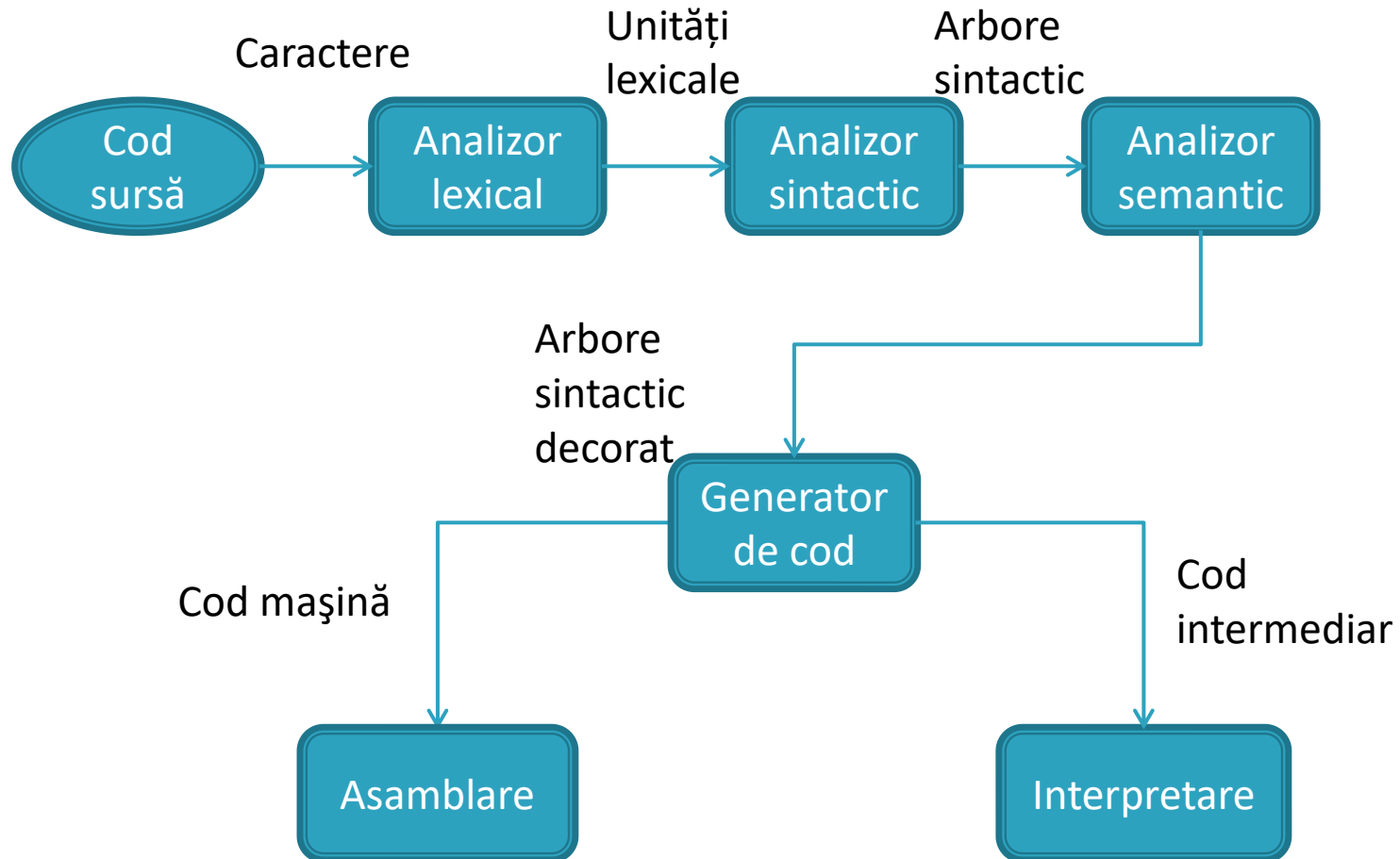


Limbaje formale, automate și compilatoare

Curs 12

Compilare



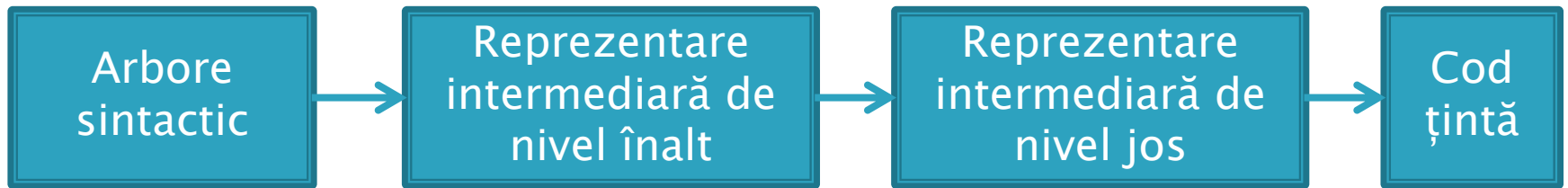
Recapitulare

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

Generare de cod

- ▶ Transformă analiza semantică în cod executabil
- ▶ Limbajul trebuie transformat diferit pentru orice sistem
 - Generare de cod intermediar – detașează partea de analiză și interpretare de generarea de cod mașină

Generare de cod



Compilatoarele pot folosi multiple straturi de reprezentări intermediare.

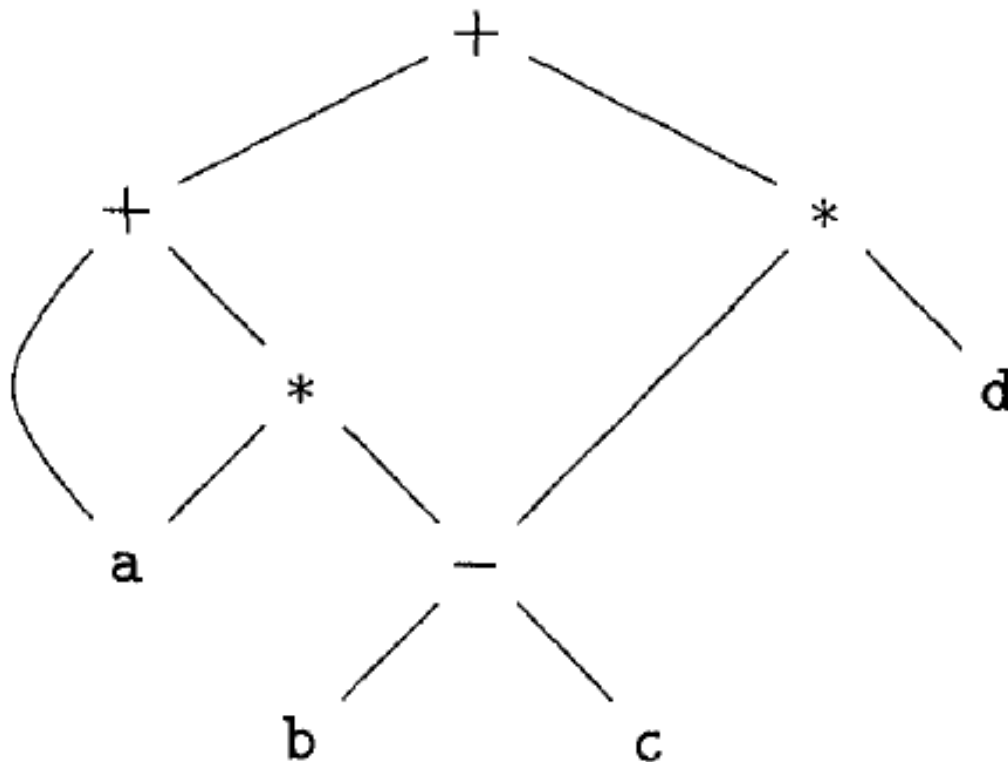
Cod intermediar

- ▶ Arbori sintactici decorați
- ▶ Cod cu trei adrese
 - $x = y \text{ op } z$
- ▶ Cod intermediar
 - De nivel înalt
 - apropiați de limbajul sursă (arbori sintactici)
 - buni pentru sarcini de nivel înalt (ex. verificare de tip)
 - De nivel jos
 - alocarea de memorie și regiștri
 - selecția de instrucțiuni

Arbori sintattici

► Grafuri orientate aciclice

- $a + a * (b - c) + (b - c) * d$



Grafuri aciclice orientate

- ▶ Gramatica de mai jos poate construi arbori sintactici sau arbori aciclici orientați
- ▶ Funcțiile *Leaf* și *Node* vor crea noduri noi dacă nu există deja noduri egale
- ▶ Dacă nodul există deja, va fi returnat în locul unui nod nou

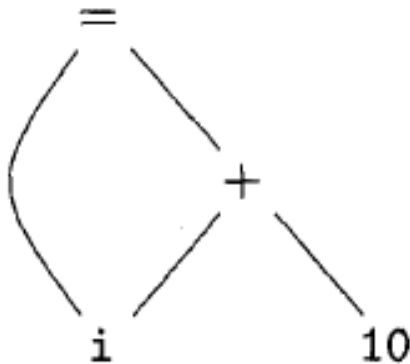
1) $E \rightarrow E_1 + T$	$E.node = \text{new Node}('+', E_1.node, T.node)$
2) $E \rightarrow E_1 - T$	$E.node = \text{new Node}('-', E_1.node, T.node)$
3) $E \rightarrow T$	$E.node = T.node$
4) $T \rightarrow (E)$	$T.node = E.node$
5) $T \rightarrow \text{id}$	$T.node = \text{new Leaf}(\text{id}, \text{id.entry})$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

Grafuri aciclice orientate

- 1) $p_1 = \text{Leaf}(\mathbf{id}, \text{entry-}a)$
- 2) $p_2 = \text{Leaf}(\mathbf{id}, \text{entry-}a) = p_1$
- 3) $p_3 = \text{Leaf}(\mathbf{id}, \text{entry-}b)$
- 4) $p_4 = \text{Leaf}(\mathbf{id}, \text{entry-}c)$
- 5) $p_5 = \text{Node}('-', p_3, p_4)$
- 6) $p_6 = \text{Node}('*', p_1, p_5)$
- 7) $p_7 = \text{Node}('+', p_1, p_6)$
- 8) $p_8 = \text{Leaf}(\mathbf{id}, \text{entry-}b) = p_3$
- 9) $p_9 = \text{Leaf}(\mathbf{id}, \text{entry-}c) = p_4$
- 10) $p_{10} = \text{Node}('-', p_3, p_4) = p_5$
- 11) $p_{11} = \text{Leaf}(\mathbf{id}, \text{entry-}d)$
- 12) $p_{12} = \text{Node}('*', p_5, p_{11})$
- 13) $p_{13} = \text{Node}('+', p_7, p_{12})$

Grafuri aciclice orientate – reprezentare cu tablouri

- ▶ Nodurile arborelui sunt reprezentate într-un tablou
 - Fiecare rând reprezintă un nod
 - Prima celulă reprezintă operatorul
 - Fiecare nod are o valoare asociată (pointer sau constantă)



1	id	→ to entry for i	
2	num	10	
3	+	1	2
4	=	1	3
5	...		

Grafuri aciclice orientate – reprezentare cu tablouri

- ▶ Nodurile sunt referite prin indicele rândului
 - Indicii nodului sau expresiei sunt numiți *value number*
 - În practică sunt folosiți pointeri
 - Valorile numerice pot fi folosite pentru a construi grafuri aciclice orientate

Metoda value number de construcție de arbori orientați

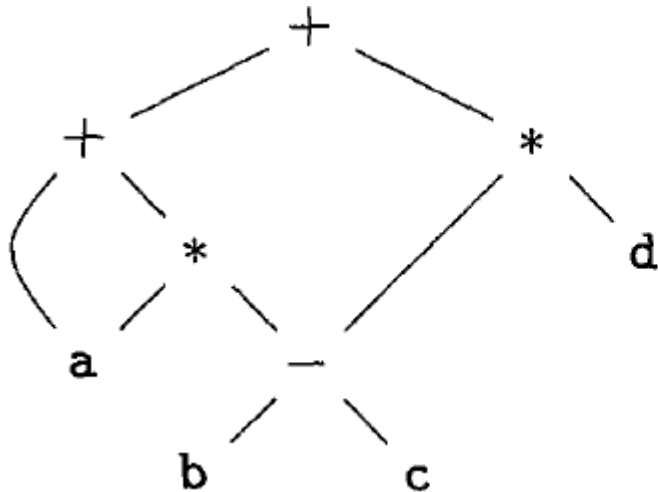
- ▶ **Intrare:** eticheta operatorului și membrii operației
- ▶ **Ieșire:** Valoarea numerică pentru nodul cu semnatura $\langle op, l, r \rangle$
- ▶ **Algoritm**
 - Caută în tablou nodul M cu eticheta op, și fiii stânga – dreapta l și r
 - Dacă există, return M
 - Altfel, creează un nou nod N, cu eticheta op și fiii l și r; return N

Coduri cu trei adrese

- ▶ Reprezentat ca o secvență de maxim trei elemente
 - Operator
 - Operand stânga
 - Operand dreapta
 - Ex: $a + b * c$
 - $t1 = b * c$
 - $t2 = a + t1$
- ▶ Poate fi folosit pentru descompunerea de operații aritmetice, instrucțiuni de control etc.
- ▶ Util pentru generare de cod și optimizare

Coduri cu trei adrese

- ▶ Este o descriere liniară a unui arbore sintactic sau a unui arbore orientat
- ▶ Pentru graful discutat anterior, traducerea în cod cu trei adrese este dată mai jos



```
t1  =  b - c
t2  =  a * t1
t3  =  a + t2
t4  =  t1 * d
t5  =  t3 + t4
```

Adrese și instrucțiuni

- ▶ Codul cu trei adrese este construit pe baza a două concepte: adrese și instrucțiuni
- ▶ O adresă poate fi:
 - Nume (va fi înlocuit cu o adresă de memorie la generare)
 - Constante (sunt necesare conversii de tip)
 - Variabile temporare generate de compilator (folosite în general pentru optimizări)

Instrucțiuni pentru cod cu trei adrese

- ▶ Atribuirii de forma $c = a \text{ op } b$
- ▶ Atribuirii de forma $a = \text{op } b$
- ▶ Copiere: $a = b$
- ▶ Salt: *goto L*; instrucțiunea cu adresa L va fi următoarea executată
- ▶ Salturi condiționale: *if a goto L* sau *if false a goto L*
- ▶ Salturi condiționale: *if a relop b goto L*; dacă a este în relația *relop* cu b, execută instrucțiunea L, dacă nu execută următoarea adresă după instrucțiunea curentă

Instrucțiuni pentru cod cu trei adrese

- ▶ Apeluri de funcții:
 - param x1
 - param x2
 - ...
 - param xn
 - call p, n
- ▶ Copiere indexată: $a = b[i]$ sau $a[i] = b$
 - $a = b[i]$ modifică valoarea adresei care vine la i poziții după adresa lui b
- ▶ Asignare de adrese sau pointeri: $a = \&b$, $a = *b$, $*a = b$

Exemplu

- ▶ Fie instrucțiunea
 - do $i = i + 1$; while ($a[i] < v$);

```
L:  t1 = i + 1  
    i = t1  
    t2 = i * 8  
    t3 = a [ t2 ]  
    if t3 < v goto L
```

```
100: t1 = i + 1  
101: i = t1  
102: t2 = i * 8  
103: t3 = a [ t2 ]  
104: if t3 < v goto 100
```

Tipuri și declarații (1)

- ▶ Verificări și traduceri
 - Verificarea are în vedere compatibilitatea de tipuri
 - Traducerea are în vedere spatiul necesar pentru memorarea unei variabile
- ▶ Expresii pentru tipuri.
 - Un tip de bază
 - Un nume de tip
 - O înregistrare este o structură de date cu câmpuri cu nume
 - Expresii pentru tipuri pot fi formate cu constructorul pentru funcții. $s \rightarrow t$ semnifică o funcție de la tipul s la tipul t

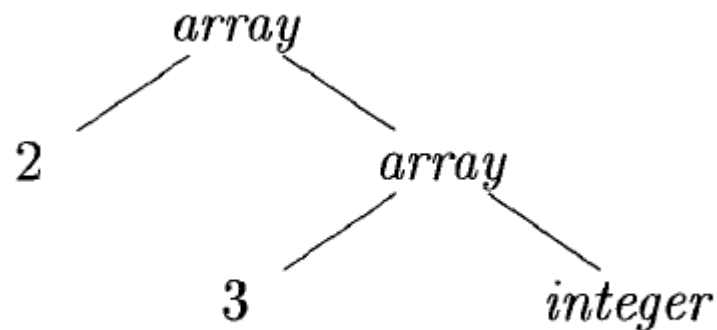
Tipuri și declarații (2)

- ▶ Expresii pentru tipuri
 - Dacă s și t sunt expresii pentru tipuri, atunci sxt este expresie pentru tip
 - Expresiile pentru tip pot conține variabile ale căror valori sunt expresii pentru tip
- ▶ Expresiile pentru tip pot fi reprezentate ca un graf aciclic orientat
 - Nodurile interne sunt constructori
 - Frunzele sunt expresii de tip elementare

Tipuri și declarații (3)

► Exemplu

- *int[2][3]*
- Este interpretată ca un tablou de tablouri de int
- Expresia pentru tip echivalentă este `array(2, array(3, integer))`



Echivalența de tipuri (1)

- ▶ Calculată cu reguli de potrivire
 - Dacă două expresii sunt egale, atunci returnează tipul, altfel eroare
 - Pot să apară ambiguități atunci când tipurile sunt redenumite, iar redenumirile sunt folosite în expresii pentru tip

Echivalența de tipuri (2)

- ▶ În cazul reprezentării cu grafuri aciclice orientate, două tipuri sunt echivalente structural dacă și numai dacă
 - Sunt același tip de bază
 - Sunt obținute prin aplicarea aceluiași constructor peste tipuri echivalente structural
 - Unul este o redenumire al celuilalt

Stocarea locală de tipuri (1)

- ▶ Știind tipul unui identificador, putem determina cantitatea de memorie necesară acelui identificador
 - La compilare sunt folosite pentru a asocia fiecărui identificador o adresă relativă (memorate în tabelul de valori)
 - Pentru tipuri de date cu dimensiune variabilă (ex. string) sau a căror dimensiune nu poate fi cunoscută până la runtime, este alocată o cantitate de memorie pornind de la o adresă

Stocarea locală de tipuri (2)

$$\begin{array}{l} T \rightarrow B \\ C \end{array} \quad \{ t = B.type; w = B.width; \}$$

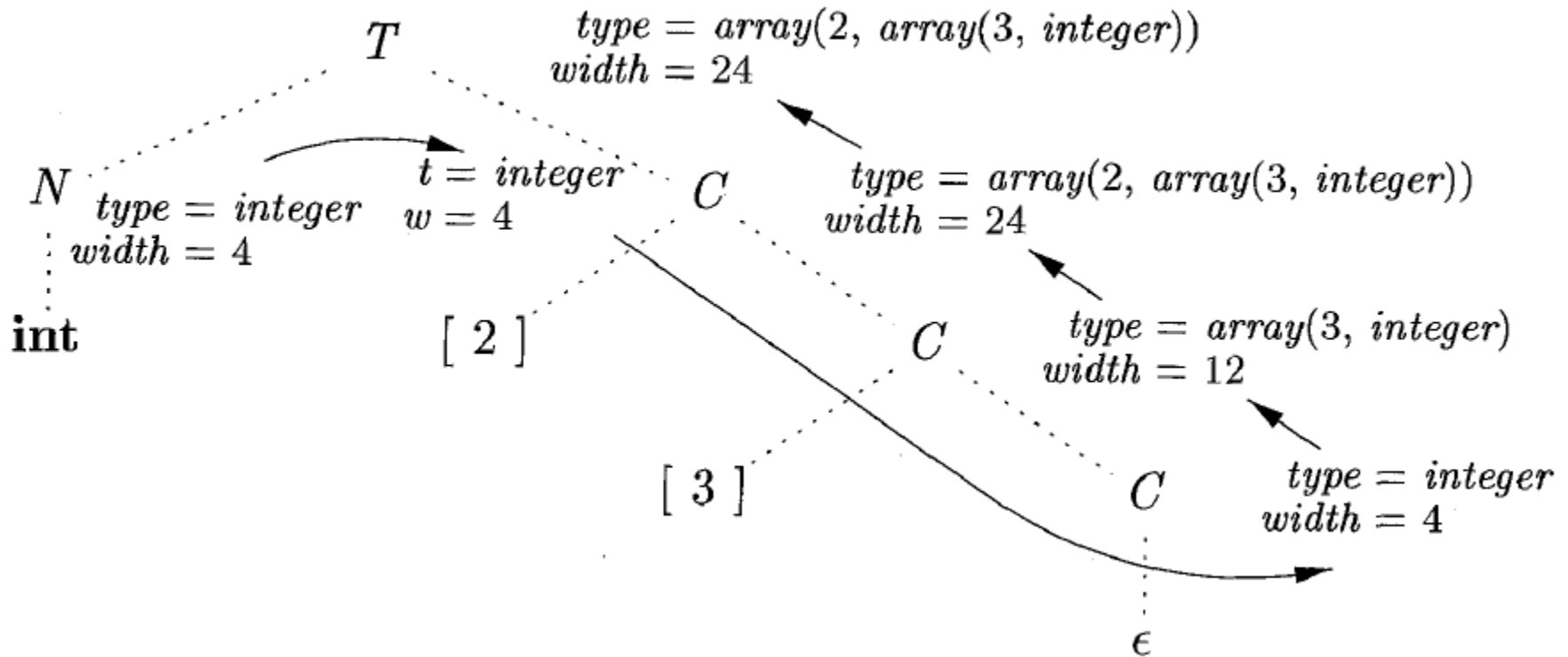
$$B \rightarrow \mathbf{int} \quad \{ B.type = integer; B.width = 4; \}$$

$$B \rightarrow \mathbf{float} \quad \{ B.type = float; B.width = 8; \}$$

$$C \rightarrow \epsilon \quad \{ C.type = t; C.width = w; \}$$

$$C \rightarrow [\mathbf{num}] C_1 \quad \{ \text{array}(\mathbf{num.value}, C_1.type); \\ C.width = \mathbf{num.value} \times C_1.width; \}$$

Stocarea locală de tipuri (3)



Bibliografie

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