Uvod u programske prevodioce

školska 2022/2023

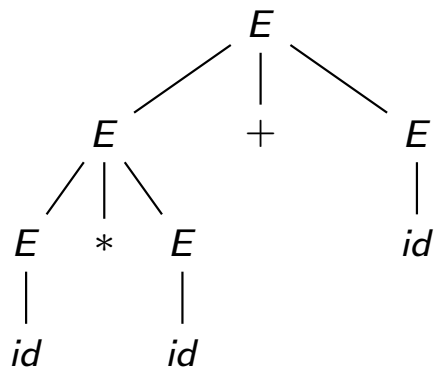
Dvosmislenost gramatike

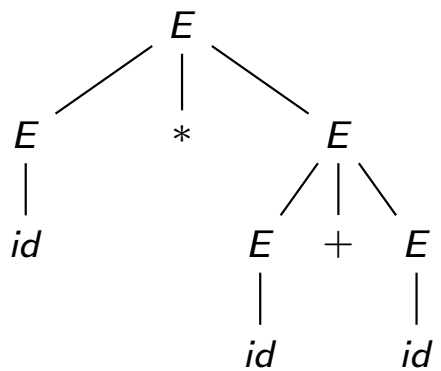
Primer.

Neka je data gramatika $G = (\{E\}, \{id, +, *\}, E, \mathcal{P})$ gde je

$$\mathcal{P}: \quad E \to E \ + \ E \mid E \ * \ E \mid id$$

Drvo izvođenja reči $id * id + id$:





Nije poželjno

Dvosmislenost gramatike

Primer.

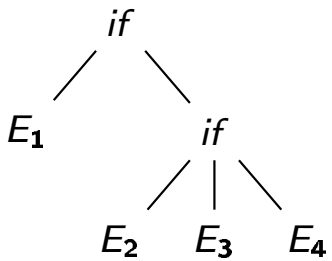
Gramatika kojok se opisuje korektna IF naredba u nekom jeziku

$$E \to \mathit{if}\ E\ \mathit{then}\ E \mid \mathit{if}\ E\ \mathit{then}\ E\ \mathit{else}\ E \mid \ldots$$

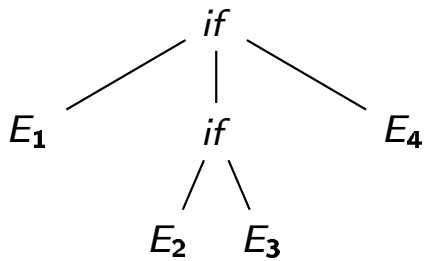
Drvo izvođenja reči

$$\mathit{if}\ E_1\ \mathit{then}\ \mathit{if}\ E_2\ \mathit{then}\ E_3\ \mathit{else}\ E_4$$

Očekivano drvo izvođenja



Moguće drvo izvođenja



else treba da se odnosi na najbliže *if*

Dvosmislenost gramatike

Primer.

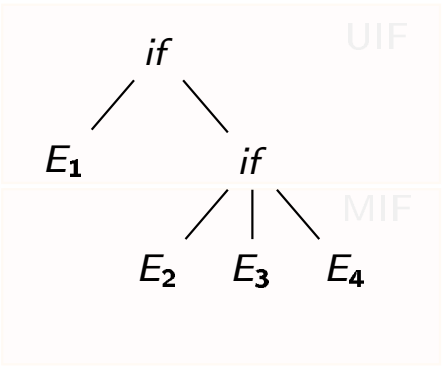
Gramatika kojok se opisuje korektna IF naredba u nekom jeziku

$$E \to \mathit{if}\ E\ \mathit{then}\ E \mid \mathit{if}\ E\ \mathit{then}\ E\ \mathit{else}\ E \mid \ldots$$

Drvo izvođenja reči

$$\mathit{if}\ E_1\ \mathit{then}\ \mathit{if}\ E_2\ \mathit{then}\ E_3\ \mathit{else}\ E_4$$

Očekivano drvo izvođenja



Drugačije definisana gramatika:

$E \to MIF \mid UIF$
$MIF \to \mathit{if}\ E\ \mathit{then}\ MIF\ \mathit{else}\ MIF \mid \ldots$
$UIF \to \mathit{if}\ E\ \mathit{then}\ E \mid \mathit{if}\ E\ \mathit{then}\ MIF\ \mathit{else}\ UIF$

Dvosmislenost gramatike

Primer.

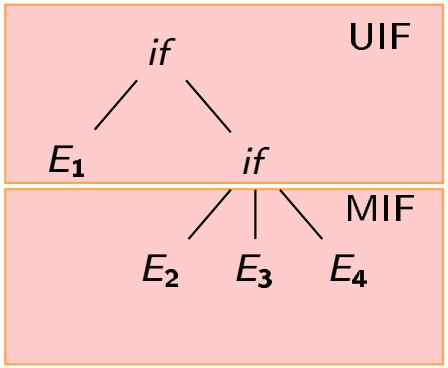
Gramatika kojok se opisuje korektna IF naredba u nekom jeziku

$$E \to \mathit{if}\ E\ \mathit{then}\ E \mid \mathit{if}\ E\ \mathit{then}\ E\ \mathit{else}\ E \mid \ldots$$

Drvo izvođenja reči

$$\mathit{if}\ E_1\ \mathit{then}\ \mathit{if}\ E_2\ \mathit{then}\ E_3\ \mathit{else}\ E_4$$

Očekivano drvo izvođenja



Drugačije definisana gramatika:

$$E \to MIF \mid UIF$$
$$MIF \to \mathit{if}\ E\ \mathit{then}\ MIF\ \mathit{else}\ MIF \mid \ldots$$
$$UIF \to \mathit{if}\ E\ \mathit{then}\ E \mid \mathit{if}\ E\ \mathit{then}\ MIF\ \mathit{else}\ UIF$$

Dvosmislenost gramatike

- Ne postoji uopšteni pristup u rešavanju dvosmislenosti
- Dvosmislene gramatike nekada olakšavaju opisivanje jezika jer omogućavaju prirodnije definicije
- Poseban mehanizam za rešavanja dvosmislenosti
  - definisanje prioriteta i/ili asocijativnosti
  - %*left* +
  - Ne menja se gramatika, a povećava čitljivost, smanjuje kompleksnost, omogućava brže parsiranje

Familije KS jezika

Neka je $w \in \Sigma^*$ i $Prvi_k(w)$ prefiks dužine k reči w i za $|w| < k$ $Prvi_k(w) = w$.

Дефиниција

Kontekstno slobodna gramatika $G = (\Gamma, \Sigma, S, \mathcal{P})$ *je* $LL(k)$*-gramatika ako za* $u, v, v' \in \Sigma^*$ *i* $X \in \Gamma$ *važi da iz*

- $S \rightarrow_L^* uX\beta \rightarrow u\alpha\beta \rightarrow_L^* uv$
- $S \rightarrow_L^* uX\beta \rightarrow u\alpha'\beta \rightarrow_L^* uv'$

i $Prvi_k(v) = Prvi_k(v')$ *sledi* $\alpha = \alpha'$,
pri čemu $\rightarrow_L^*$ *predstavlja najlevlje izvodjenje.*

Prefiks dužine k odredjuje koje će se sledeće pravilo primeniti

Дефиниција

Ako je neki jezik generiasn $LL(k)$*-gramatikom, onda je on* $LL(k)$*-jezik.*

Familije KS jezika

Neka je $w \in \Sigma^*$ i $Prvi_k(w)$ prefiks dužine k reči w i za $|w| < k$ $Prvi_k(w) = w$.

Definicija

Kontekstno slobodna gramatika $G = (\Gamma, \Sigma, S, \mathcal{P})$ je LR(k)-gramatika ako za $u, u', v, v' \in \Sigma^*$, $X \in \Gamma$ i $\rho \in (\Sigma \cup \Gamma)^*\Gamma$ važi da iz

- $S \rightarrow_R^* \beta X u \rightarrow \beta\alpha u \rightarrow_R^* \rho v$
- $S \rightarrow_R^* \beta' X' u' \rightarrow \beta'\alpha' u' \rightarrow_R^* \rho v'$

i $Prvi_k(v) = Prvi_k(v')$ sledi $X = X'$ i $\alpha = \alpha'$,
pri čemu $\rightarrow_R^*$ predstavlja najdešnje izvodjenje.

Definicija

Ako je neki jezik generiasn LR(k)-gramatikom, onda je on LR(k)-jezik.

Sintaksni analizatori

- Univerzalni - dopušta analizu ma koje KS gramatike.
  Polazeći od početnog simbola ispituju se sva moguća izvodjenja.
  Složenost $O(n^3)$ - izuzetno spor
- Analizator naniže (Top-Down) - konstruiše abstraktno sintaksno stablo polazeći od korena do listova
  Odgovara infisknom obilasku stabla
  Predstavnik *LL*-gramatike
- Analizator naviše (Bottom-Up) - konstruiše abstraktno sintaksno stablo polazeći od listova do korena
  Može se analizirati šira klas nego analizatorom naniže, ali su složeniji
  Predstavnik *LR*-gramatike

Dvosmislenost kod CFG

Primer.

Neka je data gramatika $G_1 = (\{S, E\}, \{v, i, +\}, S, \mathcal{P})$ gde je

$$\mathcal{P} : S \to E, \quad E \to E + E | v | i.$$

Izvođenje reči $v + i + i$

Top-down pristup

Reč	Pimena pravila
S	$S \to E$
E	$E \to E + E$
$E + E$	$E \to v$
$v + E$	$E \to E + E$
$v + E + E$	$E \to i$
$v + i + E$	$E \to i$
$v + i + i$	

Bottom-up pristup

Reč	Primena pravila
$v + i + i$	$E \to i$
$v + i + E$	$E \to i$
$v + E + E$	$E \to E + E$
$v + E$	$E \to v$
$E + E$	$E \to E + E$
E	$S \to E$
S	

Dvosmislenost kod CFG

Primer.

Neka je data gramatika $G_1 = (\{S, E\}, \{v, i, +\}, S, \mathcal{P})$ gde je

$$\mathcal{P} : S \to E, \quad E \to E + E | v | i.$$

Izvođenje reči $v + i + i$

Top-down pristup

Reč	Pimena pravila
S	$S \to E$
E	$E \to E + E$
$E + E$	$E \to v$
$v + E$	$E \to E + E$
$v + E + E$	$E \to i$
$v + i + E$	$E \to i$
$v + i + i$	

Bottom-up pristup

Reč	Primena pravila
$v + i + i$	$E \to i$
$v + i + E$	$E \to i$
$v + E + E$	$E \to E + E$
$v + E$	$E \to v$
$E + E$	$E \to E + E$
E	$S \to E$
S	

Dvosmislenost kod CFG

Primer.

Neka je data gramatika $G_1 = (\{S, E\}, \{v, i, +\}, S, \mathcal{P})$ gde je

$$\mathcal{P} : S \to E, \quad E \to E + E | v | i.$$

Izvođenje reči $v + i + i$

Top-down pristup

Reč	Pimena pravila
S	$S \to E$
E	$E \to E + E$
$E + E$	$E \to v$
$v + E$	$E \to E + E$
$v + E + E$	$E \to i$
$v + i + E$	$E \to i$
$v + i + i$	

Bottom-up pristup

Reč	Primena pravila
$v + i + i$	$E \to i$
$v + i + E$	$E \to i$
$v + E + E$	$E \to E + E$
$v + E$	$E \to v$
$E + E$	$E \to E + E$
E	$S \to E$
S	

Dvosmislenost kod CFG

Primer.

Neka je data gramatika $G_1 = (\{S, E\}, \{v, i, +\}, S, \mathcal{P})$ gde je

$$\mathcal{P} : S \to E, \quad E \to E + E | v | i.$$

Izvođenje reči $v + i + i$

Top-down pristup

Reč	Pimena pravila
S	$S \to E$
E	$E \to E + E$
$E + E$	$E \to v$
$v + E$	$E \to E + E$
$v + E + E$	$E \to i$
$v + i + E$	$E \to i$
$v + i + i$	

Bottom-up pristup

Reč	Primena pravila
$v + i + i$	$E \to i$
$v + i + E$	$E \to i$
$v + E + E$	$E \to E + E$
$v + E$	$E \to v$
$E + E$	$E \to E + E$
E	$S \to E$
S	

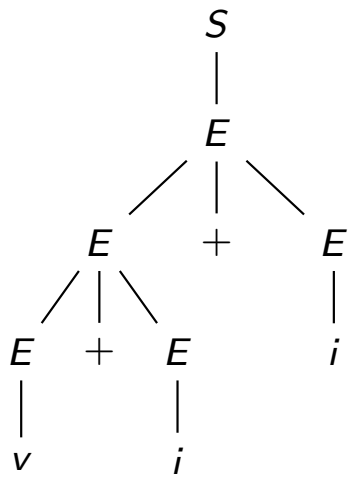
Dvosmislenost kod CFG

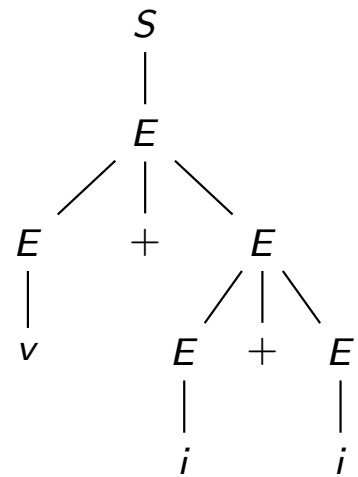
Primer.

Neka je data gramatika $G_1 = (\{S, E\}, \{v, i, +\}, S, \mathcal{P})$ gde je

$$\mathcal{P} : S \to E, \quad E \to E + E | v | i.$$

Izvođenje reči $v + i + i$





Zašto je to bitno?
Šta će biti rezultat `hello + 5 + 5`? `hello55` ili `hello10`?

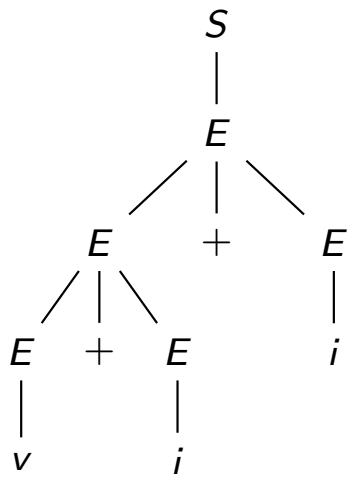
Dvosmislenost kod CFG

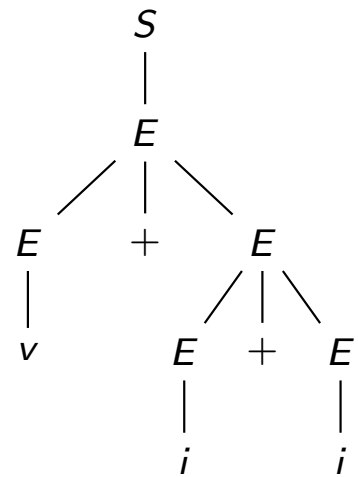
Primer.

Neka je data gramatika $G_1 = (\{S, E\}, \{v, i, +\}, S, \mathcal{P})$ gde je

$$\mathcal{P} : S \to E, \quad E \to E + E | v | i.$$

Izvođenje reči $v + i + i$





Zašto je to bitno?
Šta će biti rezultat `hello + 5 + 5`? `hello55` ili `hello10`?

Dvosmislenost kod CFG

Kada se definišu binarni operatori uobičajeno je se jedan argument definiše kao atomski term.

Primer.

Drugačija definisanje gramatike $G_2 = (\{S, E, T\}, \{v, i, +\}, S, \mathcal{P})$ gde je

$$\mathcal{P} : S \to E, \quad E \to E + T | T, \quad T \to v | i.$$

Ovime gramatika više nije dvosmislena, jer su ostala samo leva izvođenja, a jezik koji se generiše je ostao isti.

Dvosmislenost kod CFG

Kada se definišu binarni operatori uobičajeno je se jedan argument definiše kao atomski term.

Primer.

Drugačija definisanje gramatike $G_2 = (\{S, E, T\}, \{v, i, +\}, S, \mathcal{P})$ gde je

$$\mathcal{P} : S \to E, \quad E \to E + T | T, \quad T \to v | i.$$

Ovime gramatika više nije dvosmislena, jer su ostala samo leva izvođenja, a jezik koji se generiše je ostao isti.

Dvosmislenost kod CFG

Kada se definišu binarni operatori uobičajeno je se jedan argument definiše kao atomski term.

Primer.

Drugačija definisanje gramatike $G_2 = (\{S, E, T\}, \{v, i, +\}, S, \mathcal{P})$ gde je

$$\mathcal{P} : S \to E, \quad E \to E + T | T, \quad T \to v | i.$$

Ovime gramatika više nije dvosmislena, jer su ostala samo leva izvođenja, a jezik koji se generiše je ostao isti.

Strategije parsiranja

Top-down - Započinje početnom promenljivom i traži se niz transformacija kojima se izvodi ulazni niz tokena (program se izvodi iz početnog simbola gramatike)

Bottom-up - Započinje ulaznim stringom (programom) i traži se niz pravila kojima se izvodi dati ulaz. Kada se poklapanje dogodi, ceo niz ulaznih tokena se menja korenom poklopljenog pravila. Ako se ulazni string svede na početnu promenljivu, parsiranje je uspešno.

Strategije parsiranja

Problemi:

- Višeznačne gramatike - koje pravilo upotrebiti u kom koraku
- "Ručno" parsiranje ima uvid u ceo ulazni string. Ovakva pretpostavka nije realna u praksi jer ulazni programi mogu biti jako dugački
  - Obično je moguće videti samo jedan naredni token, ne ceo string
  - To dovodi do toga da više pravila može biti primenjeno/zamenjeno u jedno koraku što otežava pravljenje algoritma parsiranja

Strategije parsiranja

Rešenje:

- Backtracing - pravilo se bira na osnovu trenutnih informacija. Ukoliko odluka ne dovede do pozitivnog zaključka, primenjuje se backtracking kako bi se ispitala sledeća mogućnost
  - Jednostvno se primenjuje na gramatike u raznim formama
  - Jako spor pristup kod "velikih"gramatika (npr. programskih jezika)
- Uvođenjem ograničenja dobijaju se efikasnije metode
  - Top-down
  - Bottom-up

Klase pravila

- Direktno rekurzivna

  Simbol sa leve strane se pojavljuje i sa desne strane:

$$\begin{aligned} A &\to A\alpha \\ C &\to C\gamma C \end{aligned}$$

- Levo rekurzivna
  - Simbol sa leve strane pravila se pojavljuje **skroz levo** na desno strani pravila

$$A \to A\alpha$$

- Desno rekurzivna
  - Simbol sa leve strane pravila se pojavljuje **skroz desno** na desno strani pravila

$$B \to \beta B$$

LL gramatike

- LL(1) gramatike su podskup kontekstno slobodnih gramatika koje se parsiraju jednostavnim algoritmom.
  - Left-to-right - čitanje tokena
  - Leftmost derivation - prvo se menja najlevlji simbol tokom pravila
  - (1) - jedan preduvidni simbol
- Da bi se obezbedilo da je gramatika LL(1) potrebno je uraditi sledeće
  - Ukloniti dvosmislenost
  - Eliminisati sve leve rekurzije
  - Eliminisati sve zajedničke prefikse
- Nakon ovih koraka, pokazuje se da je gramatika LL(1) generisanjem First and Follow skupova
- Koristeći prethodno definisane skupove kreira se tabela parsiranja i ukoliko ona ne sadrži konflikte gramatika je LL(1)

LL gramatike

- LL(1) gramatike su podskup kontekstno slobodnih gramatika koje se parsiraju jednostavnim algoritmom.
  - Left-to-right - čitanje tokena
  - Leftmost derivation - prvo se menja najlevlji simbol tokom pravila
  - (1) - jedan preduvidni simbol
- Da bi se obezbedilo da je gramatika LL(1) potrebno je uraditi sledeće
  - Ukloniti dvosmislenost
  - Eliminisati sve leve rekurzije
  - Eliminisati sve zajedničke prefikse
- Nakon ovih koraka, pokazuje se da je gramatika LL(1) generisanjem First and Follow skupova
- Koristeći prethodno definisane skupove kreira se tabela parsiranja i ukoliko ona ne sadrži konflikte gramatika je LL(1)

LL gramatike

- LL(1) gramatike su podskup kontekstno slobodnih gramatika koje se parsiraju jednostavnim algoritmom.
  - Left-to-right - čitanje tokena
  - Leftmost derivation - prvo se menja najlevlji simbol tokom pravila
  - (1) - jedan preduvidni simbol
- Da bi se obezbedilo da je gramatika LL(1) potrebno je uraditi sledeće
  - Ukloniti dvosmislenost
  - Eliminisati sve leve rekurzije
  - Eliminisati sve zajedničke prefikse
- Nakon ovih koraka, pokazuje se da je gramatika LL(1) generisanjem First and Follow skupova
- Koristeći prethodno definisane skupove kreira se tabela parsiranja i ukoliko ona ne sadrži konflikte gramatika je LL(1)

LL gramatike

- LL(1) gramatike su podskup kontekstno slobodnih gramatika koje se parsiraju jednostavnim algoritmom.
  - Left-to-right - čitanje tokena
  - Leftmost derivation - prvo se menja najlevlji simbol tokom pravila
  - (1) - jedan preduvidni simbol
- Da bi se obezbedilo da je gramatika LL(1) potrebno je uraditi sledeće
  - Ukloniti dvosmislenost
  - Eliminisati sve leve rekurzije
  - Eliminisati sve zajedničke prefikse
- Nakon ovih koraka, pokazuje se da je gramatika LL(1) generisanjem First and Follow skupova
- Koristeći prethodno definisane skupove kreira se tabela parsiranja i ukoliko ona ne sadrži konflikte gramatika je LL(1)

Eliminisanje leve rekurzije

- U LL(1) gramatici ne mogu se naći pravila oblika $A \to A\alpha$ ili ma koje pravilo $A \to B\beta$ takvo da $B \to^* A\gamma$.
- Pravilo $E \to E + T$ je levo-rekurzivno. Ako se umesto njega uvede pravilo $E \to T + E$ elimiše se leva rekurzija, ali definisani operator postaje desno asocijativan i neće generisati isti jezik.
- Pravila izbođenja oblika

$$A \to A\alpha_1 | A\alpha_2 | \ldots | \beta_1 | \beta_2 | \ldots$$

menjaju se pravilima

$$A \to \beta_1 A' | \beta_2 A' | \ldots$$

$$A' \to \alpha_1 A' | \alpha_2 A' | \ldots | \varepsilon$$

Eliminisanje leve rekurzije

- U LL(1) gramatici ne mogu se naći pravila oblika $A \to A\alpha$ ili ma koje pravilo $A \to B\beta$ takvo da $B \to^* A\gamma$.
- Pravilo $E \to E + T$ je levo-rekurzivno. Ako se umesto njega uvede pravilo $E \to T + E$ elimiše se leva rekurzija, ali definisani operator postaje desno asocijativan i neće generisati isti jezik.
- Pravila izbođenja oblika

$$A \to A\alpha_1 | A\alpha_2 | \ldots | \beta_1 | \beta_2 | \ldots$$

menjaju se pravilima

$$A \to \beta_1 A' | \beta_2 A' | \ldots$$

$$A' \to \alpha_1 A' | \alpha_2 A' | \ldots | \varepsilon$$

Eliminisanje leve rekurzije

- U LL(1) gramatici ne mogu se naći pravila oblika $A \to A\alpha$ ili ma koje pravilo $A \to B\beta$ takvo da $B \to^* A\gamma$.
- Pravilo $E \to E + T$ je levo-rekurzivno. Ako se umesto njega uvede pravilo $E \to T + E$ elimiše se leva rekurzija, ali definisani operator postaje desno asocijativan i neće generisati isti jezik.
- Pravila izbođenja oblika

$$A \to A\alpha_1 | A\alpha_2 | \ldots | \beta_1 | \beta_2 | \ldots$$

menjaju se pravilima

$$A \to \beta_1 A' | \beta_2 A' | \ldots$$

$$A' \to \alpha_1 A' | \alpha_2 A' | \ldots | \varepsilon$$

Eliminisanje leve rekurzije

Primer.

$G_3 = (\{S, E, E', T\}, \{v, i, +\}, S, \mathcal{P})$ gde je

$$\mathcal{P} : S \to E, \quad E \to TE', \quad E' \to +TE'|\varepsilon, \quad T \to v|i.$$

Izvođenje reči $v + i + i$:

$$S \to E \to TE' \to vE' \to v + TE' \to v + iE'$$
$$\to v + i + TE' \to v + i + iE' \to v + i + i$$

Eliminisanje leve rekurzije

Primer.

$G_3 = (\{S, E, E', T\}, \{v, i, +\}, S, \mathcal{P})$ gde je

$$\mathcal{P} : S \to E, \quad E \to TE', \quad E' \to +TE'|\varepsilon, \quad T \to v|i.$$

Izvođenje reči $v + i + i$:

$$\begin{aligned} S &\to E \to TE' \to vE' \to v + TE' \to v + iE' \\ &\to v + i + TE' \to v + i + iE' \to v + i + i \end{aligned}$$

Eliminisanje zajedničkog prefiksa

- Ukoliko gramatika ima više pravila kod kojih je ista glava i isti prefiks tela pravila, uvodi se pravilo kojim se ovi prefiksi eliminišu
- Pravila oblika

$$A \to \alpha\beta_1 | \alpha\beta_2 | \ldots$$

se zamenjuju pravilima

$$A \to \alpha A'$$

$$A' \to \beta_1 | \beta_2 | \ldots$$

Eliminisanje zajedničkog prefiksa

- Ukoliko gramatika ima više pravila kod kojih je ista glava i isti prefiks tela pravila, uvodi se pravilo kojim se ovi prefiksi eliminišu
- Pravila oblika

$$A \to \alpha\beta_1|\alpha\beta_2|\ldots$$

se zamenjuju pravilima

$$A \to \alpha A'$$

$$A' \to \beta_1|\beta_2|\ldots$$

Eliminisanje zajedničkog prefiksa

Primer.

Ako gramatikom definišemo identifikator, referenciranje na element niza ili poziv funkcije, gramatika će sadržati pravila
$\mathcal{P} : S \to E, \quad E \to id \mid id[E] \mid id(E)$

Navedena pravila menjamo na sledeći način:
$\mathcal{P} : S \to E, \quad E \to id\ E', \quad E' \to [E] \mid (E) \mid \varepsilon$

First and Follow skupovi

- First(α) sadrži skup terminala (uključujući ε) koji se potencijalno mogu pojaviti na početku neke reči izvedene iz α
- Follow(A) sadrži set terminala (uključujući \$) koji se mogu pojaviti nakon nekog izvođenja iz neterminala A

Određivanje FIRST skupova

Computing First Sets for a Grammar G

FIRST(α) is the set of terminals that begin all strings given by α, including ϵ if $\alpha \Rightarrow \epsilon$.

For Terminals:
For each terminal $a \in \Sigma$: FIRST(a) = $\{a\}$

For Non-Terminals:
Repeat:
For each rule $X \rightarrow Y_1 Y_2 ... Y_k$ in a grammar G:
Add a to FIRST(X)
if a is in FIRST(Y_1)
or a is in FIRST(Y_n) and $Y_1 ... Y_{n-1} \Rightarrow \epsilon$
If $Y_1 ... Y_k \Rightarrow \epsilon$ then add ϵ to FIRST(X).
until no more changes occur.

For a Sentential Form α:
For each symbol $Y_1 Y_2 ... Y_k$ in α:
Add a to FIRST(α)
if a is in FIRST(Y_1)
or a is in FIRST(Y_n) and $Y_1 ... Y_{n-1} \Rightarrow \epsilon$
If $Y_1 ... Y_k \Rightarrow \epsilon$ then add ϵ to FIRST(α).

Određivanje FOLLOW skupova

Computing Follow Sets for a Grammar G

FOLLOW(A) is the set of terminals that can come after non-terminal A, including \$ if A occurs at the end of the input.

FOLLOW(S) = {\$} where S is the start symbol.

Repeat:
    If A $\rightarrow \alpha B \beta$ then:
        add FIRST(β) (excepting ϵ) to FOLLOW(B).
    If A $\rightarrow \alpha B$ or FIRST(β) contains ϵ then:
        add FOLLOW(A) to FOLLOW(B).
until no more changes occur.

First and Follow skupovi

Primer.

Data je gramatika $G_4 = (\{S, E, E', T, T', F\}, S, \{+, *, i, (,)\}, \mathcal{P})$

1. $S \to E$
2. $E \to TE'$
3. $E' \to +TE'$
4. $E' \to \varepsilon$
5. $T \to FT'$
6. $T' \to *FT'$
7. $T' \to \varepsilon$
8. $F \to (E)$
9. $F \to i$

	S	E	E'	T	T'	F
First	$(, i$	$(, i$	$+, \varepsilon$	$(, i$	$*, \varepsilon$	$(, i$
Follow	$\$$	$), \$$	$), \$$	$+,), \$$	$+,), \$$	$+, *,), \$$

First and Follow skupovi

Primer.

Data je gramatika $G_4 = (\{S, E, E', T, T', F\}, S, \{+, *, i, (,)\}, \mathcal{P})$

1. $S \to E$
2. $E \to TE'$
3. $E' \to +TE'$
4. $E' \to \varepsilon$
5. $T \to FT'$
6. $T' \to *FT'$
7. $T' \to \varepsilon$
8. $F \to (E)$
9. $F \to i$

	S	E	E'	T	T'	F
First	$(, i$	$(, i$	$+, \varepsilon$	$(, i$	$*, \varepsilon$	$(, i$
Follow	$\$$	$), \$$	$), \$$	$+,), \$$	$+,), \$$	$+, *,), \$$

Parsiranje ne osnovu tabele

LL(1) tabela za parsiranje služi da se odredi koje pravilo se primenjuje za ma koju kombinaciju neterminala na steku i sledeći terminal na ulazu

Parsiranje ne osnovu tabele

LL(1) tabela za parsiranje služi da se odredi koje pravilo se primenjuje za ma koju kombinaciju neterminala na steku i sledeći terminal na ulazu

LL(1) Parse Table Construction.
Given a grammar G and alphabet Σ, create a parse table $T[A, a]$ that selects a rule for each combination of non-terminal $A \in G$ and terminal $a \in \Sigma$.

For each rule $A \to \alpha$ in G:
    For each terminal a (excepting ϵ) in FIRST(α):
        Add $A \to \alpha$ to $T[A, a]$.
    If ϵ is in FIRST(α):
        For each terminal b (including \$) in FOLLOW(A):
            Add A $\to \alpha$ to $T[A, b]$.

Kreiranje tabele

Primer.

Data je gramatika $G_4 = (\{S, E, E', T, T', F\}, S, \{+, *, i, (,)\}, \mathcal{P})$

1. $S \to E$
2. $E \to TE'$
3. $E' \to +TE'$
4. $E' \to \varepsilon$
5. $T \to FT'$
6. $T' \to *FT'$
7. $T' \to \varepsilon$
8. $F \to (E)$
9. $F \to i$

	S	E	E'	T	T'	F
First	$(, i$	$(, i$	$+, \varepsilon$	$(, i$	$*, \varepsilon$	$(, i$
Follow	\$	$), \$$	$), \$$	$+,), \$$	$+,), \$$	$+, *,), \$$

	i	$+$	$*$	$($	$)$	\$
S	1			1		
E	2			2		
E'		3			4	4
T	5			5		
T'		7	6		7	7
F	9			8		

Kreiranje tabele

Primer.

Data je gramatika $G_4 = (\{S, E, E', T, T', F\}, S, \{+, *, i, (,)\}, \mathcal{P})$

1. $S \to E$
2. $E \to TE'$
3. $E' \to +TE'$
4. $E' \to \varepsilon$
5. $T \to FT'$
6. $T' \to *FT'$
7. $T' \to \varepsilon$
8. $F \to (E)$
9. $F \to i$

	S	E	E'	T	T'	F
First	$(, i$	$(, i$	$+, \varepsilon$	$(, i$	$*, \varepsilon$	$(, i$
Follow	$\$$	$), \$$	$), \$$	$+,), \$$	$+,), \$$	$+, *,), \$$

	i	$+$	$*$	$($	$)$	$\$$
S	1			1		
E	2			2		
E'		3			4	4
T	5			5		
T'		7	6		7	7
F	9			8		

Algoritam parsiranja

LL(1) Table Parsing Algorithm.
Given a grammar G with start symbol P and parse table T,
parse a sequence of tokens and determine whether they satisfy G.

Create a stack S.
Push \$ and P onto S.
Let c be the first token on the input.

While S is not empty:
    Let X be the top element of the stack.
    If X matches c:
        Remove X from the stack.
        Advance c to the next token and repeat.
    If X is any other terminal, stop with an error.
    If $T[X, c]$ indicates rule $X \to \alpha$:
        Remove X from the stack.
        Push symbols α on to the stack and repeat.
    If $T[X, c]$ indicates an error state, stop with an error.

Kreiranje tabele

Primer.

Data je gramatika $G_4 = (\{S, E, E', T, T', F\}, S, \{+, *, i, (,)\}, \mathcal{P})$

1. $S \to E$
2. $E \to TE'$
3. $E' \to +TE'$
4. $E' \to \varepsilon$
5. $T \to FT'$
6. $T' \to *FT'$
7. $T' \to \varepsilon$
8. $F \to (E)$
9. $F \to i$

	i	$+$	$*$	$($	$)$	\$
S	1			1		
E	2			2		
E'		3			4	4
T	5			5		
T'		7	6		7	7
F	9			8		

Parsiranje za reč $i * i$

Stack	Input	Action
$S\$$	$i * i\$$	pr 1: $S \to E$
$E\$$	$i * i\$$	pr 2: $E \to TE'$
$TE'\$$	$i * i\$$	pr 5: $T \to FT'$
$FT'E'\$$	$i * i\$$	pr 9: $F \to i$
$iT'E'\$$	$i * i\$$	match i
$T'E'\$$	$*i\$$	pr 6: $T' \to *FT'$
$*FT'E'\$$	$*i\$$	match $*$
$FT'E'\$$	$i\$$	pr 9: $F \to i$
$iT'E'\$$	$i\$$	match i
$T'E'\$$	$\$$	pr 7: $T' \to \varepsilon$
$E'\$$	$\$$	pr 4: $E' \to \varepsilon$
$\$$	$\$$	match \$

Kreiranje tabele

Primer.

Data je gramatika $G_4 = (\{S, E, E', T, T', F\}, S, \{+, *, i, (,)\}, \mathcal{P})$

1. $S \to E$
2. $E \to TE'$
3. $E' \to +TE'$
4. $E' \to \varepsilon$
5. $T \to FT'$
6. $T' \to *FT'$
7. $T' \to \varepsilon$
8. $F \to (E)$
9. $F \to i$

	i	$+$	$*$	$($	$)$	\$
S	1			1		
E	2			2		
E'		3			4	4
T	5			5		
T'		7	6		7	7
F	9			8		

Parsiranje za reč $i * i$

Stack	Input	Action
$S\$$	$i * i\$$	pr 1: $S \to E$
$E\$$	$i * i\$$	pr 2: $E \to TE'$
$TE'\$$	$i * i\$$	pr 5: $T \to FT'$
$FT'E'\$$	$i * i\$$	pr 9: $F \to i$
$iT'E'\$$	$i * i\$$	match i
$T'E'\$$	$*i\$$	pr 6: $T' \to *FT'$
$*FT'E'\$$	$*i\$$	match $*$
$FT'E'\$$	$i\$$	pr 9: $F \to i$
$iT'E'\$$	$i\$$	match i
$T'E'\$$	$\$$	pr 7: $T' \to \varepsilon$
$E'\$$	$\$$	pr 4: $E' \to \varepsilon$
$\$$	$\$$	match \$

Kreiranje tabele

Primer.

Data je gramatika $G_4 = (\{S, E, E', T, T', F\}, S, \{+, *, i, (,)\}, \mathcal{P})$

1. $S \to E$
2. $E \to TE'$
3. $E' \to +TE'$
4. $E' \to \varepsilon$
5. $T \to FT'$
6. $T' \to *FT'$
7. $T' \to \varepsilon$
8. $F \to (E)$
9. $F \to i$

	i	$+$	$*$	$($	$)$	$\$$
S	1			1		
E	2			2		
E'		3			4	4
T	5			5		
T'		7	6		7	7
F	9			8		

Parsiranje za reč $i * i$

Stack	Input	Action
$S\$$	$i * i\$$	pr 1: $S \to E$
$E\$$	$i * i\$$	pr 2: $E \to TE'$
$TE'\$$	$i * i\$$	pr 5: $T \to FT'$
$FT'E'\$$	$i * i\$$	pr 9: $F \to i$
$iT'E'\$$	$i * i\$$	match i
$T'E'\$$	$*i\$$	pr 6: $T' \to *FT'$
$*FT'E'\$$	$*i\$$	match $*$
$FT'E'\$$	$i\$$	pr 9: $F \to i$
$iT'E'\$$	$i\$$	match i
$T'E'\$$	$\$$	pr 7: $T' \to \varepsilon$
$E'\$$	$\$$	pr 4: $E' \to \varepsilon$
$\$$	$\$$	match $\$$

LR gramatike

- LL(1) gramatike i top-down parsiranje su jednostavni za implementaciju, ali ne omogućavaju reprezentovanje svih struktura koje se mogu naći u mnogim programskim jezicima
- Za većinu programskih jezika neophodno je koristiti LR(1) gramatiku i odgovarajuće bottom-up parsiranje
- LR(1) je skup gramatika koje se mogu parsirati shift-reduce tehnikom razmatrajući sledeći token
- LR(1) je nadskup skupa LL(1) i podržava levu rekurziju i zajednički prefiks, čime se mnogu konstrukcije mogu predstaviti na prirodniji način

LR gramatike

- LL(1) gramatike i top-down parsiranje su jednostavni za implementaciju, ali ne omogućavaju reprezentovanje svih struktura koje se mogu naći u mnogim programskim jezicima
- Za većinu programskih jezika neophodno je koristiti LR(1) gramatiku i odgovarajuće bottom-up parsiranje
- LR(1) je skup gramatika koje se mogu parsirati shift-reduce tehnikom razmatrajući sledeći token
- LR(1) je nadskup skupa LL(1) i podržava levu rekurziju i zajednički prefiks, čime se mnogu konstrukcije mogu predstaviti na prirodniji način

LR gramatike

- LL(1) gramatike i top-down parsiranje su jednostavni za implementaciju, ali ne omogućavaju reprezentovanje svih struktura koje se mogu naći u mnogim programskim jezicima
- Za većinu programskih jezika neophodno je koristiti LR(1) gramatiku i odgovarajuće bottom-up parsiranje
- LR(1) je skup gramatika koje se mogu parsirati shift-reduce tehnikom razmatrajući sledeći token
- LR(1) je nadskup skupa LL(1) i podržava levu rekurziju i zajednički prefiks, čime se mnogu konstrukcije mogu predstaviti na prirodniji način

LR gramatike

- LL(1) gramatike i top-down parsiranje su jednostavni za implementaciju, ali ne omogućavaju reprezentovanje svih struktura koje se mogu naći u mnogim programskim jezicima
- Za većinu programskih jezika neophodno je koristiti LR(1) gramatiku i odgovarajuće bottom-up parsiranje
- LR(1) je skup gramatika koje se mogu parsirati shift-reduce tehnikom razmatrajući sledeći token
- LR(1) je nadskup skupa LL(1) i podržava levu rekurziju i zajednički prefiks, čime se mnogu konstrukcije mogu predstaviti na prirodniji način

Shift-Reduce parsiranje

- *Bottom-up* strategija koja započinje tokenom i potragom za pravilom koje se može primeniti u cilju smanjivanja ulaza na ne-terminalne simbole.
- Ako niz redukcija dovede do početne promenljive, parsiranje je uspešno
- Shift akcija prihvata jedan token sa ulaza i smešta ga na stek
- Reduce akcija primenjuje pravilo oblika $A \rightarrow \alpha$ iz gramatike, menjajući na steku reč α ne-terminalnim simbolom A

Shift-Reduce parsiranje

- *Bottom-up* strategija koja započinje tokenom i potragom za pravilom koje se može primeniti u cilju smanjivanja ulaza na ne-terminalne simbole.
- Ako niz redukcija dovede do početne promenljive, parsiranje je uspešno
- Shift akcija prihvata jedan token sa ulaza i smešta ga na stek
- Reduce akcija primenjuje pravilo oblika $A \rightarrow \alpha$ iz gramatike, menjajući na steku reč α ne-terminalnim simbolom A

Shift-Reduce parsiranje

- *Bottom-up* strategija koja započinje tokenom i potragom za pravilom koje se može primeniti u cilju smanjivanja ulaza na ne-terminalne simbole.
- Ako niz redukcija dovede do početne promenljive, parsiranje je uspešno
- Shift akcija prihvata jedan token sa ulaza i smešta ga na stek
- Reduce akcija primenjuje pravilo oblika $A \rightarrow \alpha$ iz gramatike, menjajući na steku reč α ne-terminalnim simbolom A

Shift-Reduce parsiranje

- *Bottom-up* strategija koja započinje tokenom i potragom za pravilom koje se može primeniti u cilju smanjivanja ulaza na ne-terminalne simbole.
- Ako niz redukcija dovede do početne promenljive, parsiranje je uspešno
- Shift akcija prihvata jedan token sa ulaza i smešta ga na stek
- Reduce akcija primenjuje pravilo oblika $A \rightarrow \alpha$ iz gramatike, menjajući na steku reč α ne-terminalnim simbolom A

Shift-Reduce parsiranje

- Ideja - podeliti ulaza na dva dela
  - jedan koji je donekle sveden na promenljive (Stack)
  - drugi ko sadrži samo terminale (Input)
- U svakom koraku se donosi odluka
  - Da li se terminal prebacuje na stek - **shift**
  - Da li se stek (ili njegov deo) redukuje promenljivom - **reduce**
- Višeznačne gramatike mogu da naprave konflikte tipa
  - reduce/reduce - više od jednog izbora za redukciju
  - shift/reduce - da li primeniti shift ili reduce operaciju

Shift-Reduce parsiranje

Primer.

Data je gramatika $G_5 = (\{S, E, T\}, \{id, +, (,)\}, S, \mathcal{P})$ gde je

$\mathcal{P}$:
1. $S \to E$
2. $E \to E + T$
3. $E \to T$
4. $T \to id(E)$
5. $T \to id$

Parsiranje za $id(id + id)$

Stack	Input	Action
	$id(id + id)\$$	shift
id	$(id + id)\$$	shift
$id($	$id + id)\$$	shift
$id(id$	$+id)\$$	reduce $T \to id$
$id(T$	$+id)\$$	reduce $E \to T$
$id(E$	$+id)\$$	shift
$id(E+$	$id)\$$	shift
$id(E + id$	$)\$$	reduce $T \to id$
$id(E + T$	$)\$$	reduce $E \to E + T$
$id(E$	$)\$$	shift
$id(E)$	$\$$	reduce $T \to id(E)$
T	$\$$	reduce $E \to T$
E	$\$$	reduce $S \to E$
S	$\$$	accept

Ulaz je prihvaćen, ali ne postoji objašnjenje za redosled operacija
Npr. zašto je u drugom koraku shift, a ne reduce id na T?

Shift-Reduce parsiranje

Primer.

Data je gramatika $G_5 = (\{S, E, T\}, \{id, +, (,)\}, S, \mathcal{P})$ gde je

$\mathcal{P}$:
1. $S \to E$
2. $E \to E + T$
3. $E \to T$
4. $T \to id(E)$
5. $T \to id$

Parsiranje za $id(id + id)$

Stack	Input	Action
	$id(id + id)\$$	shift
id	$(id + id)\$$	shift
$id($	$id + id)\$$	shift
$id(id$	$+id)\$$	reduce $T \to id$
$id(T$	$+id)\$$	reduce $E \to T$
$id(E$	$+id)\$$	shift
$id(E+$	$id)\$$	shift
$id(E + id$	$)\$$	reduce $T \to id$
$id(E + T$	$)\$$	reduce $E \to E + T$
$id(E$	$)\$$	shift
$id(E)$	$\$$	reduce $T \to id(E)$
T	$\$$	reduce $E \to T$
E	$\$$	reduce $S \to E$
S	$\$$	accept

Ulaz je prihvaćen, ali ne postoji objašnjenje za redosled operacija
Npr. zašto je u drugom koraku shift, a ne reduce id na T?

Shift-Reduce parsiranje

Primer.

Data je gramatika $G_5 = (\{S, E, T\}, \{id, +, (,)\}, S, \mathcal{P})$ gde je

$\mathcal{P}$:
1. $S \to E$
2. $E \to E + T$
3. $E \to T$
4. $T \to id(E)$
5. $T \to id$

Ulaz je prihvaćen, ali ne postoji objašnjenje za redosled operacija
Npr. zašto je u drugom koraku shift, a ne reduce *id* na T?

Parsiranje za $id(id + id)$

Stack	Input	Action
	$id(id + id)\$$	shift
id	$(id + id)\$$	shift
$id($	$id + id)\$$	shift
$id(id$	$+id)\$$	reduce $T \to id$
$id(T$	$+id)\$$	reduce $E \to T$
$id(E$	$+id)\$$	shift
$id(E+$	$id)\$$	shift
$id(E + id$	$)\$$	reduce $T \to id$
$id(E + T$	$)\$$	reduce $E \to E + T$
$id(E$	$)\$$	shift
$id(E)$	$\$$	reduce $T \to id(E)$
T	$\$$	reduce $E \to T$
E	$\$$	reduce $S \to E$
S	$\$$	accept

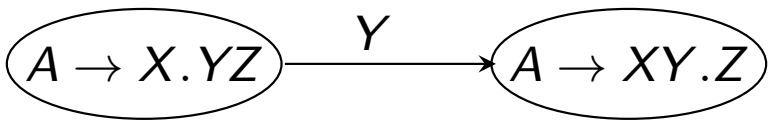
Algoritam parsiranja LR(k)

- Left to right - čitanje ulaza
- Rightmost derivation - najdešnje izvođenje
- (k) - koristi se k preduvidnih tokena

- Većina programskih jezikase može parsirati ovim algoritmom
- Nema značajnih ograničenja na formu gramatike
- Pravljenje tabele parsiranja nije jednostavno

Algoritam parsiranja LR

- Definisan je automatom - stanja i prelazi između njih
- Configuration - pravilo koje sadrži simbol "." sa desne strane čime se označava koji deo pravila je poklopljen sa ulazom
  Npr. $S \rightarrow AB.C$ označava da su AB već na steku i da se čeka pojavljivanje C da bi se primenila redukcija
- Configuration set - skup stanja u kojima može da se nađe automat
- Učitanim simbolom/redukovanom promenljivom vrši se prelaz iz jednog stanja u drugo



LR(0) automat

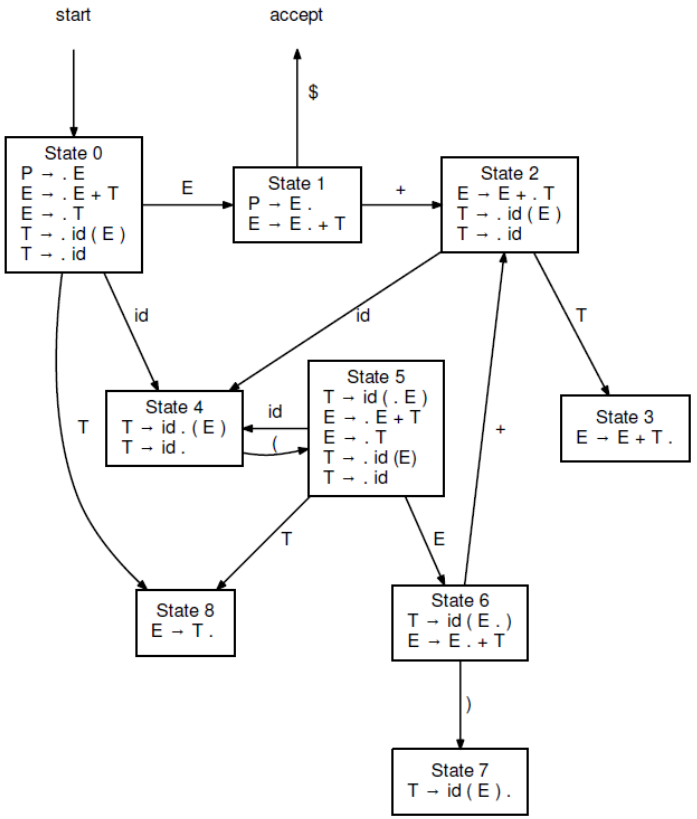
- LR(0) automat predstavlja sva moguća pravila koja bi mogla da se primene *shift-reduce* parserom
- 0 ukazuje na to da ne uzima u obzir naredni token, pri donošenju odluke o primeni pravila

LR(0) automat

- LR(0) automat predstavlja sva moguća pravila koja bi mogla da se primene *shift-reduce* parserom
- 0 ukazuje na to da ne uzima u obzir naredni token, pri donošenju odluke o primeni pravila

LR(0) automat

- LR(0) automat predstavlja sva moguća pravila koja bi mogla da se primene *shift-reduce* parserom
- 0 ukazuje na to da ne uzima u obzir naredni token, pri donošenju odluke o primeni pravila



Konflikti u LR

- LR(0) automat iako sadži sve mogućnosi za primenu akcija, ne rešava pomenute konflikte
- Simple LR (SLR) parsiranje, LR(1) automat, Lookahead LR (LALR) parsiranje, . . .

$$LL(1) \subset SLR \subset LALR \subset LR(1) \subset CFG$$

Konflikti u LR

- LR(0) automat iako sadži sve mogućnosi za primenu akcija, ne rešava pomenute konflikte
- Simple LR (SLR) parsiranje, LR(1) automat, Lookahead LR (LALR) parsiranje, ...

$$LL(1) \subset SLR \subset LALR \subset LR(1) \subset CFG$$

Konflikti u LR

- LR(0) automat iako sadži sve mogućnosi za primenu akcija, ne rešava pomenute konflikte
- Simple LR (SLR) parsiranje, LR(1) automat, Lookahead LR (LALR) parsiranje, ...

$$\text{LL(1)} \subset \text{SLR} \subset \text{LALR} \subset \text{LR(1)} \subset \text{CFG}$$