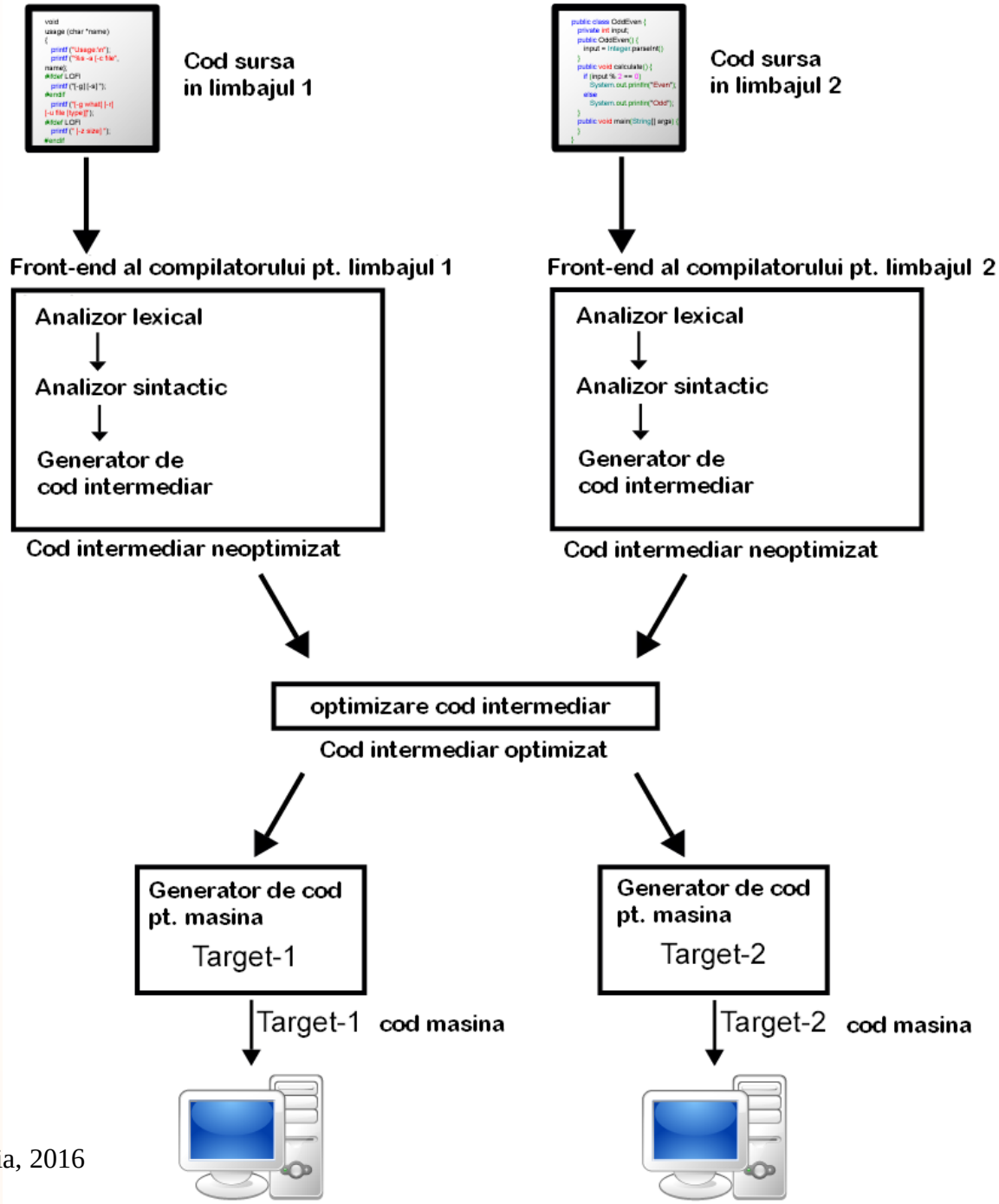
Compilator: multi-limbaj, multi-target (*masina*)

in stransa legatura cu separarea fazelor



Sursa imagine: Wikipedia, 2016

Exemple:

CIL **- Common Intermediate Language**

– Microsoft .Net

anterior cunoscut sub numele MSIL

(Microsoft Intermediate Language)

RTL - register transfer language

- GNU Compiler Collection

- multe alte compilatoare

Codul intermediar

- limbaj intermediar:
  - usor de transcris din arborele sintactic
  - usor de translatat in cod masina
  - proiectat inclusiv pt. a fi inteles/utilizat de oameni
  - mai apropiat de limbajul procesorului decat limbajul sursa

Reprezentari intermediare:

"*intre*" arborele de analiza sintactica si ASM

- high-level: mentine structura limbajului
- mid-level: independent de limbaj si masina
- low-level: dependent de masina

(tinde sa fie)

Codul intermediar

- Arbore sintactic abstract
- Forma poloneza postfixata
- Cod intermediar cu 3 adrese

  Reprezentari:
  - cvadruple
  - triplete
  - triplete indirecte

Arbore sintactic abstract

"***Abstract syntax trees***"

Alte denumiri **(intalnite in literatura de specialitate, in lb/ romana)**

arbore de sintaxa abstracta, arbore sintactic

- reprezentare apropiata de structura sintactica a programelor
- nu este arborele de derivare, ci o varianta simplificata

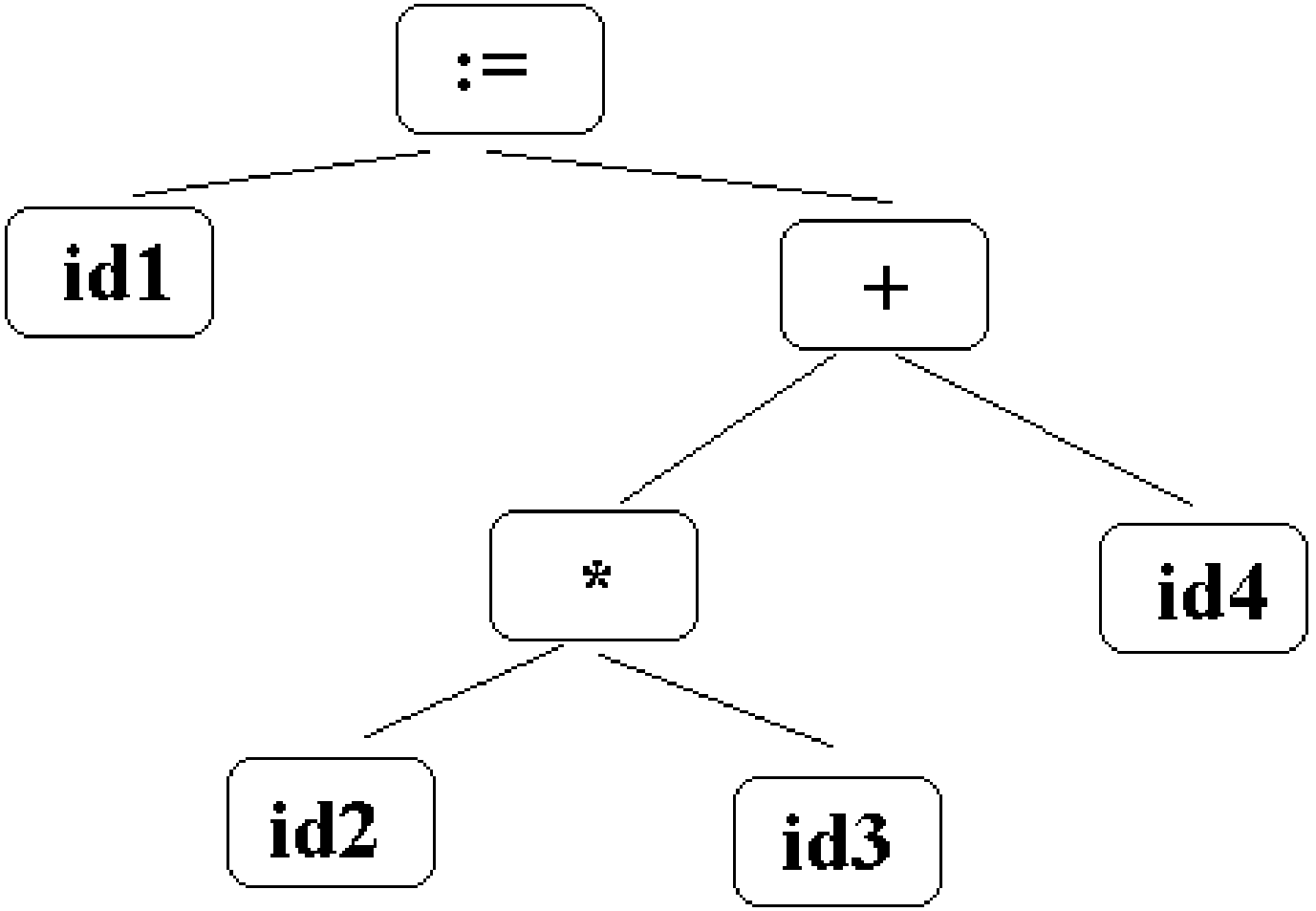
Proprietati:

- nodurile interioare sunt operatori
- descendentii unui nod sint *operanzii* lui

  ➔ fiecare subarbore formeaza o "*unitate logica*"

Arbore sintactic abstract

Exemplu: id1:=id2*id3+id4



Arbore sintactic abstract

Exercitii propuse:

Descrieti arborele sintactic abstract pentru urmatoarele instructiuni:

- if id1>id2 then id3:=id2
  else id3:=id1
- while id1>id2 do
  id1:=id2-id1

Forma poloneza

- aceeasi idee ca si la:

  forma poloneza postfixata pentru expr. aritmetice

Exemplu:

id1 := id2 * id3 + id4 => id1 id2 id3 * id4 + :=

Proprietate:

- operatorii apar in ordinea in care se executa operatiile

Avantaj:

evaluarea se face parcurgand o singura data expresia si executand operatiile tinand cont de aritatea lor

Cod intermediar cu 3 adrese

- secventa de instructiuni cu forma generala:
  <rezultat> := <arg1> <operator> <arg2>
- operatii:
  - binare
  - unare – se reprezinta doar un operand
- reprezentare
  - cvadruple
  - triplete
  - triplete indirecte

Pentru triplete si triplete indirecte vom lucra numai cu exemple cu atribuiri si expresii aritmetice.

3 adrese – reprezentare cvadruple

- structura tip inregistrare ce contine 4 campuri:

operator	arg1	arg2	rez

Exemplu:

A := B*(C+D)

operator	arg1	arg2	rez
…	…	…	…
+	C	D	T1
*	B	T1	T2
:=	T2		A

3 adrese – reprezentare triplete

- structura tip inregistrare ce contine 3 campuri:

operator	arg1	arg2

- se renunta la introducerea numelor temporare ce stocheza rezultate intermediare
- se considera ca instructiunea care calculeaza o valoare temporara retine acea valoare

3 adrese – reprezentare triplete

Exemplu:

A := B*(C+D)

	operator	arg1	arg2
…	…	…	…
(51)	+	C	D
(52)	*	B	(51)
(53)	:=	A	(52)

3 adrese – reprez. triplete indirecte

- codul contine instructiunile intr-o ordine oarecare
- pentru a obtine ordinea in care se executa operatiile, se foloseste un tabel suplimentar cu 2 campuri:

nr. de ordine a operatiei	nr. operatiei propriu-zise

3 adrese – reprez. triplete indirecte

nr. de ordine a operatiei	nr. operatiei propriu-zise
51	131
52	132
53	133

	operator	arg1	arg2
…	…	…	…
(131)	+	C	D
(132)	*	B	(131)
(133)	:=	A	(132)

Cvadruple. Conventii cu care vom lucra noi

- operanzi: constanta numerica
  valoarea unei variabile
- **operanzi speciali**
  - **@ adresa variabilei**
  - **^ variabila de la adresa indicata de valoarea variabilei**
- operatii
  - aritmetice binare: +, * , …
  - aritmetice unare: -
  - de atribuire (copiere): :=
  - relatii < , > , …
  - salt neconditionat goto et
  - salt conditionat g<operlogic> exp1 exp2 et

Optimizare cod intermediar

rearanjarea codului intermediar in vederea obtinerii unui program mai eficient

Exemple:

optimizari locale

1. realizarea unor calcule in mom. compilarii
2. eliminarea operatiilor redundante si a expresiilor comune
3. eliminarea codului inaccesibil *(secvente moarte)*
4. scurtcircuitarea expresiilor logice
5. factorizarea invariantilor de cicluri

optimizarea ciclurilor