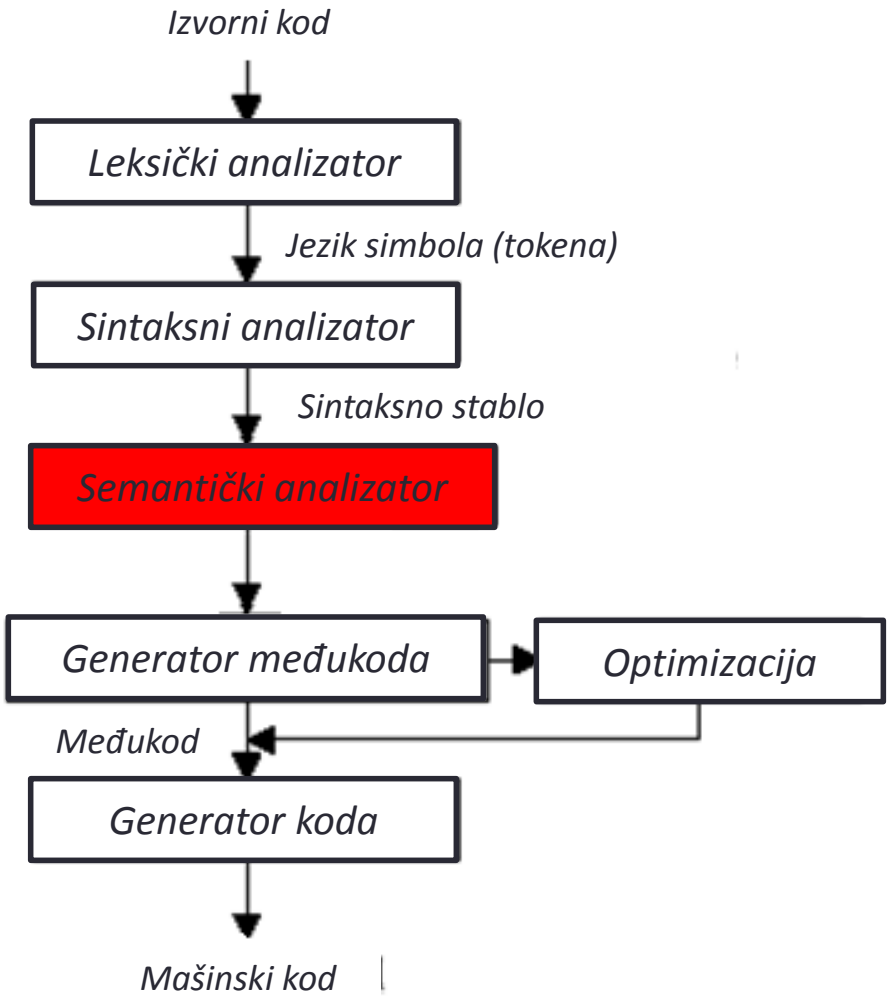
PROGRAMSI PREVODIOCI

- Sementička analiza i atributne gramatike -

Struktura kompilatora



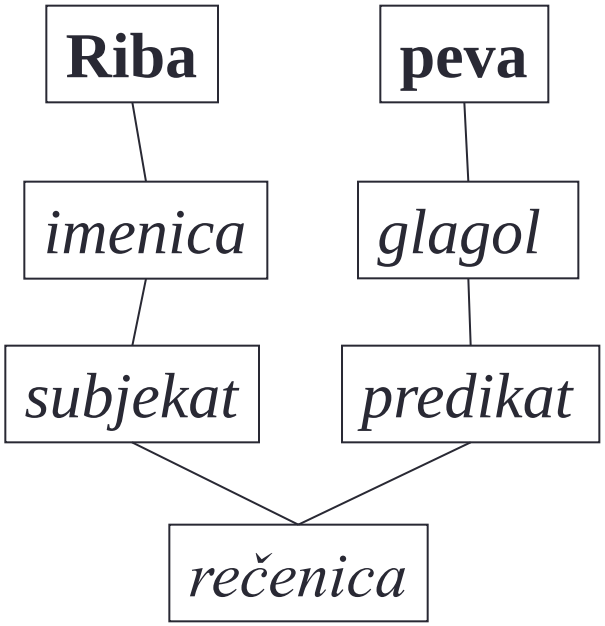
Sintaksna ↔ Semantička analiza u prirodnim jezicima

- Sintaksa analiza određuje strukturu rečenice.
- Semantička analiza utvrđuje njeno značenje.

Sintaksna ↔ Semantička analiza u prirodnim jezicima

- Analizirajmo rečenicu: **Slon peva.**

Sintaksna analiza



Zaključak: rečenica ispravna

Semantička analiza

Primer semantičkog pravila:
Glagol pevati može da stoji samo uz imena živih bića koja mogu da proizvedu zvuk.

Zaključak: rečenica neispravna

Sintaksna ↔ Semantička analiza u programskim jezicima

- Sintaksna analiza- utvrdjuje da li je struktura programa korektna
- Semantička analiza – utvrđuje:
  - Da li su različiti strukturni elementi medjusobno usaglašeni
  - Zato se naziva još i kontekstno-zavisna analiza (*context-sensitive*)
    - Da li su sve promenljive koje se koriste u programu deklarisane,
    - Da li je promenljivoj dodeljena vrednost pre njenog korišćenja,
    - Da li je promenljiva „vidljiva“ u tački u kojoj se koristi (opseg važenja)
    - Da li su operatori primenjeni nad operandima odgovarajućeg tipa,
    - Da li je određeno ime, ime funkcije, promenljive, klase...
    - Da li je lista stvarnih parametara u pozivu funkcije usaglašena sa listom fiktivnih parametara u definiciji funkcije,
    - Da li klasa sartži član koji se koristi,
    - Da li je korišćenje određenog člana klase u skladu sa njegovim pravom pristupa
    - ...

Sintaksna ↔ Semantička analiza u programskim jezicima

- Sintaksa programskog jezika - strogo formalno definisana
- Semantika – obično definisana govornim jezikom

Sintaksno-upravljana semantička analiza

- Rutine za proveru semantičke ispravnosti koda se pridružuju pravilima (smenama gramatike), odnosno semantičke rutine se izvršavaju u neterminalnim čvorovima sintaksnog stabla.
- Generalizacija ove ideje – **sintaksno-upravljano prevođenje** - sve naredne faze prevođenja se realizuju tako što se odgovarajuće rutine izvršavaju u čvorovima sintaksnog stabla.

Atributne gramatike

- Da bi se proverila semantička ispravnost (ili realizovala bilo koja naredna faza prvođenja) često je potrebno da se uz simbole koji učestvuju u smenama pamte i neke dodatne informacije o njima.
- Dodatne informacije o simbolima se nazivaju **atributi simbola**, a ovakve gramatike **atributne gramatike**.
- Sintaksno stablo obogaćeno vrednostima atributa simbola se naziva **obeleženo (notirano) sintaksno stablo**.

Određivanje vrednosti atributa

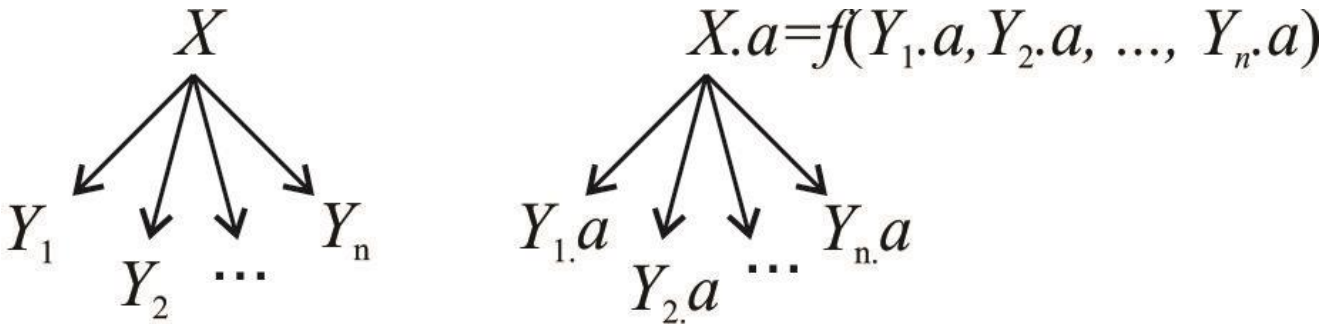
- Vrednost atributa simbola određuju se semantičkom rutinom koja je pridružena produkcionom pravilu (smeni gramatike) koje je primenjena u odgovarajućem čvoru sintaksnog stabla.
- Na osnovu toga kako se propagiraju vrednosti atributa, atributi se dele na:
  - **Generisane**
  - **Nasleđene**

Generisani atributi

- Vrednosti atributa terminalnih simbola određuje leksički analizator.
- Vrednosti atributa neterminalnih simbola se izračunavaju kao funkcije atributa simbola koji se nalaze u čvorovima potomcima odgovarajućeg čvora.
- Vrednosti ovih atributa se propagiraju kroz sintaksno stablo (odozdo naviše).

$$X \rightarrow Y_1\ Y_2 \ldots Y_n$$

$$X.a = f(Y_1.a, Y_2.a, \ldots, Y_k.a, \ldots, Y_n.a)$$



Generisani atributi - Primer

digit → 0 { digit.value = 0; }
| 1 { digit.value = 1; }
| 2 { digit.value = 2; }
...
| 9 { digit.value = 9; }

Prosleđivanje atributa naviše:

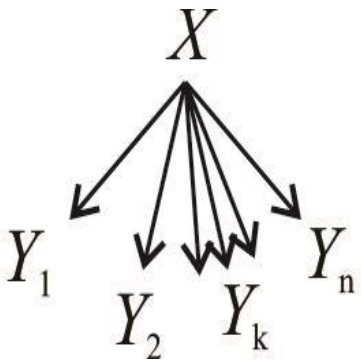
num1 → digit { num1.value = digit.value}
| num2 digit { num1.value = num2.value*10
+ digit.value; }

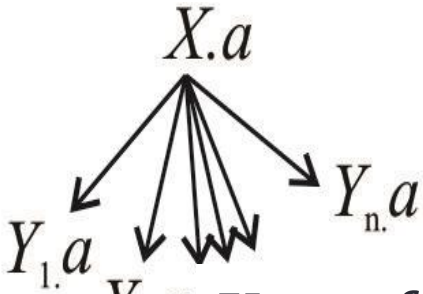
Nasleđeni atributi

- Vrednosti ovih atributa se izračunavaju kao funkcije atributa simbola koji se nalazi u roditeljskom čvoru i atributa simbola na istom nivou levo od simbola čiji se atribut određuje.
- Ovi atributi se propagiraju kroz sintaksno stablo odozgo-naniže.

$$X \rightarrow Y_1\ Y_2 \ldots Y_n$$

$$Y_k.a = f(Y_1.a, Y_2.a, \ldots, Y_{k-1}.a)$$





$$\boldsymbol{Y_k.a = f(Y_1.a, Y_2.a, \ldots, Y_{k-1}.a)}$$

Metode za realizaciju sintaksno-upavljanog prevođenja

- Jedno-prolazni prevodioci
  - Svi atributi su istog tipa (nasleđeni ili generisani) prilagođeni algoritmu sintaksne analize
    - Kod top-down algoritama se koriste nasleđeni atributi
    - Kod bottom-up algoritama generisani
- Više-prolazni prevodioci
  - Mogu da budu definisani i nasleđeni i generisani atributi
  - U prvom prolazu se generiše samo sintaksno stablo
  - Sintaksno stablo se obilazi više puta i prilikom svakog obilaska se izračunavaju novi atributi

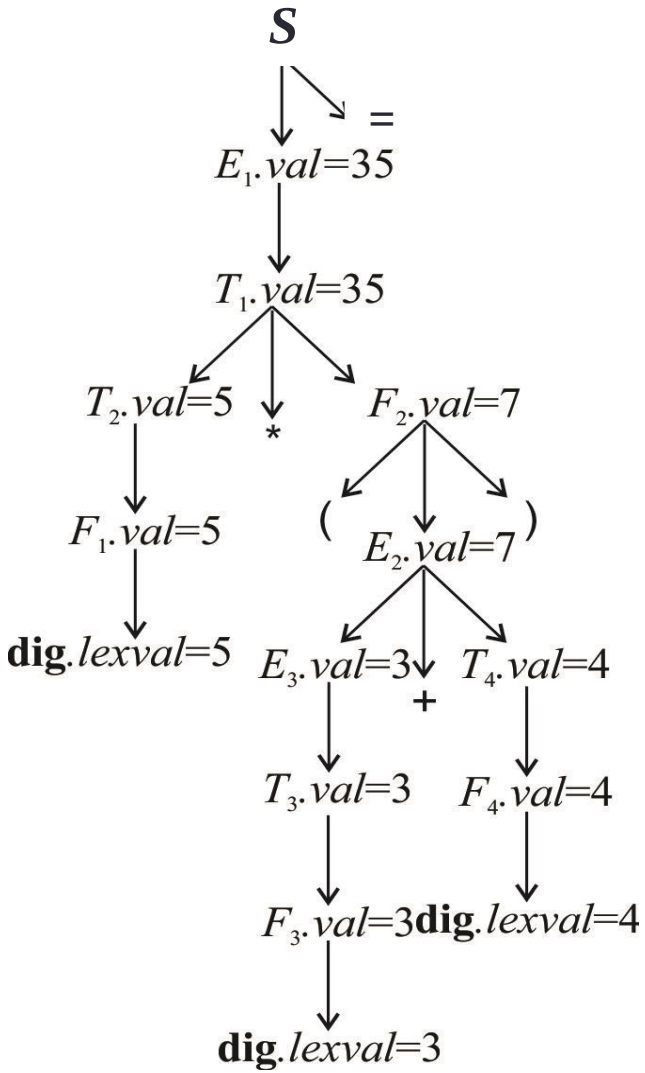
Jedno-prolazno bottom-up sintaksno-upravljano prevođenje - Primer

Gramatika sa semantičkim rutinama za realizaciju jednostavnog kalkulatora (interpretatora aritmetičkih izraza)

Pravilo gramatike	**Semantička rutina**
$S \rightarrow E =$	print (E.val)
$E \rightarrow E_1 + T$	E.val := E_1.val + T.val
$E \rightarrow T$	E.val := T.val
$T \rightarrow T_1 * F$	T.val := T_1.val * F.val
$T \rightarrow F$	T.val := F.val
$F \rightarrow (E)$	F.val := E.val
$F \rightarrow$ digit	F.val := top.get(digit.lexval)

Jedno-prolazno bottom-up sintaksno-upravljano prevođenje - Primer

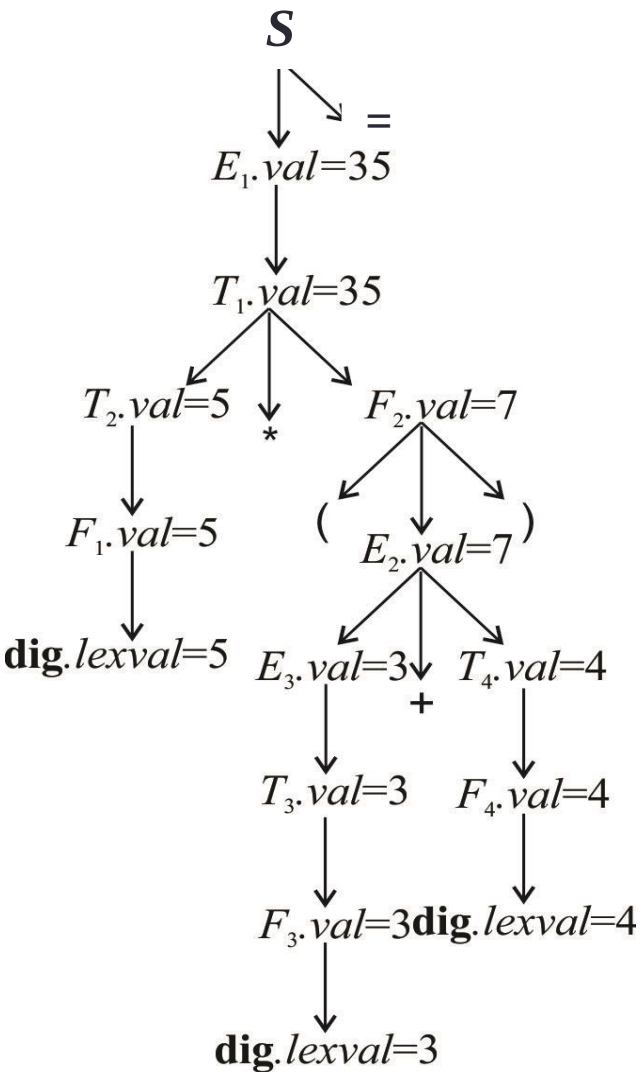
Anotirano (označeno) sintaksno stablo za izraz **5*(3+4)**



Pravilo gramatike	Semantička rutina
S → E =	print (E.val)
E → E + T	E.val := E.val + T.val
E → T	E.val := T.val
T → T * F	T.val := T.val * F.val
T → F	T.val := F.val
F → (E)	F.val := E.val
F → digit	F.val := top.get(digit.lexval)

Jedno-prolazno bottom-up sintaksno-upravljano prevođenje - Primer

Redosled izračunavanja vrednosti atributa za izraz **5*(3+4)**



Čvor	Atribut	Vrednost
F_1	F_1.val= digit.lexval	5
T_2	T_2.val := F_1.val	5
F_3	F_3.val= digit.lexval	3
T_3	T_4.val := F_3.val	3
E_3	E_3.val := T_4.val	3
F_4	F_4.val= digit.lexval	4
T_4	T_3.val := F_4..val	4
E_2	E_2.val := E_3.val + T_3.val	3+4=7
F_2	F_2.val := E_2.val	7
T_1	T_1.val := T_2.val * F_2.val	5*7=35
E_1	E_1.val := T_1.val	35
S	print(E_1.val)	35

Provera tipova – *type checker*

- Najbitnija komponenta semantičkog analizatora
- Gruba definicija: *Type checker* proverava da li su broj operanada i njihovi tipovi u izrazima odgovarajući
- Tip definiše oseg vrednosti podataka koje se mogu predstaviti i skup operacija koje se nad njima mogu izvoditi
- Tip imaju:
  - Promenljive
  - Konstante
  - Funkcije
  - Izrazi
- Postoje ugrađeni tipovi podataka i korinički definisani tipovi.

Zadaci *type checker*-a

- Primeri:
  - Da li su oba operanda u binarnim aritmetičkim izrazima numeričkog tipa,
  - Da li je indeks polja celobrojnog tipa,
  - Da li tip svakog stvarnog parametra u pozivu funkcije odgovara tipu formalnog parametra,
  - Da li postoji klasa navedena kao roditeljska u definiciji izvedene klase,
  - Da li klasa koja implementira interfejs sadrži sve funkcije definisane u interfejsu
  - ...

Problemi u realizaciji *type checker*-a

- Statička i dinamička tipizacija
  - Kod jezika sa statičkom tipizacijom svaka promenljiva koja se koristi u programu mora biti deklarisana
    - Provera tipova se radi u vreme prevođenja (compile-time)
  - Kod jezika sa dinamičkom tipizacijom promenljive se ne deklarišu
    - Provera tipova se radi u vreme izvršenja programa (run-time)
- Jedinstvenost imena
  - U nekim jezicima imena su jedinstvena na nivou celog projekta
  - U nekim jezicima imena su jedinstvena u jednom opsegu važenja (*scope*-u) – u podopsezima se ne mogu definisati ista imena
  - U nekim jezicima imena su jedinstvena u okviru tekućeg opsega, ali ne i u podopsezima – u podopsegu je moguće definisati isto ime sa drugim značenjem

Tabele simbola

- Za proces semantičke analize (pre svega *type checker*-a) za svako upotrebljeno simboličko ime u programu:
  - Koju vrstu entiteta imenuje,
  - Različite dodatne informacije (zavisno od tipa entiteta koji imenuje)
- To se pamti u pomoćnim strukturama podataka koje se nazivaju **tabele simbola** (**symbol tables**).

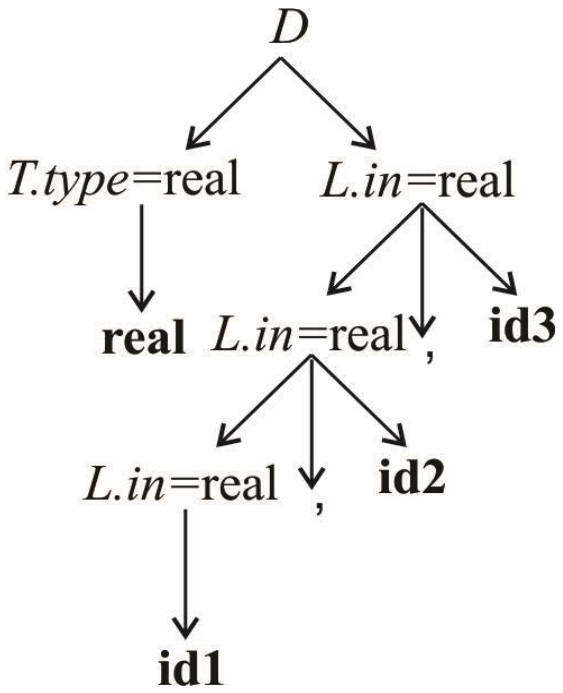
Realizacija *type checker*-a korišćenjem atributnih gramatika – Primer 1

Gramatika sa semantičkim rutinama za opis tipova promenljivih

Pravilo gramatike	**Semantička rutina**	**Vrsta atributa koji se izračunava**
D → T L	L.in := T.type	Nasleđeni
T → int	T.type := integer	Generisani
T→ real	T.type := real	Generisani
$L \rightarrow L_1$, id	L_1.in:= L.in addtype(id.entry, L.in)	Nasleđeni
L → id	addtype(id.entry, L.in)	Nasleđeni

Realizacija *type checker*-a korišćenjem atributnih gramatika – Primer 1

Notirano (označeno) sintaksno stablo za definiciju tipa:
`real id1, id2, id2;`

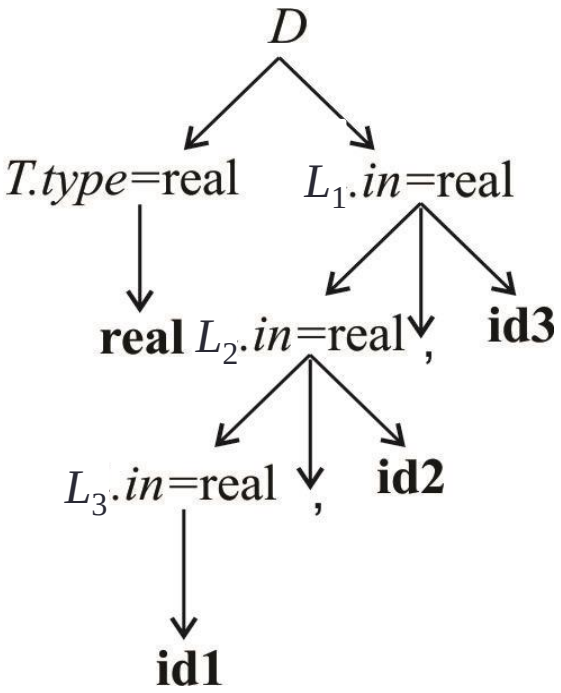


Pravilo gramatike	**Semantička rutina**
D → T L	L.in := T.type
T → int	T.type := integer
T→ real	T.type := real
$L \rightarrow L_1$, id	L_1.in:= L.in addtype(id.entry, L.in)
L → id	addtype(id.entry, L.in)

Realizacija *type checker*-a korišćenjem atributnih gramatika – Primer 1

Redosled izračunavanja atributa pri analizi definicije tipa:

`real id1, id2, id2;`



Čvor	Pravilo	Vrednost
T	T.type= real	real
D	L_1.in := T.type	L_1.in :=real
L_1	L_2.in:= L_1.in addtype(id3.entry, L_1.in)	L_2.in:=real U TS se upisuje (id3.entry, real)
L_2	L_3.in:= L_2.in addtype(id2.entry, L_2.in)	L_3.in :=real U TS se upisuje (id2.entry, real)
L_3	addtype(id1.entry, L_3.in)	U TS se upisuje (id1.entry, real)

Realizacija type checker-a korišćenjem atributnih gramatika – Primer 2

Provera tipova u izrazima

Pravilo gramatike	**Semantička rutina**
E→E1 + T	E.type := if (E1.type is numeric and T.type is numeric) then getMorePreferedType(E1.type,T.type) else type_error
E → T	E.type := T.type

Realizacija type checker-a korišćenjem atributnih gramatika – Primer 2

Provera tipova u izrazima

Pravilo gramatike	**Semantička rutina**
T→T1 mod F	E.type := if (T1.type=int and F.type=int) then int else type_error
T→ F	T.type := F.type
F→ (E)	F.type := E.type
F→ id	F.type := get_type(id.entry)

Realizacija type checker-a korišćenjem atributnih gramatika – Primer 2

Provera tipova u naredbama

Pravilo gramatike	**Semantička rutina**
S →id:=E	S.type = if (get_type(id.entry) = E.type) then void else type_error
S →if E then S1	S.type = if (E.type = bool and S1.type=void) then void else type_error

Realizacija type checker-a korišćenjem atributnih gramatika – Primer 2

Provera tipova u naredbama

Pravilo gramatike	**Semantička rutina**
S→while E do S1	S.type = if (E.type = bool and S1.type=void) then void else type_error
S → S1; S2	S.type = if (S1.type = void and S2.type=void) then void else type_error

Atributne gramatike u YACC-u

- Podržan je rad sa generisanim atributima.
- Ako se ne promeni podrazumevani tip atributa je **int**.
- Ako je potrebno da atributi nekih simbola imaju drugačije tipove, pre definicije terminalnih simbola, navodi de definicija %union u okviru koje se navode svi mogući tipovi atributa.
- Primer:

```
%union
{
        int index;
        float value;
}
```

Atributne gramatike u YACC-u

- Definicija tipa atributa za pojedine simbole:
  - Za teriminalne u definiciji %token

    %token<ime_clana_unije> ime_tokena1, ime_tokena2,…

    Primer:

    %token<index> id;

    %token<value> const;
  - Za neterminalne u definiciji %type

    %type<ime_clana_unije> ime_tokena1, ime_tokena2,…

    Primer:

    %type<value> E, T, F;

Atributne gramatike u YACC-u

- Korišćenje atributa simbola:
  - Semantičke rutine se pišu kao akcije pridružene smenama i izvršavaju se u trenutku kada se vrši redukcija po toj smeni.
  - Atribut simbola sa leve strane smene je uvek **$$**.
  - Atributi simbola sa desne strane smene su **$k** (gde je k redni broj simbola na desnoj strani smene.
  - Primer:

    X : Y1 Y2 ... Yn { $$=f($1,$2,…,$n); };

Atributne gramatike u YACC-u

- Primer – YACC specifikacija za generisanje jednostavnog kaklulatora:

```
%%
        S :  E '='        { printf("%d", $1); };
        E :  E '+' T      { $$ = $1 + $3; }
           | T            { $$ = $1; };
        T :  T '*' F      { $$ = $1 * $3; }
           | F            { $$ = $1; };
        F :  '(' E ')'     { $$ = $2; }
           | DIGIT         { $$ = $1; };
```

Atributne gramatike u CUP-u

- Podržan je rad sa generisanim atributima, takodje.
- U klasi *Symbol*, atribut **value** (tipa *Object*) je predvidjen za dodatne atribute.
- Da se ne bi vršilo stalno kastovanje atributa u konkretan tip, konkretni tipovi dodatnih atributa mogu biti navedeni:
  - U definiciji termimal (za terminalne simbole)

    ```
    terminal tip ime_tokena1, ime_tokena2,…
    ```

  - U definiciji nonterminal (za neterminalne simbole)

    ```
    nonterminal tip ime_simbola1, ime_simbola2,…
    ```

Atributne gramatike u CUP-u

- Korišćenje atributa simbola:
  - Semantičke rutine se pišu kao akcije pridružene smenama i izvršavaju se u trenutku kada se vrši redukcija po toj smeni.
  - Atribut simbola sa leve strane smene je uvek **result**.
  - Atributi simbola sa desne strane smene moraju biti navedeni u smeni na način:

    `ime_simbola : ime_atributa`

  - Primer:

```
X : Y1:y1 Y2:y2 ... Yn:yn   {:  result=f(y1,y2,…,yn); };
```

Atributne gramatike u CUP-u

- Primer – CUP specifikacija za generisanje jednostavnog kaklulatora:

```
terminal Integer DIGIT;
terminal ASSIGN, ADD, MUL, LEFTPAR, RIGHTPAR;
nonterminal Integer E, T, F;

S :  E:e ASSSIGN       {: System.out.println(e) :};
E :  E:e ADD T:t
     {: result = new Integer( e.intValue() + t.intValue(); :}
   |  T:t              {: result = t; :};
T :  T:t MUL F:f
     {: result = new Integer( t.intValue() * f.intValue(); :}
    | F:f       {: result = f; :};
F :  LEFTPAR E:e RIGHTPAR      {: result = e; :};
   |  DIGIT:d   {: result = d; :};
```