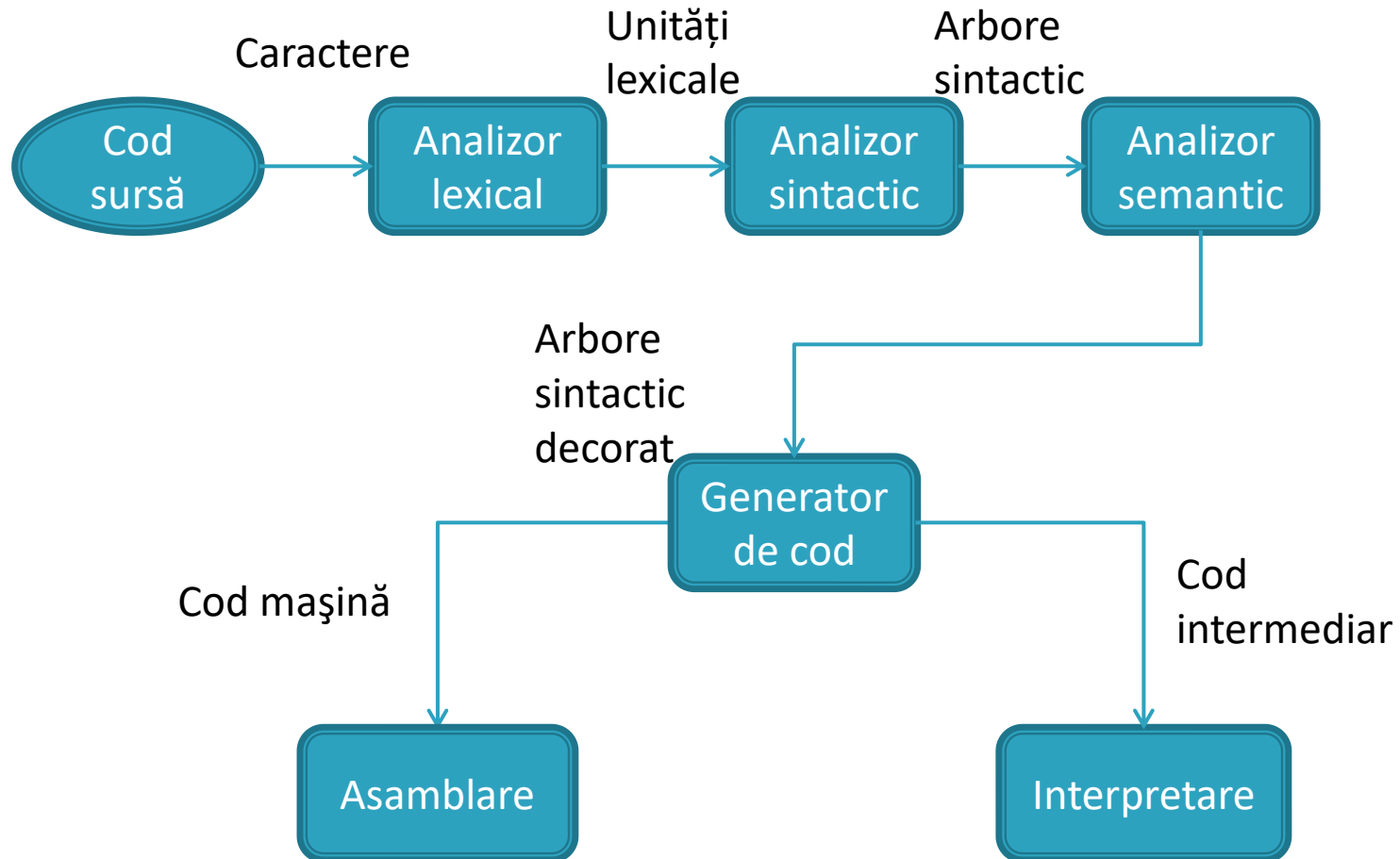


Limbaje formale, automate și compilatoare

Curs 11

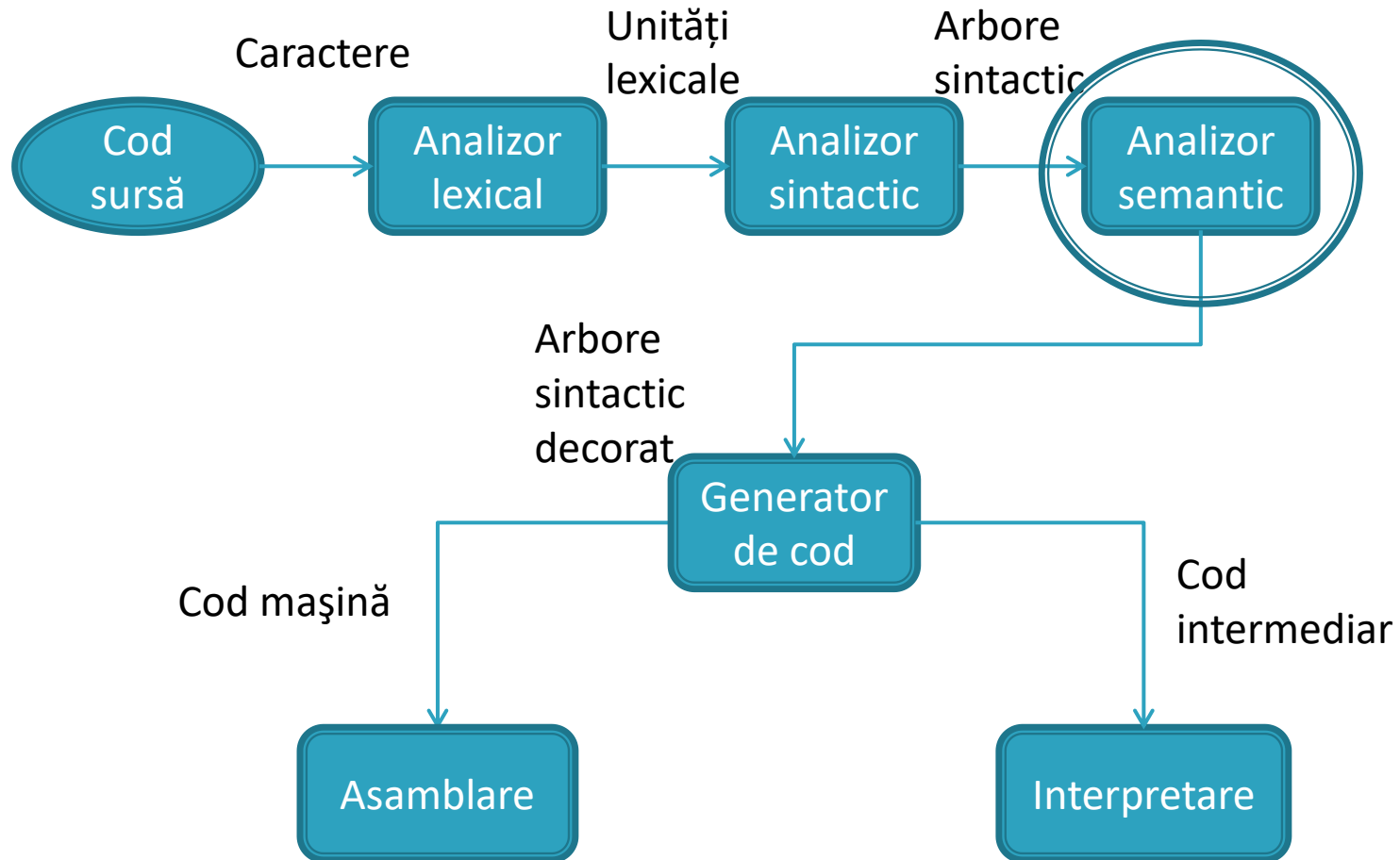
Compilare



Recapitulare

- ▶ Analiza lexicală
 - Validează tokeni
- ▶ Analiza sintactică
 - Validează arborele sintactic
- ▶ Analiza semantică
 - Detectează toate celelalte erori
 - Ultimul pas de analiză

Compilare



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

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

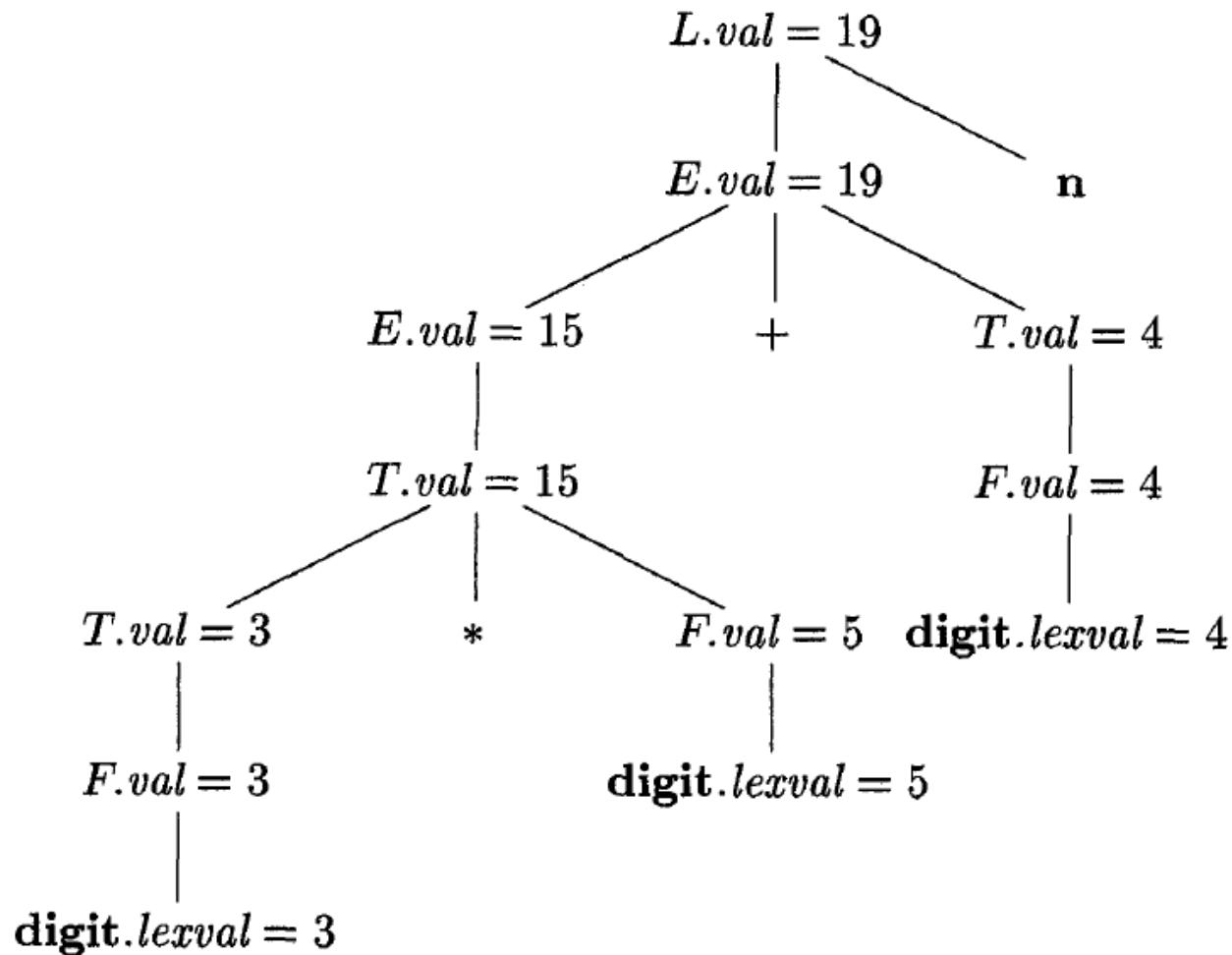
Definiții direcționate sintactic

- ▶ O gramatică de tip 2, cu attribute și reguli semantice
- ▶ Attributele sunt reprezentate de câmpuri de date asociate nodurilor din arbori sintactici
- ▶ Attributele 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

Attribute sintetizate

Reguli sintactice	Reguli semantice
$L \rightarrow E_n$	$L.val = E.val$
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow \text{digit}$	$F.val = \text{digit.lexval}$

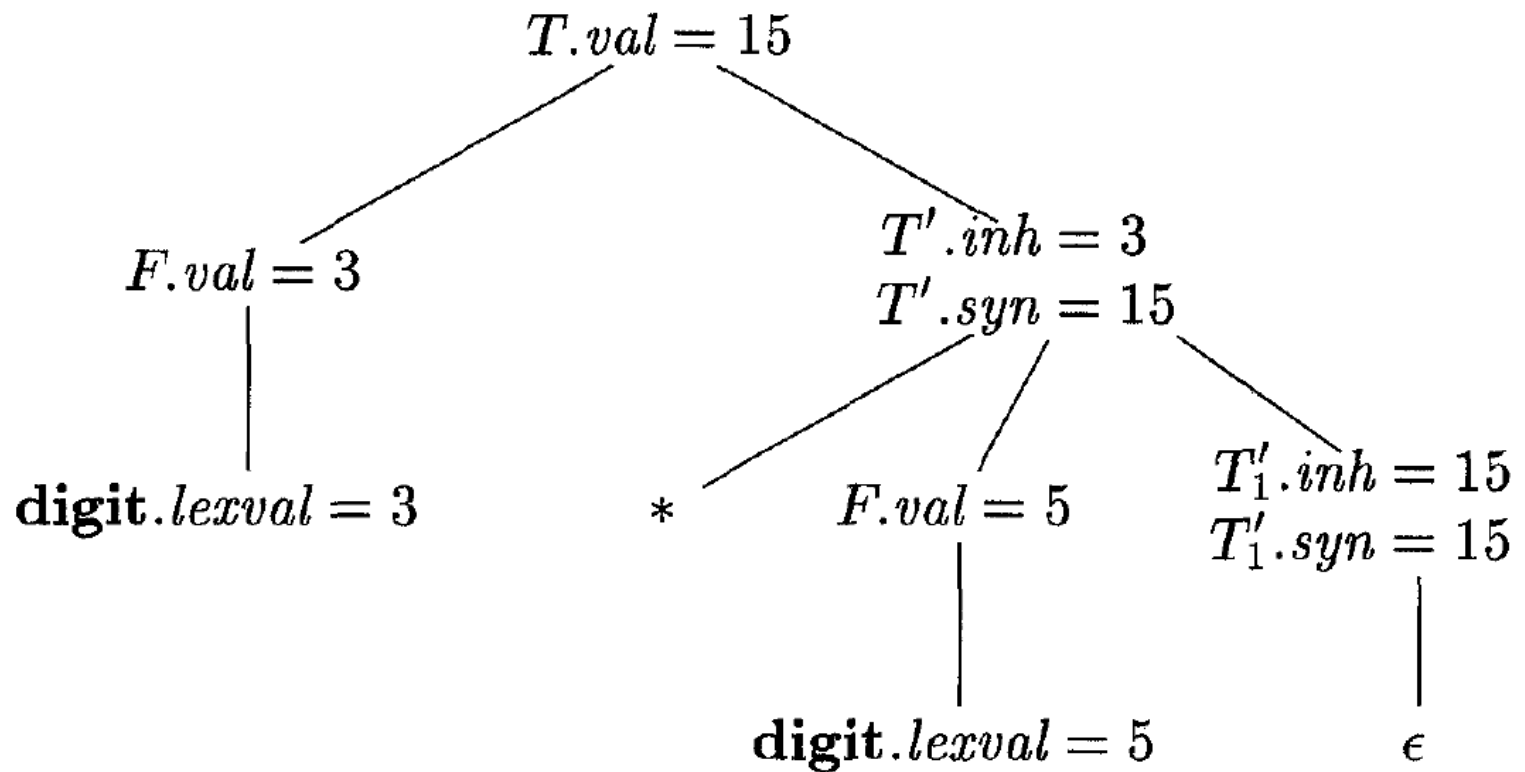
Atribute sintetizate



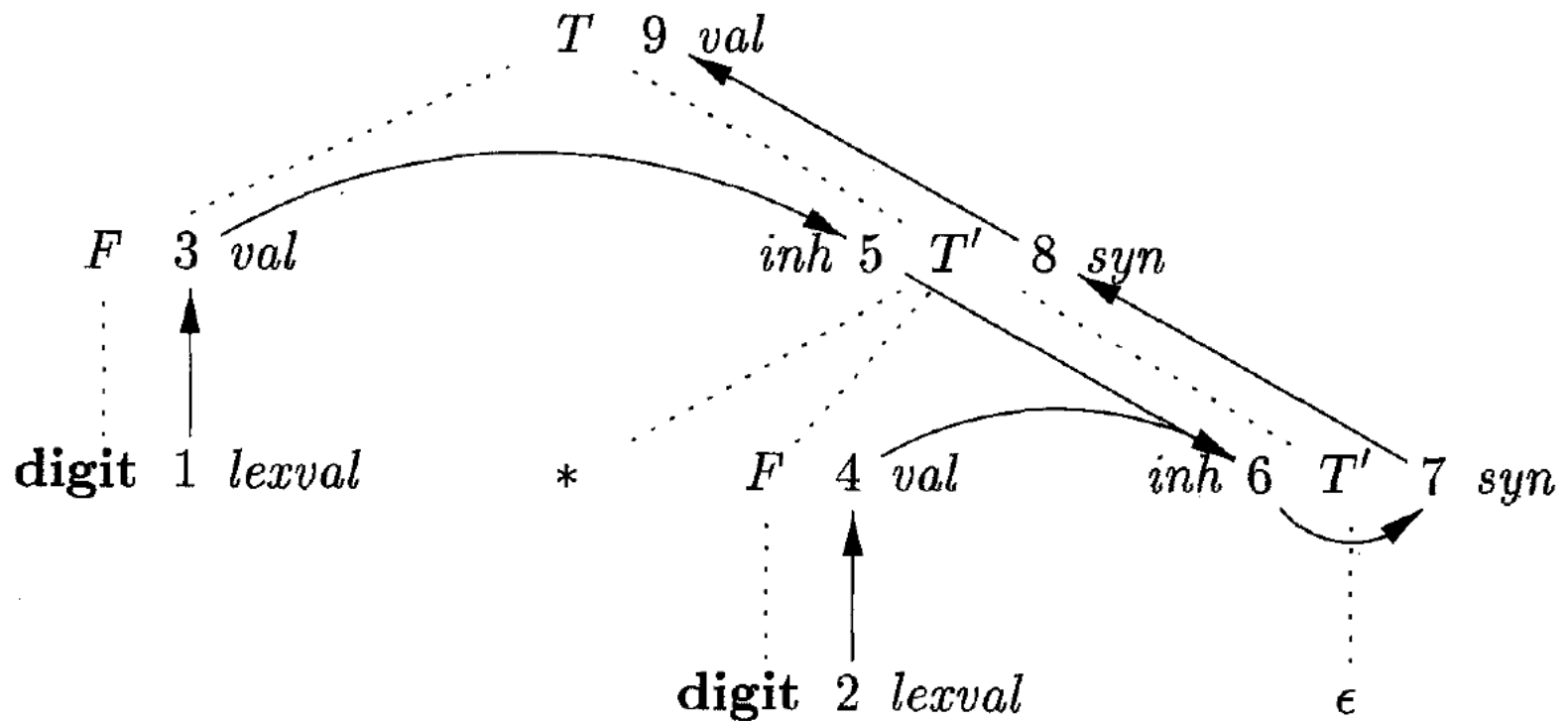
Attribute moştenite

$T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow^* FT'_1$	$T'_1.inh = T'.inh * F.val$ $T'.syn = T'_1.syn$
$T' \rightarrow \epsilon$	$T'.syn = T'.inh$
$F \rightarrow cifra$	$F.val = cifra.val$

Attribute moștenite



Ordinea de evaluare în DDS



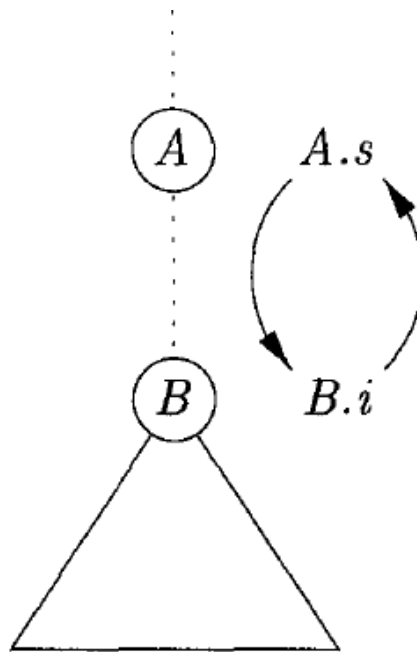
Ordinea de evaluare în DDS

- ▶ Este dată de un graf de dependență (orientat) care descrie fluxul de informație dintre attribute
 - 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 dependență de la X.a la B.c.

Ordinea de evaluare în DDS – circuite

$A \rightarrow B$

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



Ordinea de evaluare în DDS

- ▶ Determinarea existenței unor arbori de parsare pentru care grafurile de dependență 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 dependență are o muchie de la nodul M la nodul N, atunci atributul corespunzător lui M trebuie evaluat înaintea atributului din N
- ▶ Graful de dependență 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 attributele 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 $A \rightarrow X_1 X_2 \dots X_i \dots X_n$)
 - 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 $E_1 + E_2$ conține trei câmpuri: unul pentru operator și câte unul pentru fiecare operand

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

Aplicații pentru TDS

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

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

- ▶ **$S \rightarrow \text{while} (C) S1$**
 - `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

- ▶ $S \rightarrow \text{while (}$
 - `{ 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

- ▶ **Conținutul acestei prelegeri se bazează pe cartea**
 - **Compilers – Principles Techniques and Tools by *Alfred Aho, Monica Lam, Ravi Sethi, Jeffrey Ullman* – Second Edition**