Uvod v računalništvo (UvR)

Prevajalniki

Danijel Skočaj
Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Literatura: Invitation to Computer Science, poglavje 11

v1.0
Št. leto 2013/14

Cilji predavanja

- Našteti in opisati faze delovanja tipičnega prevajalnika
- Demonstrirati kako razbiti niz na lekseme
- Razumeti gramatična pravila podana v formatu BNF in zgraditi sintaksna drevesa
- Razložiti pomen rekurzivnih definicij in izogibanja dvoumnostim pri uporabi gramatik
- Razložiti kako semantična analiza uporablja semantične zapise za ugotavljanje pomena
- Razložiti nekaj pristopov k lokalni optimizaciji kode
- Opisati primer pristopa h globalni optimizaciji

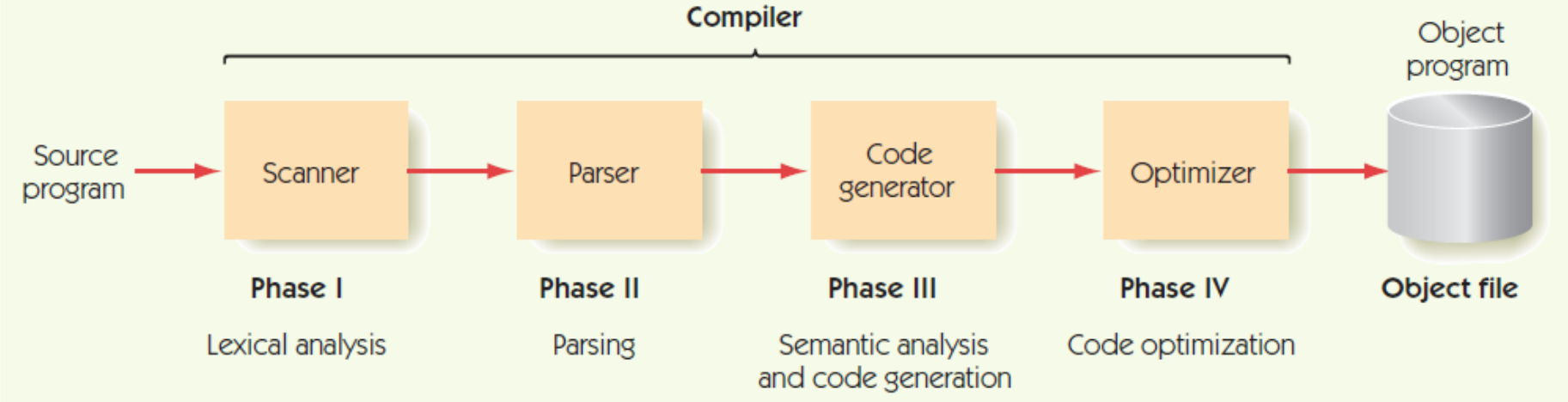
Uvod

- CPE izvajajo samo strojni jezik
  - vsak program se mora prevesti v strojni jezik
- Prevajanje iz zbirnega v strojni jezik je trivialno
  - prevajanje 1:1
- Prevajanje iz visoko-nivojskih jezikov je veliko bolj kompleksno
  - prevajanje 1:M
- Prevajalniki prevajajo programe iz visoko-nivojskih jezikov

- Dve glavni zahtevi za prevajalnike:
  - pravilnost: strojni ukazi morajo narediti natančno to kar pomenijo visoko-nivojski ukazi
  - učinkovitost in jedrnatost: strojna koda mora biti optimizirana in se mora izvajati hitro

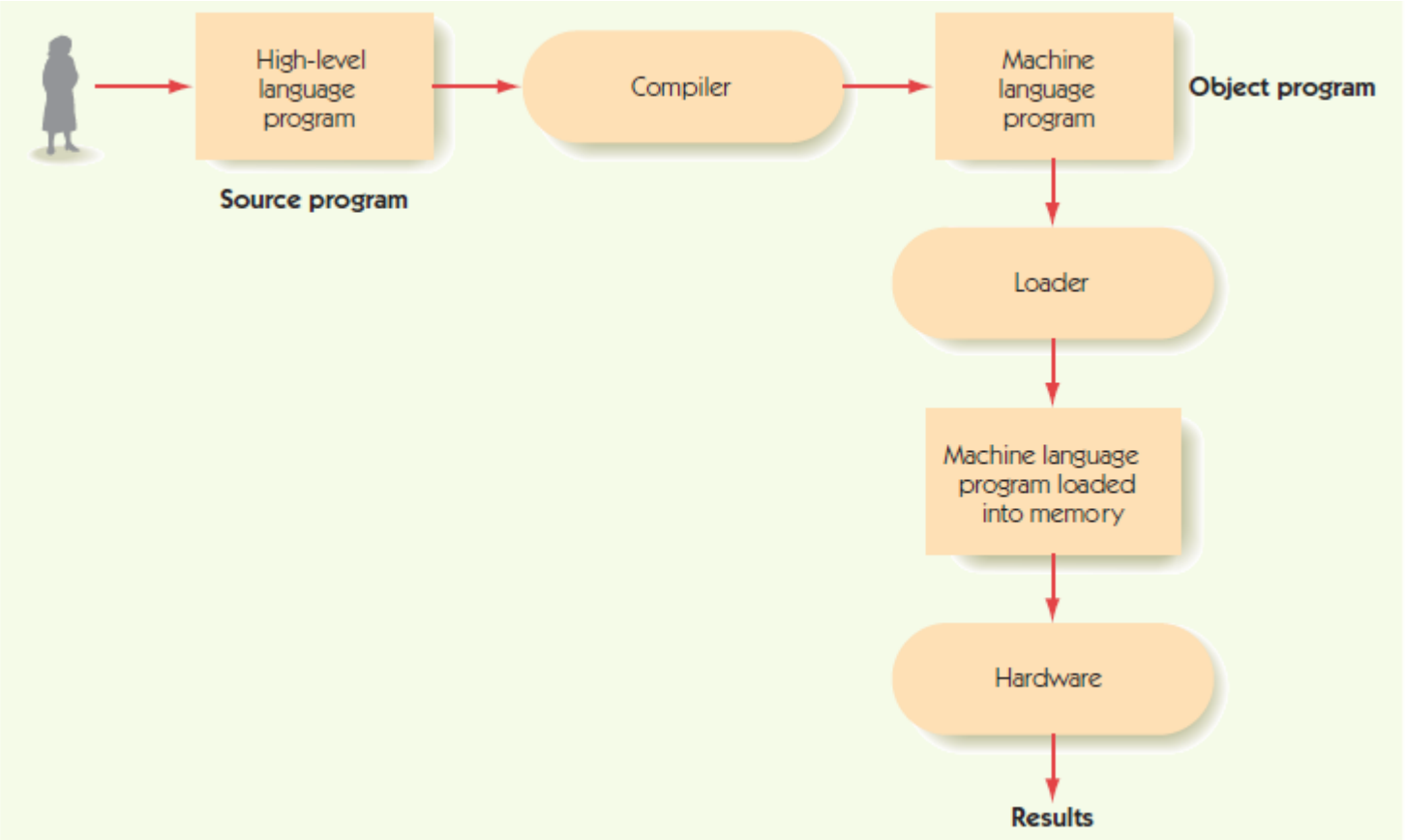
Proces prevajanja

1. Leksikalna analiza
   - združevanje znakov v lekseme
2. Sintaksna analiza
   - preverjanje sintakse in gradnja notranje predstavitve programa
3. Semantična analiza in generiranje kode
   - analiza pomena in generiranje strojnih ukazov
4. Optimizacija kode
   - izboljševanje časovne in prostorske učinkovitosti kode



Proces izvajanja programa

- Preveden program se nato izvede



1. Leksikalna analiza

- Leksikalni analizator (scanner)
  - združuje znake v lekseme (tokens)
  - izpusti nepomembne znake (presledke, komentarje, …)
  - določi tip posameznega leksema (simbol, število, oklepaj, itn.)
  - postopek:
    1. izpusti nepomembne znake in poišči začetek leksema
    2. združuj znake dokler ne zaznaš konec leksema

  - primer klasifikacije:
    - običajno je št. leksemov precej večje (>50)

Token Type	Classification Number
symbol	1
number	2
=	3
+	4
–	5
;	6
==	7
if	8
else	9
(	10
)	11

2. Sintaksna analiza

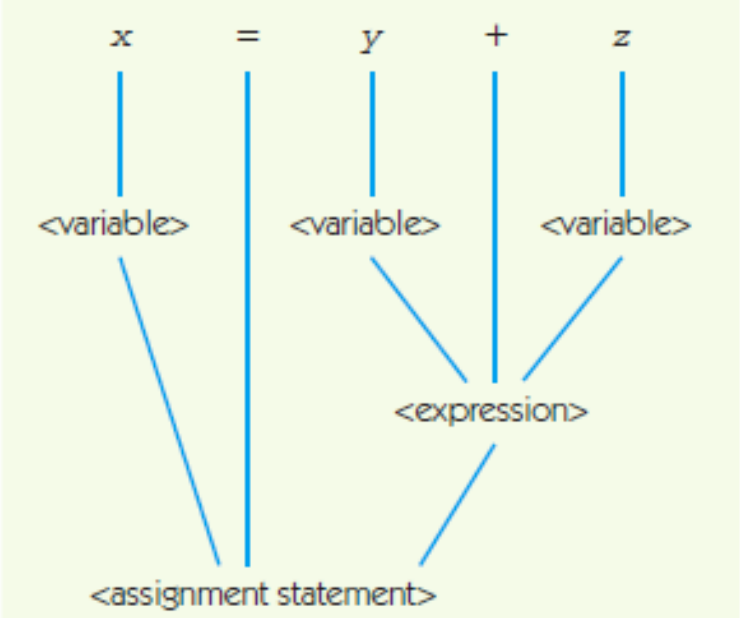
- Sintaksni analizator (parser) analizira lekseme
  - ugotovi gramatično strukturo
  - zgradi sintaksno drevo
- Sintaksa = gramatična struktura
  - definirana je z množico pravil (produkcij)
  - BNF (Backus-Naur Form) notacija za opis pravil
- Gramatika je množica pravil, ki definirajo jezik
- Pravila: `levaStran::=desnaStran`
  - `levaStran`: gramatična kategorija
  - `desnaStran`: vzorec, ki zajema strukturo kategorije
    - končni simbol (terminal): leksemi iz sintaksnega analizatorja, se ne členijo naprej
    - vmesni simbol (neterminal): gramatična kategorija (<…>)
    - začetni simbol (goal symbol): končni vmesni simbol
    - metasimboli: `<>`, `::=`, `|` (ali), `Λ` (prazni niz)
- Zaporedje leksemov je sintaksno pravilno, če jih lahko sintaksni analizator z zaporedno aplikacijo pravil pretvori v začetni simbol

2. Sintaksna analiza – primer

- Primer: prirejanje s seštevanjem spremenljivk x, y in z
- Prvi poizkus:

Number	Rule
1	<assignment statement> ::= <variable> = <expression>
2	<expression> ::= <variable> | <variable> + <variable>
3	<variable> ::= x | y | z

- Primer: `x=y+z`
- Zelo pomemben je vrstni red
  - sintaksna analiza s pogledom naprej (look-ahead parsing)
- Kaj pa `x=x+y+z` ?
- Dve zahtevi za gramatiko:
  - mora vključevati vse veljavne stavke iz gramatike
  - ne sme vključiti nobenega neveljavnega stavka



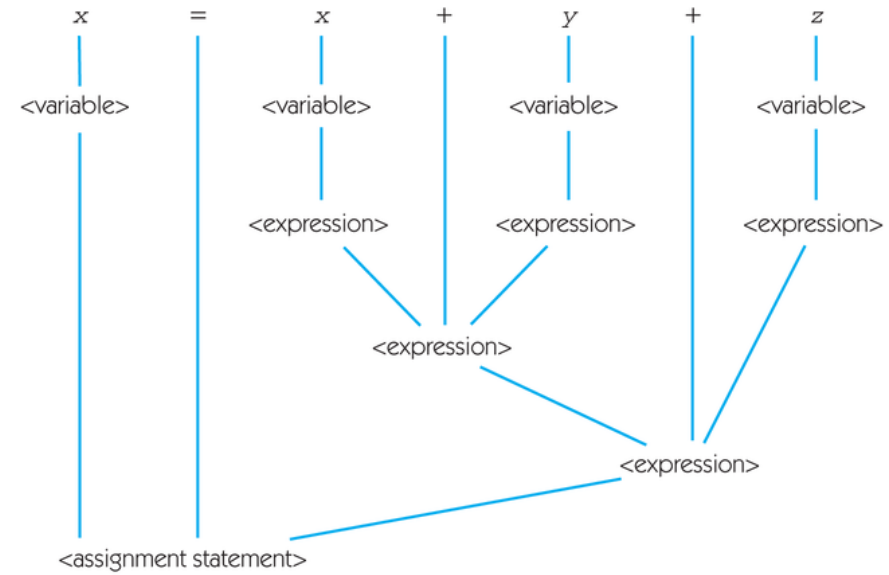
2. Sintaksna analiza – primer

- Drugi poizkus
  - poljubna dolžina vzorcev – rekurzivna definicija:

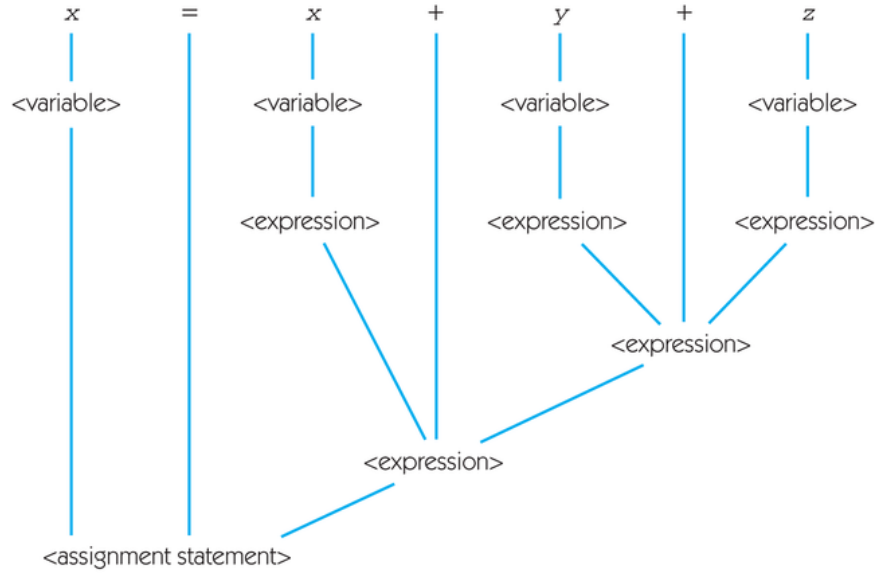
Number	Rule
1	<assignment statement> ::= <variable> = <expression>
2	<expression> ::= <variable> | <expression> + <expression>
3	<variable> ::= x | y | z

- Dvoumnost:

`x=(x+y)+z`



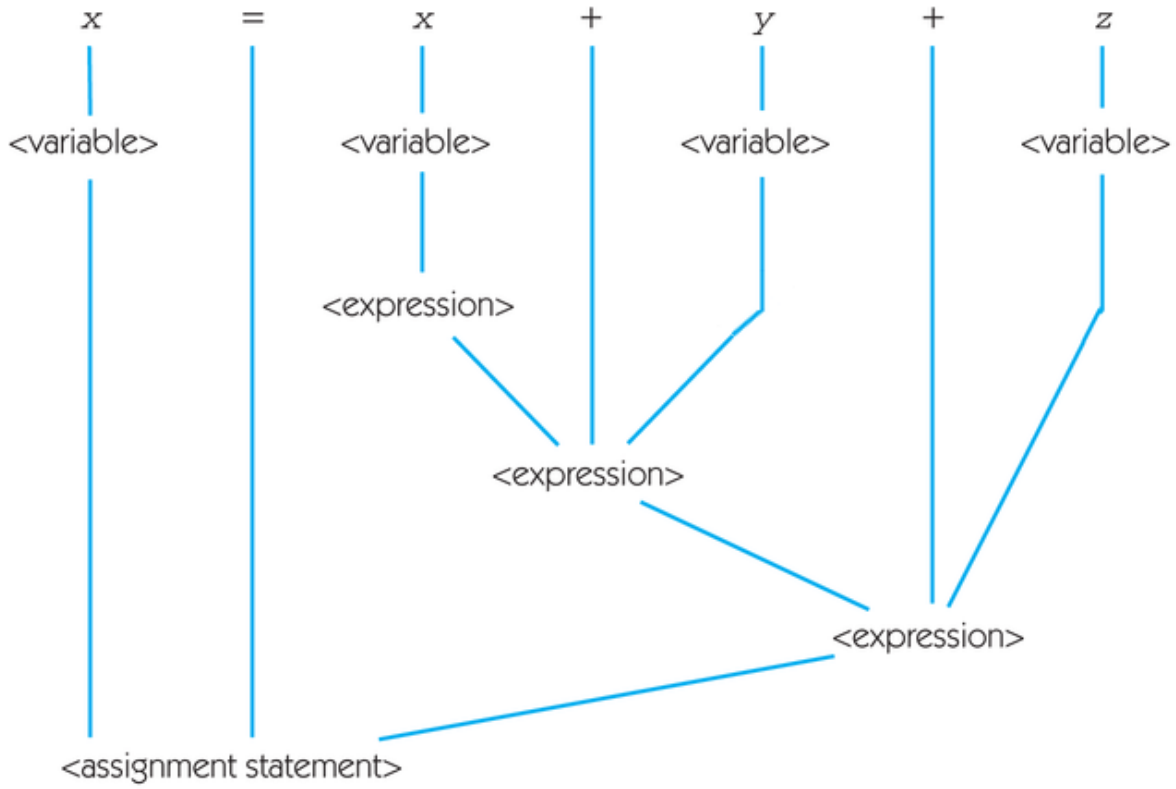
`x=x+(y+z)`



2. Sintaksna analiza – primer

- Tretji poizkus

Number	Rule
1	<assignment statement> ::= <variable> = <expression>
2	<expression> ::= <variable> | <expression> + <variable>
3	<variable> ::= x | y | z



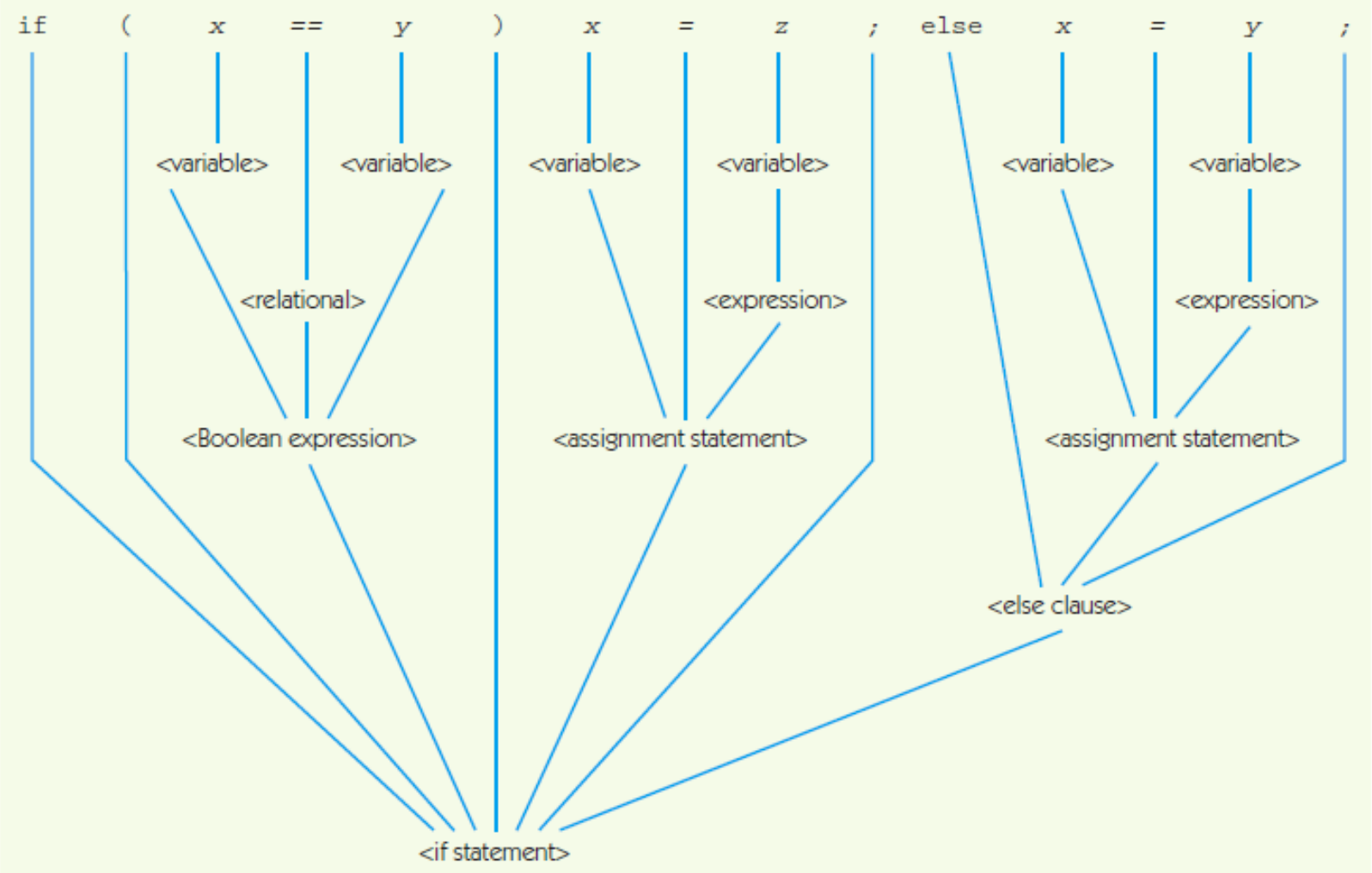
2. Sintaksna analiza – if-else

- Gramatika za poenostavljen if-else stavek:

Number	Rule
1	<if statement> ::= if (<Boolean expression>) <assignment statement> ; <else clause>
2	<Boolean expression> ::= <variable> | <variable> <relational> <variable>
3	<relational> ::= == | < | >
4	<variable> ::= x | y | z
5	<else clause> ::= else <assignment statement> ; | Λ
6	<assignment statement> ::= <variable> = <expression>
7	<expression> ::= <variable> | <expression> + <variable>

2. Sintaksna analiza – if-else

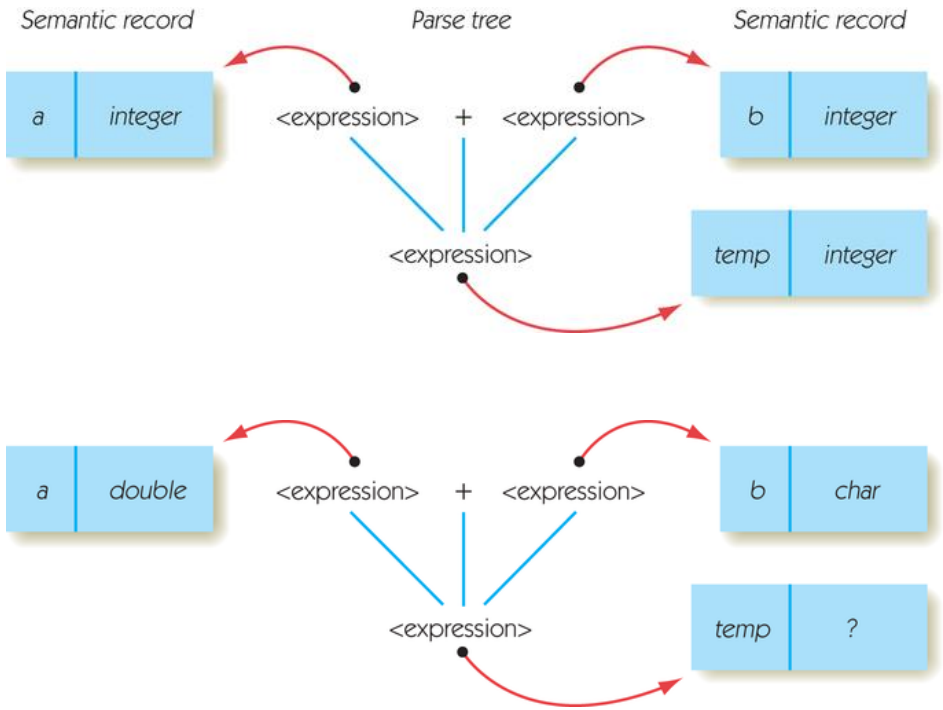
- primer: `if (x==y) x=z; else x=y;`



3. Semantična analiza

- Semantična analiza
  - preverja semantično pravilnost sintaksnega drevesa
  - preverja, da je pomen pravi
- Semantični zapisi
  - hrani informacije o vmesnih simbolih (ime, podatkovni tip,…)

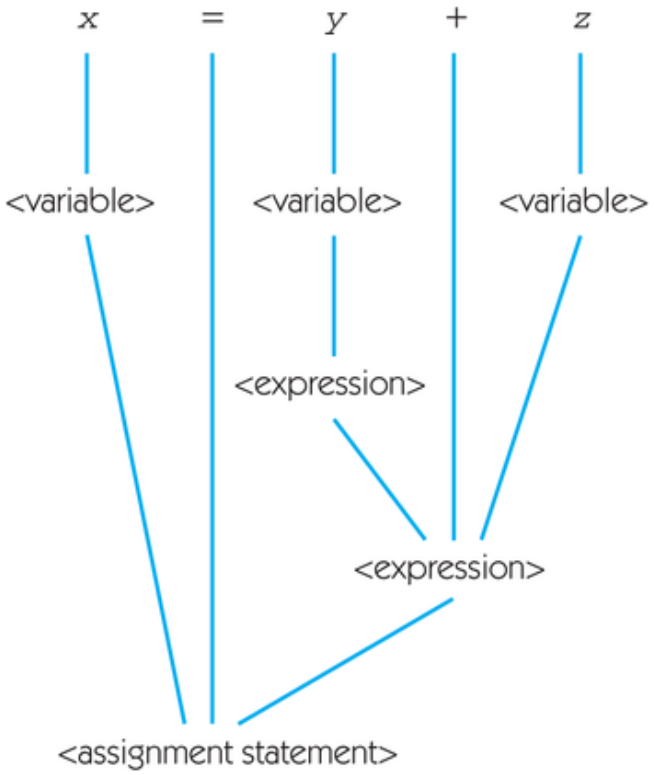
- V prvem prehodu čez kodo se pregleduje, če so vse veje semantično veljavne
  - gleda se semantične zapise
  - primerja se podatkovne tipe,…



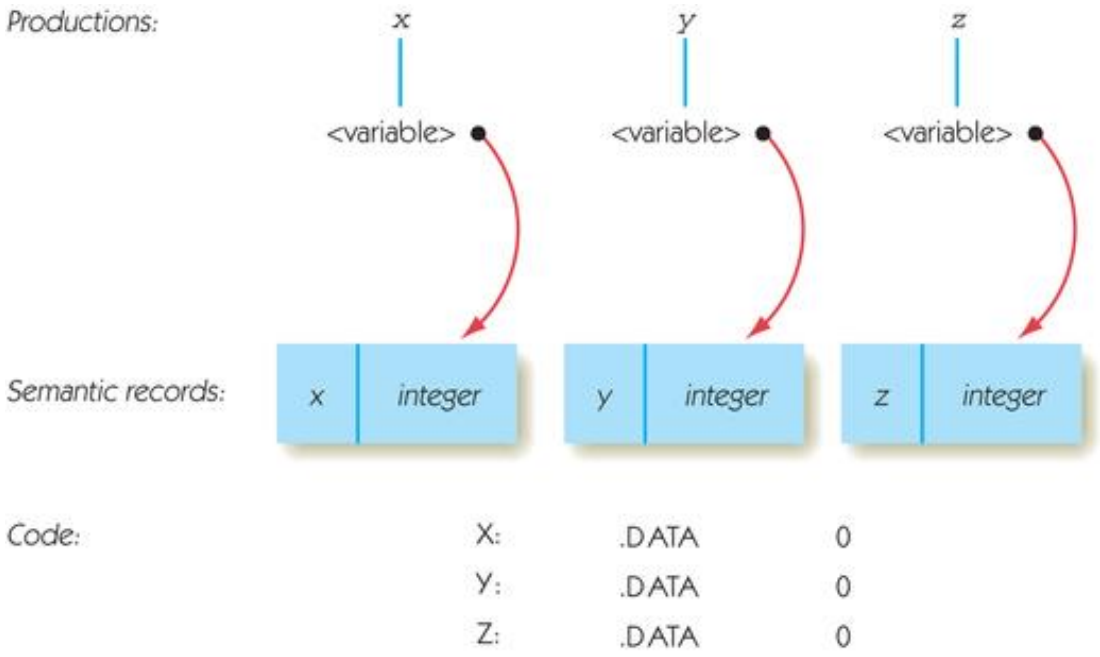
3. Generiranje kode

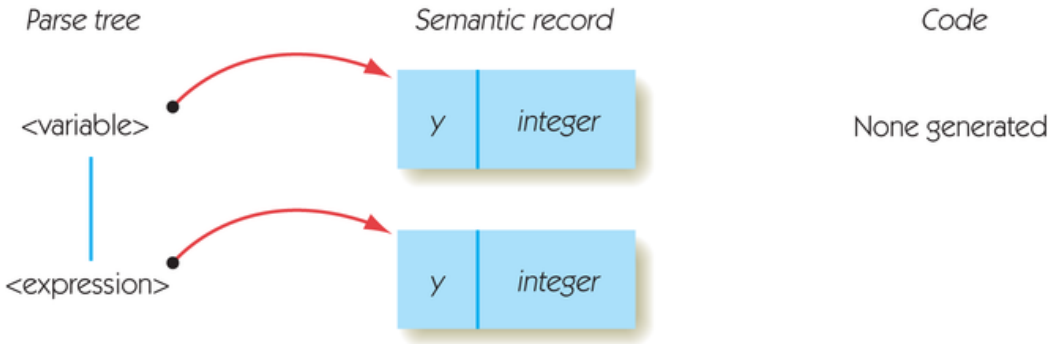
- Generiranje kode
  - Dele sintaksnega drevesa prevede v kodo v zbirnem jeziku
  - Sproti dodaja tudi semantične zapise
  - Vsi deli sintaksnega drevesa ne proizvedejo kode

- Primer:
  - `x=y+z`

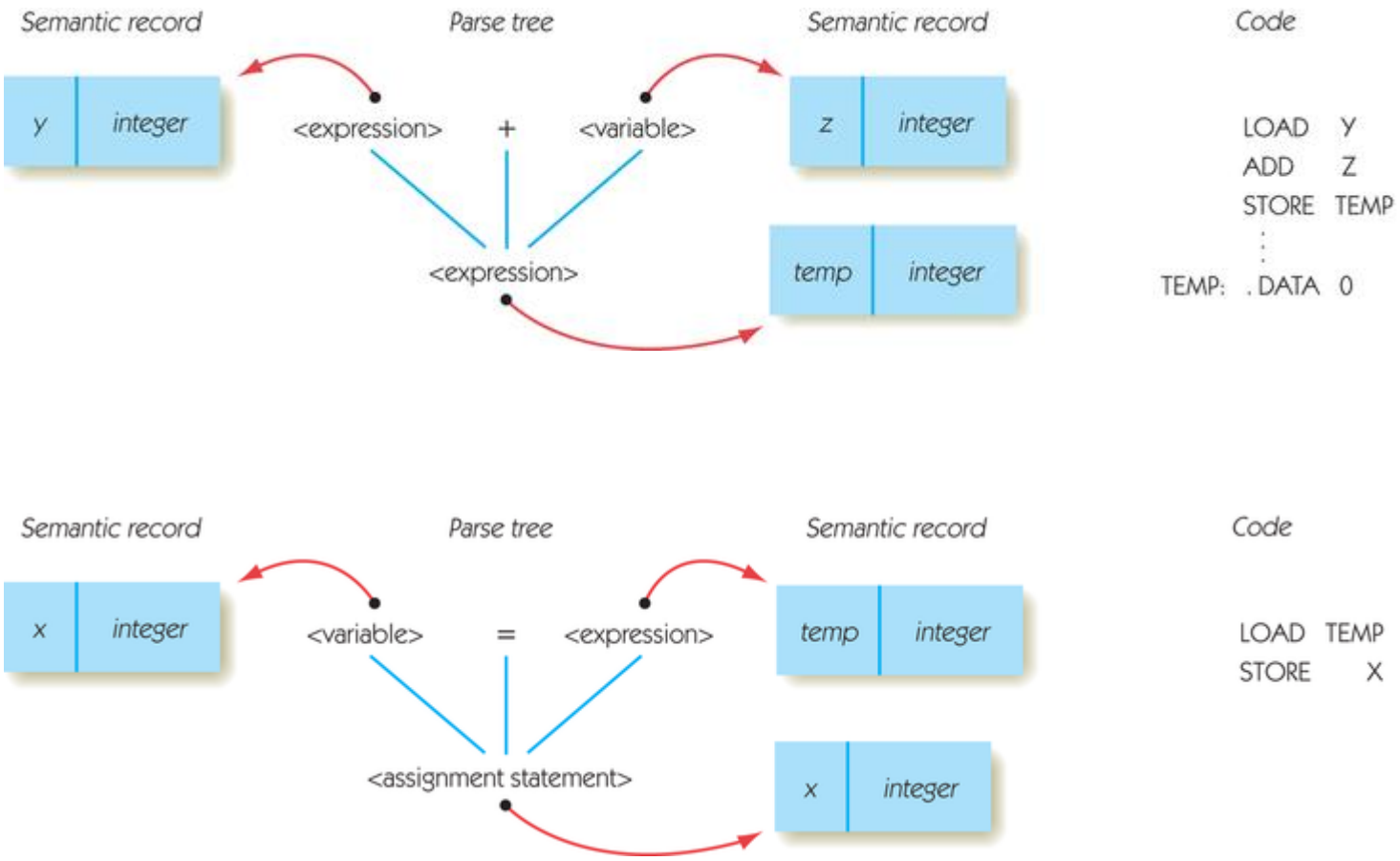


3. Generiranje kode - primer



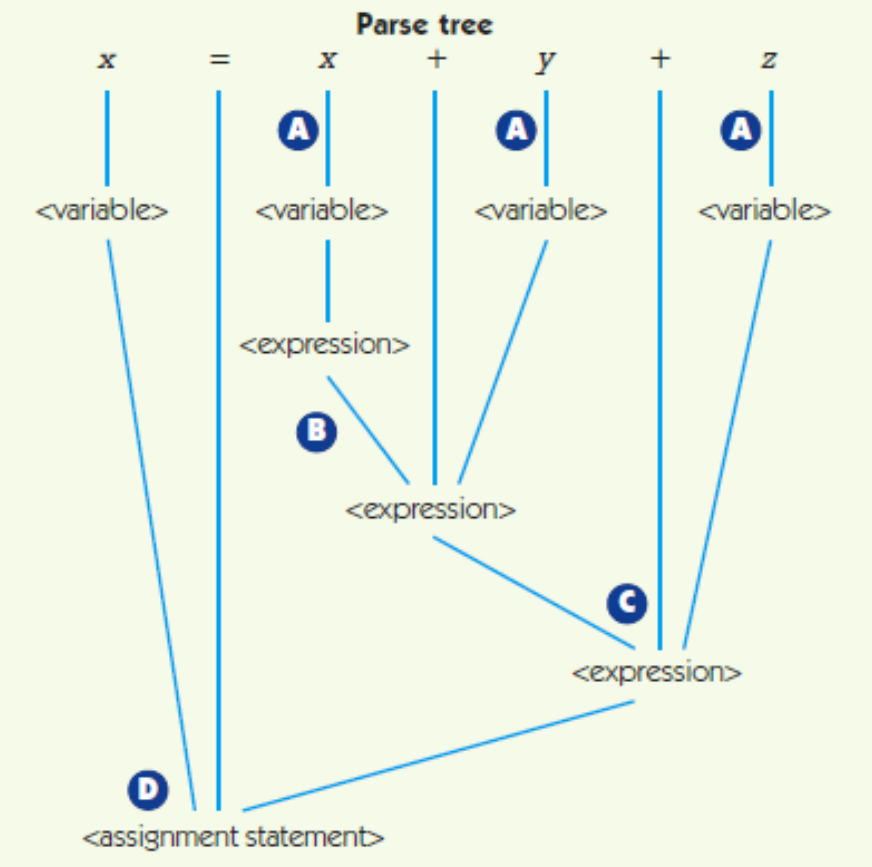


3. Generiranje kode - primer



3. Generiranje kode - primer

- Generiranje kode za izraz `x=x+y+z`



Generated code

```
--Here is the code for the production labeled B
        LOAD    X
        ADD     Y
        STORE   TEMP     --Temp holds the expression (x + y)
--Here is the code for the production labeled C
        LOAD    TEMP
        ADD     Z
        STORE   TEMP2    --Temp2 holds (x + y + z)
--Here is the code for the production labeled D
        LOAD    TEMP2
        STORE   X        --X now holds the correct result
                         --The remainder of the program goes here
--These next three pseudo-ops are generated by the productions labeled A
X:      .DATA   0
Y:      .DATA   0
Z:      .DATA   0
--The pseudo-ops for these temporary variables are generated
by productions B and C
TEMP:   .DATA   0
TEMP2:  .DATA   0
```

4. Optimizacija kode

- Optimizacija kode
  - izboljšanje časovne ali prostorske kompleksnosti proizvedene kode

- Lokalna optimizacija
  - preverja majhne dele kode v zbirniku (<5 ukazov)
  1. Evaluacija konstant
     - če se da, v naprej izračuna rezultate aritmetičnih operacij
  2. Redukcija po moči
     - počasne aritmetične operacije so zamenjane s hitrejšimi
  3. Odstranjevanje nepotrebnih operacij
     - odstranjevanje sicer pravilnih a nepotrebnih operacij

4. Optimizacija kode - primer

Generated code

```
--Here is the code for the production labeled B
        LOAD    X
        ADD     Y
        STORE   TEMP     --Temp holds the expression (x + y)
--Here is the code for the production labeled C
        LOAD    TEMP
        ADD     Z
        STORE   TEMP2    --Temp2 holds (x + y + z)
--Here is the code for the production labeled D
        LOAD    TEMP2
        STORE   X        --X now holds the correct result
                         --The remainder of the program goes here
--These next three pseudo-ops are generated by the productions labeled A
X:      .DATA   0
Y:      .DATA   0
Z:      .DATA   0
--The pseudo-ops for these temporary variables are generated
by productions B and C
TEMP:   .DATA   0
TEMP2:  .DATA   0
```

```
        LOAD    X
        ADD     Y
        ADD     Z
        STORE   X
        .
        .
        .
X:      .DATA   0
Y:      .DATA   0
Z:      .DATA   0
```

4. Optimizacija kode

- Globalna optimizacija
  - gleda večje bloke programa
    - while zanke
    - if stavke
    - procedure
  - veliko težje
  - veliko večji učinek

- Tudi najboljša optimizacija kode ne more popraviti slabega algoritma!

4. Optimizacija kode

- V preteklosti: „Strojna oprema je draga, programerji so poceni.“
- Zdaj: „Strojna oprema je poceni, programerji so dragi.“

- Moderni prevajalniki optimizirajo kodo, a zdaj je poudarek na:
  - vizualnih razvojnih orodjih (IDE) za lažje in bolj informirano programiranje
  - vgrajenih razhroščevalnikih za lažje razhroščevanje
  - programskimi knjižnicami in orodji za ponovno uporabo

Povzetek

- Visoko-nivojski jeziki potrebujejo prevajalnike za prevajanje v zbirni jezik
- Prevajanje iz visoko-nivojskih jezikov je veliko težje kot iz zbirnega jezika v strojni jezik

- Faze prevajanja:
  1. Leksikalna analiza
  2. Sintaksna analiza
  3. Semantična analiza in generiranje kode
  4. Optimizacija kode