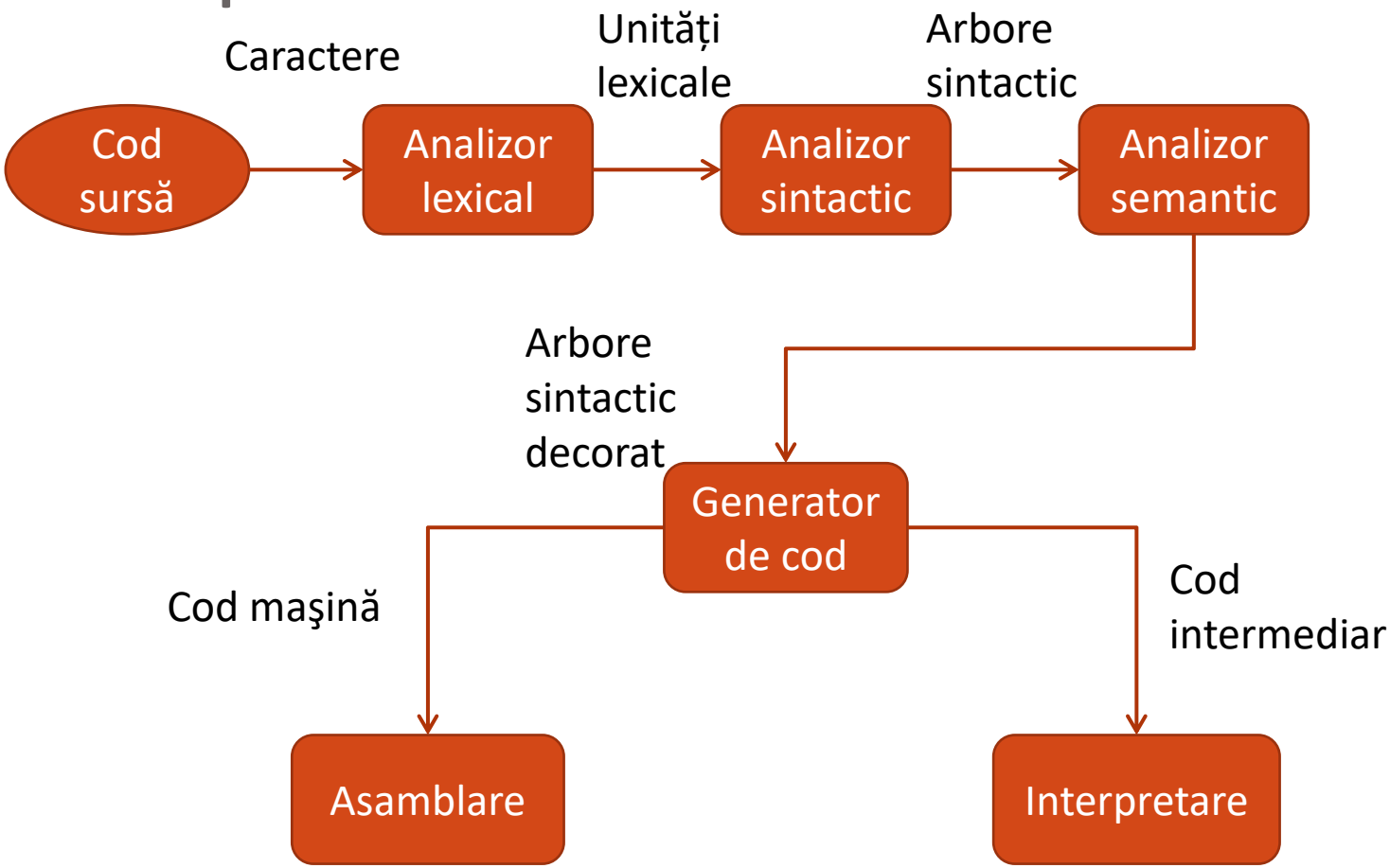
Limbaje formale, automate şi compilatoare

Curs 11

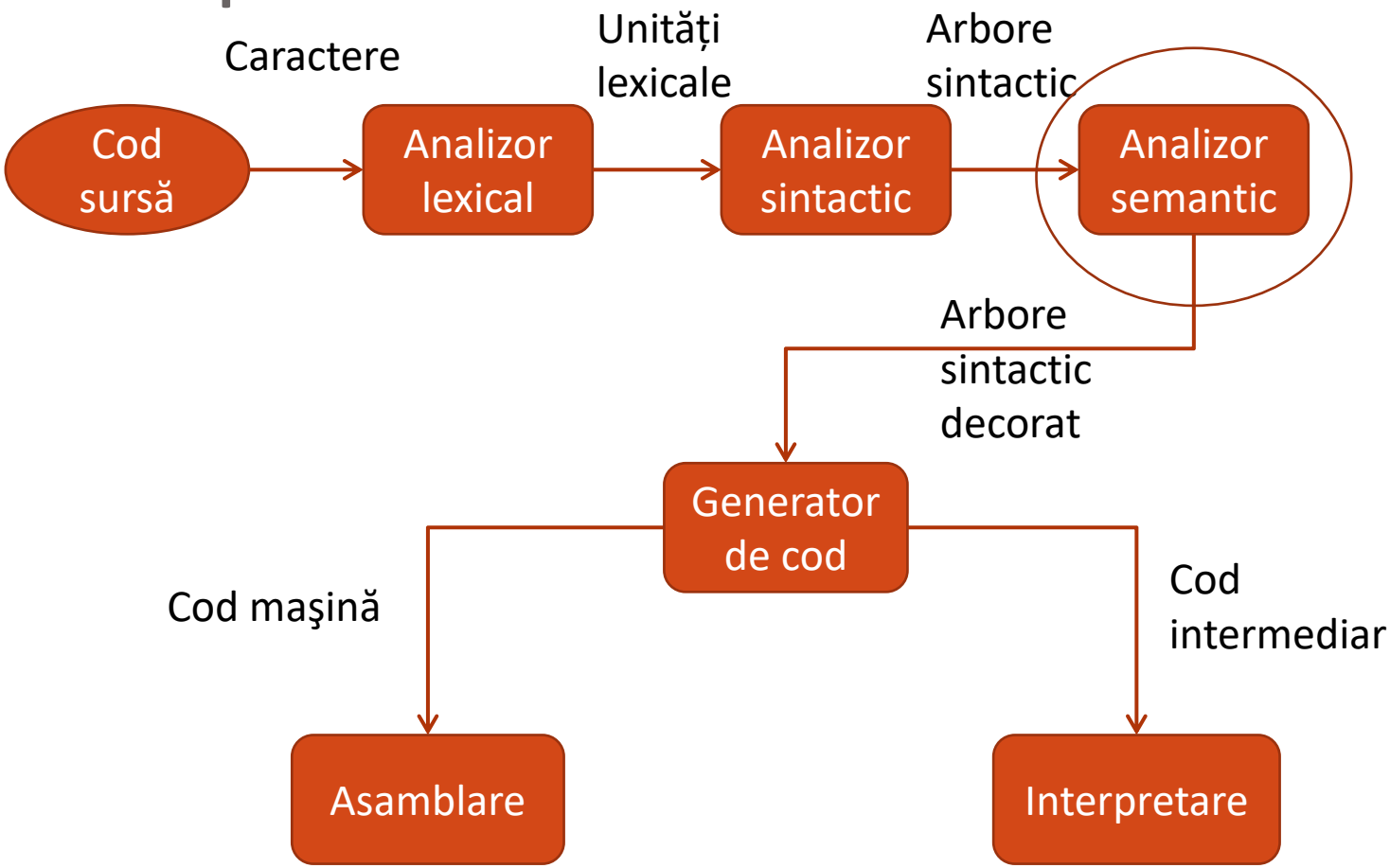
Compilare


Caractere
Unități lexicale
Arbore sintactic
Cod sursă
Analizor lexical
Analizor sintactic
Analizor semantic
Arbore sintactic decorat
Generator de cod
Cod maşină
Cod intermediar
Asamblare
Interpretare

Recapitulare

- Analiza lexicală
  - Validează tokeni
- Analiza sintactică
  - Validează arborele sintactic
- Analiza semantică
  - Detectează restul erorilor
  - Ultimul pas de analiză

Compilare


Caractere
Unități lexicale
Arbore sintactic
Cod sursă
Analizor lexical
Analizor sintactic
Analizor semantic
Arbore sintactic decorat
Generator de cod
Cod maşină
Cod intermediar
Asamblare
Interpretare

Analiza semantică

- Verificări
  - Toti identificatorii sunt declarați
  - Verificarea tipurilor
  - Relații de moștenire
  - Structurile definite de utilizator unic declarate
  - Metode unic definite
  - Identificatorii rezervați corect folosiți
  - Etc.

Analiza semantică

- Cum asociem sens parsării sintactice?
- Sunt necesare elemente sau structuri suplimentare?
- Sunt toate elementele semantice independente de context?

Traduceri direcționate sintactic (TDS)

- Traduceri de limbaje folosind gramatici de tip 2
  - Informația este transmisă folosind atribute asociate elementelor gramaticii
    - $E \rightarrow E_1 + T;$ $\quad E.val = E_1.val + T.val$

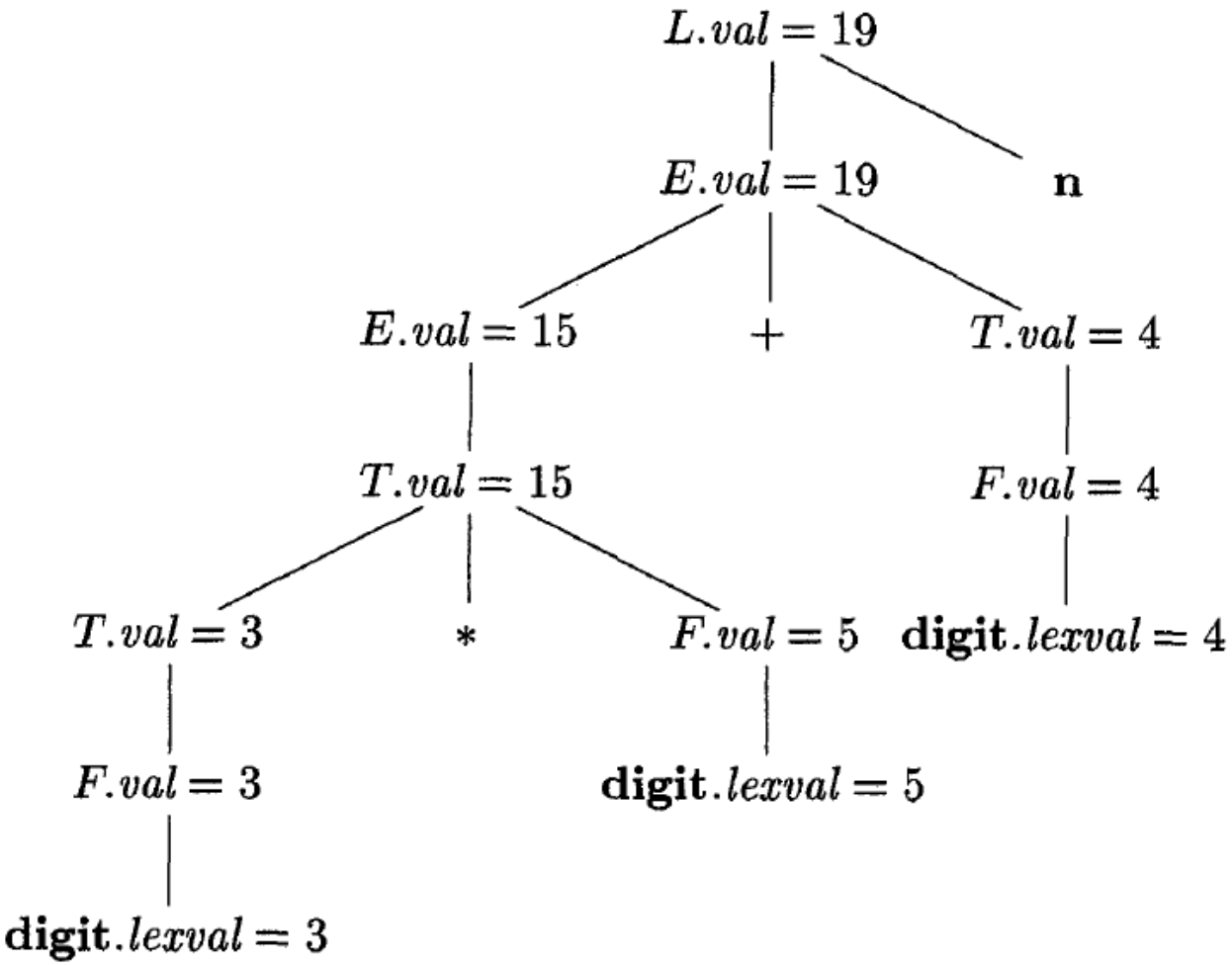
Definiții direcționate sintactic (DDS)

- O gramatică de tip 2, cu atribute și reguli semantice
- Atributele sunt reprezentate de câmpuri de date asociate nodurilor din arbori sintactici
- Atributele pot fi:
  - Sintetizate
    - Definite pentru un neterminal A de la nodul N din arborele sintactic printr-o regulă asociată producției de la nodul N
  - Moștenite
    - Definite pentru un neterminal B de la nodul N din arborele de parsare printr-o regulă asociată producției din părintele nodului N

Atribute sintetizate

Reguli sintactice	Reguli semantice
$L \rightarrow E_n$	L.val = 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

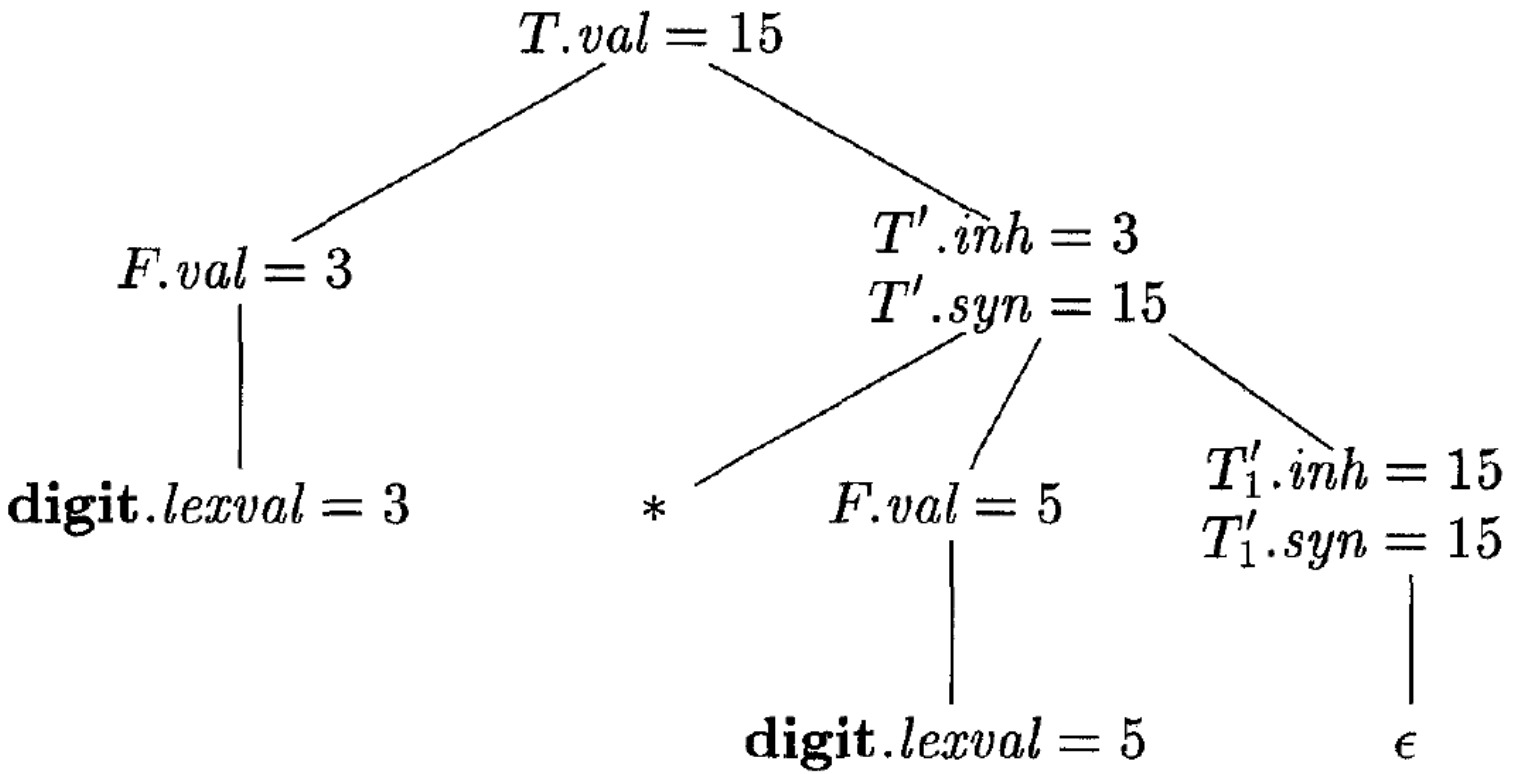
Atribute sintetizate



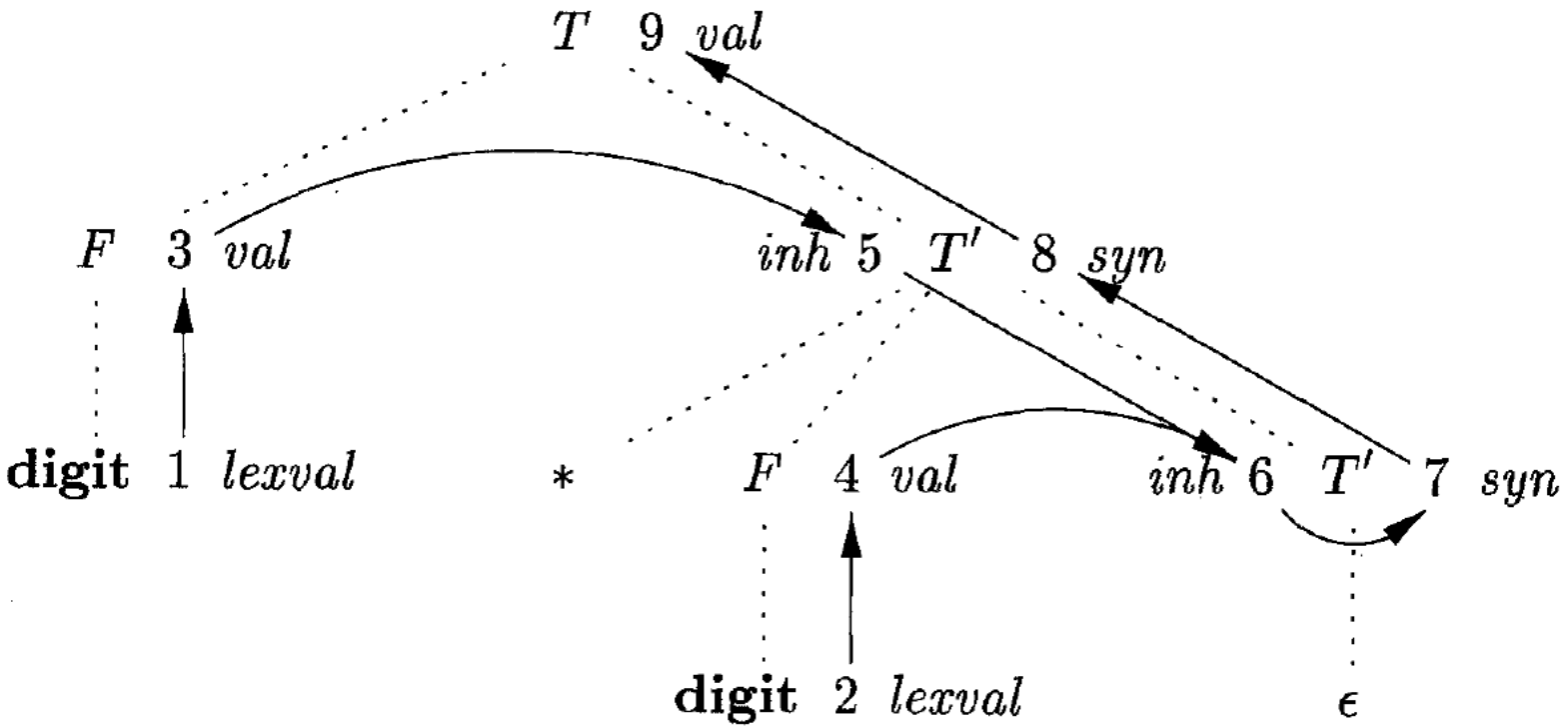
Atribute moștenite

$T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow * FT_1'$	$T_1'.inh = T'.inh * F.val$ $T'.syn = T_1'.syn$
$T' \rightarrow \epsilon$	$T'.syn = T'.inh$
$F \rightarrow cifra$	$F.val = cifra.val$

Atribute moștenite



Ordinea de evaluare în DDS



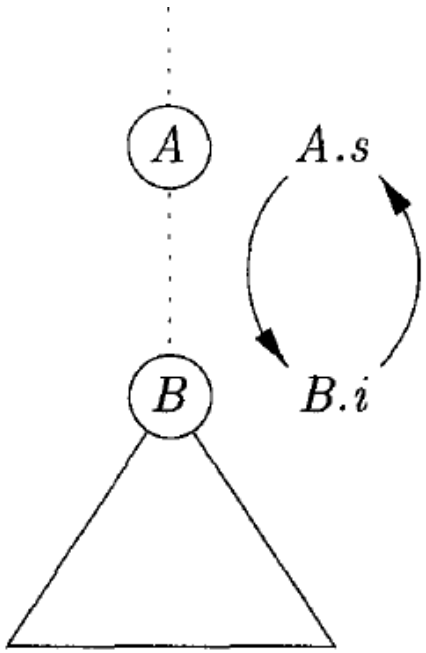
Ordinea de evaluare în DDS

- Este dată de un graf de dependență (orientat) care descrie fluxul de informație dintre atribute
  - Pentru fiecare nod din arborele sintactic, graful de dependență are un nod asociat fiecărui atribut
  - Dacă valoarea atributului sintetizat A.b depinde de X.c, atunci există o muchie în graful de dependență de la X.c la A.b
  - Dacă valoarea atributului moștenit B.c depinde de X.a, atunci există o muchie în graful de dependeță de la X.a la B.c.

Ordinea de evaluare în DDS - circuite

$A \to B$

$A.s = B.i;$
$B.i = A.s + 1$



Ordinea de evaluare în DDS

- Determinarea existenței unor arbori de parsare pentru care grafurile de dependeță nu au circuite este de complexitate exponențială
- Traducerile pot fi implementate folosind clase de DDS care garantează existența unei ordini de evaluare

Ordonarea evaluării atributelor

- Dacă graful de dependeță are o muchie de la nodul M la nodul N, atunci atributul corespunzător lui M trebuie evaluat înaintea atributului din N

- Graful de dependeță trebuie ordonat topologic (adus la o formă liniară astfel încât toate muchiile sunt îndreptate dinspre primul atribut evaluat către ultimul atribut evaluat)

Definiții S-Atribuite

- Toate atributele sunt sintetizate
- Ordinea corectă pentru evaluare este dată de orice parcurgere ascendentă a arborelui de parsare
  - Ex. Traversare postordine
  - Poate fi realizată în timpul parsării ascendente (care corespunde unei traversări în postordine)

Definiții L-Atribuite

- Muchiile grafului de dependență pot fi orientate de la stânga la dreapta, dar nu invers
- Atributele pot fi:
  - Sintetizate
  - Atributele moștenite pentru X_i (dată regula de producție $A \rightarrow X_1 X_2 \ldots X_i \ldots X_n$)
    - De la părintele A
    - De la un frate de la stânga lui X_i
    - Atributele moștenite sau sintetizate pentru X_i care nu produc circuite

Definiții L-Atribuite : exemplu

- Declarațiile de variabile cu tip pot fi interpretate semantic folosind definiții L-atribuite

Aplicații pentru TDS

- Construcția de arbori sintactici din expresii matematice
- Un nod care reprezintă E1+E2 conține trei câmpuri: unul pentru operator și câte unul pentru fiecare operand

1)	$E \rightarrow E_1 + T$	$E.node = \textbf{new}\ Node('+', E_1.node, T.node)$
2)	$E \rightarrow E_1 - T$	$E.node = \textbf{new}\ Node('-', E_1.node, T.node)$
3)	$E \rightarrow T$	$E.node = T.node$
4)	$T \rightarrow (E)$	$T.node = E.node$
5)	$T \rightarrow \textbf{id}$	$T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$
6)	$T \rightarrow \textbf{num}$	$T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$

Aplicații pentru TDS

1)	$E \to T\ E'$	$E.node = E'.syn$ $E'.inh = T.node$
2)	$E' \to +\ T\ E'_1$	$E'_1.inh = \textbf{new}\ Node('+', E'.inh, T.node)$ $E'.syn = E'_1.syn$
3)	$E' \to -\ T\ E'_1$	$E'_1.inh = \textbf{new}\ Node('-', E'.inh, T.node)$ $E'.syn = E'_1.syn$
4)	$E' \to \epsilon$	$E'.syn = E'.inh$
5)	$T \to (\ E\ )$	$T.node = E.node$
6)	$T \to \textbf{id}$	$T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$
7)	$T \to \textbf{num}$	$T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$

Cod intermediar

- *S.next* (inh): începutul codului care urmează instrucțiunii S
- *S.code* (syn): implementarea în cod intermediar pentru S; sare la S.next după execuție.
- *C.true* (inh): Începutul codului executat dacă C este adevărat.
- *C.false* (inh): Începutul codului executat dacă C este fals.
- *C.code* (syn): implementarea în cod intermediar pentru condiția C; sare la C.true sau C.false.

Cod intermediar - DDS

- **S →while (C) S1**
  - `L1 = new();`
  - `L2 = new();`
  - `S1.next = L1;`
  - `C.false = S.next;`
  - `C.true = L2;`
  - `S.code = label || L1 || C.code || label || L2 || S1.code`

Generare de cod intermediar - TDS

- S → while (
  - `{ L1 = new(); L2 = new(); C.false = S.next; C.true = L2;}`
- C)
  - `{ S1.next = L1;}`
- S1
  - `{ S.code = label || L1 || C.code || label || L2 || S1 .code;}`

Bibliografie

- A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, Second Edition. Addison-Wesley, 2007