Compilatoare

Curs 1

CPL
IR crunch

Cursul acesta

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

Sa ne cunoastem

Curs + Laboratoare + Teme

Bogdan Nițulescu – bogdan.nitulescu@cs.pub.ro
Alex Guduleasa
Andrei Țuicu
Cristian Enciu

Diana Picuș
Lavinia Ghica
Marius Geantă
Mihai Pîrvu

CPL
IR crunch

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

CPL
IR crunch

Ce vom face la CPL in acest an

Cum se implementeaza un limbaj de programare

- Analiza lexicala, sintactica, semantica
- Generare de cod intermediar
- Sistemul de runtime
- 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.

CPL
IR crunch

Tehnologii

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

CPL
IR crunch

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 / 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

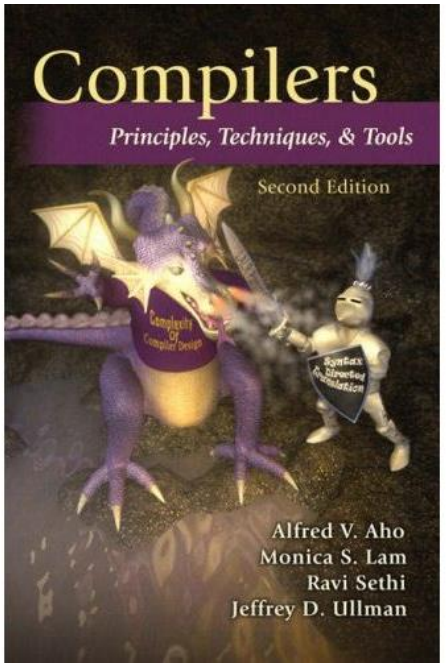
- 100 puncte = nota 10.
- Puteți strânge puncte la examen (40p), teme (50p), teste la curs (10p), laboratoare (10+p), participând la concurs (10p)
- Condiții de promovare: minim 20p examen, minim 30p în timpul semestrului.
- Nota 10 fara examen pentru primii clasati la concurs care au rezolvat foarte bine și temele.
- 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

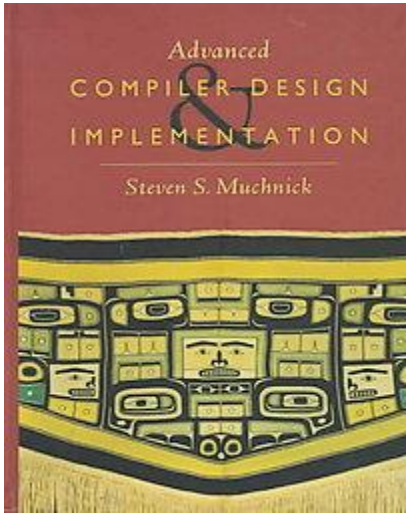


Exista un numar mare de cursuri online bazate pe Dragon Book
(de ex. coursera.org)

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 compilarii
- Analiza sintactica

De ce un curs de compilatoare

CPL
IR crunch

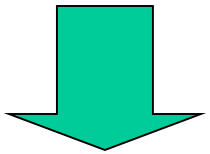
- Scopul cursului: cum se implementeaza un limbaj de programare.
- De ce e nevoie de multe limbaje…
  - Fiecare domeniu de aplicatie are cerintele proprii (aplicatii de sistem, baze de date, calcul stiintific, aplicatii web)
- Dinamica limbajelor de programare
  - Este relativ usor de scris un limbaj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

Domenii conexe

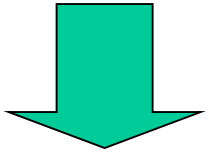
Designul limbajelor de programare

paradigme: procedurala, obiectuala, generica, functionala...

sisteme de tipuri | limbaje formale



Implementarea limbajelor de programare



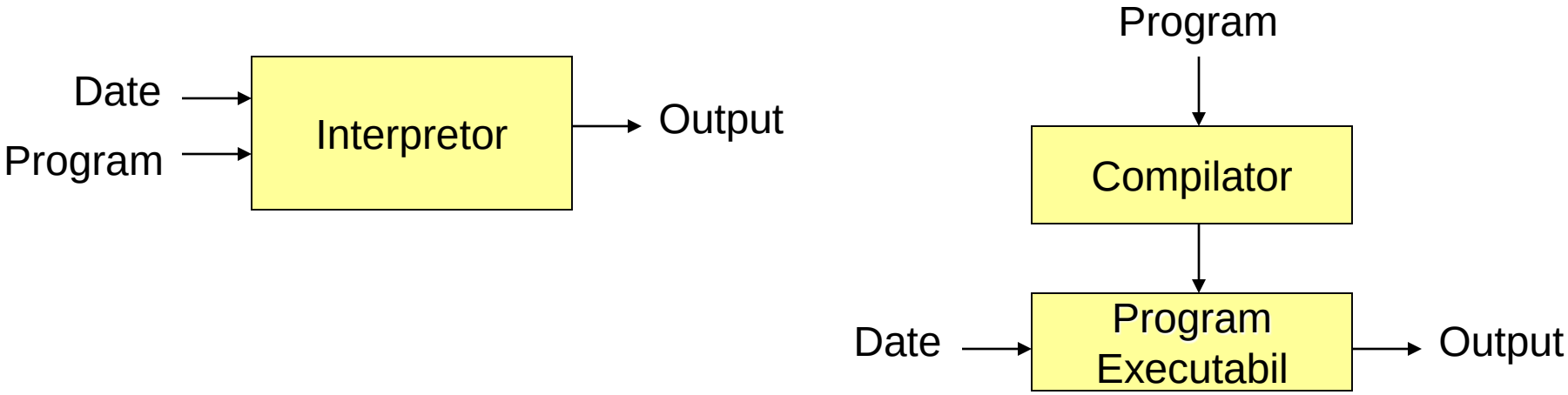
Sisteme de operare | Microarhitecturi

De ce un curs de compilatoare

CPL
IR crunch

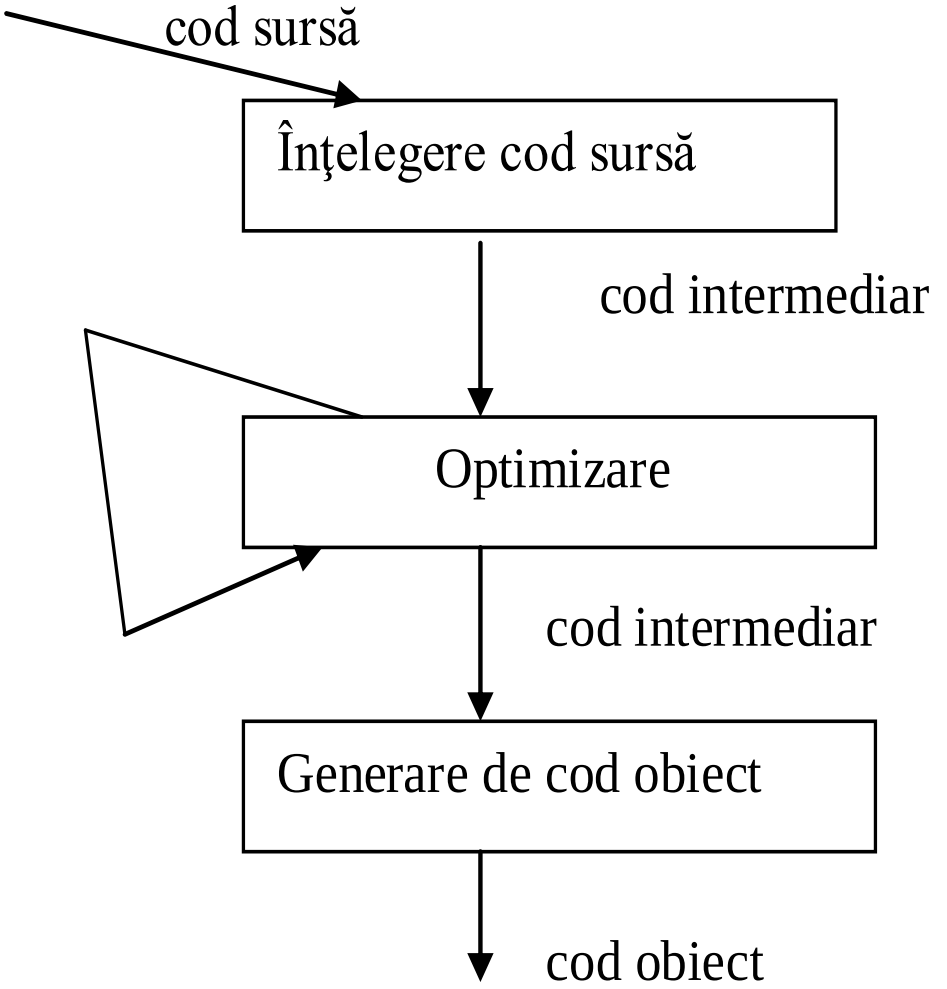
- Sunt peste tot
  - Toolchains; IDE-uri; analiza statică; execuție simbolică; scripting; domain specific languages; baze de date; tehnoredactare; servere HTTP; framework-uri web; browsere; drivere video
- Combina cunostinte de hardware, software, inginerie, matematica
- Multe probleme grele/complexe, 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

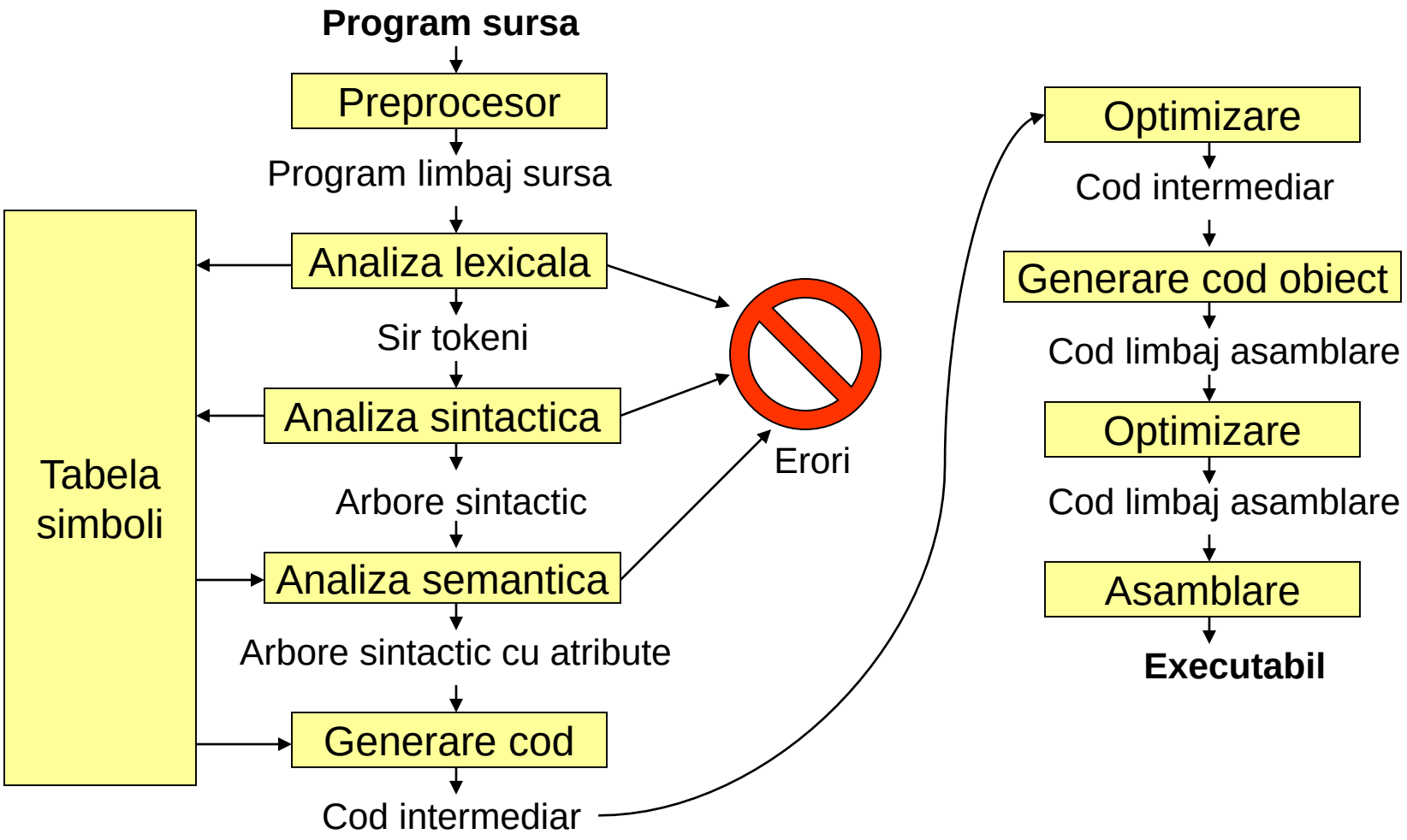


Front end
(independent de maşină)

optimizare
(independent de limbaj şi maşină)

Back end
(independent de limbaj)

Structura detaliata



Cursul acesta

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

CPL
IR crunch

Fazele compilarii

- Preprocesorul - macro substituţie, 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 = îmbunătăţirea calitatii traducerii (performanţelor programului tradus).
- Fazele nu sunt neapărat clar separate ci sunt realizate printr-un ansamblu de funcţii care cooperează

CPL
IR crunch

Analiza lexicala

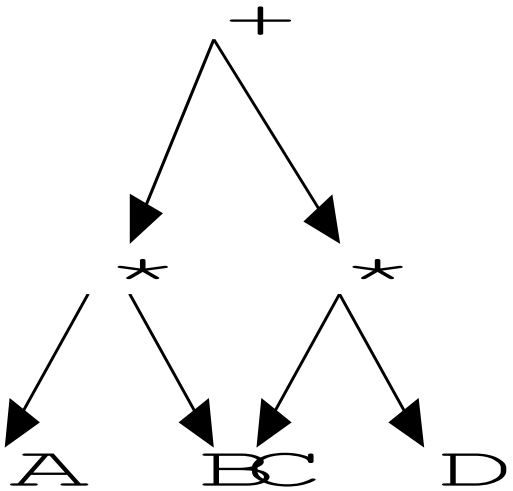
- Automate finite, expresii regulate – LFA!!!
- Imparte programul in unitati lexicale (atomi lexicali, tokeni)
  - Cuvinte cheie
  - Numere
  - Nume
- Prima decizie – stabilirea atomilor lexicali ai limbajului

Analiza lexicala (2)

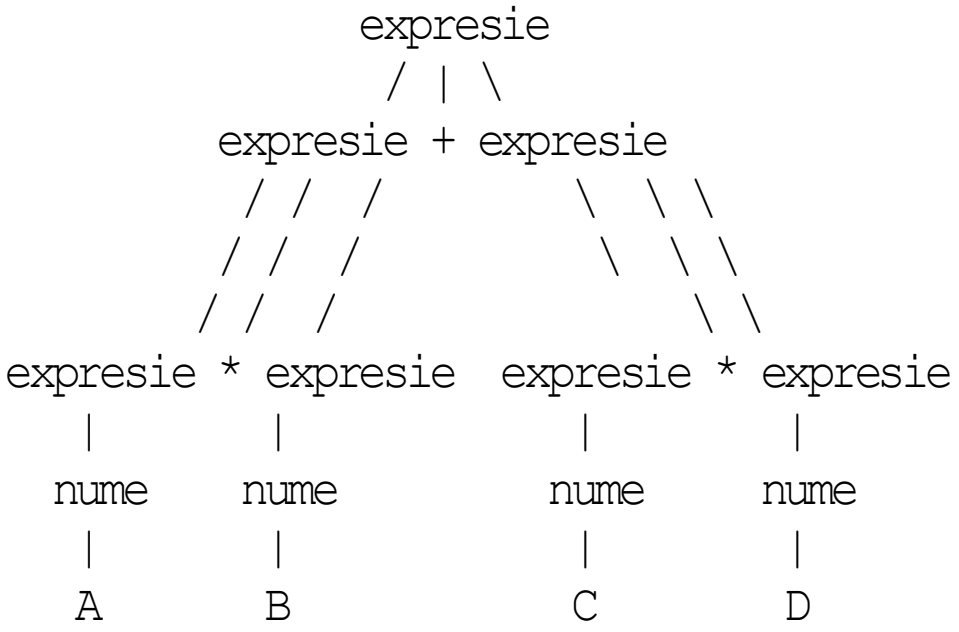
- Un atom lexical are
  - Clasa (cod numeric) – folosit in analiza sintactica
  - Atribute specifice clasei – analiza semantica, generare de cod
- De obicei, analizorul lexical are o interfata simpla
  - lexer.getNextToken()

Analiza sintactica

- Descompune textul programului sursa în componentele sale "gramaticale"
  - A * B + C * D

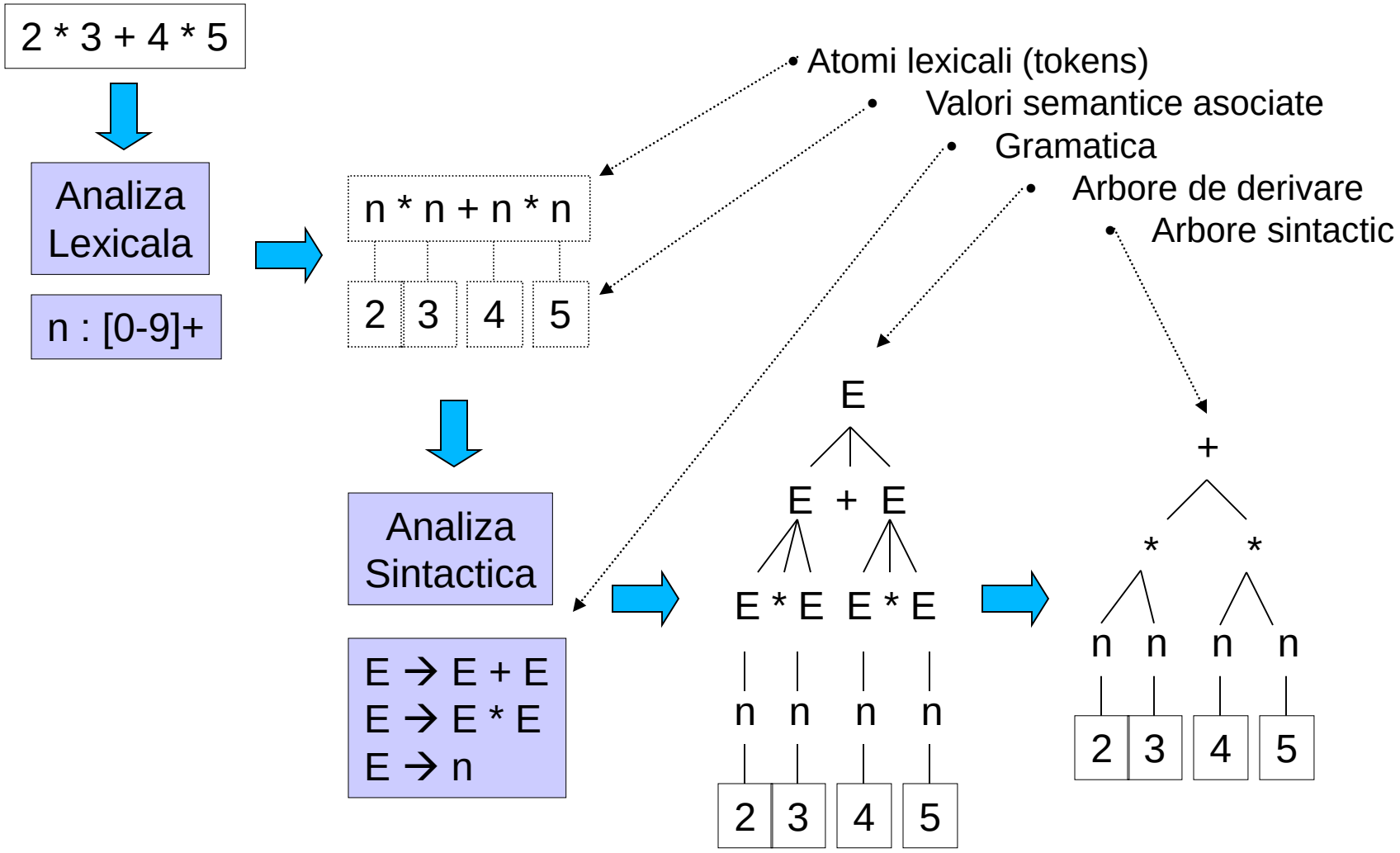


Arborele sintactic



Arborele de derivare

Arbore de derivare / sintactic



CPL
IR crunch

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

CPL
IR crunch

Tabela de simboli

- identificatorii utilizaţi în program şi informaţii despre acestia.
- Pt. fiecare identificator - câmpuri pentru atributele posibile.
  - Tip
  - Domeniu de valabilitate
  - Signatura (pt functii) – nr. si 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 şi semantic.

Detectarea erorilor

- Analiza lexicală - şir care nu corespunde unui atom lexical
  - 0x1234ABFG
- Analiza sintactica - erori legate de structura instrucţiunilor
  - 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

CPL
IR crunch

- 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

CPL
IR crunch

Câteva exemple :

- Constant folding
  - int sec = ore*60*60;
- Calcularea subexpresiilor 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

CPL
IR crunch

Generarea de cod obiect

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

Cursul acesta

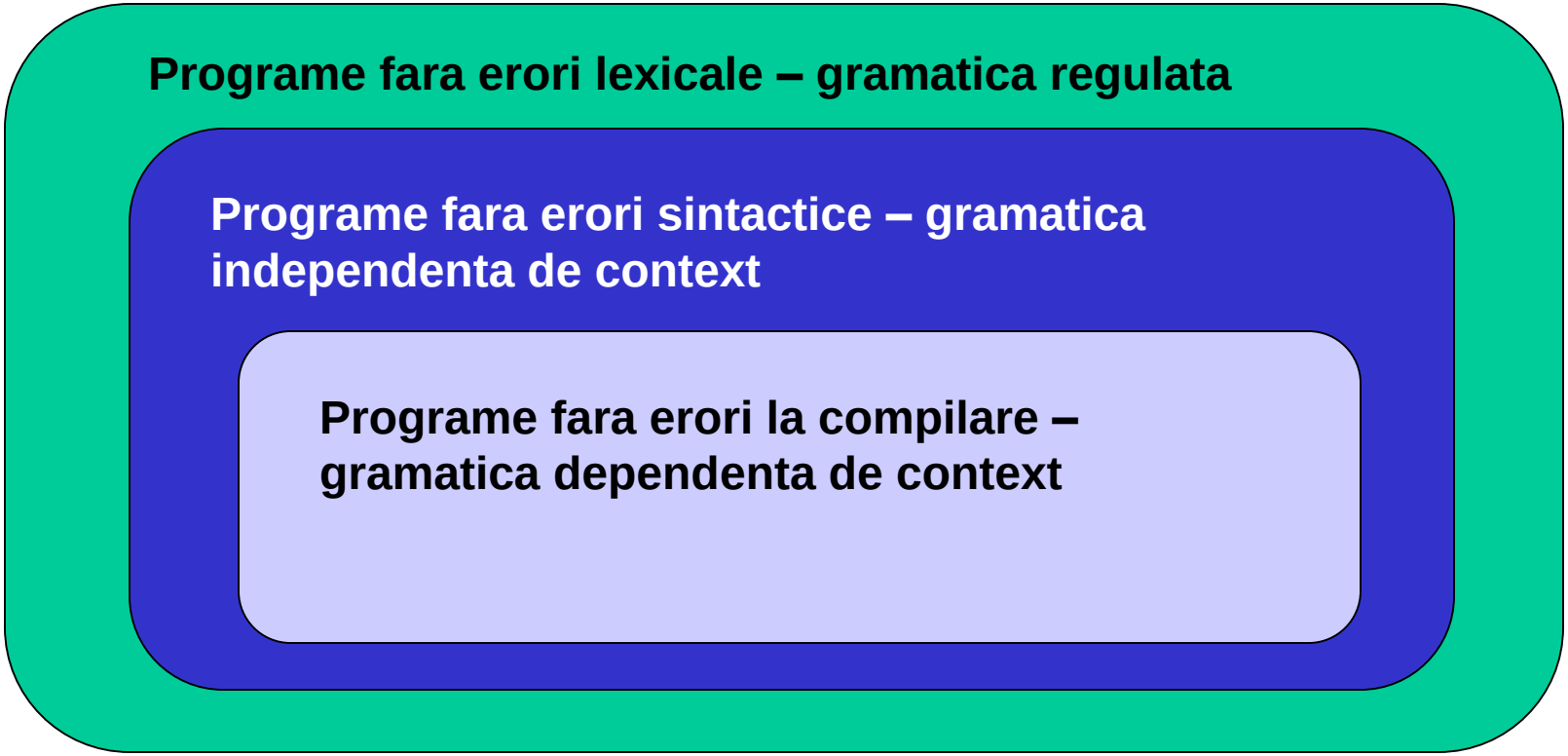
- Introducere/Administrative
- Structura compilatoarelor
- Fazele compilarii
- Analiza sintactica

CPL
IR crunch

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



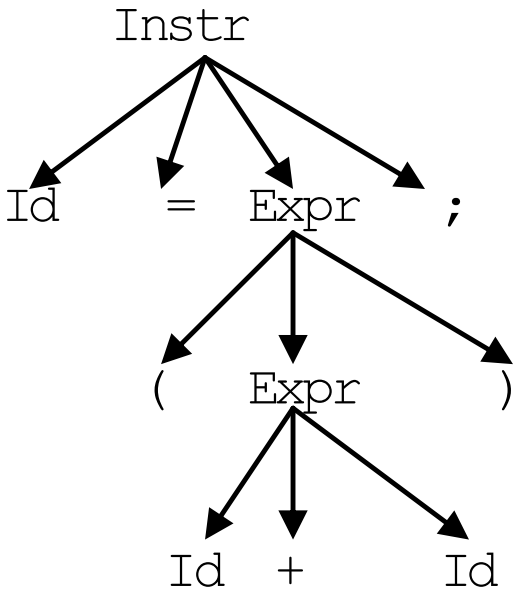
CPL
IR crunch

Analiza sintactica

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

Exemplu

- ALFA = (BETA + GAMA);
- id = (id + id) ;
- Instr → id = Expr ;
  Expr → Expr + Expr
  | Expr * Expr
  | (Expr)
  | id



Tipuri de analiza sintactica

- Descendenta (top-down, de sus in jos)
  - Inlocuieste cate un neterminal cu partea dreapta a unei productii, pana ramane doar cu terminali
- Ascendenta (bottom-up, de jos in sus)
  - Porneste de la sirul de atomi lexicali, abstractizeaza din sir simbolul de start prin reduceri succesive
- Analiza descendenta – derivare stanga
  - Tot timpul inlocuim cel mai din stanga neterminal
- Analiza ascendenta - derivarea dreapta
  - primul neterminal înlocuit este cel mai din dreapta din forma propoziţională curentă

Derivare stânga , top down

Instr	id = (id + id) ;
id = **Expr** ;	id = (id + id) ;
id = (**Expr**) ;	id = (id + id) ;
id = (**Expr** + Expr) ;	id = (id + id) ;
id = (id + **Expr**) ;	id = (id + id) ;
id = (id + id) ;	id = (id + id) ;

- LL: Șirul de tokeni se parcurge din stânga (L)
- Se deriveaza non-terminalul cel mai din stânga (L)
- Cum alegem producția folosită pentru derivare?
- Backtracking dacă alegem producția greșită

Derivare stânga , top down

Instr	id = (id) + (id) ;
id = **Expr** ;	id = (id) + (id) ;
id = **Expr** + Expr ;	id = (id) + (id) ;
id = (**Expr**) + Expr ;	id = (id) + (id) ;
id = (id) + **Expr** ;	id = (id) + (id) ;
id = (id) + (**Expr**) ;	id = (id) + (id) ;
id = (id + id) ;	id = (id) + (id) ;

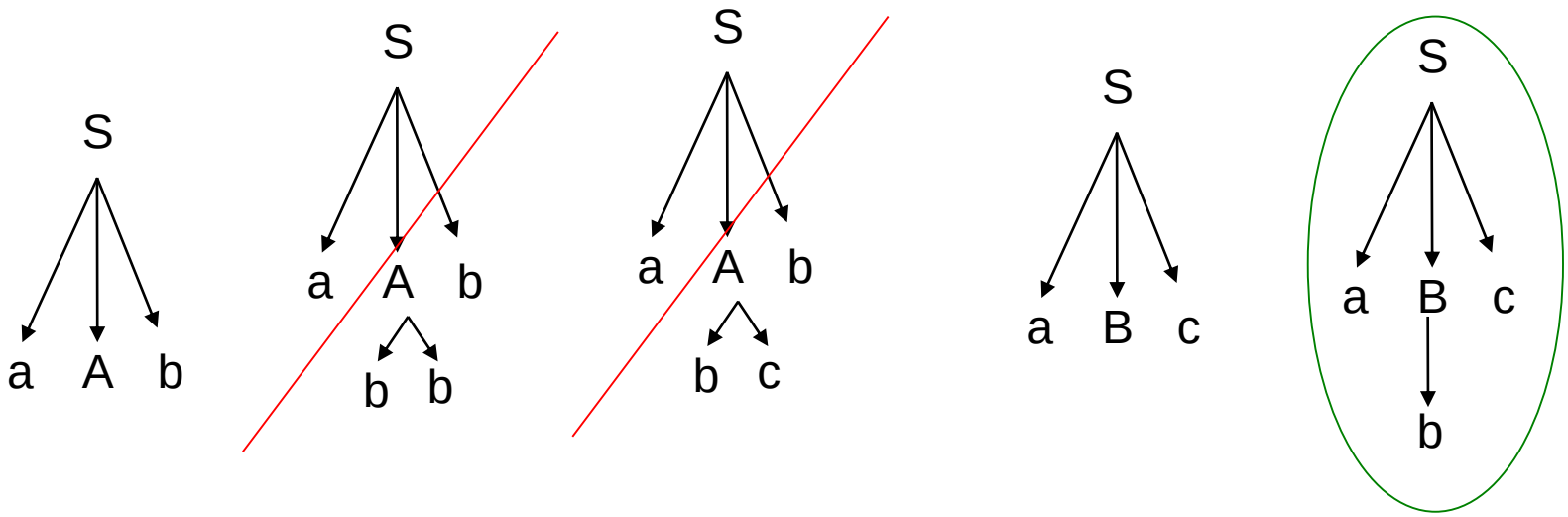
- Exemplu similar, dar trebuie alese alte producții pentru o derivare corectă fără backtracking
- Este necesară o metodă de predicție

Top down cu backtracking

- S → a A b | a B c
- A → bb | bc
- B → b

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

Arborele de derivare pentru şirul abc (abordare descendenta):



Derivare dreapta, bottom up

```
                id = ( id + id ) ;
id                 = ( id + id ) ;
id =                 ( id + id ) ;
id = (                 id + id ) ;
id = ( id                + id ) ;
id = ( Expr              + id ) ;
id = ( Expr +              id ) ;
id = ( Expr + id             ) ;
id = ( Expr + Expr           ) ;
id = ( Expr + Expr )           ;
id = ( Expr )                  ;
id = Expr                      ;
id = Expr ;
Instr
```

- LR: șirul de tokeni se parcurge din stânga (L)
- Tokenii sunt adăugați pe o stivă
- Se compara partea dreaptă a stivei (R) cu partea dreaptă a unei producții
- De ce primul **id** nu a fost transformat în **Expr** ?
- Decizie : SHIFT sau REDUCE

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.

Backup slides

Algoritmi generici de parsare

Algoritmi generici top-down

- Unger (1968)
- Acopera limbajele independente de context.
- Divide et impera

  Exemplu: S→ABC | DE ; input: pqrs

  Subprobleme:

A→p , B→q, C→rs	D→p , E→qrs
A→p , B→qr, C→s	D→pq , E→rs
A→pq , B→r, C→s	D→pqr , E→s

- Trebuie detectate derivarile care pot forma bucle.
- Complexitate?

Algoritmi generici bottom-up

- CYK (Cocke-Younger-Kasami, 1967)
- Acopera limbajele independente de context.
- Reducerea gramaticilor la forma normala Chomsky

  Toate regulile de forma A→a sau A→BC

  A→ε|γ ; B→α A β se transforma in

  B→α A′ β; B→αβ; A′→γ
- Programare dinamica
- Complexitate?