Formalni jezici i jezični procesori I

PARSERI

prof. dr. sc. Sanda Martinčić - Ipšić
smarti@uniri.hr

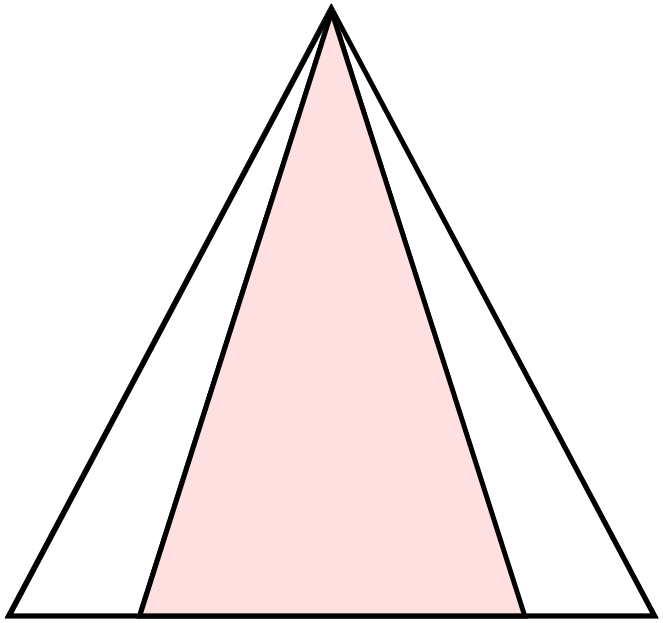
Parsiranje

- postupak prepoznavanja niza i gradnja generativnog stabla na temelju zadanih produkcija kontekstno neovisne gramatike

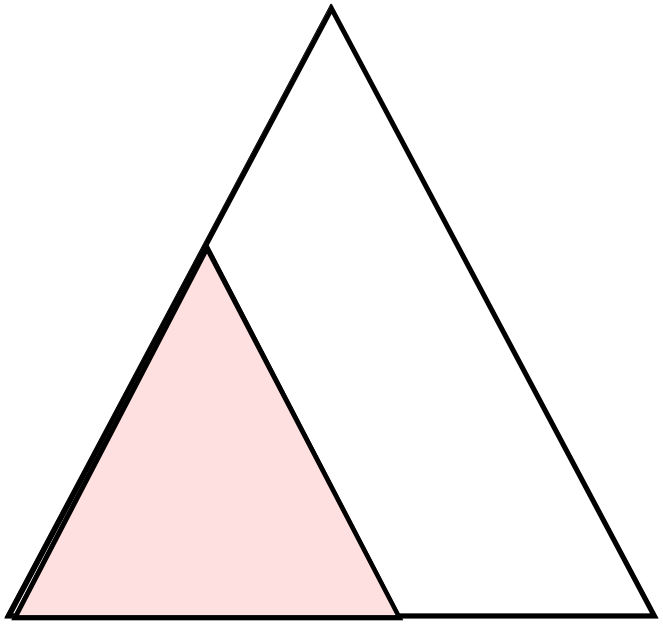
- želimo odrediti je li ulazni niz (program) generirala gramatika (je li u skladu sa sintaksnim pravilima zapisanima u gramatici)

Parsiranje Od-vrha-prema-dnu vs. Od-dna-prema-vrhu

- veličina stabla parsiranja



Top-down



Bottom-up

Parsiranje niza

- prepoznavanje niza – određivanje
  - pripada li niz w jeziku L(G)?
  - generira li gramatika G niz w?
  - w – C program, SQL query, XML dokument, ...
- **generativno stablo** (stablo parsiranja) koristi se za interpretaciju niza (u listovima isključivo završni znakovi)
- **parsiranje** – prepoznavanje niza + gradnja generativnog stabla (za niz w i gramatiku G)
  - od vrha prema dnu
  - od dna prema vrhu

Generativno stablo za gramatiku G=(V, T, P, S)

- **čvorovi** stabla: $V \cup T \cup \{\varepsilon\}$
- **korijen** stabla: S – početni nezavršni znak
- **unutrašnji čvorovi**: $A \in V$ (primjena produkcija P iz G)
  - čvor n ima djecu: $n_1, n_2, ..n_k$ i čvor n je označen s A a čvorovi $n_1, n_2, ..n_k$ sa $X_1, X_2, ..X_k$ onda je $A \rightarrow X_1,X_2..X_k$ produkcija iz P
- **ε**: isključivo označava listove stabla (to je jedino dijete svojih roditelja – unutrašnjih čvorova)
- listovi stabla: $T \cup \{\varepsilon\}$
  - ako ih čitamo s lijeva na desno dobijemo niz w iz L(G)
  - produkcije P se primjenjuju dok u čvorovima nisu isključivo završni znakovi niza w

Parsiranje od vrha prema dnu

- početak: korijen je S – početni nezavršni znak
- primijenimo produkcije P za gradnju ostalih čvorova
- postupak ponavljamo sve dok listovi nisu označeni isključivo završnim znakovima
- učinkovito i jednostavno za programiranje
- LL(k) parsiranje-
  - 1. L : od lijeva na desno čita se niz
  - 2. L : produkcije se primjenjuju na krajnje lijevi nezavršni znak u generiranom međunizu
  - k : odluka o primjeni produkcije donosi se na osnovu k pročitanih znakova; k≥1

Jednoznačna gramatika

- **G**=({C,S,S1,S2},{begin, end, while, do, ;, :=, ≠, var}, P, C)
- **P**={C→ begin S1 end

  S1→S S2

  S2→ ; S1 | ε

  S→var := var | while var ≠ var do S | begin S1 end }
- **niz w:**

begin var:= var;

while var ≠ var do

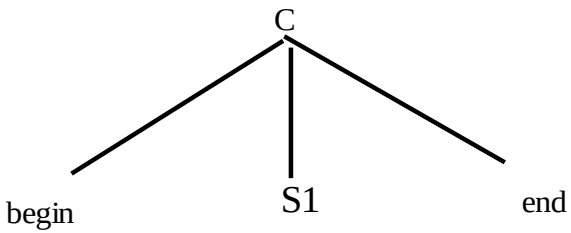
var:= var

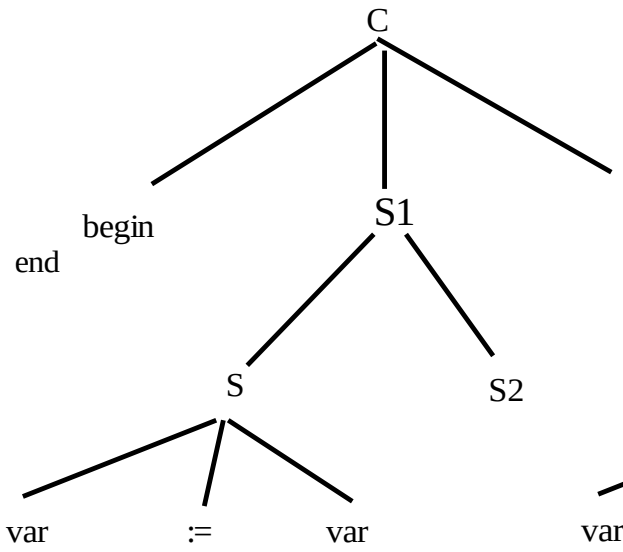
end

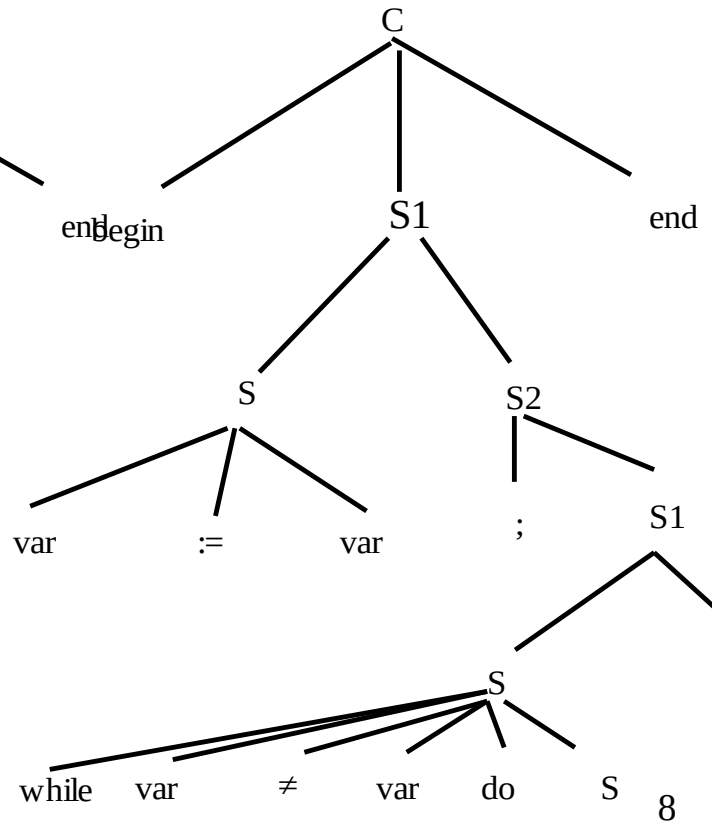
Generativno stablo

niz:begin var:= var;
while var ≠ var do
var:= var
end

P={C→ begin S1 end
S1→S S2
S2→ ; S1 | ε
S→var := var | while var ≠ var do S | begin S1 end }







begin var:=
var ; while
var ≠ var do
var:= var
end

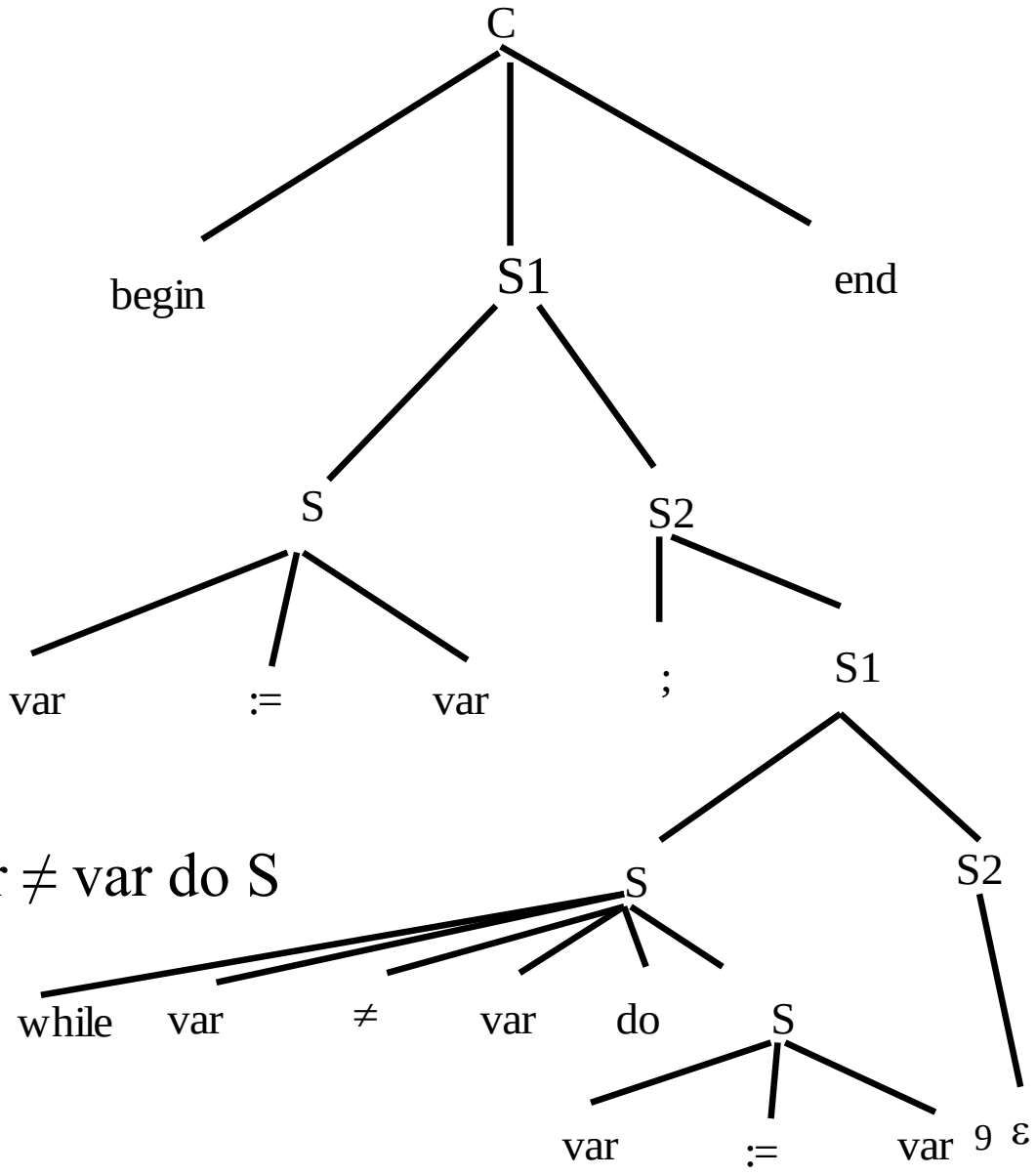
begin var:=
var **;** while
var ≠ var do
var:= var
end

begin var:= var **;** while var ≠ var do var:= var end

Generativno stablo

```
begin var:= var;
    while var ≠ var
   do
    var:= var
end
```

C→ begin S1 end
S1→S S2
S2→ ; S1 | ε
S→var := var | while var ≠ var do S
| begin S1 end



LL parser i LL gramatika

- prvi L: ulazni niz se čita s lijeva na desno (left-to-right)
- drugi L: produkcije se primjenjuju na krajnje lijevi nezavršni znak u generiranom međunizu (leftmost derivation)

- LL(1) – parsira se čitanjem 1 znaka u nizu
- LL(k) – parsira se čitanjem k znakova u nizu

Primjer izgradnje parsera za gramatiku

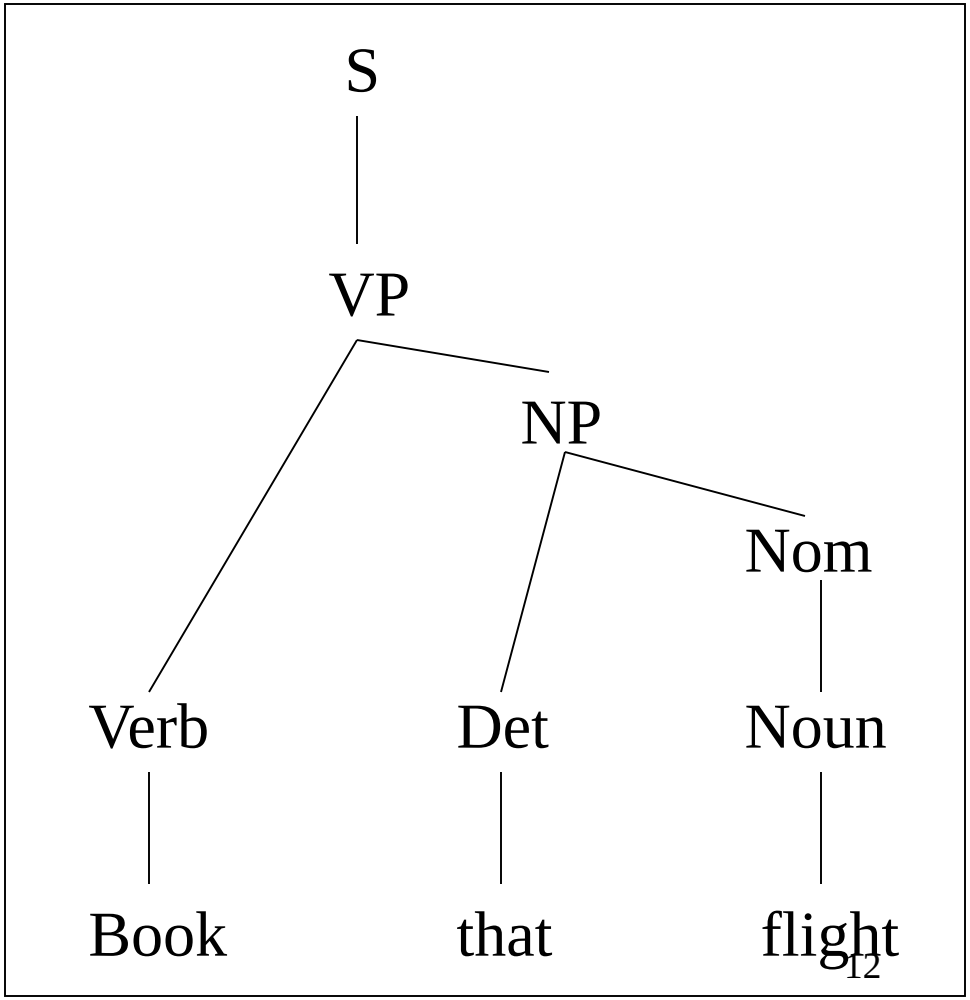
S → NP VP	Det → that | this |a
S → Aux NP VP	Noun → book | flight | meal | money
S → VP	Verb → book | include | prefer
NP → Det Nominal	Aux → does
Nominal → Noun	Proper-Noun → Houston | TWA
Nominal → Noun Nominal	Prep → from | to | on
NP → Proper-Noun	
VP → Verb	
VP → Verb NP	
Nominal → Nominal PP	

Book that flight.

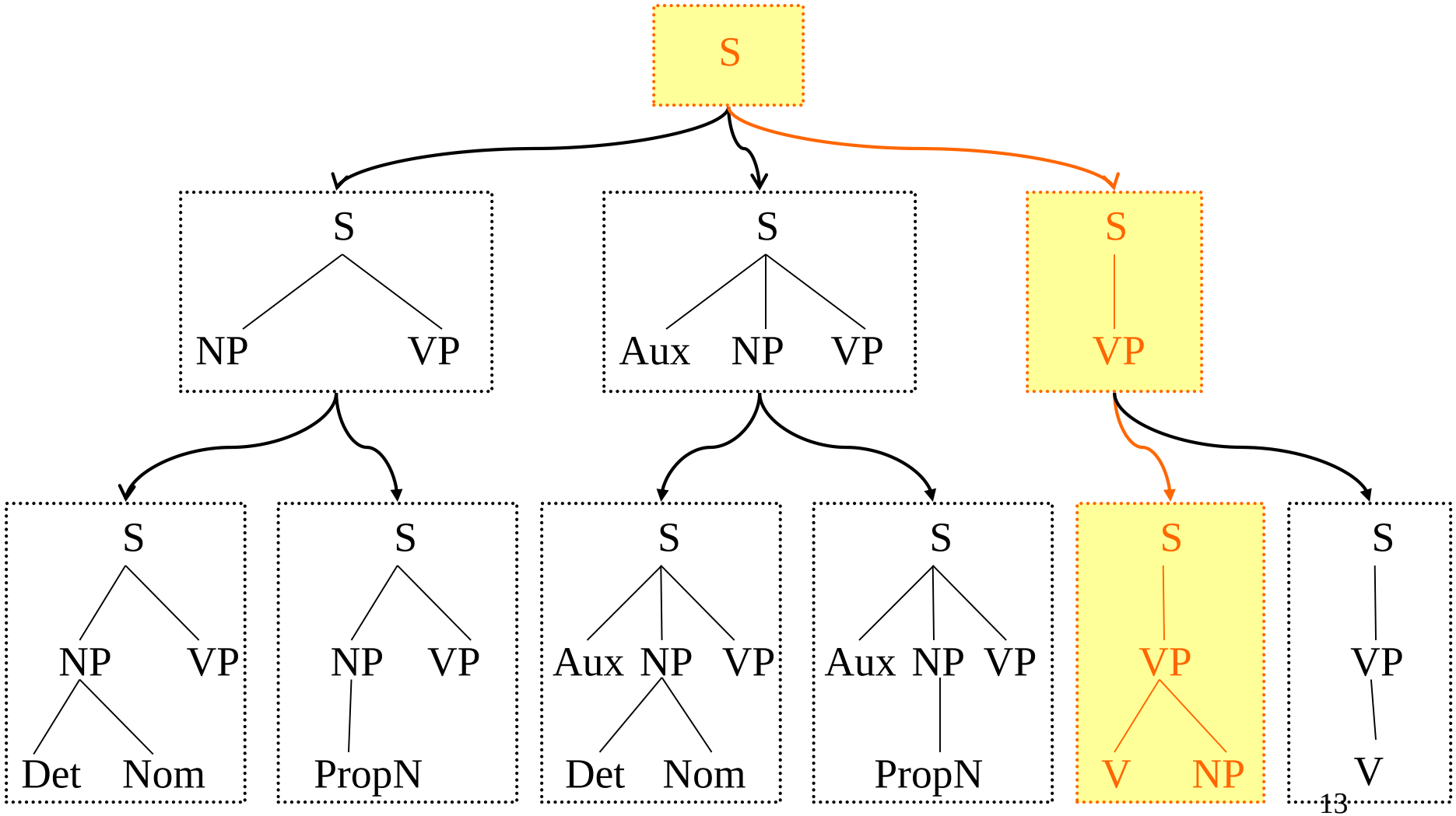
Generativno stablo

Book that flight

S → VP
VP → Verb NP
NP → Det Nominal
Nominal → Noun
Verb → book
Det → that
Noun → flight



Primjer parsiranja od vrha prema dnu



Rekurzivni spust I

- uporaba rekurzije za parsiranje od vrha prema dnu
- nezavršnim znakovima pridružuju se potprogrami koji ispituju da li desna strana produkcije odgovara pročitanom nizu
- završni znakovi desnih stana produkcija uspoređuju se sa nizom
- $\perp$ - oznaka za kraj niza

Programska realizacija algoritma:

1. u glavnom programu pročita se prvi znak niza w i pozove se potprogram za početni nezavršni znak gramatike
   - ako se pri izvođenju pročita oznaka za kraj niza $\perp$ niz se prihvaća u suprotnom se ne prihvaća

Rekurzivni spust II

```
Glavni program( )
     {
        ulaz = prvi znak iz niza w;
        potprogram pridružen početnim
              nezavršnom znaku S;
        ako (ulaz !=⊥)
           ispiši (“w  ∉ L(G)”);
        inače
           ispiši (“w  ∈ L(G)”);
     }
```

Rekurzivni spust III

2. za svaku produkciju svakog nezavršnog znaka gradi se poseban potprogram $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid .. \mid a_n\beta_n$

```
A( )
{
slučaj(ulaz)
{
            a1:
                ispitaj slijedeće znakove prema β1;
            a2:
                ispitaj slijedeće znakove prema β2;
              ....
             an:
                ispitaj slijedeće znakove prema βn;
        svi ostali znakovi:
                ispiši ("w  ∉ L(G)");
            }
}
```

Rekurzivni spust IV

3. za desne stane produkcija $\beta_i : A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid .. \mid a_n\beta_n$

- za bilo koji **završni znak** b na desnoj stani produkcije $A \rightarrow a\alpha b\gamma$ koji nije na krajnje lijevom mjestu

```
A( )
{    ...
     ulaz = slijedeći znak iz ulaznog niza w;
     ako (ulaz != b)
               ispiši ("w  ∉ L(G)");
     ... }
```

- za bilo koji **nezavršni znak** B na desnoj stani produkcije $A \rightarrow a\alpha B\gamma$ koji nije na krajnje lijevom mjestu

```
A( )
{    ...
     ulaz = slijedeći znak iz ulaznog niza w;
     B( );
     ...
}
```

Rekurzivni spust V

- ako je nezavršni znak B na na krajnje lijevom mjestu produkcije $A \rightarrow Ba\alpha\gamma$

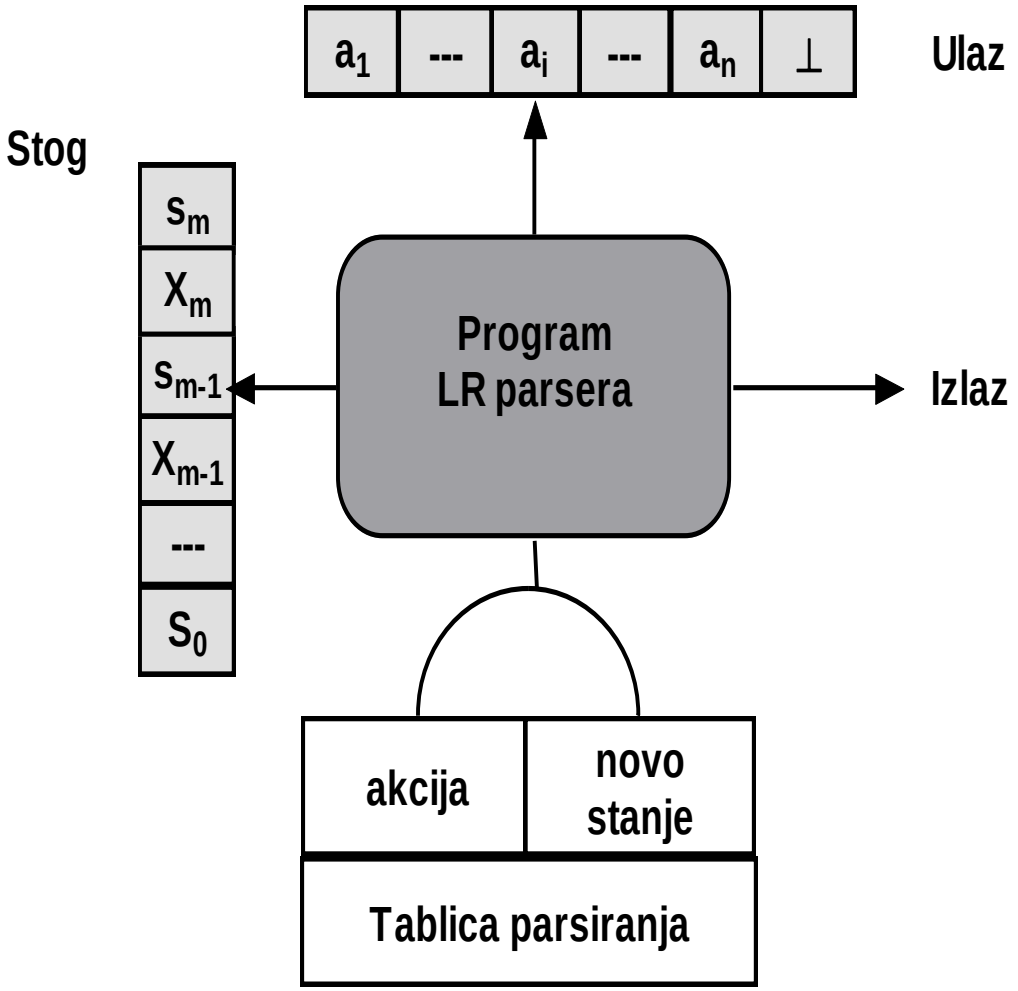
```
A( )
{   ...
    B( );
    ...
}
```

Parsiranje od dna prema vrhu

- gradnja započinje od listova, završnih znakova gramatike
- ako je međuniz niza w jednak desnoj strani produkcije onda se zamijeni lijevom
- na taj način gradi se stablo uključujući i korijen
- broj znakova se postepeno smanjuje prema vrhu zato produkcije nazivamo redukcijama
- metoda primjerena za generatore parsera
- LR(k) parsiranje –
  - L niz čitamo s lijeva na desno
  - R stablo gradimo s desna na lijevo
  - k : najviše k znakova treba pročitati da se primjeni redukcija; $k \geq 1$

Model LR parsera

- LR parser gradi se posebnim generatorom
- LR parseri različitih gramatika razlikuju se samo u **tablici parsiranja**
- u radu koristi potisni stog (LIFO)



Rad LR parsera I

- tablica parsiranja: tablica **Akcija** i tablica **Novo stanje**
- čita se **vrh stoga** s_m i **znak na ulazu** a_i $[s_m, a_i]$
- na temelju $[s_m, a_i]$ se odredi akcija iz tablice:
  - Pomak (odredi stanje),
  - Reduciraj $A \rightarrow \beta$ (primjeni produkciju gramatike),
  - Prihvati,
  - Odbaci
- **konfiguracija LR parsera** se dobiva sadržajem stoga i nepročitanim dijelom ulaznoga niza (X_1 su znakovi gramatike)
  - sadržaj stoga nepročitani dio ulaznoga niza

    $(s_0 X_1 s_1 X_2 . s_{2...} X_m s_m \quad a_i a_{i+1...} a_n \perp)$

Rad LR parsera II

- **Akcija [s_m, a_i]=Pomak s** promjeni konfiguraciju LR parsera u ($s_0 X_1 s_1 X_{2.} s_{2...} X_m$ **$s_m a_i s$,** $a_{i+1...} a_n \perp$)
- **Akcija [s_m, a_i]= Reduciraj** $A \rightarrow \beta$
  - s vrha stoga uzme se 2r znakova (ako je r broj znakova u β)
  - na vrhu stoga ostaje stanje s_{m-r}, i na stog se stavi A (nezavršni znak produkcije) a ulazni niz ostaje nepromijenjen
  - promjeni konfiguraciju LR parsera u

$$(s_0 X_1 s_1 X_{2.} s_{2...} X_{m-r} s_{m-r} \mathbf{A s}, a_i a_{i+1...} a_n \perp)$$

- **Akcija [s_m, a_i]= Prihvati**
  - prihvaća se niz w, zaustavlja parsiranje
- **Akcija [s_m, a_i]= Odbaci**
  - odbacuje se niz w, zaustavlja parsiranje

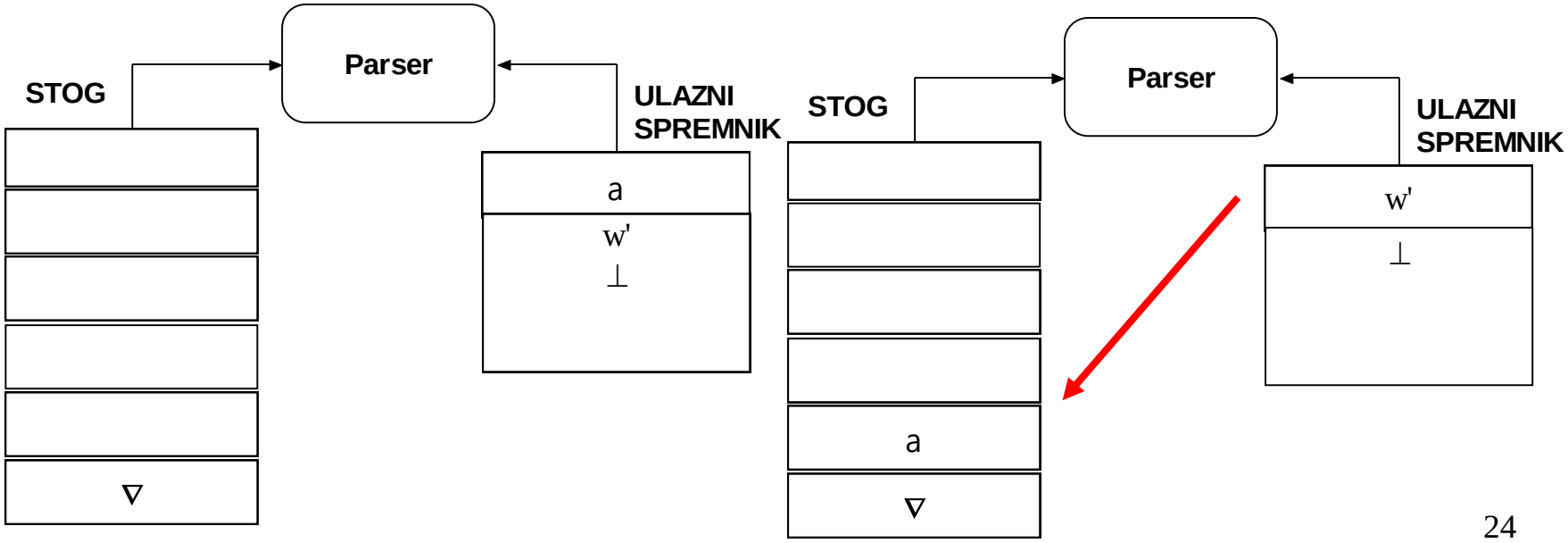
Rad bottom-up parsera

- koristi se potisni stog i ulazni spremnik
  - na jednom koraku se čita više znakova stoga (ne samo vrh)
  - na stogu su završni i nezavršni nizovi
  - u ulaznom spremniku je niz w i svi međunizovi koji se parsiraju
    - oznaka dna stoga: Δ
    - oznaka kraja niza: $\perp$
- akcije:
  - *Pomakni; Reduciraj; Prihvati; Odbaci*

Rad bottom-up parsera: *Pomakni*

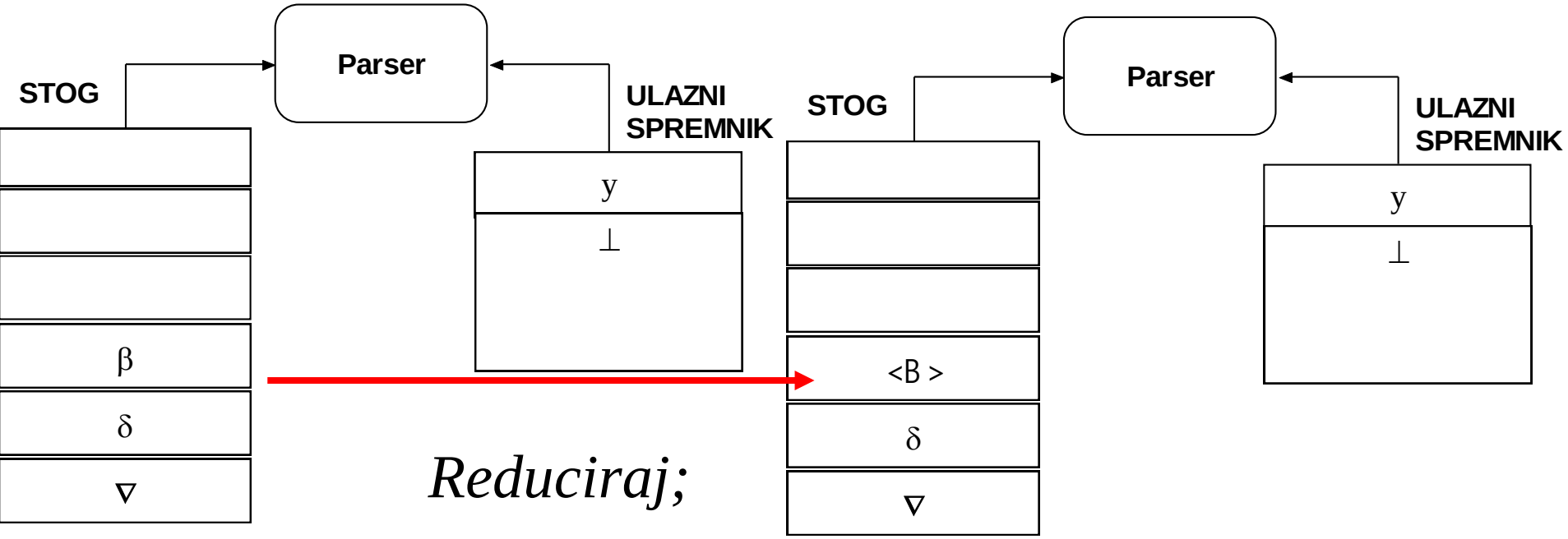
- na početku je *stog prazan*
- čita se prvi znak ulaznog niza

- *Pomakni*; prvi znak ulaznog niza se zapisuje na vrh stoga
- kazaljka na ulaznom nizu se postavlja za jedan znak desno
  - sljedeći znak



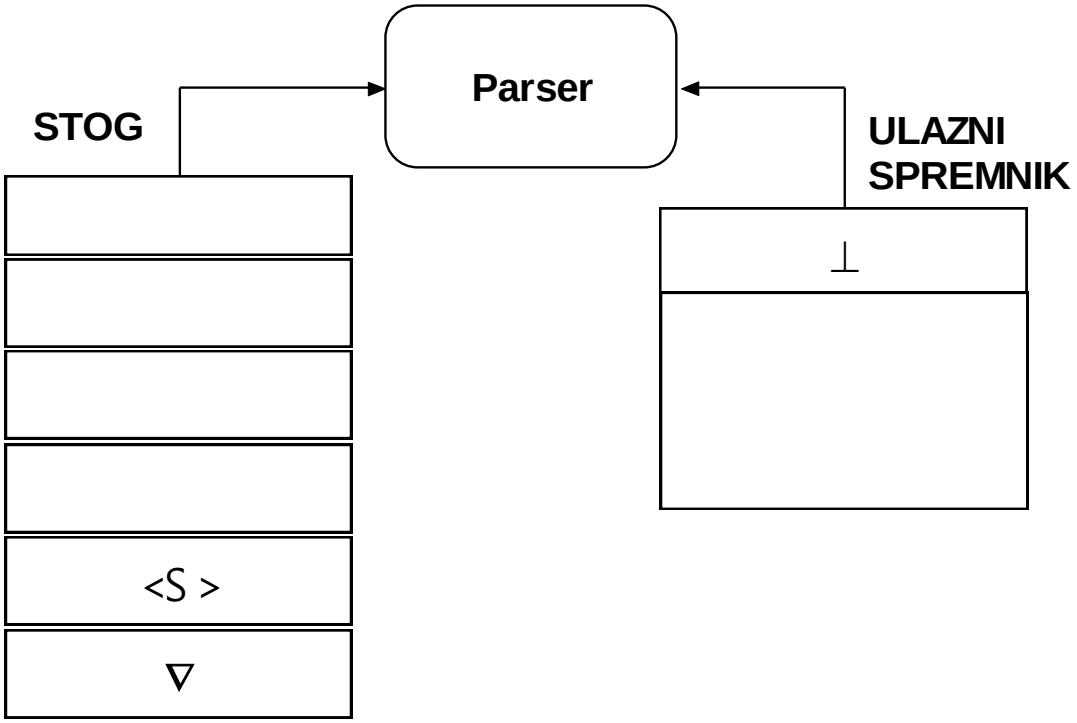
Rad bottom-up parsera: *Reduciraj;*

- na vrhu stoga je β i postoji produkcija <B>→β
- na vrhu stoga se zamijeni β s nezavršnim znakom <B>
- međuniz δβγ <= δ<B>γ



Rad bottom-up parsera: *Prihvati;*

- Pročitani svi znakovi ulaznog niza: do $\perp$
- na stogu je početni nezavršni znak <S> i oznaka kraja Δ



Primjer: rad parsera

gramatika: 1. **E→ E+T** 2. **E→ T** 3. **T→ T*F** 4. **T→ F** 5. **F→ (E)** 6. **F→var**
ulazni niz: **var+var*var**

Tablica parsera:

akcije **si:** stavi stanje si na stog **rj:** reduciraj produkcijom j **prihvati: odbaci:** sve prazne ćelije **početno stanje** 0

	Akcija						***NovoStanje***		
	var	**+**	*****	**(**	**)**	**⊥**	**E**	**T**	**F**
0	s5			s4			1	2	3
1		s6				prihvati			
2		r2	s7		r2	r2			
3		**r4**	**r4**		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Primjer: rad parsera II

Ulazni niz	Stog	Akcija
var+var*var⊥	Δ0	s5
+var*var⊥	Δ0var5	r6: redukcija F→var, skidam 2 znaka, stanje 0,F=3
+var*var⊥	Δ0F3	r4: redukcija T→ F, skidam 2 znaka, stanje 0,T=2
+var*var⊥	Δ0T2	r2: redukcija E→ T, skidam 2 znaka, stanje 0,E=1
+var*var⊥	Δ0E1	s6: stavi + i stanje 6
var*var⊥	Δ0E1+6	s5: stavi var i stanje 5
*var⊥	Δ0E1+6var5	r6: redukcija F→var, skidam 2 znaka, stanje 6,F=3
*var⊥	Δ0E1+6F3	r4: redukcija T→F, skidam 2 znaka, stanje 6,T=9
*var⊥	Δ0E1+6T9	s7: stavi * i stanje 7
var⊥	Δ0E1+6T9*7	s5: stavi var i stanje 5
⊥	Δ0E1+6T9*7var5	r6: redukcija F→var, skidam 2 znaka, stanje 7,F=10
⊥	Δ0E1+6T9*7F10	r3: redukcija T→ T*F, skidam 6 znakova, stanje 6,T=9
⊥	Δ0E1+6T9	r1: redukcija E→ E+T, skidam 6 znakova, stanje 0,E=1
⊥	Δ0E1	Prihvati

Algoritam LR parsera

- *ulaz:* niz w i tablica parsiranja
- *izlaz:* ili se w prihvaća ili odbaci
- *postupak:* na početku rada na stogu je stanje s_0, a na ulazu w⊥, parser izvodi program parsera sve dok se ne izvede akcija prihvati ili odbaci

Program LR parsera:

```
ProgramLRParsera()

{     postavi kazaljku na 1. znak u nizu w ⊥;

      dok(1)        //ponavljaj zauvijek
      {             //s je stanje na vrhu stoga
                    //kazaljka pokazuje na znak a u nizu w
```

(nastavak)

Program LR parsera

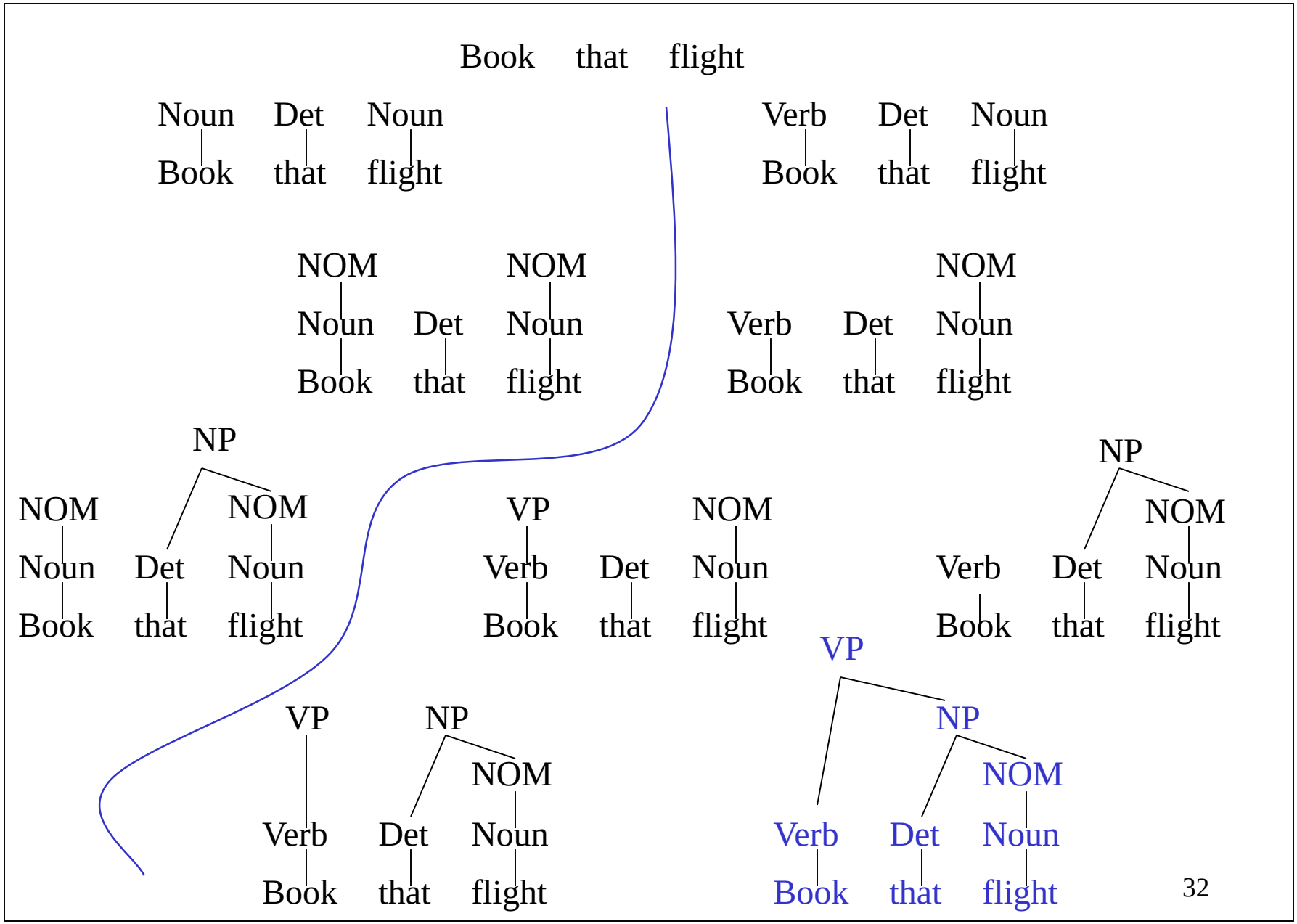
```
slučaj(akcija[s,a])
{     pomak s':
            stavi a na stog;
            stavi s' na stog;
            pomakni kazaljku na slijedeći znak iz w;
      reduciraj A→β:
            skini sa stoga 2*|β| znakova;
                  // s' stanje na vrhu nakon skidanja
            stavi A na stog;
            stavi NovoStanje[s',A] na stog;
            ispiši("primjenjena produkcija A→β");
      prihvati:
            ispiši("niz w se prihvaća");
            završi;
      odbaci:
            ispiši("niz w se ne prihvaća");
            završi;
}}}
```

Primjer izgradnje parsera za gramatiku

S → NP VP	Det → that | this |a
S → Aux NP VP	Noun → book | flight | meal | money
S → VP	Verb → book | include | prefer
NP → Det Nominal	Aux → does
Nominal → Noun	Proper-Noun → Houston | TWA
Nominal → Noun Nominal	Prep → from | to | on
NP → Proper-Noun	
VP → Verb	
VP → Verb NP	
Nominal → Nominal PP	

Book that flight.

Primjer parsiranja od dna prema vrhu



Parsiranje: Formalni II

- Parsiranje od vrha prema dnu
  - prvo u dubinu / prvo u širinu
  - S-gramatika / Q-gramatika
  - LL(1)-gramatika /rekurzivni spust
- Parsiranje od dna prema vrhu
  - Pomakni-pronađi / Pomakni-reduciraj
  - Prednosti operatora
  - LR parsiranje
  - Earlyev algoritam
  - CYK (Cocke-Younger-Kasami) algoritam

LITERATURA

- S. Srbljić: *Jezični procesori I + II*, Element, Zagreb, 2002.
- J.E. Hopcroft, J.D. Ullman: *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, USA,1979.
- A.V. Aho, R. Sethi, J.D. Ullman: *Compilers Principles, Techniques and Tools*, 1987.
- Michael Sipser, *Introduction to the Theory of Computation, second edition*, Course Technology, MIT, 2005.