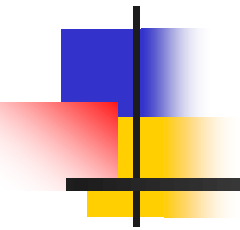


Programski prevodioci 1

Lekcija – Parsiranje od vrha ka dnu (Top Down)

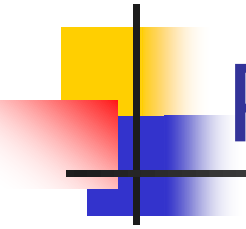
- Uvod
- Konstrukcija parsera od vrha ka dnu
- Selekcioni skupovi i LL(1) gramatike
- Transformacije gramatika u LL(1)
- Sintaksno-upravljano prevođenje od vrha ka dnu

Koncept parsiranja od vrha ka dnu

- parser na ulazu ima kompletnu ulaznu sekvencu (i poznatu gramatiku)
- potrebno je da parser rekonstruiše celokupno levo izvođenje ulazne sekvence, počev od startnog neterminala

Koncept parsiranja od vrha ka dnu

- parser u svakom koraku rada pokušava da upari tekući simbol iz sentencijalne forme koju trenutno razmatra, sa delom preostalog ulaza
  - ako je tekući simbol terminal, onda mora biti isti terminal na ulazu. Tada se samo prelazi na razmatranje sledećeg simbola iz tekuće sentencijalne forme i sledećeg ulaznog simbola.
  - ako je tekući simbol neterminal, bira se jedna od smena kojom se taj neterminal zamenjuje i na taj način dobija nova sentencijalna forma za razmatranje. Izbor smene: tekući ulaz mora biti u ***selekcionom skupu*** izabrane smene.
- procedura se ponavlja sve dok se ne rekonstruiše celokupno levo izvođenje (ukoliko je zadata ulazna sekvenca u jeziku razmatrane gramatike).



Primer

1. <S> → a <A> <B>
2. <S> → b
3. <A> → c <B>
4. <B> → d

<S> => ? => acdd

Primer

1. <S> → a <A> <B>
2. <S> → b
3. <A> → c <B>
4. <B> → d

- parser počinje rad razmatranjem startnog neterminala i prvog znaka ulazne sekvence

<S> => ? => acdd
↑ ↑ tekući ulaz
tekući neterminal

Primer

1. <S> → a <A> <B>
2. <S> → b
3. <A> → c <B>
4. <B> → d

- na parser bira smenu za <S> na osnovu tekućeg ulaza. U primeru, to je smena 1.

  $<S> \Rightarrow_{lm} a <A><B> \Rightarrow ? \Rightarrow acdd$

- U opštem slučaju, smena se bira ako je tekući ulaz neki terminalni simbol iz ***selekcionog skupa*** te smene.

Primer

1. <S> → a <A> <B>
2. <S> → b
3. <A> → c <B>
4. <B> → d

- Novo stanje:

<S> $\Rightarrow_{lm}$ a <A><B>=> ? => acdd

tekući simbol tekući ulaz

- Sada vidimo da se a u sentencijalnoj formi i a u završnoj sentenci uparuju, pa tekući postaju:

<S> $\Rightarrow_{lm}$ a <A><B>=> ? => acdd

tekući simbol tekući ulaz

Primer

1. <S> → a <A> <B>
2. <S> → b
3. <A> → c <B>
4. <B> → d

- biramo smenu 3. za zamenu <A>:

<S> $\Rightarrow_{lm}$ a <A><B> $\Rightarrow_{lm}$ a c <B> <B> => ? => acdd

tekući simbol — tekući ulaz

- Uparivanje tekućeg terminala i tekućeg ulaza:

<S> $\Rightarrow_{lm}$ a <A><B> $\Rightarrow_{lm}$ a c <B> <B> => ? => acdd

tekući simbol — tekući ulaz

- biramo smenu 4 na osnovu tekućeg ulaznog simbola:

<S> $\Rightarrow_{lm}$ a <A><B> $\Rightarrow_{lm}$ a c <B> <B> $\Rightarrow_{lm}$ a c d <B> => ? => acdd

tekući simbol — tekući ulaz

Primer

- uparivanje tekućeg terminala i tekućeg ulaza:

<S> $\Rightarrow_{lm}$ a <A><B> $\Rightarrow_{lm}$ a c <B> <B> $\Rightarrow_{lm}$ a c d <B> => ? => acdd

tekući simbol tekući ulaz

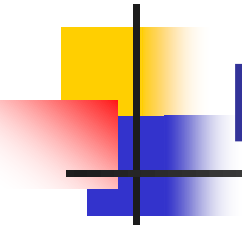
- sada biramo smenu 4:

<S> $\Rightarrow_{lm}$ a <A><B> $\Rightarrow_{lm}$ a c <B> <B> $\Rightarrow_{lm}$ a c d d => ? => acdd

tekući simbol tekući ulaz

- uparivanjem tekućeg terminala i tekućeg ulaza konstatujemo da je rekonstrukcija levog izvođenja uspešno okončana

- Uvod
- Konstrukcija parsera od vrha ka dnu
- Selekcioni skupovi i LL(1) gramatike
- Transformacije gramatika u LL(1)
- Sintaksno-upravljano prevođenje od vrha ka dnu



Metodi konstrukcije TD parsera

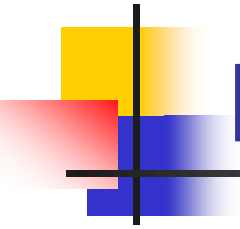
- metod rekurzivnog spusta
- tabelarni (u vidu potisnog automata)

Metod rekurzivnog spusta

- Primer gramatike za koju želimo da konstruišemo parser:

1. <E> $\rightarrow$<T> <E_list>
2. <E_List> $\rightarrow$ + <T> <E_List>
3. <E_list> $\rightarrow \varepsilon$
4. <T> $\rightarrow$ <P> <T_list>
5. <T_List> $\rightarrow$ * <P> <T_List>
6. <T_list> $\rightarrow \varepsilon$
7. <P> $\rightarrow$ (<E>)
8. <P> $\rightarrow$ I

Principi konstrukcije



- Programsko rešenje čini glavni program i niz uzajamno rekurzivnih potprograma. Za svaki neterminal gramatike obezbeđuje se jedan potprogram (u konkretnom slučaju imaćemo potprograme PROCE, PROCE_LIST, PROCT, PROCT_LIST i PROCP).
- Potprogram za određeni neterminal ima ulogu da prepozna primenu neke od smena sa datim neterminalom na levoj strani i upari neterminal sa ulazom u skladu sa prepoznatom smenom.

Primer

globalna promenljiva IN predstavlja tekući ulazni znak

```
procedure PROCP:
        case IN of
            'I'   : goto P8;
            '('   : goto P7;
            '+', '*', ')', '┤' : REJECT;
        end case;
   P8: kod za prepoznavanje                    8. smene
   P7: kod za prepoznavanje                    7. smene
end procedure;
```

primena pojedinih smena prepoznaje se koristeći njihove ***selekcione*** skupove

Kod za prepoznavanje smene

- generalno, segmenti koda odgovaraju pojedinim gramatičkim simbolima desne strane smene
- neterminalu odgovara poziv odgovarajućeg potprogrma, npr. za <E> je PROCE
- terminalu generalno odgovara sekvenca u kojoj se uparuje terminal u smeni sa tekućim ulazom:
  - ispituje da li je tekući ulaz jednak terminalu,
  - ako jeste, procedurom NEXTCHAR se čita sledeći znak sa ulaza ()
  - ako nije, akcijom REJECT se odbija ulazna sekvenca

Primer za 7. <P> → (<E>) :

```
P7:
   IN = NEXTCHAR();
   call PROCE;
   if ( IN = ‘)’ )
        then  IN = NEXTCHAR();
                        return;
        else   REJECT;
   end if
```

Glavni program

- inicijalizuje promenljivu IN da sadrži prvi ulazni znak.
- poziva potprograma za startni neterminal čime inicira uparivanje ulazne sekvence
- potom ispituje se da li je procesiran ceo ulaz što označava da se ulazna sekvenca prihvata.
- U suprotnom, ako tekući znak nije marker kraja, sekvenca se odbija.

Kompletan primer

```
Glavni program:
      IN = NEXTCHAR();
      call PROCE;
      if (IN = '⊣')
        then    ACCEPT;
        else    REJECT;
      end if;
end;

procedure PROCE:
      case IN of
        'I', '(': goto P1;
        '+','*',')','⊣':REJECT;
      end case;
P1: call PROCT;
      call PROCE_LIST;
      return;
end procedure;

procedure PROCP:
      case IN of
        'I'   : goto P8;
        '('   : goto P7;
        '+','*',')','⊣':REJECT;
      end case;
P8: IN = NEXTCHAR();
      return;
P7: IN = NEXTCHAR();
      call PROCE;
      if ( IN = ')')
        then IN=NEXTCHAR();
        else REJECT;
      end if;
end procedure;
procedure PROCT:
   case IN of
      'I', '(':  goto P4;
      '+','*',')','⊣': REJECT;
   end case;
P4:call PROCP;
      call PROCT_LIST;
      return;
end procedure;

procedure PROCE_LIST:
   case IN of
      '+' : goto P2;
      ')', '⊣'  : goto P3;
      'I', '(', '*'   : REJECT;
   end case;
P2: IN = NEXTCHAR();
      call PROCT;
      call PROCE_LIST;
      return;;
P3: return;
end procedure;

procedure PROCT_LIST:
   case IN of
      '*'    : goto P5;
      '+', ')', '⊣'   : goto P6;
      'I', '('  : REJECT;
   end case;
P5: IN = NEXTCHAR();
      call PROCP;
      call PROCT_LIST;
      return;
P6: return;
end procedure;
```

- Uvod
- Konstrukcija parsera od vrha ka dnu
- Selekcioni skupovi i LL(1) gramatike
- Transformacije gramatika u LL(1)
- Sintaksno-upravljano prevođenje od vrha ka dnu

Selekcioni skupovi

- Skup ulaznih simbola koji mogu predstavljati tekući ulaz u trenutku prepoznavanja smene <X> $\rightarrow \alpha$ od strane potisnog automata, naziva se **selekcioni skup** smene, u oznaci SELECT(<X> $\rightarrow \alpha$)

LL(1) gramatike

- Da bi parser jednoznačno prepoznao smenu, selekcioni skupovi smena sa istom levom stranom moraju biti disjunktni:
  - SELECT(<X> $\rightarrow \alpha$) $\cap$ SELECT(<X> $\rightarrow\beta$) = $\varnothing$, $\forall\alpha,\beta$: $\alpha\neq\beta$
- Gramatike kod kojih je ovo ispunjeno nazivaju se **LL(1) gramatike**
- Značenje skraćenice: LL(1)
  - parsira se sa (L)eva na desno
  - nalazi se (L)evo izvođenje ulazne sekvence
  - koristi se po (1) simbol sa ulaza unapred (enlg. lookahead symbol)

Računanje selekcionih skupova

- Za proizvoljnu produkciju <A> $\rightarrow \alpha$, važi da je

  SELECT(<A> $\rightarrow \alpha$) = FIRST(α),

ako α nije poništiva sekvenca, ili

  SELECT(<A>$\rightarrow\alpha$)=FIRST(α) $\cup$ FOLLOW(<A>),

u suprotnom.

Primer

- Za sledeću gramatiku sa startnim simbolom <A>:
  1. <A>$\rightarrow$ a <B> <C>
  2. <A> $\rightarrow$ b <B>
  3. <A> $\rightarrow \varepsilon$
  4. <B> $\rightarrow$ a <B> b
  5. <B> $\rightarrow \varepsilon$
  6. <C> $\rightarrow$ b <C>
  7. <C> $\rightarrow$ c
- Odrediti SELECT skupove za svaku smenu i ispitati da li se radi o LL(1) gramatici.

Primer

- Moramo odrediti poništive neterminale, FIRST i FOLLOW skupove:

1. <A> →a<B><C> 2. <A> → b<B> 3. <A> → ε 4. <B> → a<B>b 5. <B> → ε 6. <C> → b<C> 7. <C> → c	Pon = { <A>, <B>} FIRST(<A>)= {a, b} FIRST(<B>)={a} FIRST(<C>)={b, c} FOLLOW(<A>)= {⊣} FOLLOW(<B>)= FIRST(<C>)∪FOLLOW(<A>)∪{b}= {b,c, ⊣} FOLLOW(<C>) = FIRST(ε)∪FOLLOW(<A>)= {⊣}

Primer

SELECT(1)={a}

SELECT(2)={b}

SELECT(3)=FOLLOW(<A>)={⊣}

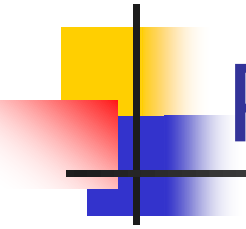
SELECT(4)={a}

SELECT(5)=FOLLOW(<B>)={ b, c, ⊣ }

SELECT(6)={b}

SELECT(7)={c}

- Ovo jeste LL(1) gramatika jer ispunjava uslov da smene koje imaju identične leve strane, imaju disjunktne selekcione skupove.



Primer

- Provera za smene 1. , 2. i 3. :

  $SELECT(1) \cap SELECT(2) = \varnothing$

  $SELECT(1) \cap SELECT(3) = \varnothing$

  $SELECT(2) \cap SELECT(3) = \varnothing$

- Provera za smene 4. i 5. :

  $SELECT(4) \cap SELECT(5) = \varnothing$

- Provera za smene 6. i 7. :

  $SELECT(6) \cap SELECT(7) = \varnothing$

- Uvod
- Konstrukcija parsera od vrha ka dnu
- Selekcioni skupovi i LL(1) gramatike
- Transformacije gramatika u LL(1)
- Sintaksno-upravljano prevođenje od vrha ka dnu

Transformacije gramatika za dovođenje u LL(1) oblik

- Uklanjanje direktne leve rekurzije
- Uklanjanje indirektne leve rekurzije
- Leva faktorizacija

Uklanjanje direktne leve rekurzije

<X> $\rightarrow$ <X> α

<X> $\rightarrow \beta$

- Ideja transformacije: <X> opisuje sekvencu koja počinje jednim od β simbola, a nastavlja se nizom α simbola.
- Ovaj niz α simbola posle početnog β, možemo opisati uvođenjem novog neterminala <Y> i korišćenjem desno rekurzivnih smena

Uklanjanje direktne leve rekurzije

$\langle X \rangle \rightarrow \langle X \rangle \alpha_1$

....

$\langle X \rangle \rightarrow \langle X \rangle \alpha_n$

$\langle X \rangle \rightarrow \beta_1$

.....

$\langle X \rangle \rightarrow \beta_n$

transformiše se u:

$\langle X \rangle \rightarrow \beta_1 \langle Y \rangle$

....

$\langle X \rangle \rightarrow \beta_n \langle Y \rangle$

$\langle Y \rangle \rightarrow \alpha_1 \langle Y \rangle$

....

$\langle Y \rangle \rightarrow \alpha_n \langle Y \rangle$

$\langle Y \rangle \rightarrow \varepsilon$

Uklanjanje indirektne leve rekurzije

- Primer indirektne leve rekurzije:

  <X> $\rightarrow$ <Y> a

  <Y>$\rightarrow$ <X> b

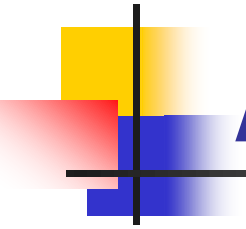
  <Y>$\rightarrow$ c

Uklanjanje indirektne leve rekurzije

- Algoritam:

Poređati neterminale u nekom redosledu A1, A2, …, An
for i $\leftarrow$ 1 to n
    for j $\leftarrow$ 1 to i – 1
        zameniti svaku smenu oblika
            $\langle A \rangle_i \rightarrow \langle A \rangle_j \gamma$
        sa
            $\langle A \rangle_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \ldots \mid \delta_k\gamma$,
        gde su $\langle A \rangle j \rightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$ sve aktuelne smene za $\langle A \rangle_j$
    end_for
    eliminisati direktno levo rekurzivne smene za $\langle A \rangle_i$
    novodobijene neterminale dodati na kraj
end_for

- pretpostavka algoritma je da inicijalna gramatika ne sadrži cikluse ($\langle A \rangle_i \Rightarrow^+ \langle A \rangle_i$)



Primer

- Ukloniti levu rekurziju iz sledeće gramatike:

<S> → <E>

<E> → <E> + <T>

<E> → <T>

<T> → <E> - <T>

<T> → id

Primer

- usvajamo redosled <S>, <E>, <T>
- prva iteracija, i=1, <A>i = <S>, ne ulazi se u unutrašnju petlju, nema direktne leve rekurzije po <S>, nema promene gramatike
- i=2, j=1, <A>i = <E>, <A>j = <S>, u unutrašnjoj petlji nema promena, u spoljašnjoj eliminacija dir. rekurzije po <E>:

STARO

<S> → <E>
<E> → <E> + <T>
<E> → <T>
<T> → <E> - <T>
<T> → id

NOVO

<S> → <E>
<E> → <T> <E'>
<E'> → + <T> <E'>
<E'>→ ε
<T> → <E> - <T>
<T> → id

Primer

- i=3, j=1, <A>i = <T>, <A>j = <S>, nema promene
- i=3, j=2, <A>i = <T>, <A>j = <E>, menja se 5. smena

STARO

<S> → <E>
<E> → <T> <E'>
<E'> → + <T> <E'>
<E'>→ ε
<T> → <E> - <T>
<T> → id

NOVO

<S> → <E>
<E> → <T> <E'>
<E'> → + <T> <E'>
<E'>→ ε
<T> → <T> <E'> - <T>
<T> → id

Primer

- Eliminacija direktne leve rekurzije po <T>:

STARO

<S> → <E>
<E> → <T> <E'>
<E'> → + <T> <E'>
<E'>→ ε
<T> → <T> <E'> - <T>
<T> → id

NOVO

<S> → <E>
<E> → <T> <E'>
<E'> → + <T> <E'>
<E'>→ ε
<T> → id <T'>
<T'>→ <E'> - <T> <T'>
<T'> → ε

Primer

- i=4, <A>i = <E'>, za j=1, 2, 3 nema promene
- i=5, <A>i = <T'>, za j=1, 2, 3 nema promene, za j=4, <A>j = <E'>:

STARO

<S> → <E>
<E> → <T> <E'>
<E'> → + <T> <E'>
<E'>→ ε
<T> → id <T'>
<T'>→ <E'> - <T> <T'>
<T'> → ε

NOVO

<S> → <E>
<E> → <T> <E'>
<E'> → + <T> <E'>
<E'>→ ε
<T> → id <T'>
<T'>→ +<T><E'> - <T> <T'>
<T'>→ - <T> <T'>
<T'> → ε

Leva faktorizacija

- Pravilo leve faktorizacije glasi (uvodi se novi neterminal <S'>):

$$\langle S\rangle \to \alpha\ \beta 1 \quad \ldots \quad \langle S\rangle \to \alpha\ \beta n \qquad \to \qquad \langle S\rangle \to \alpha\ \langle S'\rangle \quad \langle S'\rangle \to \beta 1 \quad \ldots \quad \langle S'\rangle \to \beta n$$

- Da li ove transformacije garantuju dovođenje u LL(1) oblik bilo koje bezkontekstne gramatike?
  - ne, problem transformacije u LL(1) je neodlučiv.

- primer jezika koji se ne može opisati LL(1) gramatikom:

  $\{ a^n 0 b^n \mid n \geq 1 \} \cup \{ a^n 1 b^{2n} \mid n \geq 1 \}$

- gramatika za ovaj jezik – nije LL(1):

  <S> → <S'>

  <S> → <S">

  <S'> → a0b

  <S'>→ a<S'>b

  <S"> → a1bb

  <S">→ a<S">bb

- 0 i 1 ne mogu biti u istoj smeni
- sa ulaza se moraju pročitati svi a simboli, kojih ima proizvoljan broj, da bi se odredilo izvođenje

- Uvod
- Konstrukcija parsera od vrha ka dnu
- Selekcioni skupovi i LL(1) gramatike
- Transformacije gramatika u LL(1)
- Sintaksno-upravljano prevođenje od vrha ka dnu

Sintaksno-upravljano prevođenje od vrha ka dnu

- Kojim redom rekurz. parser pri prepoznavanju smene <A> → <B><C> određuje atribute?
- ***nasleđeni*** *atributi X su* ***ulazni*** *parametri za PROCX*
- ***sintetizovani*** *atr. X su* ***izlazni*** *parametri za PROCX*

1. nasleđeni atributi od <A> su već poznati (prosleđuje pozivalac).
2. nasleđeni atributi od <B> zavise od nasleđenih od <A> i oni se izračunavaju
3. posle uparivanja <B> sa ulazom saznaćemo sintetizovane atribute <B>
4. nasleđeni atributi od <C> zavise od nasleđenih od <A> i bilo kog atributa <B>

Sintaksno-upravljano prevođenje od vrha ka dnu

- Kojim redom rekurz. parser pri prepoznavanju smene <A> → <B><C> određuje atribute?

5. posle uparivanja sa ulazom <C> saznajemo sintetizovane atribute od <C>
6. kad uparimo celu desnu stranu izračunavamo sintetizovane atribute od <A>
7. vraćanje kontrole pozivaocu sa sint. atr. <A>

- moraju se uvesti ograničenja u pravila izračunavanja atributa, da bi parser paralelno sa konstrukcijom stabla izvođenja (od vrha ka dnu) računao vrednosti svih atributa).
- Gramatike koje zadovoljavaju ova pravila nazivaju se **L-atributivne gramatike**

Uslovi L-atributivnosti gramatike

- **Nasleđeni atributi simbola sa desne strane** smene mogu da zavise od nasleđenog atributa simbola sa leve strane i bilo kog atributa bilo kog simbola koji je na desnoj strani ispred posmatranog simbola.
- **Sintetizovani atribut simbola sa leve strane** smene može da zavisi od nasleđenih atributa tog simbola i od bilo kog atributa bilo kog simbola sa desne strane.
- **Sintetizovani atribut akcionog simbola** sme da zavisi isključivo od nasleđenog atributa istog akcionog simbola.

Primer

- Sledeće smene su deo neke atributivne gramatike. U svakoj od njih, *p*, *q* i *r* su nasleđeni atributi, a *s* i *t* sintetizovani. Za svaku od smena pronaći od kojih sve atributa mogu da zavise atributi *p* i *r* da bi smena bila L-atributivna.

a) $\langle S \rangle_{s,q} \rightarrow a_u \langle S \rangle_{t,p} \langle A \rangle_r$

b) $\langle S \rangle_{s,q} \rightarrow \{c_p\}\ b_u \langle S \rangle_{t,r}$

c) $\langle S \rangle_{s,q} \rightarrow c_u \langle A \rangle_r \langle S \rangle_{t,p}$

Rešenje

a) $\langle S \rangle_{s,q} \rightarrow a_u \langle S \rangle_{t,p} \langle A \rangle_r$

$p = f_1(q, u)$, $r = f_2(q, u, t, p)$

$s = f_3(q, u, t, p, r)$

b) $\langle S \rangle_{s,q} \rightarrow \{c_p\}\ b_u \langle S \rangle_{t,r}$

$p = f_1(q)$,

$r = f_2(q, p, u)$ u - sintetizovan.

c) $\langle S \rangle_{s,q} \rightarrow c_u \langle A \rangle_r \langle S \rangle_{t,p}$

$p = f_1(q, u, r)$

$r = f_2(q, u)$

Konstrukcija sintaksno-upravljanog procesora za L-atributivne LL(1) translacione gramatike

- ***Osnovne ideje***
- Nasleđeni atributi se prenose po vrednosti, sintetizovani atributi se prenose po adresi
- Svi atributi koji su sa desne strane postaju lokalne promenljive.
- Svi atributi istog simbola sa leve strane moraju imati isto ime (formalni parametri)
- Pravilo x $\leftarrow$ y može se eliminisati tako da se x preimenuje u y ili obrnuto (pri tome se ne smeju menjati imena atributa sa leve strane)

Primer

1. $\langle S \rangle_{q} \rightarrow a_{x1} \langle A \rangle_{x,y,z} \langle S \rangle_{q1} \{b_{w}\}$
   $w \leftarrow y * z + q1 \quad x \leftarrow x1 \quad q \leftarrow q1$
2. $\langle S \rangle_{r1} \rightarrow b\ a_{y} \{c_{w}\} \langle A \rangle_{u,r,x} \{b_{v}\}$
   $w \leftarrow y + 3 \quad r1 \leftarrow r \quad v \leftarrow r * x - w \quad u \leftarrow w + 2$
3. $\langle A \rangle_{x,y,z1} \rightarrow a_{q} \{c_{v}\} \langle A \rangle_{x1,z,u1} \langle A \rangle_{u,t,y1}$
   $v \leftarrow x - q \quad x1 \leftarrow x \quad y \leftarrow y1 \quad z1 \leftarrow z$
   $u \leftarrow 3$
4. $\langle A \rangle_{t,s,s1} \rightarrow b\ a_{q} \{b_{t1}\}\ a_{s2}$
   $t1 \leftarrow t \quad (s,s1) \leftarrow s2$

Preuređena gramatika

1. $\langle S \rangle_q \rightarrow a_x \langle A \rangle_{x,y,z} \langle S \rangle_q \{b_w\}$
   (x1 postalo x, q1 postalo q)
   $w \leftarrow y * z + q$
- $\langle S \rangle_q \rightarrow b\ a_y \{c_w\} \langle A \rangle_{u,r,x} \{b_v\}$
  (r1 postalo q, r postalo q)
  $w \leftarrow y + 3 \qquad v \leftarrow q * x - w \qquad u \leftarrow w + 2$
- $\langle A \rangle_{x,y,z1} \rightarrow a_q \{c_v\} \langle A \rangle_{x,z1,u1} \langle A \rangle_{u,t,y}$
  (x1 postalo x, z postalo z1, y1 postalo y)
  $v \leftarrow x - q \qquad u \leftarrow 3$
- $\langle A \rangle_{x,y,z1} \rightarrow b\ a_q \{b_x\}\ a_{s2}$
  (t i t1 postali x, s postalo y, s1 postalo z1)
  $(y,z1) \leftarrow s2$

Implementacija parsera (pseudokod)

```
Program Parser;
    type ulaz =  record of
                    ime:char;
                    atrib: integer;
                 end;
    var q: integer;
        U: ulaz;
 procedure ADVANCE;
  begin
          U.ime = sledeći ulazni simbol;
          U.atrib = njegov atribut;
  end;
 procedure ACCEPT;
    begin
          kraj rada sa prihvatanjem ulaza
    end;
 procedure REJECT;
    begin
          kraj rada sa neprihvatanjem ulaza
    end;
```

Implementacija parsera (pseudokod)

```
procedure S (var q :integer)
        var x,y,z,w,u,v :integer;
        begin
            case U.ime of
              'a': x := U.atrib;
                   ADVANCE;
                   A(x,y,z);
                   S(q);
                   w := y * z + q;
                   out (b,w);
               'b': ADVANCE;
                   if U.ime <> 'a'  then REJECT;
                   y := U.atrib;
                   ADVANCE;
                   w := y + 3;
                   out ('c', w);
                   u := w + 2;
                   A(u,q,x);
                   v := q * x - w;
                   out ('b', v);
             default: REJECT;
             end; {od "case"}
        end; {od "begin"}
```

Implementacija parsera (pseudokod)

```
procedure A(x:integer,var y,z1:integer)
        var q,v,u,u1,s2, t:integer;
        begin
            case U.ime of
                'a': q:=U.atrib;
                     ADVANCE;
                     v:=x-q;
                     u:=3;
                     out ('c', v);
                     A(x,z1,u1);
                     A(u,t,y);
                'b': ADVANCE;
                     if U.ime <> 'a' then REJECT;
                     q:=U.atrib;
                     ADVANCE;
                     out ('b', x);
                     if U.ime <>'a' then REJECT;
                     s2:=U.atrib;
                     ADVANCE;
                     y:=s2;
                     z1:=s2;
                 default: REJECT;
             end;
        end;
```

Implementacija parsera (pseudokod)

```
begin {main}
    ADVANCE;
    S(q);
    if U.ime=ENDMARKER then ACCEPT
    else REJECT;
end.
```

Oporavak od grešaka

- Procedure za neterminale imaju dodatni parametar koji prima niz terminala.
- Razmotrimo PROCA(termset) povezanu sa nekim neterminalom A. Kada se pozove PROCA, svaki član termset-a može biti ispravan tekući ulaz posle povratka iz PROCA. Ako se unutar PROCA detektuje greška, na ulazu se gutaju simboli dok ne naiđe neki iz termseta. Potom se vrši povratak iz PROCA.

Oporavak od grešaka, primer

- Gramatika:
  1. <S> → [E]
  2. <S> → (E)
  3. <E> → a

- Termset koji se prosleđuje proc. E obuhvata ‘]’ ako je pozvano iz koda za prvu smenu, ‘)’ ako je pozvano iz druge smene, plus simboli iz termseta pozivajuce proc. S.

- Pošto marker kraja ulaza $ sledi S, svaki termset uključuje $. U najgorem slučaju, ulaz programa guta se do $, posle čega će biti izvršen povratak iz svih aktivnih procedura.

Primer

Gramatika:

1. <S> → [E]
2. <S> → (E)
3. <E> → a

```
procedure S(termset)
  case IN of
  '[' : IN = NEXTCHAR();
        call E(termset U ']');
        if IN <> ']'
             call RECOVERY(']', termset);
        endif;
  '(':  IN = NEXTCHAR();
        call E(termset U '}');
        if IN <> ']'
             call RECOVERY('}', termset);
        endif;
   default: call RECOVERY('([',termset);
   end case;
end procedure;

procedure E(termset)
  if IN == 'a'
  then IN=NEXTCHAR();
  else call RECOVERY('a', termset);
  endif
end procedure;
```

```
procedure RECOVERY( exp, termset)
   call PRINT( 'ERROR Expected', exp,
                    'but found', IN);
   while IN not in termset do
     IN=NEXTCHAR();
   end;
end procedure;

Glavni program:
  IN=NEXTCHAR();
  call S('$');
  if IN <> '$'
     call RECOVERY('$', '$');
  endif;
end;
```