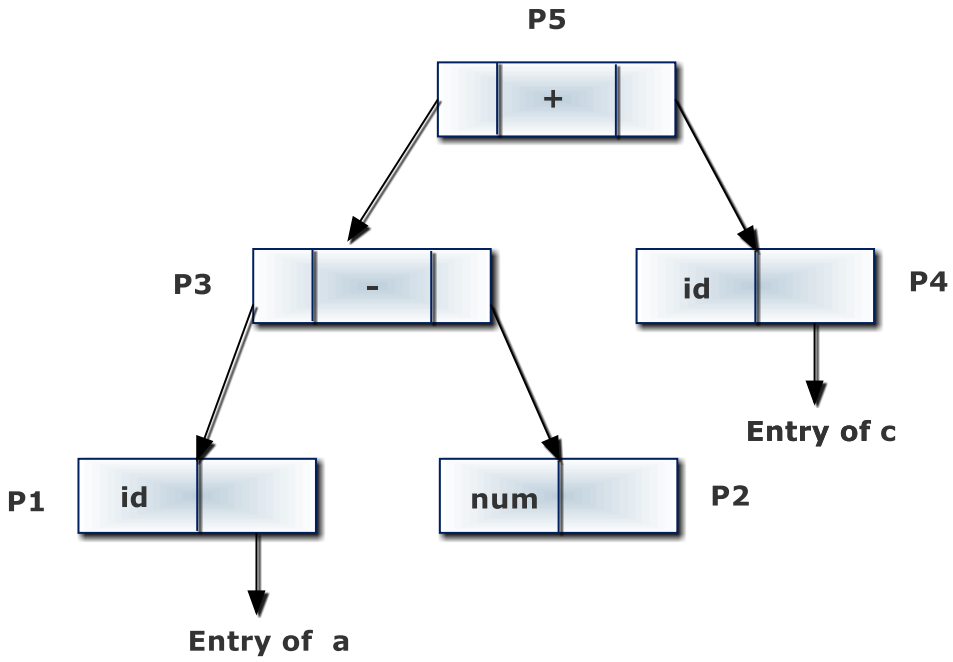
- **Number**: one field with label num and another to keep the value of the number

`mkleaf(num,val)`

Constructing AST for Expression

The following sequence of function calls creates a tree for `a - 4 + c`

```
P1 = mkleaf (id, entry.a)
P2 = mkleaf (num, 4)
P3 = mknode (-, P1 , P2 )
P4 = mkleaf (id, entry.c)
P5 = mknode (+, P3 , P4 )
```



Syntax Directed Definition for Constructing Tree

$E \rightarrow E_1 + T$

$E \rightarrow T$

$T \rightarrow T_1 * F$

$T \rightarrow F$

$F \rightarrow (E)$

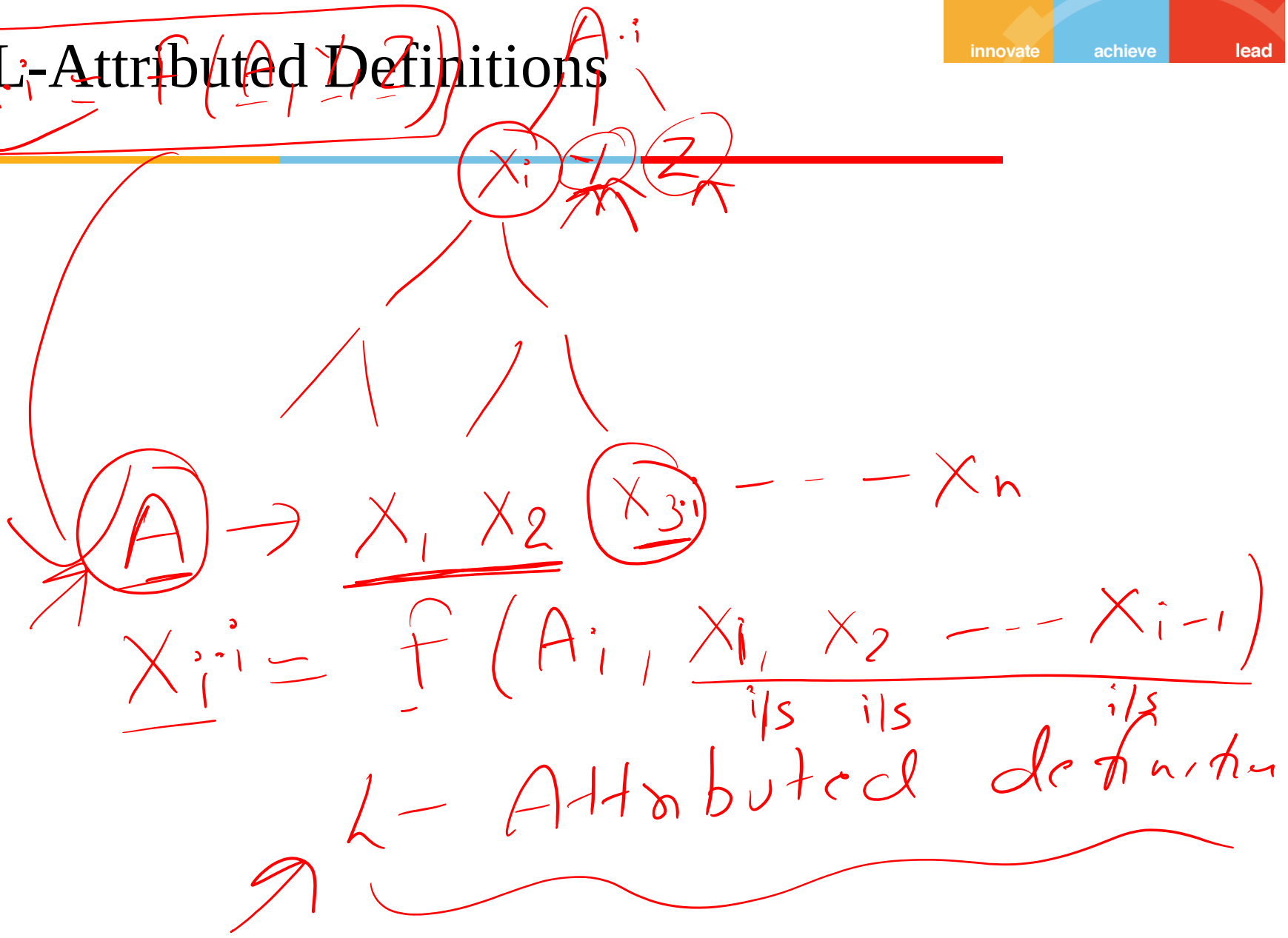
$F \rightarrow id$

$F \rightarrow num$

Expression Grammar

• $E \rightarrow E_1 + T$	**E.ptr = mknode(+, E_1 .ptr, T.ptr)**
• $E \rightarrow T$	**E.ptr = T.ptr**
• $T \rightarrow T_1 * F$	**T.ptr := mknode(*, T1 .ptr, F.ptr)**
• $T \rightarrow F$	**T.ptr := F.ptr**
• $F \rightarrow (E)$	**F.ptr := E.ptr**
• $F \rightarrow id$	**F.ptr := mkleaf(id, entry.id)**
• $F \rightarrow num$	**F.ptr := mkleaf(num,val)**

L-Attributed Definitions



L-Attributed Definitions

L-Attributed Definitions contain both synthesized and inherited attributes

A syntax directed definition is L-Attributed if each inherited attribute of X_j in a production

$A \rightarrow X_1 \; . \; . \; . \; X_j \; . \; . \; . \; X_n$, depends only on

- The attributes of the symbols to the left (this is what L in L-Attributed stands for) of Xj , i.e., **$X_1 \; X_2 \; . \; . \; . \; X_{j-1}$**, and
- The inherited attributes of A.
- A → BC {B.S = A.S} is an L-attributed definition.

L-Attributed Definitions

When translation takes place during parsing, order of evaluation is linked to the order in which nodes are created

- In S-attributed definitions, parent's attribute evaluated after child's attribute.

A natural order in both top-down and bottom-up parsing is depth first-order

- **L-attributed definition**: where attributes can be evaluated in depth-first order

Few Examples

innovate achieve lead

A → LM

- L.i = f1 (A.i)
- M.i = f2 (L.s)
- A.s = f3 (M.s)

L-ATTRIBUTED DEFINITIONS

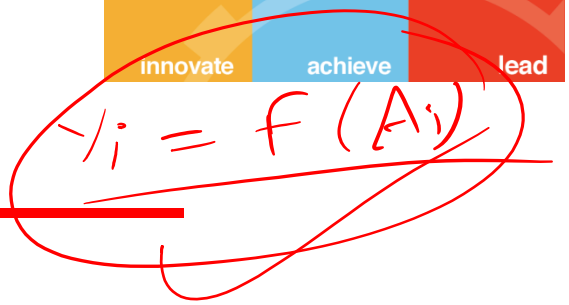
A → QR

- R.i = f4 (A.i)
- Q.i = f5 (R.s)
- A.s = f6 (Q.s)

NOT L-ATTRIBUTED DEFINITIONS

Translation Schemes

$A \rightarrow XY2$



$A \rightarrow X_1 X_2 X_3 \cdots X_n \{A\}$

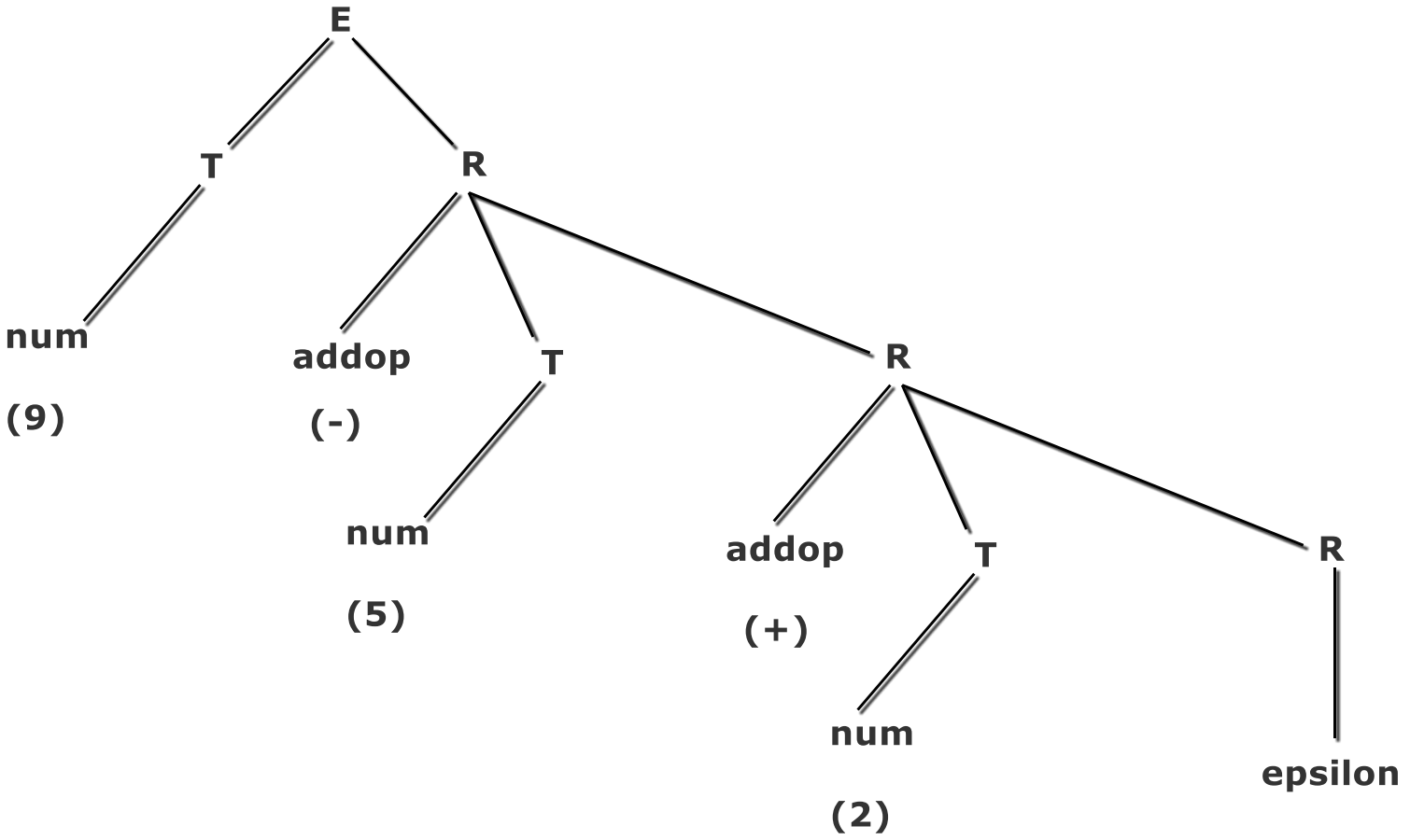
$A \rightarrow X_1 X_2 [A] X_3 \cdots X_n$

Translation Schemes

A CFG where semantic actions occur within the RHS of production

- E→ T R
- R→ addop T R | ε
- T→ num
- addop → + | –

Parse Tree for 9-5+2



Translation Schemes

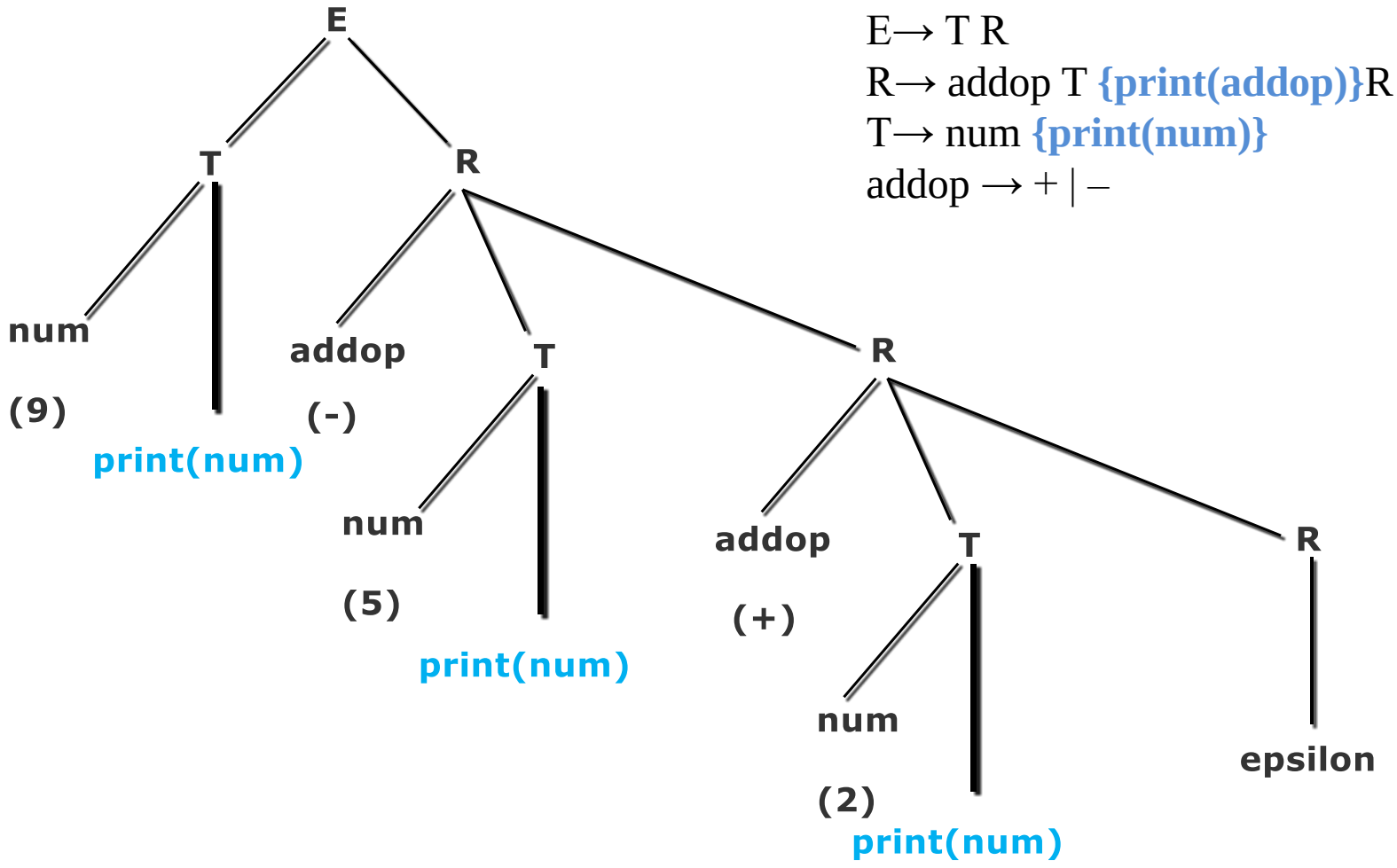
A CFG where semantic actions occur within the RHS of production

- E→ T R
- R→ addop T R | ε
- T→ num
- addop → + | –

Example: A translation scheme to map infix to postfix

- E→ T R
- R→ addop T **{print(addop)}**R | ε
- T→ num **{print(num)}**
- addop → + | –

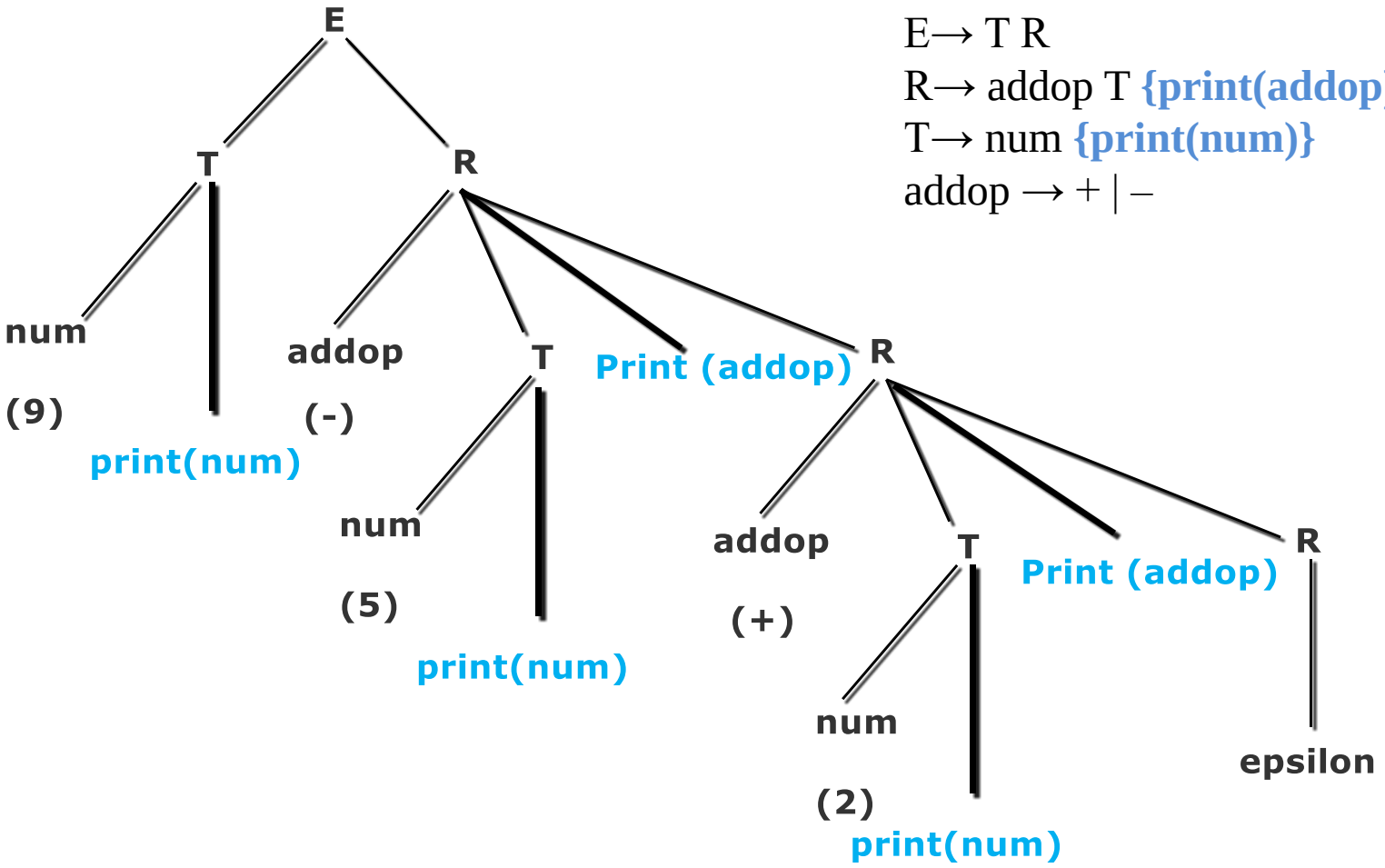
Parse Tree for 9-5+2



E→ T R
R→ addop T {print(addop)}R | ε
T→ num {print(num)}
addop → + | –

Parse Tree for 9-5+2

E→ T R
R→ addop T {print(addop)}R | ε
T→ num {print(num)}
addop → + | –



Evaluation of Translation Schemes

innovate achieve lead

Assume actions are terminal symbols

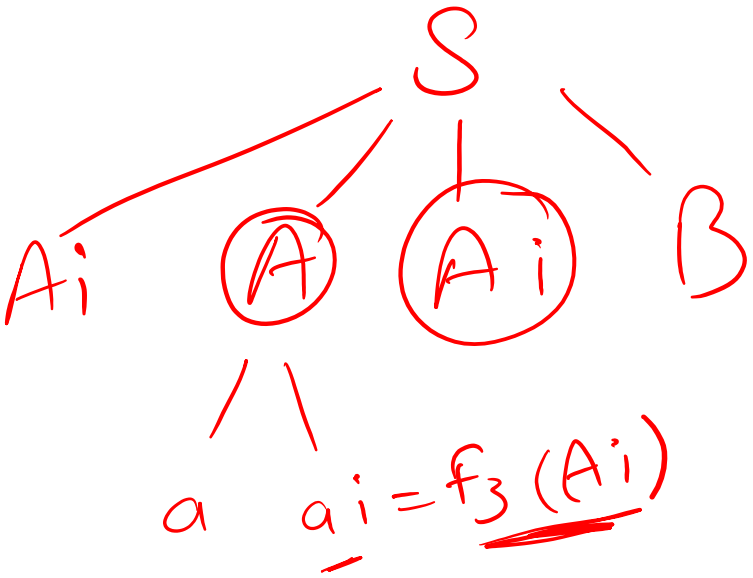
- Perform depth first order traversal to obtain 9 5 – 2 +

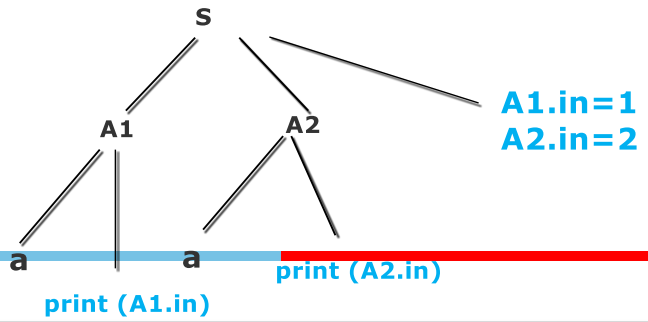
When designing translation scheme, ensure attribute value is available when referred to

$S \rightarrow A_i\ A\ A_i\ B$ { $A_i = f_1()$ }

$A \rightarrow a$ | $a_i = f_3(A_i)$ { $B_i = f_2()$ }

$B \rightarrow b$





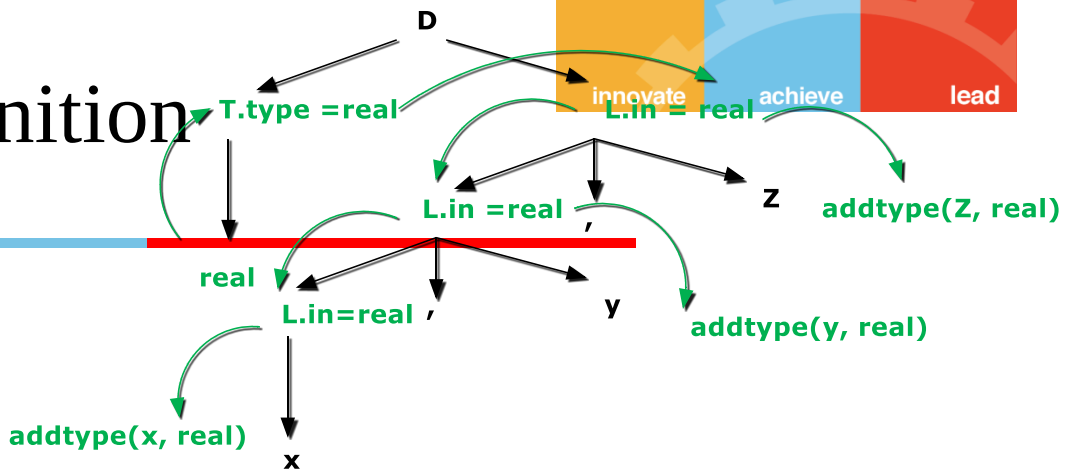
An inherited attribute for a symbol on RHS of a production must be computed in an action before that symbol

- S → A1 A2 **{A1 .in = 1,A2 .in = 2}**
- A → a **{print(A.in)}**

Depth first order traversal gives error **(undefined)**

- A synthesized attribute for the non terminal on the LHS can be computed after all attributes it references, have been computed. The action normally should be placed at the end of RHS.

Syntax Directed Definition



- D → T L — `L.in = T.type`
- T → real — `T.type = real`
- T → int — `T.type = int`
- L → L_1 , id — `L1 .in = L.in`

  `addtype(id.entry, L.in)`
- L → id — `addtype (id.entry,L.in)`

Translation Schemes: An Example

Consider the Translation Scheme for the L-Attributed Definition for "type declarations":

D → T **{L.in := T .type }** L

T → int **{T .type :=int }**

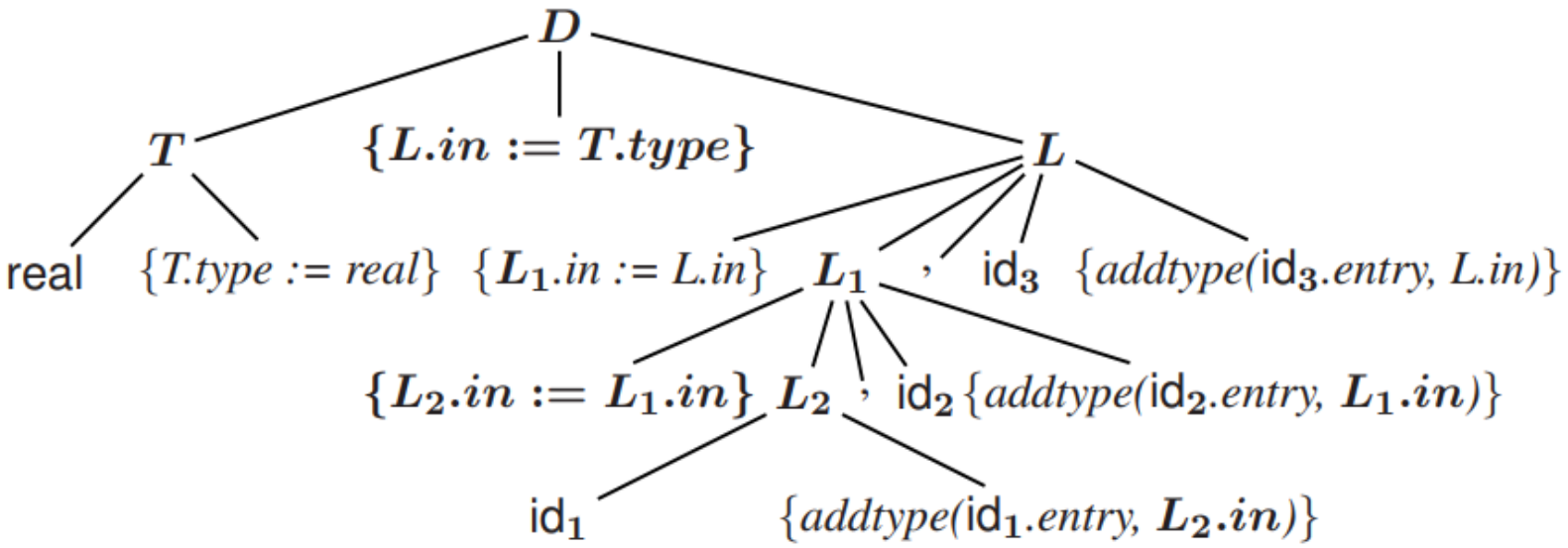
T → real **{T .type :=real }**

L → **{ L 1.in := L.in }** L_1, id **{addtype(id.entry, L.in) }**

L → id **{addtype(id.entry, L.in) }**

Translation Schemes: An Example (Cont.)

The parse-tree with semantic actions for the input real id_1, id_2, id_3 is:



Traversing the Parse-Tree in depth-first order we can evaluate the attributes.

Bottom-up Evaluation of Inherited Attributes

Remove embedded actions from translation scheme

- Make transformation so that embedded actions occur only at the end of their productions.

Replace each action by a distinct marker non-terminal M and attach action at end of $M \rightarrow \varepsilon$

L-Attributed Definitions to S-Attributed Definitions

E → T R
R → + T **{print (+)}** R
R → - T **{print (-)}** R
R → Є
T→num**{print(num.val)}**

TRANSLATION SCHEMES

E → T R
R → + T M R
R → - T N R
R → Є
T → num **{print(num.val)}**
M → Є **{print(+)}**
N → Є **{print(-)}**

S-ATTRIBUTED DEFINITIONS