

Compilatoare

Curs 1



Cursul acesta

- Introducere/Administrative
- Structura compilatoarelor
- Fazele compilarii
- Analiza sintactica

Sa ne cunoastem

Curs + Laboratoare + Teme

Bogdan Nițulescu – bogdan.nitulescu@cs.pub.ro

Cristina Ciocan

Răzvan Crainea

Sorin Baltateanu

Octavian Moraru

Lavinia Ghica

Raluca Grigoraș

Silviu Grigore

Ionuț Guduleasa

Diana Picuș

Irina Preșa

Laura Vasilescu

Ce asteptari aveti de la curs?

- De ce ati ales materia aceasta
- Ce credeti ca se va face
- Ce v-ar place voua sa se faca

Ce vom face la CPL in acest an

- Structura generala a unui compilator
- Ceva analiza lexicala
- Analiza sintactica, semantica
- Generare de cod intermediar
- Optimizari
- Garbage collection, Just-In-Time compilers
- (ce mai vreti voi sa aflati)

Cunostinte necesare

- Limbaje Formale si Automate.
- Arhitecturi cu microprocesoare.
- Programare in limbaj de asamblare.
- Tehnici avansate de programare in limbaje de nivel inalt.

Tehnologii

- Limbaje de programare: **C** si **C++**
- Parsare: **flex** si **bison**
- Generare de cod si optimizari: **LLVM**

Informatii despre curs

- Pagina de web:
<http://ocw.cs.pub.ro/courses/cpl>
- Cursuri, Laboratoare, Teme, Wiki,
Grup de discutii
- Adresele noastre de e-mail, pe pagina de
web

Despre teme

- Termene stricte si penalizari pentru intarziere.
- Punctaj divizat - teste publice, teste private, la latitudinea asistentului
 - Tot punctajul e de fapt la latitudinea asistentului – el decide daca tema indeplineste cerintele!
 - Incepeti din timp – nu lasati pe ultimul moment
 - Nu uitati ca un programator bun scrie codul a.i. sa poata fi citit cu usurinta de altii!
- Cititi sectiunea de 'Reguli'
 - Luatul de pe net e considerat "copiere"! (scopul e sa invatati, nu sa terminati cat mai repede)

Notare

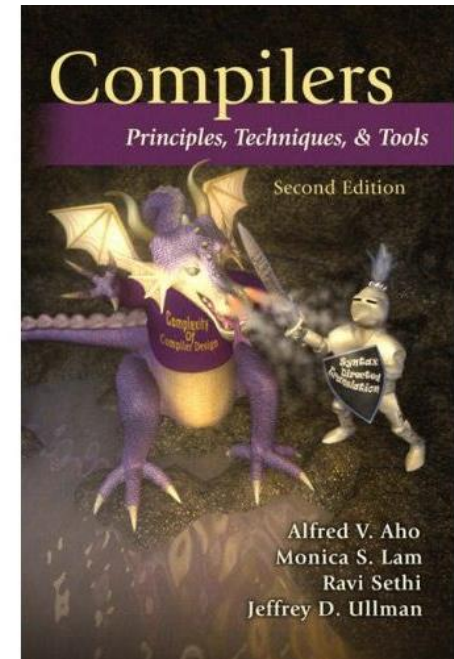
- 60% laborator, 40% examenul final
 - 50% teme
 - 10% activitatea la laborator
 - 40% examen
- 4 teme + concurs – 62.5p – 50% din nota finala
 - 50p + bonus pentru cei harnici
- Toate temele foarte bune (≥ 60 p) -> nota 10 fara examen pentru primii 5 clasati
- Anul acesta aveti nevoie de 50% din punctajul din timpul semestrului, pentru a trece (!)
- Activitatea la laborator – teste in timpul laboratorului, aprecierea asistentului

Pentru mai multe informatii

“Dragon Book”, 2nd edition

Compilers – Principles, Techniques and Tools

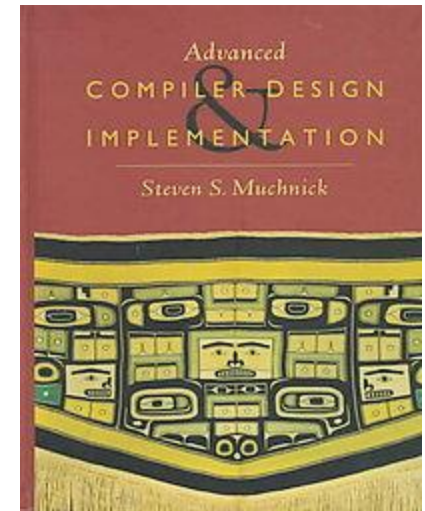
Aho, Lam, Sethi & Ullman



Exista un numar mare de cursuri online bazate pe Dragon Book
(de ex. coursera.org)

Pentru mai multe informatii

Advanced Compiler Design &
Implementation
Steven Muchnick



...ofera mai multe amanunte pe partea de optimizari

Cursul acesta

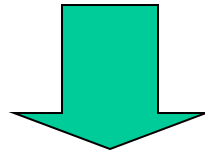
- Introducere/Administrative
- Structura compilatoarelor
- Fazele compilării
- Analiza sintactica

De ce un curs de compilatoare

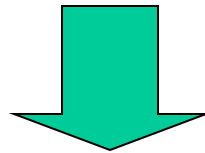
- Scopul cursului: cum se implementeaza un limbaj de programare.
- De ce e nevoie de multe limbaje...
 - Fiecare domeniu de aplicatie are cerintele proprii (aplicatii de sistem, baze de date, calcul stiintific, aplicatii web)
- Dinamica limbajelor de programare
 - Este relativ usor de scris un limbaj nou.
 - Este greu de modificat un limbaj popular.
 - Apar noi domenii de aplicatii.
 - Limbaje legate de platforma (Java, C#, Android, OS-X/iOS)

Domenii conexe

Paradigme de programare – procedurala, obiectuala, generica, functionala...



Implementarea limbajelor de programare

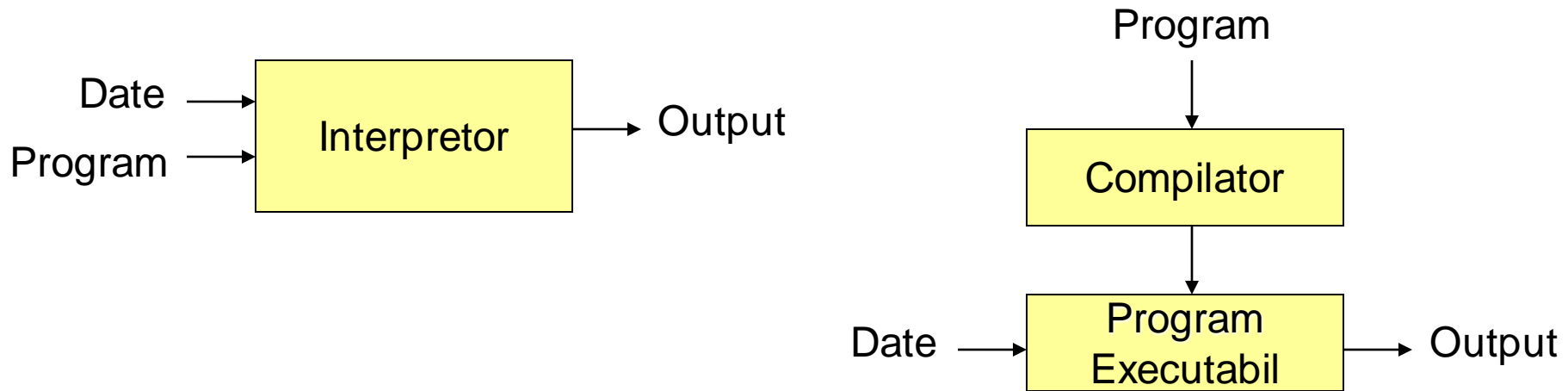


Sisteme de operare | Microarhitecturi

De ce un curs de compilatoare

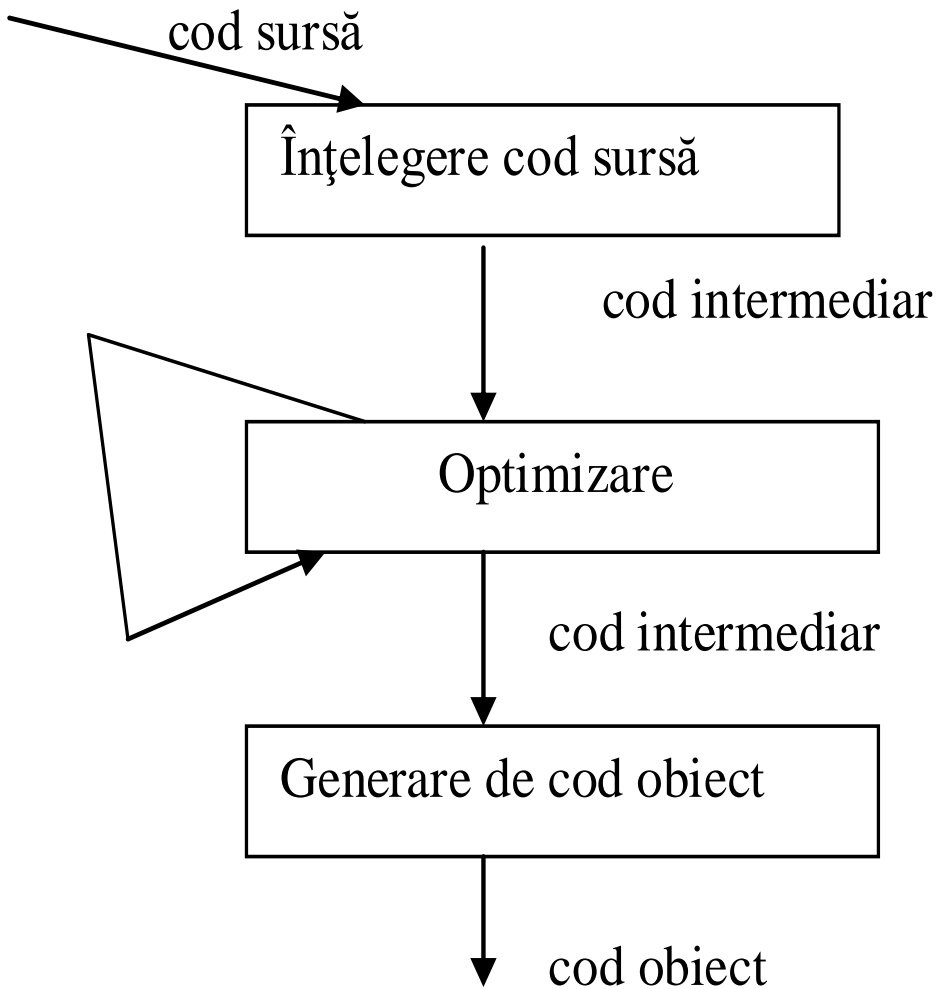
- Sunt peste tot
 - Toolchains; IDE-uri; scripting; domain specific languages; baze de date; tehnoredactare; servere HTTP; framework-uri web; browsere; drivere video...
- Combina cunostinte de hardware, software, inginerie, matematica
- Multe probleme grele/complexe, implica multa teorie dar si practica
- Compilatoarele "ghideaza" procesoarele viitoare!

Translatare vs. interpretare



- Compilatoarele transforma specificatiile
- Interpretoarele executa specificatiile
- Amandoua trebuie sa 'inteleaga' specificatiile!
- Combinatii: emulatoare, masini virtuale, just in time

Structura generala

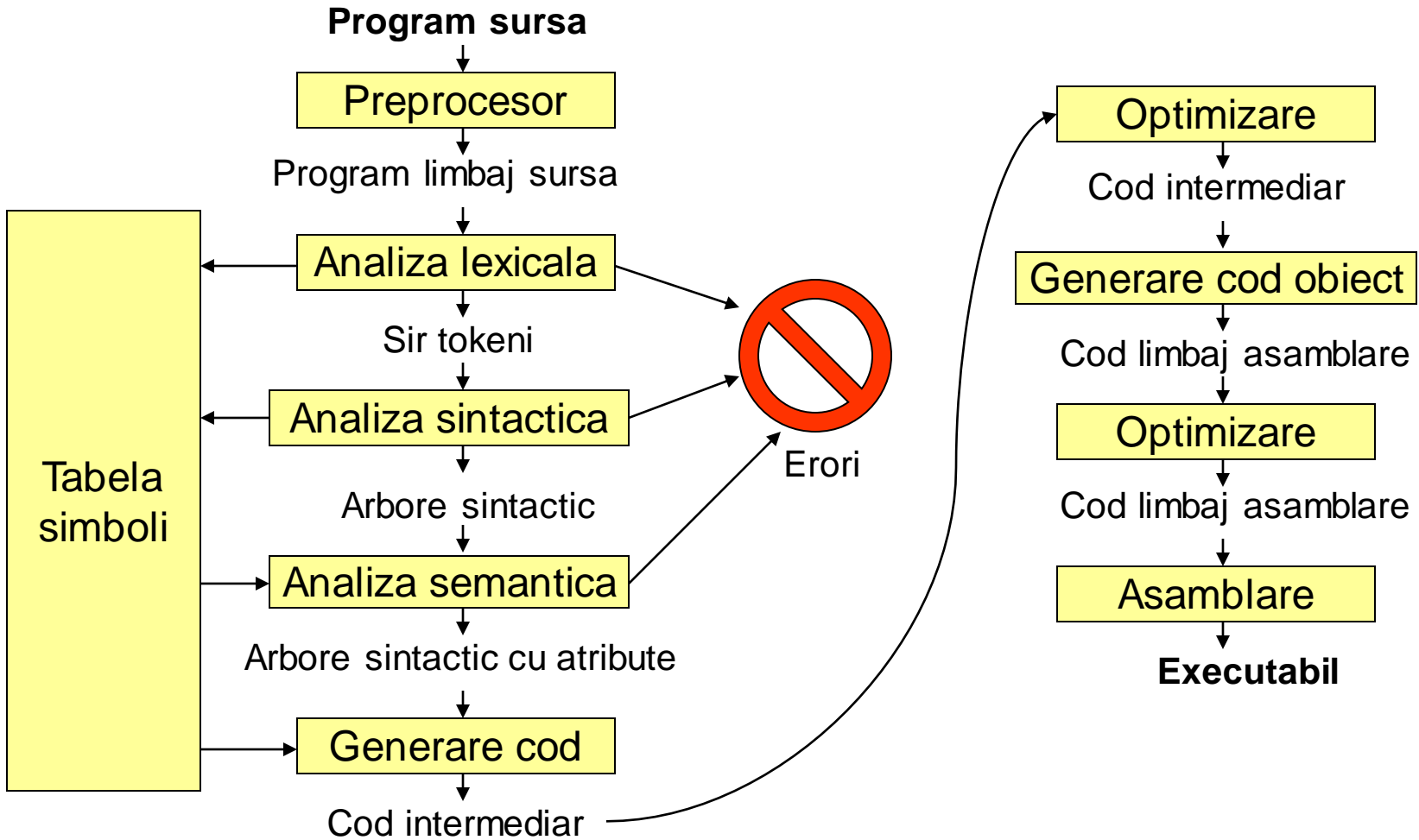


Front end
(independent de mașină)

optimizare
(independent de limbaj și mașină)

Back end
(independent de limbaj)

Structura detaliata



Cursul acesta

- Introducere/Administrative
- Structura compilatoarelor
- Fazele compilarii
- Analiza sintactica

Fazele compilării

- Preprocesorul - macro substituție, eliminarea comentariilor, etc.
- Analiză+generare de cod=componenta principală a traducerii
 - Se verifica corectitudinea formala a textului programului
 - Se traduce acest text într-o alta forma
- Optimizari = îmbunătățirea calitatii traducerii (performanțelor programului tradus).
- Fazele nu sunt neapărat clar separate ci sunt realizate printr-un ansamblu de funcții care cooperează

Analiza lexicala

- Automate finite, expresii regulate – LFA!!!
- Imparte programul in unitati lexicale (atomi lexicali, tokeni)
 - Cuvinte cheie
 - Numere
 - Nume
- Prima decizie – stabilirea atomilor lexicali ai limbajului

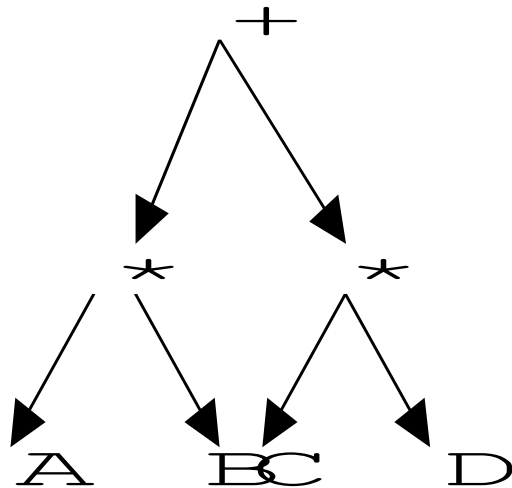
Analiza lexicala (2)

- Un atom lexical are
 - Clasa (cod numeric) – folosit in analiza sintactica
 - Attribute specifice clasei – analiza semantica, generare de cod
- De obicei, analizorul lexical are o interfata simpla
 - `lexer.getNextToken()`

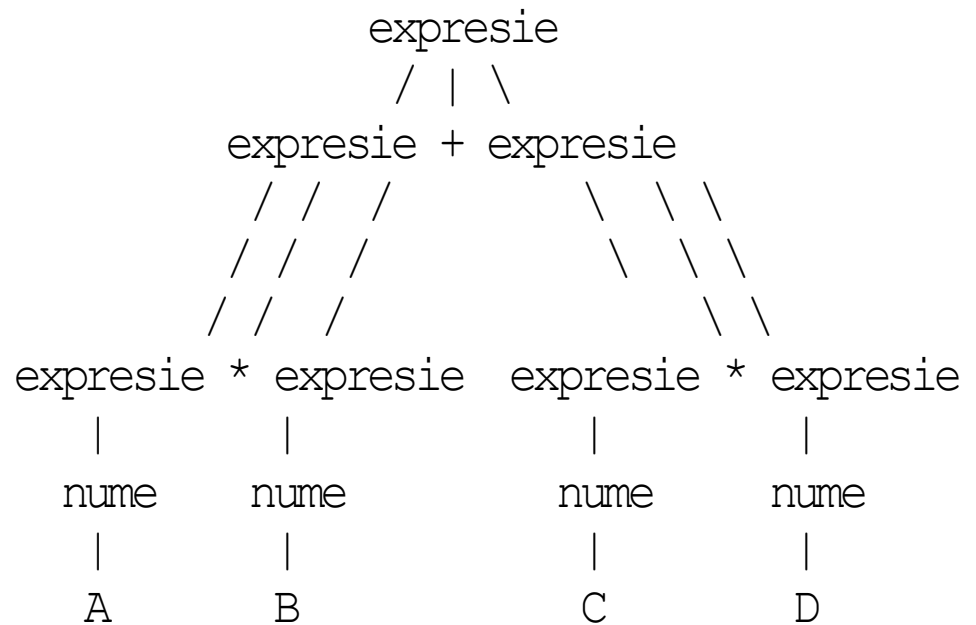
Analiza sintactica

- Descompune textul programului sursa în componentele sale "gramaticale"

• $A * B + C * D$

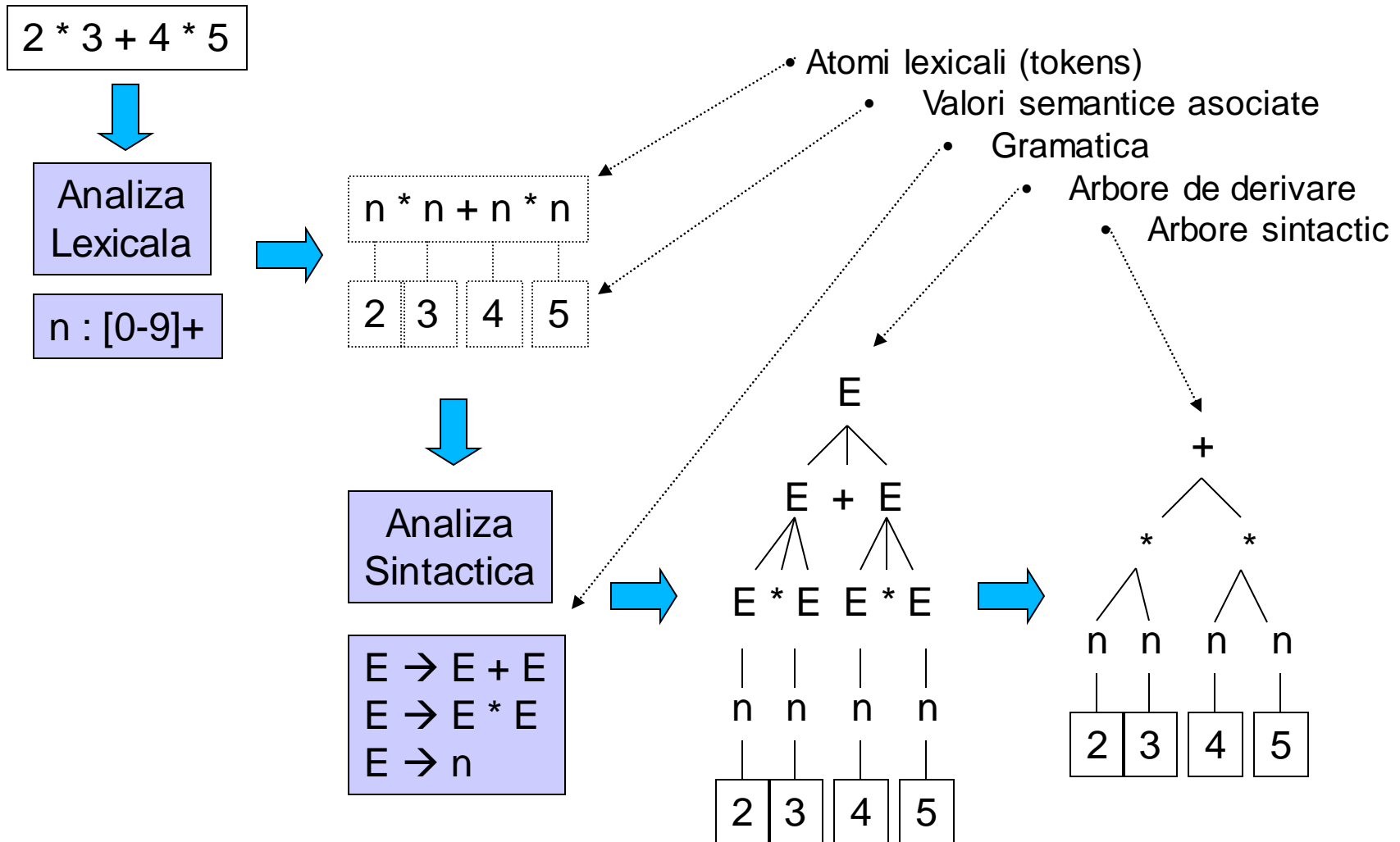


Arborele sintactic



Arborele de derivare

Arbore de derivare / sintactic



Analiza semantica

- O mare parte e integrata de obicei in analiza sintactica
- Verificarea corectitudinii semantice a propozitiilor
 - In compilatoare – verificarea consistentei programului, verificari de tip
- Adnoteaza arborele de derivare cu informatii semantice (de tip)
 - Pregateste generarea de cod

Tabela de simbolii

- identificatorii utilizați în program și informații despre acestia.
- Pt. fiecare identificator - câmpuri pentru atributele posibile.
 - Tip
 - Domeniu de valabilitate
 - Signatura (pt funcții) – nr. și tipul argumentelor, modul de transfer, tipul rezultatului
- Introducerea simbolilor în tabelă se face de către analizorul lexical.
- Atributele sunt completate în tabelă de către analizoarele sintactic și semantic.

Detectarea erorilor

- Analiza lexicală - șir care nu corespunde unui atom lexical
 - 0x1234ABFG
- Analiza sintactică - erori legate de structura instrucțiunilor
 - `int real alfa; /* corect lexical, nu sintactic */`
- Analiză semantică – erori semantice
 - `A=b+c; /*incorect daca b e de tip 'struct' */`
- Recuperarea din eroare – cum continuam analiza cand am intanit o eroare

Generarea de cod intermediar

- Uneori, direct in timpul parsarii
 - Sau prin parcurgerea arborelui de derivare/sintactic
- “cod obiect pentru o masina virtuala”
 - Permite multe optimizari comune pentru diferite frontend-uri (limbaje) si backend-uri (procesoare)
 - Unele optimizari se pot face si direct pe arbore
- $N * M$ vs. $N + M$

Optimizari pe codul intermediar

- Propagarea constantelor
 - `int sec = ore*60*60;`
- Calcularea expresiilor comune
 - `int a = x+y+z, b=x+y+t;`
- Strength reduction
 - `Int a=b*2,c=d%8`
- Scoaterea expresiilor constante in afara buclelor
- Etc.

Generarea de cod obiect

- Maparea instructiunilor din IR pe instructiunile procesorului destinatie
- Poate implica optimizari dependente de masina
- Asamblarea (codificarea) instructiunilor
- Linking

Cursul acesta

- Introducere/Administrative
- Structura compilatoarelor
- Fazele compilarii
- Analiza sintactica

Limbaje si gramatici

- Ce este o gramatica?
- Dar un limbaj?
- Tipuri de gramatici
- Ierarhia Chomsky a gramaticilor
 - Regulate
 - Independente de context
 - Dependente de context
 - Generice
- Unde se incadreaza un limbaj de programare?

Limbaje si gramatici

Programe fara erori lexicale – gramatica regulata

**Programe fara erori sintactice – gramatica
independenta de context**

**Programe fara erori la compilare –
gramatica dependenta de context**

Analiza sintactica

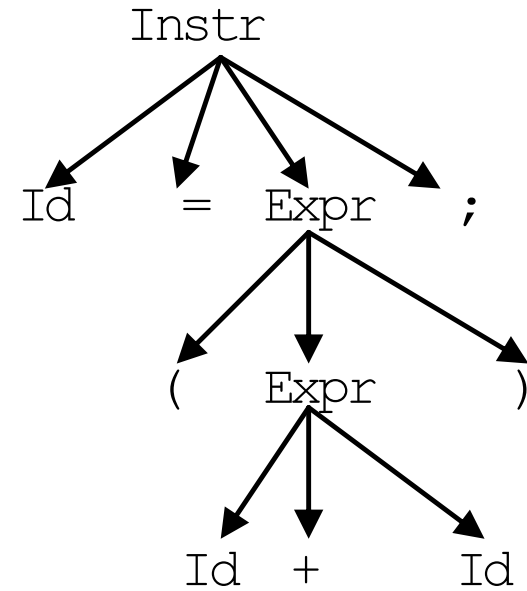
- Verifica formarea corecta (cf. gramaticii) a constructiilor din limbaj
 - Analiza lexicata – “cuvinte”
 - Analiza sintactica – “propozitii”
- Primeste un sir de atomi lexicali, construiește un arbore de derivare

Exemplu

- ALFA = (BETA + GAMA);

- id = (id + id);

- Instr \rightarrow id = Expr ;
 Expr \rightarrow Expr + Expr
 | Expr * Expr
 | (Expr)
 | id



Tipuri de analiza sintactica

- Descendentă (top-down, de sus în jos)
 - Înlocuiește câte un neterminat cu partea dreaptă a unei producții, până rămâne doar cu terminali
- Ascendentă (bottom-up, de jos în sus)
 - Porneste de la șirul de atomi lexicali, abstractizează din șir simbolul de start prin reduceri succesive
- Analiza descendentă – derivare stângă
 - Tot timpul înlocuim cel mai din stânga neterminat
- Analiza ascendentă - derivarea dreaptă
 - primul neterminat înlocuit este cel mai din dreapta din forma propozițională curentă

Top down vs bottom up

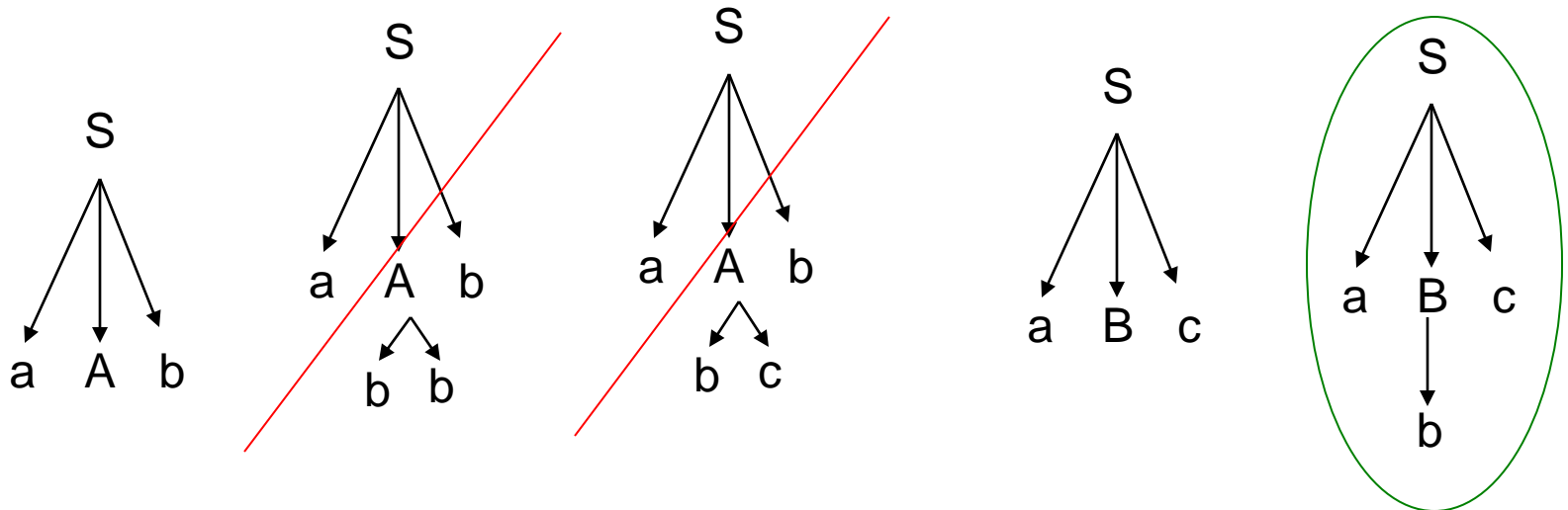
- Derivare stânga , top down:
 - $\text{Instr} \Rightarrow \text{id} = \text{Expr} ; \Rightarrow \text{id} = (\text{Expr}) ; \Rightarrow$
 $\text{id} = (\text{Expr} + \text{Expr}) ; \Rightarrow \text{id} = (\text{id} + \text{Expr}) ; \Rightarrow$
 $\text{id} = (\text{id} + \text{id}) ;$
- Derivare dreapta, bottom up:
 - $\text{id} = (\text{id} + \text{id}) ; \Leftarrow \text{id} = (\text{Expr} + \text{id}) ; \Leftarrow$
 $\text{id} = (\text{Expr} + \text{Expr}) ; \Leftarrow \text{id} = (\text{Expr}) ; \Leftarrow$
 $\text{id} = \text{Expr} ; \Leftarrow \text{Instr}$

Exemplul 2

- $S \rightarrow a A b \mid a B c$
- $A \rightarrow bb \mid bc$
- $B \rightarrow b$

Gramatica generează de fapt (abbb, abcb și abc).

Arborele de derivare pentru șirul abc (abordare descendentă):



Analiza LL, LR

- Vrem sa evitam backtrackingul
- O clasă de gramatici independente de context care permit o analiza deterministă.
 - Alg. LL(k) analizeaza left-to-right, derivare stanga
 - Alg. LR(k) analizeaza left-to-right, derivare dreapta
 - K – lookahead (cati tokeni sunt cititi)
- $LL(k) < LR(k)$
- Algoritmul folosit nu depinde de limbaj, gramatica da.