# Chapter 6: Attribute Grammars and Syntax-Directed Translation

Zhen Gao
gaozhen@tongji.edu.cn

# Outline

- **Attribute Grammar**
- **Syntax-Directed Translation**

# Attribute grammar Example

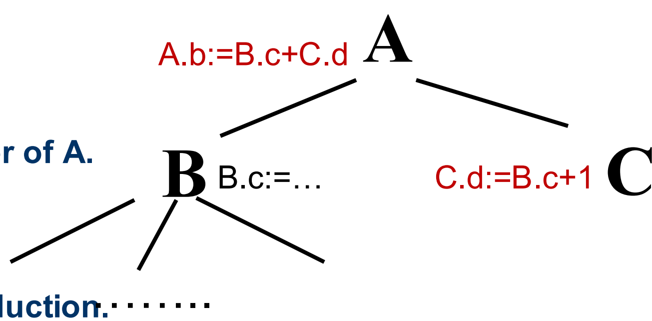
| Production | Semantic Rules |
|---|---|
| $L \rightarrow En$ | print(E.val) |
| $E \rightarrow E_1 + T$ | $E.val := E_1.val + T.val$ |
| $E \rightarrow T$ | $E.val := T.val$ |
| $T \rightarrow T_1 * F$ | $T.val := T_1.val * F.val$ |
| $T \rightarrow F$ | $T.val := F.val$ |
| $F \rightarrow (E)$ | $F.val := E.val$ |
| $F \rightarrow digit$ | $F.val := digit.lexval$ |

# Attribute Grammer


- Proposed by Knuth in 1968
- Based on context-free grammar, each symbol has attributes (e.g., type, address)
- Each production has a set of semantic rules: $b := f(c_1, c_2, \ldots, c_k)$
- Definition: grammar with attributes on symbols and semantic rules on productions $\rightarrow$ **Attribute Grammar**

# Attributes and Semantic Rules

- **Attributes**
  - **represent information related to grammar symbols, e.g., type, value, code sequence, symbol table content.**
  - **can be computed and passed**
- **Semantic Rules**
  - **For each production $A \rightarrow \alpha$, there is a set of semantic rules of the form: $b := f(c_1, c_2, \ldots, c_k)$**
- **Attributes Types**
  - **Synthesized attribute**
    - **b is an attribute of A.**
    - **$c_1, c_2, \ldots, c_k$ are attributes of symbols on the right-hand side or of A.**
  - **Inherited attribute**
    - **b is an attribute of a symbol X on the right-hand side.**
    - **$c_1, c_2, \ldots, c_k$ are attributes of A, itself or its siblings in the production.**



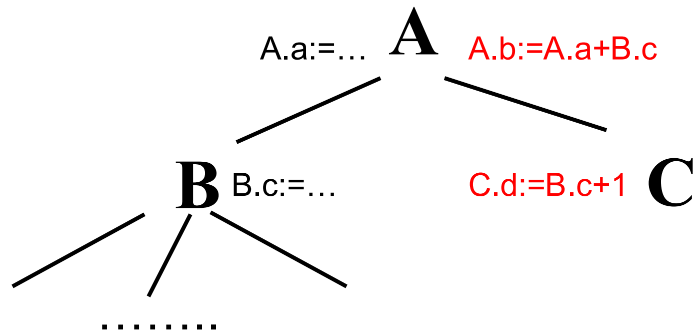
# Example: Consider nonterminals A, B, and C

| | Synthesized Attribute | Inherited Attribute |
|---|---|---|
| A | b | a |
| B | c | |
| C | | d |

The production A→BC

C.d:=B.c+1

A.b:=A.a+B.c




Q: When should the attributes A.a and B.c be computed?

# Notation for Attributes

- **For a grammar symbol $X \in V_T \cup V_N$ , its attributes are denoted as:**
  - **X.type – type of X**
  - **X.cat – category of X**
  - **X.val – value or address of X**
- **Use subscripts to distinguish multiple occurrences of the same symbol in a production.**

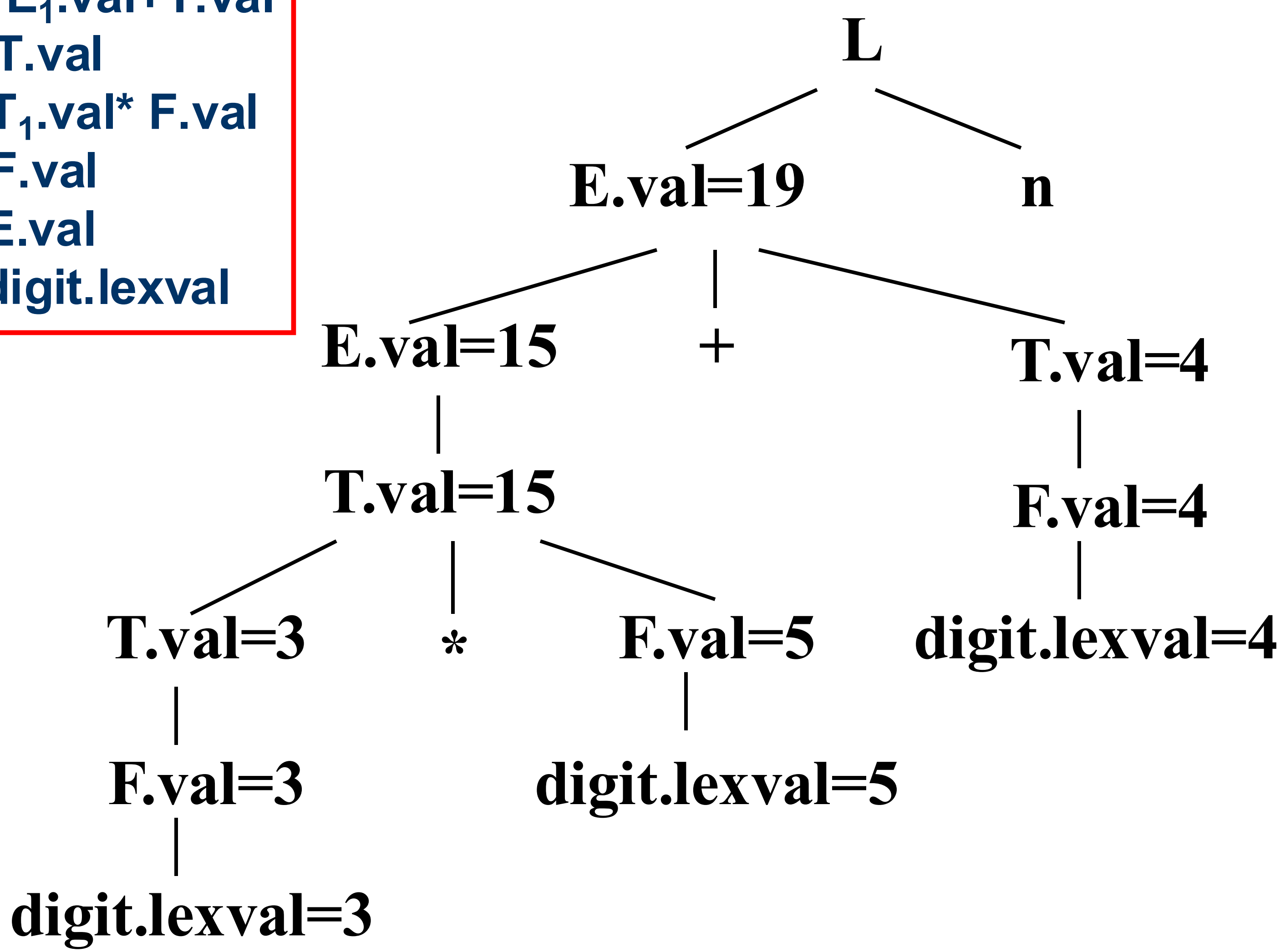
# S-Attributed Grammar

- **A grammar that uses only synthesized attributes is called an S-attributed grammar.**
- **a node's synthesized attribute is computed from its children.**
- **Computation uses bottom-up semantic rules.**

**Production/Semantic Rules**

| | |
|---|---|
| $L \rightarrow En$ | print(E.val) |
| $E \rightarrow E_1+T$ | E.val := $E_1$.val+T.val |
| $E \rightarrow T$ | E.val :=T.val |
| $T \rightarrow T_1*F$ | T.val :=$T_1$.val* F.val |
| $T \rightarrow F$ | T.val :=F.val |
| $F \rightarrow (E)$ | F.val :=E.val |
| $F \rightarrow digit$ | F.val :=digit.lexval |

**3*5+4n**




*

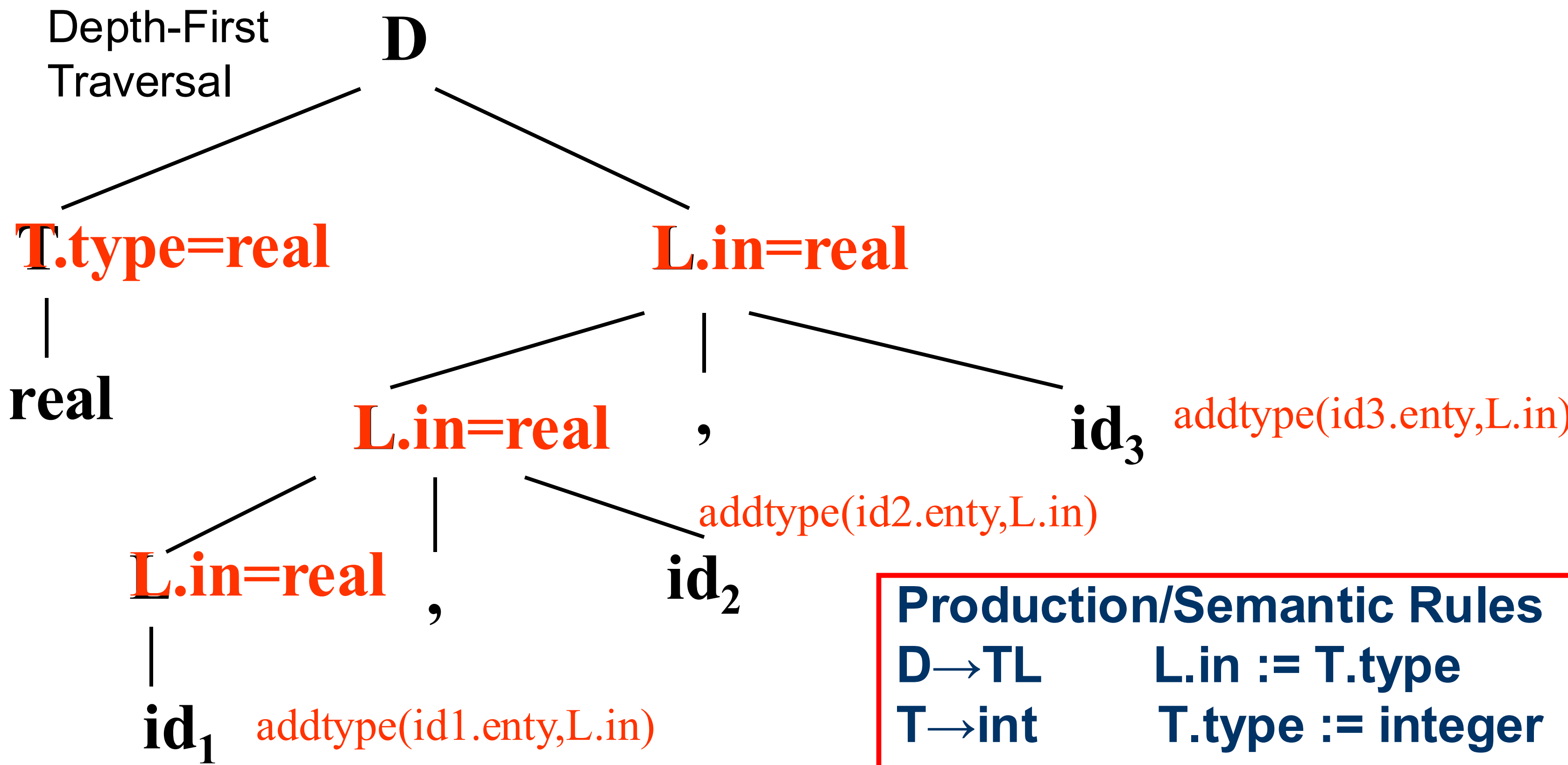
# L-Attributed Grammar

- For each production $A \rightarrow X_1 \ldots X_{j-1} X_j \ldots X_n$ each semantic rule computes either:
  - a synthesized attribute of A, or
  - an inherited attribute of $X_j$ that depends only on:
    - attributes of symbols to the left of $X_j$ ($X_1, \ldots, X_{j-1}$), and
    - inherited attributes of A.
- **S-attributed grammars** are a subset of L-attributed grammars.

# Example: Symbol Table Operations
# Attribute grammar with inherited attribute **L.in**

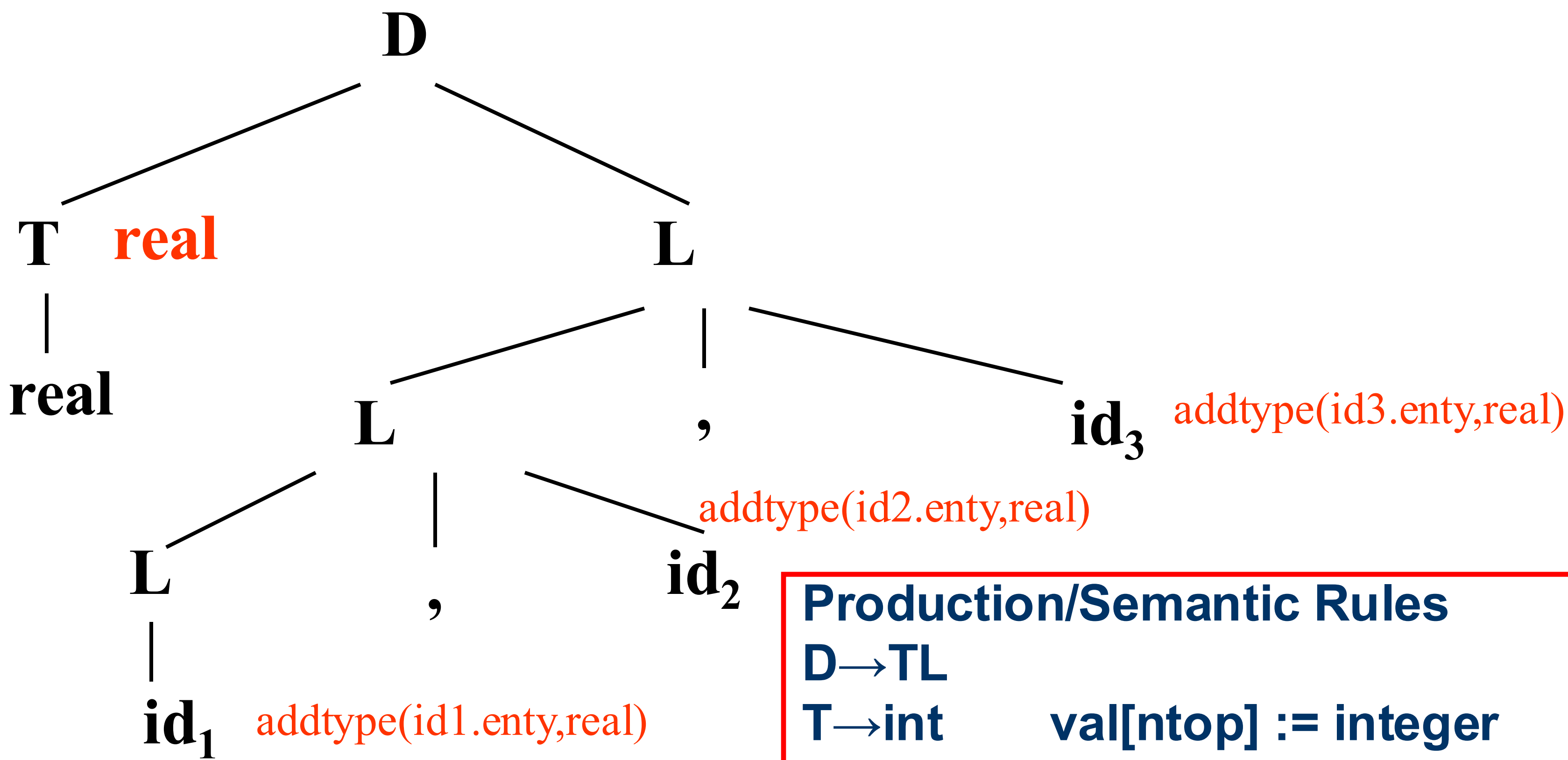
| **Production** | **Semantic Rules** |
|---|---|
| **$D \rightarrow TL$** | **L.in := T.type** |
| **$T \rightarrow$ int** | **T.type := integer** |
| **$T \rightarrow$ real** | **T.type := real** |
| **$L \rightarrow L_1$, id** | **$L_1$.in :=L.in** |
| | **addtype(id.entry, L.in)** |
| **$L \rightarrow$ id** | **addtype(id.entry, L.in)** |

# real $id_1$, $id_2$, $id_3$




| Production/Semantic Rules | |
|---|---|
| $D \rightarrow TL$ | L.in := T.type |
| $T \rightarrow int$ | T.type := integer |
| $T \rightarrow real$ | T.type := real |
| $L \rightarrow L_1$, id | $L_1$.in :=L.in<br>addtype(id.entry, L.in) |
| $L \rightarrow id$ | addtype(id.entry, L.in) |

# LR Parsing




**Production/Semantic Rules**

| | |
|---|---|
| $D \rightarrow TL$ | |
| $T \rightarrow int$ | val[ntop] := integer |
| $T \rightarrow real$ | val[ntop] := real |
| $L \rightarrow L_1$, id | addtype(val[top],val[top-3]) |
| $L \rightarrow id$ | addtype(val[top],val[top-1]) |

top:before R,   ntop:After R

*

# Example : Static Semantic Check (Type Checking)

$E \to T^1 + T^2$

{ if $T^1$.type = int and $T^2$.type= int

then E.type :=int

else error}

$E \to T^1$ or $T^2$

{ if $T^1$.type = bool and $T^2$.type=bool

then E.type :=bool

else error}

$T \to n$ { T.type := int}

$T \to b$ { T.type := bool}

## LR(0) Parsing Table

| State | action | | | | | GOTO | |
|---|---|---|---|---|---|---|---|
| | + | or | n | b | # | E | T |
| 0 | | | s4 | s3 | | 1 | 2 |
| 1 | | | | | acc | | |
| 2 | s5 | s7 | | | | | |
| 3 | r4 | r4 | r4 | r4 | r4 | | |
| 4 | r3 | r3 | r3 | r3 | r3 | | |
| 5 | | | s4 | s3 | | | 6 |
| 6 | r1 | r1 | r1 | r1 | r1 | | |
| 7 | | | s4 | s3 | | | 8 |
| 8 | r2 | r2 | r2 | r2 | r2 | | |

The **LR parser stack** is augmented with **semantic values**.

| State | Value | Symbol |
|---|---|---|
| | | |
| $S_m$ | Y.VAL | Y |
| $S_{m-1}$ | X.VAL | X |
| … | … | … |
| $S_0$ | -- | # |

TOP

*

$E \rightarrow T_1 + T_2$ {
if $T_1$.type = int and $T_2$.type= int
E.type :=int
else error }
$E \rightarrow T_1$ or $T_2$ {
if $T_1$.type = bool and $T_2$.type=bool
E.type :=bool
else error }
$T \rightarrow n$ { T.type := int}
$T \rightarrow b$ { T.type := bool}

**LR(0) Parsing Table**

| | action | | | | | GOTO | |
|---|---|---|---|---|---|---|---|
| State | + | or | n | b | # | E | T |
| 0 | | | s4 | s3 | | 1 | 2 |
| 1 | | | | | acc | | |
| 2 | s5 | s7 | | | | | |
| 3 | r4 | r4 | r4 | r4 | r4 | | |
| 4 | r3 | r3 | r3 | r3 | r3 | | |
| 5 | | | s4 | s3 | | | 6 |
| 6 | r1 | r1 | r1 | r1 | r1 | | |
| 7 | | | s4 | s3 | | | 8 |
| 8 | r2 | r2 | r2 | r2 | r2 | | |

| | | |
|---|---|---|
| 6 | T | int |
| 5 | + | --- |
| 1 | E | int |
| 0 | # | -- |

Input : n + n

*

$E \to T_1 + T_2$ {
if $T_1$.type = int and $T_2$.type= int
E.type :=int
else error }
$E \to T_1$ or $T_2$ {
if $T_1$.type = bool and $T_2$.type=bool
E.type :=bool
else error }
$T \to n$ { T.type := int}
$T \to b$ { T.type := bool}

# Input: n + b

| | | |
|---|---|---|
| 6 | T | bool |
| 5 | + | --- |
| 1 | E | error |
| 0 | # | -- |

**LR(0) Parsing Table**

| State | action | | | | | GOTO | |
|---|---|---|---|---|---|---|---|
| | + | or | n | b | # | E | T |
| 0 | | | s4 | s3 | | 1 | 2 |
| 1 | | | | | acc | | |
| 2 | s5 | s7 | | | | | |
| 3 | r4 | r4 | r4 | r4 | r4 | | |
| 4 | r3 | r3 | r3 | r3 | r3 | | |
| 5 | | | s4 | s3 | | | 6 |
| 6 | r1 | r1 | r1 | r1 | r1 | | |
| 7 | | | s4 | s3 | | | 8 |
| 8 | r2 | r2 | r2 | r2 | r2 | | |

*

# Conclusion

- **Terminals**: only synthesized attributes from the lexer.
- **Nonterminals**: synthesized and inherited attributes; start symbol's inherited attributes are initial values.
- Provide rules for **right-hand inherited and left-hand synthesized attributes**, using only symbols in the same production.
- **Left-hand inherited and right-hand synthesized attributes** are computed elsewhere or externally.
- Semantic rules can include **attribute computation, symbol table updates, type checks, code generation**, etc.

# Outline

- ✓ Attribute Grammar
- Syntax-Directed Translation

# Single-Pass Processing Method

- **Single-pass method**: compute attribute values during parsing
- **S-attributed grammars**: suited for bottom-up, single-pass parsing
- **L-attributed grammars**: suited for top-down, single-pass parsing

# Syntax-Directed Translation

- Processing driven by the source program's **syntax**.
- Attach a **semantic action** to each production and execute it during parsing.
- When a production is **matched** (top-down) or **reduced** (bottom-up), its **semantic action** executes to perform the translation and produce intermediate code.

# Role of Syntax-Directed Translation

- If semantic actions generate intermediate code, code is produced gradually during parsing.
- Functions:
  - **Generate intermediate code**
  - Generate target instructions
  - Interpret the input string during parsing

# Exercise

- Assume a grammar **G(R)** for decimal numbers:

  **R → N.N**

  **N → d**

  **N → Nd**

Define an **attribute grammar** to compute the value of decimal numbers, and draw the **attribute tree with computed values** for 23.05.

# Quiz-Canvas

- **ch6 Syntax-Directed Translation Method**

Dank u
Dutch

Merci
French

Спасибо
Russian

Gracias
Spanish

شكراً
Arabic

धन्यवाद
Hindi

감사합니다
Korean

Tack så mycket
Swedish

תודה רבה
Hebrew

Obrigado
Brazilian Portuguese

谢谢
Chinese

Dankon
Esperanto

***Thank You !***

ありがとうございます
Japanese

Trugarez
Breton

Danke
German

Tak
Danish

Grazie
Italian

நன்றி
Tamil

děkuji
Czech

ขอบคุณ
Thai

go raibh maith agat
Gaelic

*