ZBIRKA ZADATAKA
IZ
PROGRAMSKIH PREVODILACA I

UNIVERZITET U BEOGRADU

Dušan Velašević, Dragan Bojić

ZBIRKA ZADATAKA
IZ
PROGRAMSKIH PREVODILACA I

ELEKTROTEHNIČKI FAKULTET
Beograd, 2000.

prof. dr Dušan Velašević, mr Dragan Bojić
ZBIRKA ZADATAKA IZ PROGRAMSKIH PREVODILACA

Recenzenti:
dr Jovan Đorđević, vanredni profesor Elektrotehničkog fakulteta u Beogradu
dr Vladimiru Blagojević, docent Elektrotehničkog fakulteta u Beogradu

Izdaje:
Elektrotehnički fakultet Univerziteta u Beogradu
Bulevar revolucije 73, Beograd

Za izdavača:
prof. dr Vlada Teodosić

Urednik:
mr Gradimir Božilović

Tiraž:
200 primeraka

Štampa:
Zavod za grafičke tehnologije Tehnološko – metalurškog fakulteta u Beogradu

SADRŽAJ

PREDGOVOR

Zbirka zadataka iz Programskih prevodilaca treba da posluži bržem i potpunijem savladavanju teorijskih i praktičnih osnova u ovoj oblasti. S obzirom na njenu obimnost, zbirka je podeljena u dva toma. Na globalnom planu, ciljevi oba toma su postavljeni tako da obuhvate

- ilustraciju niza tehnika i algoritama na problemima čiji je stepen složenosti takav da omogućava praćenje rešenja bez većeg napora, kao i određen broj zadataka većeg stepena složenosti;
- rešenja nekoliko odabranih problema programskim putem kako bi se razmotrili i neki čisto programerski aspekti;
- korišćenje razvojnih alata LEX i YACC za realizaciju programskih prevodilaca;
- originalni doprinos autora problemu sintaksne analize generalizacijom postojećih metoda.

U strukturnom pogledu I tom je podeljen u pet poglavlja s ukupno 113 zadataka, i to:

- leksička analiza (33)
- gramatike (35)
- sintaksna analiza od vrha ka dnu (22)
- sintaksna analiza od dna ka vrhu (23)
- prilozi.

Struktura zadataka je sačinjena od :

- formulacije
- analize (gde je to potrebno)
- rešenja
- diskusije (gde je to potrebno).

U poglavlju o leksičkoj analizi zastupljeni su svi metodi i algoritmi važni za konstrukciju odgovarajućeg analizatora. Tu se iscrpno razmatraju deterministički i nedeterministički automati, njihova konstrukcija iz regularnih izraza, transformacija nedeterminističkih automata u determinističke, minimizacija determinističkih automata, leksički procesori, oporavak zbog grešaka u identifikatorima kao i programski primeri.

Poglavlje o gramatikama sadrži niz zadataka iz analize i obrade gramatika. Tu su pre svega: krajnje levo i krajnje desno izvođenje u gramatici, dvosmislenost u gramatici, nalaženje formalnog i neformalnog opisa jezika na osnovu gramatike i obratno, dokazivanje nekih osobina gramatika, gramatičke transformacije (nalaženje suvišnih neterminala, eliminisanje smena oblika $A \to B$, eliminisanje pojava startnog simbola iz desnih strana svih smena, eliminisanje

kružnih izvođenja), specijalne forme bezkontekstnih gramatika, operacije zatvaranja jezika, gramatika u Greibach-ovoj normalnoj formi i normalnoj formi Čomskog, i programski primer.

Treće poglavlje razmatra zadatke iz oblasti sintaksne analize od vrha ka dnu. To su pre svega konstrukcija potisnih automata, problem beskonačnih petlji kod potisnih automata. Zatim, analiza od vrha ka dnu koja obuhvata konstrukciju prepoznavača za S i q gramatike, nalaženje skupova FOLLOW i SELECT za date gramatike, konstrukcija prepoznavača za LL(1) gramatike na osnovu relacionog pristupa, nalaženje gramatike na osnovu zadatog potisnog automata, eliminisanje mrtvih i nedostižnih neterminala, eliminacija leve rekurzije, leva faktorizacija, konstrukcija prepoznavača na osnovu rekurzivnog spusta i programski primeri.

Poglavlje o sintaksnoj analizi od dna ka vrhu obuhvata tri pristupa: relacioni, konfiguracioni i hibridni koji je u potpunosti originalni pristup autora. U okviru relacionog modela razmatrana su dva pristupa: pomeri-identifikuj i pomeri-svedi. U okviru pristupa pomeri-identifikuj razmatrane su konstrukcije prepoznavača za bezsufiksne gramatike, gramatike slabog i prostog mešovitog prvenstva, nalaženje relacija BELOW i REDUCED_BY za date gramatike, nalaženje odgovarajuće gramatike za datu kontrolnu tabelu i optimizacija parserskih tabela. U pristupu pomeri-svedi razmatrane su konstrukcije prepoznavača za gramatike LR(0) i SLR(1), nalaženje relacija OBELOW i OFIRST. U konfiguracionom modelu razmatraju se konstrukcije prepoznavača za gramatike tipa LR(0), SLR(1), LALR(1) i LR(1), određivanje predikcionih simbola, nalaženje gramatika višeg reda (SLR(k), LALR(k), k>1). U hibridnom pristupu, izvršena je generalizacija sintaksne analize kojom se uklanja potreba za posebnim razmatranjem sintaksne analize od vrha ka dnu i od dna ka vrhu. Kao najopštija u klasi sa jednim predikatskim simbolom, gramatika LR(1) postaje veoma praktična za realizaciju, kao što je to sada bila LALR(1) gramatika, zbog drastične redukcije broja stanja. Ako se primeni hibridna metoda za konstrukciju prepoznavača za gramatiku LR(1), a gramatika je u klasi LL(1), onda će se dobiti samo kontrolna tabela dok će potisna biti prazna, znači kao da je konstruisan relacionim modelom. To se može uraditi i polazeći od konstrukcije LALR(1) prepoznavača samo što svaka LL(1) gramatika nije ujedno i LALR(1) gramatika (to su obično patološki slučajevi). Urađena su dva zadatka u kojima je prikazana konstrukcija prepoznavača za LALR(1) i LR(1) gramatike.

Prilozi su podeljeni u dva dela. Prvi deo razmatra generator leksičkog analizatora LEX, način njegove upotrebe i forma specifikacije. Drugi deo daje teorijske osnove raznih gramatičkih transformacija kao što su: leva faktorizacija, ugaona i singlton zamena, eliminacija leve rekurzije i praznih smena, dobijanje poljske translacione gramatike i pretvaranje gramatike u "pomeri-identifikuj" konzistentnu.

Na kraju, autori žele da se zahvale prof. dr Živku Tošiću sa Elektronskog fakulteta u Nišu, prof. dr Zoranu Jovanoviću i recenzentima prof. dr Jovanu Đorđeviću i doc. dr Vladimiru Blagojeviću sa Elektrotehničkog fakulteta u Beogradu za korisne sugestije u konačnom uobličavanju I toma zbirke.

Beograd, avgust 1999. godine

Autori

1. Leksička analiza

1.1. Deterministički konačni automati

Zadatak 1.1.1

a) Predstaviti zadati konačni automat (Sl. 1.1.1) u alternativnoj formi, pomoću grafa prelaza.

b) Izložiti nekoliko sekvenci koje ovaj automat prepoznaje. Navesti nekoliko sekvenci koje ovaj automat odbija.

c) Pronaći najkraću sekvencu koju dati automat prepoznaje.

	0	1	
→ A	D	A	0
B	A	C	0
C	A	F	0
D	B	C	0
E	B	C	1
F	E	A	1

Sl. 1.1.1

Analiza problema

Deterministički konačni automat opisan je uređenom petorkom (S, U, δ, S_t, P) gde:

- S predstavlja skup stanja automata, u konkretnom slučaju S = {A, B, C, D, E, F}; stanjima su obeležene vrste tabele prelaza automata prikazane na Sl. 1.1.1;
- U predstavlja skup ulaznih simbola (azbuku automata), u konkretnom slučaju U = {0, 1}; ulazni simboli obeležavaju kolone tabele prelaza.

- δ: S x U $\rightarrow$ S predstavlja funkciju prelaza; za svako stanje i svaki ulazni simbol definiše novo stanje u koje automat prelazi iz stanja s $\in$ S za ulazni simbol u $\in$ U; u konkretnom slučaju funkcija δ zadata je tabelom prelaza prikazanom na Sl. 1.1.1; u ulaz tabele u vrsti X i koloni Y upisana je vrednost δ(X,Y);
- $S_t \in S$ je startno stanje automata (stanje iz koga počinje rad). U konkretnom slučaju S_t = A; ovo stanje po konvenciji je stanje koje obeležava prvu vrstu tabele prelaza ili stanje koje je označeno strelicom.
- P $\subseteq$ S predstavlja skup stanja prihvatanja, u konkretnom slučaju P = {E, F}. Stanja iz skupa S-P nazivaju se stanjima odbijanja. Na Sl. 1.1.1 stanja prihvatanja označena su jedinicom desno od odgovarajuće vrste tabele prelaza, a stanja odbijanja nulom.

Rad automata odvija se po sledećem algoritmu:

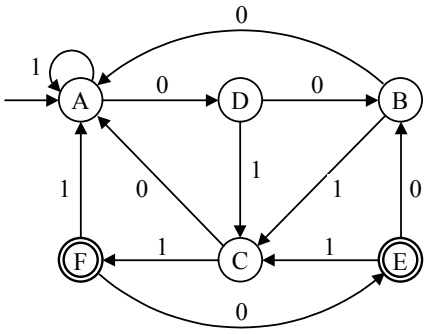
```
tekuće_stanje := St;
tekući_ulaz := prvi simbol ulazne sekvence;
while not (kraj ulazne sekvence)
        tekuće_stanje := δ( tekuće_stanje, tekući ulaz );
        tekući_ulaz := sledeći znak ulazne sekvence;
end while;
if ( tekuće_stanje ∈ P )
        then ulazna sekvenca se prihvata;
        else ulazna sekvenca se ne prihvata;
end if.
```

Skup svih sekvenci ulaznih simbola koje automat prihvata naziva se jezik tog automata.

Rešenje

a)

U grafu prelaza (Sl. 1.1.2) stanjima automata odgovaraju čvorovi grafa. Startno stanje obeleženo je ulaznom granom, stanja prihvatanja posebno se obeležavaju (dupla linija). Prelazi između stanja odgovaraju granama grafa. Ako je δ(X,Y)=Z tada iz čvora X grafa postoji grana obeležena ulaznim simbolom Y ka čvoru Z.



Sl. 1.1.2

b)

Posmatrajmo rad datog automata za ulaznu sekvencu 0110. Iz startnog stanja A imamo sledeći niz promena stanja (što je lako uočiti na Sl. 1.1.2):

$$A \xrightarrow{0} D \xrightarrow{1} C \xrightarrow{1} F \xrightarrow{0} E$$

S obzirom da je E stanje prihvatanja, znači da sekvenca 0110 pripada jeziku datog automata. Probanjem možemo utvrditi da sekvence 1011, 0011, 011011, 011011011 itd. takođe pripadaju jeziku datog automata, dok, na primer, sekvence 0, 00, 00111, 1, 11, 111, 1111 itd. ne pripadaju jer se rad automata završava u nekom od stanja odbijanja.

c)

Najkraća sekvenca koju automat prihvata, izaziva najkraći niz promena stanja počev od startnog stanja A do nekog od stanja prihvatanja E ili F. U stanje E može se doći isključivo prolazeći kroz stanje F (jer se E pojavljuje isključivo u vrsti F tabele prelaza), čime sekvence koje se završavaju u stanju E automatski otpadaju iz razmatranja.

U stanje F dolazi se isključivo iz stanja C, čime se problem pronalaženja najkraće sekvence iz A u F svodi na pronalaženje najkraće sekvence iz A u C:

$$A \to \cdots \to C \xrightarrow{1} F$$

U stanje C može se doći iz stanja B, D i E. Stanje E već je eliminisano iz razmatranja, tako da ostaje:

$$A \to \cdots \to B \xrightarrow{1} C \xrightarrow{1} F$$

$$A \to \cdots \to D \xrightarrow{1} C \xrightarrow{1} F$$

Daljim razmatranjem utvrđujemo da se u stanje B može doći iz stanja D ili E, te ova varijanta otpada, dok se u stanje D može doći iz stanja A čime smo utvrdili da je najkraća sekvenca koja se prihvata 011:

$$A \xrightarrow{0} D \xrightarrow{1} C \xrightarrow{1} F$$

Zadatak 1.1.2

Konstruisati automat sa konačnim brojem stanja koji će prepoznavati deo FORTRAN deklaracije iza ključne reči INTEGER, na primer:

INTEGER A

INTEGER X, I(3), U, V, W

INTEGER C(3, J, 4), B

pri čemu uvodimo sledeća uprošćenja: razmaci se ignorišu u ulaznom nizu, promenljive su jednoslovne, konstante su jednocifrene i ne postoji ograničenje u broju indeksa nizova.

Rešenje

U postupku sinteze automata prvo ćemo definisati ulaznu azbuku kao petočlani skup:

U = {s c , ( ) }

pri čemu ulazni simbol s označava proizvoljno slovo, c označava proizvoljnu cifru. Ovim smanjujemo veličinu automata pošto je akcija automata ista bez obzira o kom slovu, odnosno o kojoj cifri je reč.

Za konstrukciju tabele prelaza koristićemo neformalni metod numeracije ulaznih simbola. Potrebno je napisati jedan ili više karakterističnih primera ulaznih sekvenci. Deklaracije:

INTEGER A(3, J, 4), B INTEGER A, B, C INTEGER A(3)

redom daju ulazne sekvence:

s (c, s, c) , s s, s, s s(c)

Sada razmatramo prelaze između stanja traženog automata koje izazivaju ove ulazne sekvence:

s	(	c	,	s	,	c	)	,	s
1	2	3	4	5	6	5	4	7	8

s	,	s	,	s
1	2	8	2	8

s	(	c	)
1	2	3	4

2
2

7

Iz startnog stanja 1 pod ulazom s prelazi se u stanje 2, itd. Pri odluci da li za određeni prelaz uvesti novo stanje ili upotrebiti neko od postojećih, razmatra se da li je očekivani nastavak ulazne sekvence specifičan ili je nastavak isti kao za neko od postojećih stanja. Tako se u prvoj sekvenci iz stanja 4 prelazi u stanje 5 pod dejstvom simbola "," na ulazu, isto kao i iz stanja 6 pod dejstvom istog ulaznog simbola, jer se u obe situacije posle zareza očekuje sledeći indeks niza. Na sličan način prelazi iz stanja 7, odnosno 2 pod dejstvom ulaza "," su u stanje 8, čime je obezbeđeno pojavljivanje proizvoljnog broja promenljivih u deklaraciji.

Stanja se mogu neformalno opisati na sledeći način (nije neophodno za konstrukciju automata, ali je korisno radi provere):

1. očekuje se početak deklaracije;
2. prošla promenljiva; očekuje se otvorena zagrada, kraj deklaracije ili zarez koji razdvaja promenljive u deklaraciji;
3. prošla otvorena zagrada; očekuje se prvi indeks niza;
4. prošao prvi indeks; očekuje se zatvorena zagrada ili zarez koji razdvaja indekse;
5. prošao zarez koji razdvaja indekse; očekuje se naredni indeks niza;
6. prošao naredni indeks; očekuje se zatvorena zagrada ili zarez koji razdvaja indekse;
7. prošla deklaracija niza; očekuje se kraj deklaracije ili zarez koji razdvaja promenljive u deklaraciji;
8. prošao zarez koji razdvaja promenljive; očekuje se sledeća promenljiva u deklaraciji.

Na osnovu prethodnog razmatranja popunjavamo tabelu prelaza automata (Sl. 1.1.3). Stanja prihvatanja su ona stanja u kojima se očekuje kraj deklaracije, odnosno stanja 2 i 7. Ostala stanja su stanja odbijanja.

Nepopunjene ulaze tabele smatramo prelazima u posebno stanje E, takozvano stanje greške (radi se o stanju odbijanja i kada automat u toku rada jednom dođe u to stanje on ostaje u njemu do kraja ulazne sekvence). Naravno, za svaki nepopunjen prelaz prethodno treba razmotriti da li je stvarno reč o nelegalnoj ulaznoj sekvenci, ili korišćeni primeri ulaznih sekvenci nisu pokrili tu situaciju.

	v	c	,	(	)	
1	2	E	E	E	E	0
2	E	E	8	3	E	1
3	6	4	E	E	E	0
4	E	E	5	E	7	0
5	6	4	E	E	E	0
6	E	E	5	E	7	0
7	E	E	8	E	E	1
8	2	E	E	E	E	0
E	E	E	E	E	E	0

Sl. 1.1.3

Diskusija

Automat dobijen metodom enumeracije simbola ne mora nužno biti sa najmanjim mogućim brojem stanja. Na primer, u dobijenom automatu stanja 1 i 8 su oba stanja odbijanja i imaju identične vrste (znači da će se automat ponašati na isti način ukoliko nastavi rad iz stanja 1 kao i iz stanja 8). Zbog toga je moguće izbaciti jedno od ta dva stanja iz tabele prelaza automata, a sve pojave izbačenog stanja zameniti preostalim stanjem, čime se smanjuje ukupan broj stanja. Isti postupak moguće je sprovesti sa parovima stanja 3 i 5, kao i 4 i 6. Problematika minimizacije konačnih automata razmatra se u odeljku 1.2.

U cilju jednostavnije implementacije automata moguće je kraj ulazne sekvence označiti eksplicitnim markerom ⊣ i proširiti ulaznu azbuku automata ovim simbolom. Na Sl. 1.1.4 prikazana je tabela prelaza ovako proširenog automata koji se od prepoznavača time pretvara u procesor. U koloni ⊣ akcija YES odgovara stanjima prihvatanja, a akcija NO stanjima odbijanja ulazne sekvence.

Automat na Sl. 1.1.4 obavezno procesira kompletnu ulaznu sekvencu. Alternativa je prikazana na Sl. 1.1.5, gde je stanje greške eliminisano i prelazi u ovo stanje zamenjeni akcijom NO, čime se sekvenca odbija u najranijem mogućem trenutku procesiranja (ovakva vrsta procesora naziva se detektor).

	v	c	,	(	)	⊣
1	2	E	E	E	E	NO
2	E	E	8	3	E	YES
3	6	4	E	E	E	NO
4	E	E	5	E	7	NO
5	6	4	E	E	E	NO
6	E	E	5	E	7	NO
7	E	E	8	E	E	YES
8	2	E	E	E	E	NO
E	E	E	E	E	E	NO

Sl. 1.1.4

	v	c	,	(	)	⊣
1	2	NO	NO	NO	NO	NO
2	NO	NO	8	3	NO	YES
3	6	4	NO	NO	NO	NO
4	NO	NO	5	NO	7	NO
5	6	4	NO	NO	NO	NO
6	NO	NO	5	NO	7	NO
7	NO	NO	8	NO	NO	YES
8	2	NO	NO	NO	NO	NO

Sl. 1.1.5

Zadatak 1.1.3

Projektovati automat sa konačnim brojem stanja koji će prepoznavati sekvence jedinica i nula u kojima je broj jedinica paran, a broj nula neparan.

Rešenje

Traženi automat prikazan je na Sl. 1.1.6. Za svaku od četiri kombinacije parnosti i neparnosti nula i jedinica u ulaznoj sekvenci uvedeno je posebno stanje. Ako, na primer, sekvenca sadrži paran broj nula i neparan broj jedinica, automat će rad završiti u stanju 1odd0even.

	0	1	⊣
1even 0even	1even0odd	1odd0even	NO
1odd 0even	1odd0odd	1even0even	NO
1even 0odd	1even0even	1odd0odd	YES
1odd 0odd	1odd0even	1even0odd	NO

Sl. 1.1.6

Prva vrsta predstavlja startno stanje, prazna sekvenca odgovara parnom broju i nula i jedinica. Primetimo da se iz svakog ulaza u tabelu može doći do stanja prihvatanja. Ovaj automat ne poseduje stanje greške.

Zadatak 1.1.4

Projektovati automate sa konačnim brojem stanja nad ulaznom azbukom {0,1}, koji prepoznaju tačno sledeće sekvence simbola:

a) sve ulazne sekvence

b) nijednu ulaznu sekvencu

c) sekvencu 101

d) samo praznu sekvencu.

Rešenje

a) Da bi se sve sekvence prepoznale dovoljan je automat sa jednim stanjem (Sl. 1.1.7):

	0	1	
A	A	A	1

Sl. 1.1.7

Stanje A je istovremeno i stanje prihvatanja.

b) Automat koji ne prihvata nijednu ulaznu sekvencu prikazan je na Sl. 1.1.8. Automat poseduje samo jedno stanje – radi se o stanju odbijanja.

	0	1	
A	A	A	0

Sl. 1.1.8

c) Automat za prepoznavanje sekvence 101 prikazan je na Sl. 1.1.9.

	0	1	
A	E	B	0
B	C	E	0
C	E	D	0
D	E	E	1
E	E	E	0

Sl. 1.1.9

A je startno stanje u kome se očekuje početak niza simbola. Ako kao prvi simbol stigne nula odmah se prelazi u stanje greške jer se očekuje da sekvenca 101 stigne odmah a ne bilo gde u nizu signala. Stanje D je neophodno da se izbegne prihvatanje dužih sekvenci koje počinju sa 101.

d) Automat koji prihvata samo praznu sekvencu, odnosno ulaznu sekvencu dužine nula simbola prikazan je na Sl. 1.1.10. Startno stanje A je stanje prihvatanja, a svi prelazi iz stanja A su prelazi u stanje greške E.

	0	1	
A	E	E	1
E	E	E	0

Sl. 1.1.10

Zadatak 1.1.5

Projektovati automat sa konačnim brojem stanja koji će prepoznavati jednu od sledeće tri rezervisane reči ALGOL-a 60: STEP, STRING, SWITCH.

Rešenje

Metodom numeracije simbola određujemo stanja i prelaze.

```
S T E P    S T R I N G      S W I T C H
1 2 3 4 5  1 2 3 6 7 8 9    1 2 10 11 12 13 14
```

Iz datog primera vidi se generalna strategija za dodavanje novih ključnih reči. Nova reč se dovede na ulaz postojećeg automata i procesira do kraja reči (tada se završno stanje automata proglasi stanjem prihvatanja) ili do trenutka kada automat pređe u stanje greške. U ovom drugom slučaju, uvode se nova stanja za prepoznavanje preostalog dela nove reči, a poslednji primenjeni ulaz postojećeg automata, koji predstavlja prelaz u stanje greške, modifikuje se tako da postaje prelaz u prvo od novih stanja.

	S	T	E	P	R	I	N	G	W	C	H	
1	2											0
2		3							10			0
3			4		6							0
4				5								0
5												1
6						7						0
7							8					0
8								9				0
9												1
10						11						0
11		12										0
12										13		0
13											14	0
14												1
E												0

Sl. 1.1.11

Diskusija

Automat dobijen ovim postupkom nije nužno minimalan. Očigledno su stanja prihvatanja sva ekvivalentna.

Ovaj prepoznavač samo daje 0 ili 1, a ne i informaciju koja je reč prepoznata. Ukoliko se želi napraviti procesor koji daje informaciju o tome koja je reč prepoznata, ulaznu azbuku treba proširiti markerom kraja ⊣ i u istoimenoj koloni tabele prelaza ulaze koji odgovaraju stanjima prihvatanja popuniti akcijama “STEP”, “STRING”, “SWITCH”, a ostale ulaze akcijom “NO”.

Zadatak 1.1.6

Napisati (na paskaloidnom pseudojeziku) algoritam koji za proizvoljan deterministički automat i proizvoljnu ulaznu sekvencu određuje najduži prefiks ulazne sekvence koju zadati automat prihvata. Da li se pri tome uvek mora pročitati cela ulazna sekvenca?

Rešenje

Modifikovaćemo algoritam rada konačnog automata () na taj način što će se pamtiti trenutno dostignuta pozicija u ulaznom nizu kada god tekuće stanje postane neko od stanja prihvatanja. Rad automata se završava ili kada se procesira kompletna ulazna sekvenca, ili ako se pređe u stanje greške, u kom slučaju nema više mogućnosti ulaska u stanje prihvatanja u daljem procesiranju ulaza, pa nije potrebno razmatrati ostatak ulaza. Po okončanju rada, deo ulazne sekvence od prve do poslednje zapamćene pozicije predstavlja traženi prefiks. Sledi modifikovani algoritam. Promenljiva tekuća_pozicija daje poziciju tekućeg ulaznog simbola u ulaznom nizu, a promenljiva pozicija_prihvatanja pamti poziciju dostignutu u poslednjem dolasku u stanje prihvatanja.

```
tekuća_pozicija := 0;
pozicija_prihvatanja := -1;
tekuće_stanje := S_t
tekući_ulaz := prvi simbol ulazne sekvence;
while not (kraj ulazne sekvence ili tekuće_stanje = stanje_greške)
        if ( tekuće_stanje ∈ P )
                then pozicija_prihvatanja := tekuća_pozicija;
        end if;
        tekuće_stanje := δ( tekuće_stanje, tekući ulaz );
        tekući_ulaz := sledeći simbol ulazne sekvence;
        tekuća_pozicija := tekuća_pozicija + 1;
end while;
if ( pozicija_prihvatanja = -1 )
        then ne postoji traženi prefiks jer nije bilo prolaska kroz stanje prihvatanja;
        else
                if ( pozicija_prihvatanja = 0 )
                        then traženi prefiks je prazna sekvenca;
                        else  traženi prefiks je deo ulazne sekvence od
                              1. pozicije do pozicije_prihvatanja.
                end if;
end if.
```

1.2. Minimizacija automata

Zadatak 1.2.1

Pronaći sva suvišna stanja u automatu prikazanom na Sl. 1.2.1.

	0	1	2	
→ A	C	E	G	0
B	J	E	G	0
C	J	A	H	1
D	F	A	G	0
E	E	J	H	1
F	D	I	A	0
G	H	A	J	0
H	G	J	B	1
I	D	F	G	0
J	B	H	G	0

Sl. 1.2.1

Analiza problema

Suvišna stanja automata su ona stanja u koja se nikad ne može stići pod dejstvom ulaznih simbola iz startnog stanja, to jest, stanja koja nemaju uticaja na rad automata. Najčešće se ne radi samo o jednom suvišnom stanju već postoji skup suvišnih stanja unutar koga se može kružiti ali se u njega nikako ne može doći iz početnih stanja. Postupak nalaženja suvišnih stanja je indirektan: zapravo se traže sva stanja u koja se može doći iz početnog.

Rešenje

1. Startno stanje A očigledno nije suvišno, a ni C, E i G jer se u njih može stići iz A (pojavljuju se u vrsti A). Dakle skup dostižnih stanja je: {A, C, E, G}
2. Razmatramo redom elemente skupa dostižnih stanja da bismo odredili preostala dostižna stanja. U vrsti C tabele prelaza pojavljuju se stanja J, A, H. Skup dostižnih stanja je:

   {A, C, E, G, J, H}
3. Razmatramo vrstu E: postoje prelazi u E, J, H - skup dostižnih stanja se ne menja.
4. Razmatramo vrstu G: postoje prelazi u H, A, J - skup se ne menja.
5. Razmatramo vrstu J: postoje prelazi u B, H, G. Skup je: {A, C, E, G, J, H, B}
6. Razmatramo vrstu H: postoje prelazi u G, J, B - skup se ne menja.
7. Razmatramo vrstu B: postoje prelazi u J, E, G - skup se ne menja. Pošto su sva stanja iz skupa dostižnih razmotrena, postupak se okončava.

Znači suvišna su stanja: {D, F, I} i možemo jednostavno izbaciti odgovarajuće vrste iz tabele prelaza.

Zadatak 1.2.2

a) Dati primer dva ekvivalentna konačna automata M i N, gde jednom stanju automata N odgovara više ekvivalentnih stanja automata M. Dokazati da primer zadovoljava uslove zadatka.

b) Dati primer dva ekvivalentna konačna automata M i N, gde automat M ima stanje koje nije ekvivalentno ni sa jednim stanjem automata N. Dokazati da primer zadovoljava uslove zadatka.

Analiza problema

Stanje S_x automata X ekvivalentno je stanju S_y automata Y ako i samo ako automat X startujući od stanja S_x prihvata tačno isti skup sekvenci kao i automat Y startujući od stanja S_y.

Ako stanja S_x automata X i S_y automata Y nisu ekvivalentna, onda postoji najmanje jedna sekvenca koju automat X, polazeći iz stanja S_x prihvata, a automat Y polazeći iz stanja S_y odbacuje, ili obratno, automat X, polazeći od stanja S_x, sekvencu odbacuje, a automat Y, polazeći od stanja S_y je prihvata. Takva sekvenca naziva se *sekvenca razlikovanja*.

Operativna definicija na osnovu koje je moguće izvršiti proveru ekvivalencije dva stanja je sledeća: Stanja S_x i S_y su ekvivalentna ako i samo ako su zadovoljeni:

1. Uslov kompatibilnosti - stanja S_x i S_y moraju oba biti ili stanja prihvatanja ili stanja odbijanja.
2. Uslov propagacije - prelazi iz ovih stanja, za svaki ulazni simbol, moraju biti u međusobno ekvivalentna stanja.

Dva automata su ekvivalentna ako i samo ako su im startna stanja ekvivalentna.

Rešenje

a)

Na Sl. 1.2.2 prikazan je proizvoljan automat M koji ima dva ekvivalentna stanja B1 i C1 (oba predstavljaju stanja prihvatanja i vrste B1 i C1 su identične tako da su zadovoljeni uslovi kompatibilnosti i propagacije). Uklanjanjem stanja B i preimenovanjem istoimenog ulaza u prvoj vrsti u C2 dobijamo automat N.

Ispitajmo da li su startna stanja ova dva automata ekvivalentna. Uslov kompatibilnosti je zadovoljen, jer se radi o stanjima odbijanja. U cilju ispitivanja uslova propagacije beležimo parove novih stanja za svaki od ulaznih simbola u takozvanu tabelu ekvivalencije stanja.

	x	y
A1, A2	B1, C2	C1, C2

Automat M:

	x	y	
→ A1	B1	C1	0
B1	B1	A1	1
C1	B1	A1	1

Automat N:

	x	y	
→ A2	C2	C2	0
C2	C2	A2	1

Sl. 1.2.2

Sada se problem svodi na ispitivanje ekvivalencije stanja B1 i C2, kao i stanja C1 i C2. Oba para zadovoljavaju uslov kompatibilnosti jer se radi o stanjima prihvatanja. U cilju ispitivanja zadovoljenosti uslova propagacije nastavljamo popunjavanje tabele ekvivalencije:

	x	y
A1, A2	B1, C2	C1, C2
B1, C2	B1, C2	A1, A2
C1, C2	B1, C2	A1, A2

Pošto se u tabeli nisu pojavili novi parovi stanja, popunjavanje tabele je završeno. S obzirom da nije došlo do pojave parova nekompatibilnih stanja, zaključujemo da svi parovi iz tabele predstavljaju međusobno ekvivalentna stanja. Dakle automati M i N su međusobno ekvivalentni, a iz tabele vidimo da su stanja B1 i C1 automata M ekvivalentna sa stanjem C2 automata N čime su zadovoljeni uslovi zadatka.

b)

Automatu M iz tačke a) dodaćemo stanje D1, a automat N ostaje isti (Sl. 1.2.3) .

Automat M:

	x	y	
→ A1	B1	C1	0
B1	B1	A1	1
C1	B1	A1	1
D1	D1	D1	0

Automat N:

	x	y	
→ A2	C2	C2	0
C2	C2	A2	1

Sl. 1.2.3

Stanje D1 nije dostižno iz stanja A1, to jest nema uticaja na rad automata M, pa su automati M i N ekvivalentni kao što je pokazano u tački a). Stanje D1 nije ekvivalentno stanju C2 jer nije zadovoljen uslov kompatibilnosti. Što se tiče para D1, A2 konstrukcijom tabele ekvivalencije:

	x	y
D1, A2	D1, C2	D1, C2
D1, C2		

vidimo da uslov propagacije nije zadovoljen jer stanja D1 i C2 nisu međusobno ekvivalentna.

Zaključujemo da stanje D1 automata M nije ekvivalentno ni sa jednim stanjem automata N čime su zadovoljeni uslovi zadatka.

Zadatak 1.2.3

Pronaći minimalne ekvivalentne tabele prelaza za svaki od zadatih automata.

	0	1	
→ S1	S1	S3	0
S2	S7	S4	1
S3	S6	S5	0
S4	S1	S4	1
S5	S1	S4	0
S6	S7	S6	1
S7	S7	S3	0

(a)

	x	y	
→ 1	4	1	1
2	5	1	1
3	4	5	0
4	2	6	0
5	1	7	0
6	1	4	1
7	2	5	1

(b)

	A	C	E	L	
→ init	Error	C	Error	Error	0
C	CA	Error	CE	Error	0
CA	Error	Error	Error	CAL	0
CAL	Error	Error	Error	CALL	0
CALL	Error	Error	Error	Error	1
CE	Error	Error	Error	CEL	0
CEL	Error	Error	Error	CELL	0
CELL	Error	Error	Error	Error	1
Error	Error	Error	Error	Error	0

(c)

Sl. 1.2.4

Analiza problema

Ekvivalentni automat je onaj koji prihvata tačno isti skup sekvenci kao i polazni automat. Za određeni automat postoji mnogo različitih ekvivalentnih automata. Od interesa je među takvim automatima naći onaj sa najmanjim brojem stanja. Takav automat nema suvišnih niti

međusobno ekvivalentnih stanja (u suprotnom bismo mogli dobiti ekvivalentan automat sa manje stanja uklanjanjem suvišnih, odnosno zamenom svih međusobno ekvivalentnih stanja jednim stanjem).

U skladu sa tim postupak minimizacije automata sprovodi se u sledećim koracima:

a) uklanjanje suvišnih stanja (videti Zadatak 1.2.1);

b) nalaženje skupova međusobno ekvivalentnih stanja i zamena takvih skupova stanja sa po jednim predstavnikom za svaki od skupova.

U okviru rešenja biće demonstrirano više načina za sprovođenje koraka b) postupka minimizacije.

Rešenje

a)

Razmotrimo prvi od zadatih automata. Najpre eliminišimo suvišna stanja.. Dostižna stanja su $\{S_1, S_3, S_5, S_6, S_7, S_4\}$, što znači da je stanje S_2 suvišno, pa ga izbacujemo iz tabele prelaza, prikazane na Sl. 1.2.5.

	0	1	
$\rightarrow S_1$	S_1	S_3	0
S_2	S_7	S_4	1
S_3	S_6	S_5	0
S_4	S_1	S_4	1
S_5	S_1	S_4	0
S_6	S_7	S_6	1
S_7	S_7	S_3	0

Sl. 1.2.5

Prva varijanta postupka minimizacije:

Proveravamo ekvivalenciju stanja po parovima na način opisan u prethodnom zadatku. Potrebno je razmotriti sve parove stanja.

1. par stanja: S_1, S_3. Oba su stanja neprihvatanja. Konstruišemo tabelu ekvivalencije.

	0	1
S_1, S_3	S_1, S_6	S_3, S_5
S_1, S_6		

Stanja S_1 i S_6 nisu ekvivalentna jer nisu kompatibilna pa ni S_1 i S_3 nisu ekvivalentna.

2. par stanja: S_1, S_4. Nisu kompatibilna, pa nisu ni ekvivalentna.

3. par stanja: S_1, S_5 – kompatibilna stanja. Tabela ekvivalencije glasi:

	0	1
S_1, S_5	S_1	S_3, S_4
S_3, S_4		

Stanja S_3 i S_4 nisu ekvivalentna jer nisu kompatibilna pa ni S_1 i S_5 nisu ekvivalentna.

4. par stanja: S_1, S_6. Nisu kompatibilna, pa nisu ni ekvivalentna.
5. par stanja: S_1, S_7 – kompatibilna stanja. Tabela ekvivalencije glasi:

	0	1
S_1, S_7	S_1, S_7	S_3

Tabela je kompletirana, što znači da su stanja S_1 i S_7 ekvivalentna.

6. par stanja: S_3, S_4 – nekompatibilna stanja, pa nisu ni ekvivalentna.
7. par stanja: S_3, S_5 – kompatibilna stanja. Tabela ekvivalencije glasi:

	0	1
S_3, S_5	S_1, S_6	S_5, S_4

Stanja S_1 i S_6 nisu kompatibilna, pa ni stanja S_3 i S_5 nisu ekvivalentna.

8. par stanja: S_3, S_6. Nisu kompatibilna, pa nisu ni ekvivalentna.
9. par stanja: S_3, S_7. Nisu ekvivalentna jer su S_1 i S_7 ekvivalentna, a S_1 i S_3 nisu kako je ranije utvrđeno.
10. par stanja: S_4, S_5. Nisu kompatibilna, pa nisu ni ekvivalentna.
11. par stanja: S_4, S_6 - kompatibilna stanja. Tabela ekvivalencije glasi:

	0	1
S_4, S_6	S_1, S_7	S_4, S_6

Kako su S_1 i S_7 ekvivalentna to su i S_4 i S_6 ekvivalentna.

12. par stanja: S_4, S_7. Nisu kompatibilna, pa nisu ni ekvivalentna.
13. par stanja: S_5 , S_6. Nisu kompatibilna, pa nisu ni ekvivalentna.
14. par stanja: S_5 , S_7. Nisu ekvivalentna jer su S_1 i S_7 ekvivalentna, a S_1 i S_5 nisu kako je ranije utvrđeno.
15. par stanja: S_6 , S_7. Nisu kompatibilna, pa nisu ni ekvivalentna.

Svi parovi su ispitani i utvrđeno je da su međusobno ekvivalentni parovi stanja S_1, S_7 i S_4, S_6, te svaki od parova možemo zameniti jednim stanjem. Obeležimo stanja na sledeći način: $A = \{S_1, S_7\}$, $B = \{S_4, S_6\}$, $C = S_3$, $D = S_5$. Rezultujući minimalni automat prikazan je na Sl. 1.2.6.

	0	1	
→ A	A	C	0
B	A	B	1
C	B	D	0
D	A	B	0

Sl. 1.2.6

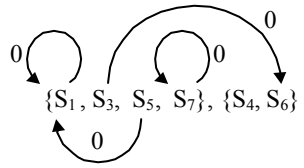
Druga varijanta postupka minimizacije (particioni metod):

Ovaj metod zasniva se na činjenici da je ekvivalencija stanja relacija ekvivalencije, što znači da se prema ovoj relaciji skup svih stanja deli na međusobno disjunktne podskupove. U svakom podskupu su stanja koja su međusobno ekvivalentna. Da bismo našli podelu skupa stanja prema ovoj relaciji, primenjujemo iterativan postupak u više koraka, u svakom koraku rafiniramo podelu skupa stanja dok ne dobijemo konačnu podelu.

1° Inicijalno, skup stanja delimo na dva podskupa - u jednom se nalaze sva stanja prihvatanja, a u drugom sva stanja odbijanja, u skladu sa optimističkom pretpostavkom da su ekvivalentna ona stanja koja zadovoljavaju uslov kompatibilnosti. Ova dva skupa čine početnu particiju.

$$P_0 = (\{S_1, S_3, S_5, S_7\}, \{S_4, S_6\})$$

2° U narednim koracima pokušavamo da dalje razbijemo pojedine skupove stanja, utvrđujući koja stanja ne zadovoljavaju uslov propagacije. Prvo ćemo razmotriti skup stanja $\{S_1, S_3, S_5, S_7\}$ i prelaz iz ovih stanja za ulazni simbol 0 (Sl. 1.2.7).

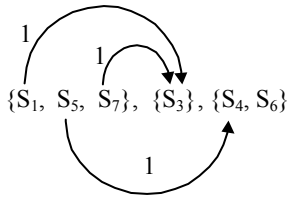


Sl. 1.2.7

Prelazi iz stanja S_1, S_5, S_7 vode u prvi skup za ulazni simbol 0 a iz stanja S_3 u drugi skup - to znači da S_3 ne može biti u istom skupu sa stanjima S_1, S_5, S_7 jer sigurno ne zadovoljava uslov propagacije. Druga particija je:

$$P_1 = (\{S_1, S_5, S_7\}, \{S_3\}, \{S_4, S_6\})$$

Prelazi iz S_1, S_5, S_7 pod dejstvom nule završavaju u istom skupu, pa treba razmotriti prelaze za ulazni simbol 1 (Sl. 1.2.8).



Sl. 1.2.8

Pod dejstvom jedinice iz S_1 i S_7 ide se u drugi skup, a iz S_5 u treći skup. Znači skup $\{S_1, S_5, S_7\}$ razbijamo na skupove $\{S_1, S_7\}$ i $\{S_5\}$.

$$P_2 = (\{S_1, S_7\}, \{S_5\}, \{S_3\}, \{S_4, S_6\})$$

Dalje posmatramo prelaze iz $\{S_1, S_7\}$ pod dejstvom 0 i 1 i vidimo da prelaze u iste skupove. Isto važi i za prelaze iz $\{S_4, S_6\}$. Sada se vidi da je P_2 istovremeno i završna particija. Stanja koja su se našla u istim skupovima u P_2 su međusobno ekvivalentna.

b)

Razmotrimo drugi automat (Sl. 1.2.9).

	x	y	
→ 1	4	1	1
2	5	1	1
3	4	5	0
4	2	6	0
5	1	7	0
6	1	4	1
7	2	5	1

Sl. 1.2.9

Stanja 1, 4, 2, 6, 5 i 7 nisu suvišna – znači suvišno je stanje 3. Minimizaciju radimo metodom particija:

$$P_0=(\{1, 2, 6, 7\}, \{4, 5\})$$

Pod dejstvom ulaza x iz stanja 6 i 7 prelazi se u stanja iz drugog skupa particije P_0, a iz stanja 1 i 2 prelazi se u stanja iz prvog skupa iste particije. Deobom prvog skupa particije P_0 dobija se particija P_1:

$$P_1=(\{1, 2\}, \{ 6, 7\}, \{4, 5\})$$

Ispitivanjem prelaza za svaki ulazni simbol može se utvrditi da dalje razbijanje nije moguće, to jest, P_1 je završna particija. Novi automat, prikazan na Sl. 1.2.10, ima samo tri stanja A = {1, 2}, B = {6, 7} i C = {4, 5}.

	x	y	
→ A	C	A	1
B	A	C	0
C	A	B	1

Sl. 1.2.10

c)

Treći automat (Sl. 1.2.11) nema suvišnih stanja.

	A	C	E	L	
→ init	Error	C	Error	Error	0
C	CA	Error	CE	Error	0
CA	Error	Error	Error	CAL	0
CAL	Error	Error	Error	CALL	0
CALL	Error	Error	Error	Error	1
CE	Error	Error	Error	CEL	0
CEL	Error	Error	Error	CELL	0
CELL	Error	Error	Error	Error	1
Error	Error	Error	Error	Error	0

Sl. 1.2.11

Postupkom minimizacije utvrđuje se da su ekvivalentna stanja: {CALL, CELL}, {CAL, CEL} i {CA, CE}. Sada se jasno vidi da su stanja ekvivalentna ako automat, startujući iz svakog od tih stanja, prihvata isti skup ulaznih sekvenci.

Diskusija

Zadatak 1.7.2 ilustruje programsku realizaciju postupka minimizacije particionim metodom.

Zadatak 1.2.4

Projektovati minimalni deterministički konačni automat koji prepoznaje FORTRAN-sku REAL deklaraciju (isključujući samu reč REAL). Primer ispravne REAL deklaracije je:

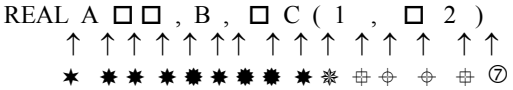
REAL A ,B, C(1,2),D

Pretpostaviti da su promenljive jednoslovne, konstante jednocifrene, razmak proizvoljan, da se deklaracija navodi u jednom redu i da se ulazna azbuka sastoji od sledećih šest simbola:

promenljiva konstanta ( ) , *razmak*

Rešenje

Pogledajmo jednu REAL deklaraciju i prepoznajmo stanja :



Stanja se mogu rečima opisati na sledeći način:

□ znak razmaka

← početno stanje; čeka se promenljiva

↑ stanje prihvatanja; deklarisana promenljiva

→ bio zarez; čeka se promenljiva

↓ bila zagrada; čeka se konstanta

° čeka se zarez ili “)”

± čeka se sledeća konstanta

″ bila “)”; čeka se zarez

Tabela prelaza automata prikazana je na Sl. 1.2.12. Prazni prelazi predstavljaju prelaze u stanje greške E koje po konvenciji nije prikazano u tabeli.

	promenljiva	*konstanta*	(	)	,	□	
1	2					1	0
2			4		3	2	1
3	2					3	0
4		5				4	0
5				7	6	5	0
6		5				6	0
7					3	7	1

Sl. 1.2.12: Tabela stanja/prelaza

Sprovedimo postupak minimizacije dobijenog automata. Sva stanja automata dostižna su iz startnog stanja. Za pronalaženje eventualnih ekvivalentnih stanja koristimo metod particija.

- Inicijalno delimo stanja na stanja prihvatanja i stanja neprihvatanja:

  $P_0 = (\{1, 3, 4, 5, 6, E\}, \{2, 7\})$

- Ako unutar jedne particije, stanja za isti ulaz prelaze u isti skup, ta stanja su ekvivalentna.

  Razmatramo P_0 za ulaz *promenljiva*: $P_1 = (\{1, 3\}, \{4, 5, 6, E\}, \{2, 7\})$

  Razmatramo P_1 za ulaz): $P_2 = (\{1, 3\}, \{4, 6, E\}, \{5\}, \{2, 7\})$

  Razmatramo P_2 za ulaz *konstanta*: $P_3 = (\{1, 3\}, \{4, 6\}, \{5\}, \{2, 7\}, \{E\})$

  Razmatramo P_3 za ulaz (: $P_4 = (\{1, 3\}, \{4, 6\}, \{5\}, \{2\}, \{7\}, \{E\})$

Proverom P_4 za sve ulazne simbole zaključujemo da smo dobili konačnu particiju, što znači da su ekvivalentna stanja 1 i 3, kao i 4 i 6.

Uz preimenovanja stanja $\{1, 3\} \equiv A$, $\{4, 6\} \equiv B$, $\{5\} \equiv C$, $\{2\} \equiv D$, $\{7\} \equiv E$ minimalni automat prikazan je na Sl. 1.2.13.

	promenljiva	*konstanta*	(	)	,	□	
A	D					A	0
B		C				B	0
C				D	B	C	0
D			B		A	D	1
E					A	E	1

Sl. 1.2.13: Minimalni automat

Zadatak 1.2.5

a) Konstruisati minimalni deterministički konačni automat koji prepoznaje rimske brojeve od I (1) do LXXXIX (89).

b) Navesti skup rimskih brojeva za koje sva stanja automata dobijenog u tački a) bivaju bar jednom posećena.

Analiza problema

Zadatak se može rešiti odvojenom konstrukcijom automata za jedinice, preciznije za skup brojeva J = {ε, I, II, III, IV, V, VI, VII, VIII i IX} i za dekade, odnosno za skup D = {ε, X, XX, XXX, XL, L, LX, LXX, LXXX}. Traženi skup brojeva R = D × J dobija se kao Dekartov proizvod skupova D i J.

Drugim rečima, svaka od sekvenci koje prepoznaje traženi automat dobija se nadovezivanjem određene sekvence koju prepoznaje automat za jedinice na sekvencu koju prepoznaje automat za dekade. Zadatak 1.3.6 demonstrira postupak konstrukcije automata za prepoznavanje skupa R na osnovu automata za skupove D i J, koji će biti primenjen i u ovom slučaju.

Rešenje

a)

Automat za prepoznavanje skupa jedinica J, prikazan je na Sl. 1.2.14(a) a automat za dekade D, prikazan je na Sl. 1.2.14(b).

	I	V	X	
S_1	I	V		1
I	II	IV	IX	1
II	III			1
III				1
IV				1
V	VI			1
VI	VII			1
VII	VIII			1
VIII				1
IX				1

(a)

	X	L	
S_2	X	L	1
X	XX	XL	1
XX	XXX		1
XXX			1
XL			1
L	LX		1
LX	LXX		1
LXX	LXXX		1
LXXX			1

(b)

Sl. 1.2.14

Na osnovu ovih automata konstruišemo jedinstveni nedeterministički automat (Sl. 1.2.15). Ulazna azbuka novog automata {I, V, X, L} je unija ulaznih azbuka polaznih automata. Skup stanja novog automata odgovara uniji skupova stanja polaznih automata. Funkcija prelaza novog automata definisana je tako da uključuje sve prelaze između stanja koji se pojavljuju u polaznim automatima, pri čemu su dodati prazni prelazi od završnih stanja automata za prepoznavanje dekada ka startnom stanju automata za prepoznavanje jedinica. Startno stanje novog automata odgovara startnom stanju automata za prepoznavanje dekada, a završna stanja odgovaraju završnim stanjima automata za prepoznavanje jedinica.

Standardnim postupkom određivanja determinističkog ekvivalenta automata sa Sl. 1.2.15, dobija se automat na Sl. 1.2.16. Ukoliko za startno stanje ovoga automata uvedemo novu oznaku S, a ostala stanja preimenujemo prema prvim elementima odgovarajućih skupova koji predstavljaju pojedina stanja, pa zatim sprovedemo postupak minimizacije, dobija se automat sa Sl. 1.2.17. Startno stanje naknadno je proglašeno stanjem odbijanja da automat ne bi prihvatao praznu sekvencu.

	I	V	X	L	ε	
→ S_2			X	L	S_1	0
X			XX	XL	S_1	0
XX			XXX		S_1	0
XXX					S_1	0
XL					S_1	0
L			LX		S_1	0
LX			LXX		S_1	0
LXX			LXXX		S_1	0
LXXX					S_1	0
S_1	I	V				1
I	II	IV	IX			1
II	III					1
III						1
IV						1
V	VI					1
VI	VII					1
VII	VIII					1
VIII						1
IX						1

Sl. 1.2.15

	I	V	X	L	
$\rightarrow \{S_2, S_1\}$	{I}	{V}	{X, S_1}	{L, S_1}	1
{I}	{II}	{IV}	{IX}		1
{V}	{VI}				1
{X, S_1}	{I}	{V}	{XX, S_1}	{XL, S_1}	1
{L, S_1}	{I}	{V}	{LX, S_1}		1
{II}	{III}				1
{IV}					1
{IX}					1
{VI}	{VII}				1
{XX, S_1}	{I}	{V}	{XXX, S_1}		1
{XL, S_1}	{I}	{V}			1
{LX, S_1}	{I}	{V}	{LXX, S_1}		1
{III}					1
{VII}	{VIII}				1
{XXX, S_1}	{I}	{V}			1
{LXX, S_1}	{I}	{V}	{LXXX, S_1}		1
{VIII}					1
{LXXX, S_1}	{I}	{V}			1

Sl. 1.2.16

b) Sekvenca LXXXVIII izaziva redom obilazak stanja {S}, {L}, {LX}, {XX, LXX}, {XXX, XL, LXXX}, {V}, {VI}, {II, VII} i {III, IV, VIII, IX} automata sa Sl. 1.2.17.

Sekvenca XIL izaziva redom obilazak stanja {S}, {X}, {I} i stanja greške.

Navedene dve sekvence pokrivaju sva stanja rezultujućeg automata.

	I	V	X	L	
{S}	{I}	{V}	{X}	{L}	0
{I}	{II,VII}	{III,IV,VIII,IX}	{III,IV,VIII,IX}		1
{V}	{VI}				1
{X}	{I}	{V}	{XX,LXX}	{XXX,XL,LXX}	1
{L}	{I}	{V}	{LX}		1
{II,VII}	{III,IV,VIII,IX}				1
{III,IV,VIII,IX}					1
{VI}	{II,VII}				1
{XX,LXX}	{I}	{V}	{XXX,XL,LXXX}		1
{XXX,XL,LXXX}	{I}	{V}			1
{LX}	{I}	{V}	{XX,LXX}		1

Sl. 1.2.17

Zadatak 1.2.6

a) Ispitati da li su automati sa Sl. 1.2.18.(a) i Sl. 1.2.18.(b) ekvivalentni. Ako nisu, navesti sekvencu razlikovanja.

b) Odrediti minimalni ekvivalent automata sa Sl. 1.2.18.(b).

	a	b	c	
→ A		B	C	0
B	B	F		1
C				1
D	D	E		1
E	D	B		0
F	B	B		0

(a)

	a	b	c	
→ P		Q	S	0
Q	R	U		1
R	Q	U		1
S				1
T				1
U	R	R		0

(b)

Sl. 1.2.18

Rešenje

a)

Jedan način rešavanja problema bio bi putem ispitivanja ekvivalencije startnih stanja ova dva automata (videti Zadatak 1.2.2). Ovde ćemo primeniti alternativni način, tako što ćemo oba automata minimizovati, a zatim utvrditi identičnost tako dobijenih minimalnih automata.

Automat (a):

Formiranje skupa dostižnih stanja: {A}.

{A, B, C}.

{A, B, C, F} $\rightarrow$ kraj

Nedostižna su stanja D i E.

Ekvivalentna stanja (particioni metod):

P_0=({A, F}, {B, C})

Delimo particiju P_0 posmatrajući prelaze za ulaz a:

P_1=({A}, {F}, {B}, {C}) $\rightarrow$ kraj

Stanje greške koje nije posebno navedeno nije ekvivalentno ni sa jednim drugim stanjem.

	a	b	c	
A		B	C	0
B	B	F		1
C				1
F	B	B		0

Sl. 1.2.19: minimalni ekvivalent automata (a)

Automat (b):

Dostižna stanja: {P}.

{P, Q, S}.

{P, Q, S, U} $\rightarrow$ kraj

Nedostižno je stanje T.

Ekvivalentna stanja:

P_0=({P, U}, {Q, R, S})

P_1=({P}, {U}, {Q, R}, {S})

P_2=({P}, {U}, {Q, R}, {S}) $\rightarrow$ kraj

Stanja Q i R su ekvivalentna.

	a	b	c	
P		X	S	0
X	X	U		1
S				1
U	X	X		0

Sl. 1.2.20: Minimalni ekvivalent automata (b)

Očigledno je da su ova dva minimalna automata potpuno ista, pri čemu treba uzeti: A = P, B = X, C = S i F = U. Pošto su im minimalni ekvivalentni automati identični, sledi da su i polazni automati međusobno ekvivalentni.

b)

Minimalni ekvivalent automata sa Sl. 1.2.18.(b) određen je u tački a) rešenja i prikazan na Sl. 1.2.19.

1.3. Nedeterministički automati

Zadatak 1.3.1

Opisati postupak kojim se za proizvoljni nedeterministički automat i proizvoljnu sekvencu ulaznih simbola utvrđuje da li je automat prihvata. Demonstrirati postupak za sekvencu abcacbc i automat sa Sl. 1.3.1.

	a	b	c	
→ A	A, B, C	C	E	0
→ B	E	A	E	0
→ C	C, A	C, B	D	1
D	D	D	B, A	0
E	E	E	E	0

Sl. 1.3.1

Analiza problema

Nedeterministički konačni automat opisan je uređenom petorkom (S, U, δ, S_t, P) gde:

- S predstavlja skup stanja automata, u konkretnom slučaju S = {A, B, C, D, E}; stanjima su obeležene vrste tabele prelaza automata prikazane na Sl. 1.3.1;

- U predstavlja skup ulaznih simbola (azbuku automata), u konkretnom slučaju U = {a, b, c}; ulazni simboli obeležavaju kolone tabele prelaza.
- δ: S x (U $\cup$ {ε}) $\rightarrow$ P(S) predstavlja funkciju prelaza (P - partitivni skup); za svako stanje i svaki ulazni simbol definiše skup $S_N \subseteq S$ mogućih novih stanja u koje automat može preći iz stanja $s \in S$ za ulazni simbol $u \in U$ ili za simbol prazne sekvence ε; u konkretnom slučaju, funkcija δ zadata je tabelom prelaza prikazanom na Sl. 1.3.1; u ulaz tabele u vrsti X i koloni Y upisana je vrednost δ(X,Y);
- $S_t \subseteq S$ je skup startnih stanja automata (stanja iz koga može početi rad). U konkretnom slučaju S_t = {A, B, C}; startna stanja automata se po konvenciji označavaju strelicama.
- $P \subseteq S$ predstavlja skup stanja prihvatanja, u konkretnom slučaju P = {C}. Stanja iz skupa S-P nazivaju se stanjima odbijanja. Na Sl. 1.3.1 stanja prihvatanja označena su jedinicom desno od odgovarajuće vrste tabele prelaza, a stanja odbijanja nulom.

Za razliku od determinističkih automata, nedeterministički automati mogu imati veći broj startnih stanja. Takođe, prelazi iz određenog stanja za određeni ulazni simbol mogu biti u jedno od većeg broja mogućih stanja onako kako je definisano funkcijom prelaza δ. Neki nedeterministički automati takođe mogu imati prelaze pod dejstvom simbola ε; pri takvim prelazima tekući ulazni simbol se ne menja.

Nedeterminizam automata odgleda se u tome da je za određenu ulaznu sekvencu moguće izvesti više različitih scenarija promene stanja, počevši od nekog od startnih stanja. Na primer, za dati automat i sekvencu ab neki od mogućih scenarija su:

$$A \xrightarrow{a} A \xrightarrow{b} C$$

$$A \xrightarrow{a} B \xrightarrow{b} A$$

$$A \xrightarrow{a} C \xrightarrow{b} C$$

$$A \xrightarrow{a} C \xrightarrow{b} B$$

$$B \xrightarrow{a} E \xrightarrow{b} E,$$

itd.

Vidimo da je završno stanje u nekim scenarijima stanje prihvatanja C, a u ostalim neko od stanja odbijanja. Po definiciji, nedeterministički automat prihvata ulaznu sekvencu ako je moguće pronaći scenario promene stanja od nekog od početnih stanja do nekog od stanja prihvatanja. U konkretnom slučaju sekvenca ab se prihvata.

Rešenje

Da bismo utvrdili da li nedeterministički automat prihvata određenu sekvencu, moramo voditi evidenciju, u svakom koraku rada automata, o skupu svih mogućih stanja u kojima se automat može nalaziti, prema sledećem algoritmu:

```
tekući_skup_stanja := S_t;
tekući ulaz := prvi simbol ulazne sekvence;
while not (kraj ulazne sekvence)
      novi_skup_stanja := ∅;
      for (∀ s ∈ tekući_skup_stanja)
          novi_skup_stanja := novi_skup_stanja ∪ δ(s, tekući_ulaz);
      end for;
      tekući_skup_stanja := novi_skup_stanja;
      tekući_ulaz := novi simbol ulazne sekvence;
end while;
if (tekući_skup_stanja ∩ P ≠ ∅)
      then ulazna sekvenca se prihvata;
      else ulazna sekvenca se ne prihvata;
end if.
```

Dakle, novi skup stanja je unija svih rezultujućih skupova kod primene funkcije prelaska δ na svako od stanja u tekućem skupu stanja. Ovakva procedura određivanja skupa mogućih stanja primenjuje se za svaki simbol ulaznog niza. Sekvenca se prihvata ukoliko se u završnom skupu stanja nalazi bar jedno stanje prihvatanja.

Za automat sa Sl. 1.3.1 i sekvencu abcacbc formiraju se sledeći skupovi stanja:

$$\{A, B, C\} \xrightarrow{a} \{A, B, C, E\} \xrightarrow{b} \{A, B, C, E\} \xrightarrow{c} \{D, E\} \xrightarrow{a}$$
$$\longrightarrow \{D, E\} \xrightarrow{c} \{A, B, E\} \xrightarrow{b} \{A, C, E\} \xrightarrow{c} \{D, E\}$$

Sekvenca se, prema tome, ne prihvata.

Diskusija

Zadatak se može alternativno rešiti određivanjem ekvivaletnog determinističkog konačnog automata zadatom nedeterminističkom automatu (vidi Zadatak 1.3.2). Prethodna procedura nalaženja skupova stanja se može shvatiti kao određivanje dela funkcije prelaza ekvivalentnog determinističkog automata za konkretnu ulaznu sekvencu.

Zadatak 1.3.2

Odrediti deterministički konačni automat ekvivalentan datom nedeterminističkom automatu (Sl. 1.3.2).

	x	y	z	
→ A	A, D	C	A	0
B	A, B			0
→ C		C, D		1
D	D	C	A	1

Sl. 1.3.2

Analiza problema

Svako stanje determinističkog automata ekvivalentnog polaznom nedeterminističkom automatu je skup onih stanja nedeterminističkog automata u kojima se isti može naći u datom trenutku procesiranja ulazne sekvence. Tabela prelaza determinističkog automata određuje se na sledeći način:

1. Startno stanje determinističkog automata određuje se kao skup startnih stanja nedeterminističkog automata. Obeležiti prvu vrstu tabele ovim stanjem.
2. Izabrati nepopunjenu vrstu tabele označenu stanjem S.
3. S je stanje prihvatanja ako sadrži bar jedno stanje prihvatanja nedeterminističkog automata, u suprotnom radi se o stanju odbijanja.
4. Svaki ulaz u izabranoj vrsti tabele popunjava se na sledeći način: za svaki ulazni simbol u odrediti skup $S' = \bigcup_{s \in S} \delta(s, u)$, gde je δ funkcija prelaza nedeterminističkog automata. Ulaz tabele u vrsti S i koloni u popuniti sa S'. Ako nijedna vrsta tabele prelaza nije označena sa S', označiti novu vrstu tabele prelaska sa S'.
5. Ako postoji nepopunjena vrsta tabele, preći na korak 2. U suprotnom, konstrukcija automata je završena.

Rešenje

Sprovođenjem opisane procedure za dati automat dobija se ekvivalentan deterministički konačan automat prikazan na Sl. 1.3.3(a). Stanju greške odgovara prazan skup stanja nedeterminističkog automata {}. Rezultujući automat uz preimenovana stanja M = {A, C}, N = {A, D}, O = {C, D}, P = {A}, Q = {C} i R = {D} prikazan je na Sl. 1.3.3(b). Prazni prelazi odgovaraju stanju greške.

	x	y	z	
→ {A, C}	{A, D}	{C, D}	{A}	1
{A, D}	{A, D}	{C}	{A}	1
{C, D}	{D}	{C, D}	{A}	1
{A}	{A, D}	{C}	{A}	0
{C}	{}	{C, D}	{}	1
{D}	{D}	{C}	{A}	1
{}	{}	{}	{}	0

(a)

	x	y	z	
→ M	N	O	P	1
N	N	Q	P	1
O	R	O	P	1
P	N	Q	P	0
Q		O		1
R	R	Q	P	1

(b)

Sl. 1.3.3

Zadatak 1.3.3

Projektovati mašinu sa konačnim brojem stanja koja prihvata reči koje se mogu sačiniti od komponenti AN, ANTN, TNN i NF. Ponavljanje komponenti je dozvoljeno ali ne i preklapanje.

Analiza problema

Sastavljene reči iz komponenti znači da se na primer prihvata: ANNF, ANTNTNN, ANANNF. Preklapanja nisu dozvoljena što znači da se ne prihvataju sekvence: ANF, ANTNN, TNNF. Primenićemo metod numeracije simbola, pri čemu ćemo ići na konstrukciju nedeterminističkog automata jer ako bi se radilo sa determinističkim bilo bi jako teško prepoznati npr. ANTNNF – teškoće nastaju jer se ne zna koje su reči prepoznate dok ne stigne poslednje slovo.

Rešenje

Uvedimo stanja nedeterminističkog automata:

ulaz		A N	A N T N	T NN	NF
stanja	S_0	A_1 N_1	A_2 N_2 T_2 N_3	T_3 N_4 N_5	N_6 F_1

U postupku određivanja stanja svaka reč je posmatrana zasebno. Uvodimo S_0 kao jedinstveno početno stanje jer je to lakše za posao konverzije nego da imamo 4 početna stanja. Nedeterministički automat prikazan je na Sl. 1.3.4.(a). Sva stanja prihvatanja ekvivalentna su sa početnim stanjem čime se omogućava ponavljanje reči. Zato je ova stanja moguće zameniti u tabeli početnim stanjem kao na Sl. 1.3.4.(b), čime se uprošćava postupak određivanja ekvivalentnog determinističkog automata (Zadatak 1.3.2).

	A	N	T	F	
$\rightarrow S_0$	A_1, A_2	N_6	T_2		1
A_1		N_1			0
N_1	A_1, A_2	N_6	T_2		1
A_2		N_2			0
N_2			T_1		0
T_1		N_3			0
N_3	A_1, A_2	N_6	T_2		1
T_2		N_4			0
N_4		N_5			0
N_5	A_1, A_2	N_6	T_2		1
N_6				F_1	0
F_1	A_1, A_2	N_6	T_2		1

(a)

	A	N	T	F	
$\rightarrow S_0$	A_1, A_2	N_6	T_2		1
A_1		S_0			0
A_2		N_2			0
N_2			T_1		0
T_1		S_0			0
T_2		N_4			0
N_4		S_0			0
N_6				S_0	0

(b)

Sl. 1.3.4

Ekvivalentni deterministički automat prikazan je na Sl. 1.3.5(a). Dobijeni automat je ujedno i minimalan. Na Sl. 1.3.5(b) prikazan je isti automat uz preimenovana stanja $S_0 = \{S_0\}$, $A_x = \{A_1, A_2\}$, $N_6 = \{N_6\}$, $T_2 = \{T_2\}$, $N_2 = \{S_0, N_2\}$, $N_4 = \{N_4\}$, $T_x = \{T_1, T_2\}$, $N_x = \{S_0, N_4\}$, $N_6 = \{S_0, N_6\}$ i praznim ulazima koji odgovaraju stanju greške {}.

	A	N	T	F	
$\rightarrow \{S_0\}$	$\{A_1, A_2\}$	$\{N_6\}$	$\{T_2\}$	{}	1
$\{A_1, A_2\}$	{}	$\{S_0, N_2\}$	{}	{}	0
$\{N_6\}$	{}	{}	{}	$\{S_0\}$	0
$\{T_2\}$	{}	$\{N_4\}$	{}	{}	0
{}	{}	{}	{}	{}	0
$\{S_0, N_2\}$	$\{A_1, A_2\}$	$\{N_6\}$	$\{T_1, T_2\}$	{}	1
$\{N_4\}$	{}	$\{S_0\}$	{}	{}	0
$\{T_1, T_2\}$	{}	$\{S_0, N_4\}$	{}	{}	0
$\{S_0, N_4\}$	$\{A_1, A_2\}$	$\{S_0, N_6\}$	$\{T_2\}$	{}	1
$\{S_0, N_6\}$	$\{A_1, A_2\}$	$\{N_6\}$	$\{T_2\}$	$\{S_0\}$	1

(a)

	A	N	T	F	
$\rightarrow S_0$	A_x	N_6	T_2		1
A_x		N_2			0
N_6				S_0	0
T_2		N_4			0
N_2	A_x	N_6	T_x		1
N_4		S_0			0
T_x		N_x			0
N_x	A_x	N_6	T_2		1
N_6	A_x	N_6	T_2	S_0	1

(b)

Sl. 1.3.5

Zadatak 1.3.4

Odrediti minimalni konačni deterministički automat koji prihvata one i samo one ulazne sekvence koje NE prihvata nedeterministički konačni automat sa slike Sl. 1.3.6, sa startnim stanjima A i B.

	a	b	
$\rightarrow$ A	C	E	0
$\rightarrow$ B	B	D	1
C	E	E	1
D	E	F	0
E	E	E	0
F	F	F	1

Sl. 1.3.6

Rešenje

Prevođenjem zadatog nedeterminističkog konačnog automata u deterministički dobijamo automat prikazan na Sl. 1.3.7(a). Isti automat sa preimenovanim stanjima prikazan je na slici Sl. 1.3.7(b).

	a	b	
→ M={A,B}	{B,C}	{D,E}	1
N={B,C}	{B,E}	{D,E}	1
O={D,E}	{E}	{E,F}	0
P={B,E}	{B,E}	{D,E}	1
Q={E}	{E}	{E}	0
R={E,F}	{E,F}	{E,F}	1

(a)

	a	b	
→ M	N	O	1
N	P	O	1
O	Q	R	0
P	P	O	1
Q	Q	Q	0
R	R	R	1

(b)

Sl. 1.3.7

Skup dostižnih stanja automata sa Sl. 1.3.7(b) je {M,N,O,P,Q,R}, dakle nema nedostižnih stanja. Sprovodimo minimizaciju particionim metodom:

$$P_0 = (\{M,N,P,R\},\{O,Q\})$$

Deljenjem particije P_0 za ulaz b dobijamo P_1:

$$P_1 = (\{M,N,P\},\{R\},\{O,Q\})$$

Ispitivanjem za sve ulazne simbole zaključujemo da dalje deljenje nije moguće. Stanja M, N i P su ekvivalentna. i u minimalnom automatu na Sl. 1.3.8 ova stanja predstavljena su stanjem X.

	a	b	
→ X	X	O	1
O	Q	R	0
Q	Q	Q	0
R	R	R	1

Sl. 1.3.8

Dobijeni automat prihvata isti skup sekvenci kao i zadati automat. Da bismo dobili automat koji prihvata komplement toga skupa, odnosno koji prihvata sve sekvence koje zadati automat odbija, a odbija sve sekvence koje zadati automat prihvata, stanja prihvatanja proglašavaju se stanjima odbijanja i obrnuto, čime dobijamo automat prikazan na Sl. 1.3.9.

	a	b	
→ X	X	O	0
O	Q	R	1
Q	Q	Q	1
R	R	R	0

Sl. 1.3.9

Diskusija rešenja

Koraci 2 i 3 mogu se međusobno zameniti ali ne i koraci 1 i 3. Proglašavanjem stanja prihvatanja za stanja odbijanja i obrnuto kod nedeterminističkog automata ne bi se postiglo komplementiranje skupa sekvenci koje se prihvataju.

Zadatak 1.3.5

a) Opisati ulazne skupove S_1 i S_2 koje prihvataju automati (a) i (b) sa slike Sl. 1.3.10. Projektovati nedeterministički automat koji prihvata skup $S_1 \cup S_2$.

b) Pronaći minimalni deterministički automat koji odgovara dobijenom nedeterminističkom automatu.

	0	1	
→ A_1	C_1	B_1	1
B_1	B_1	C_1	1
C_1	C_1	C_1	0

(a)

	0	1	
→ A_2	D_2	B_2	0
B_2	C_2	C_2	0
C_2	D_2	D_2	1
D_2	D_2	D_2	0

(b)

Sl. 1.3.10

Analiza problema

Opštija formulacija problema glasi: dokazati da je skup regularnih jezika (odnosno jezika koji mogu biti opisani konačnim automatima) zatvoren u odnosu na operaciju unije. Drugim rečima, potrebno je dokazati da je jezik L_D koji se dobija operacijom unije proizvoljna dva regularna jezika L_1 i L_2, takođe regularan. Operacija unije jezika, svodi se na uniju skupova, s obzirom da su jezici skupovi sekvenci.

Dokaz sprovodimo konstrukcijom, na osnovu zadatih automata, rezultujućeg automata čiji jezik predstavlja uniju jezika zadatih automata.

Procedura konstrukcije je opšta i ne zavisi od konkretnih automata:

- Skup ulaznih simbola novog automata je unija skupa ulaznih simbola polaznih automata.
- Skup stanja novog automata je unija skupova stanja polaznih automata.
- Funkcija prelaza novog automata sadrži sve prelaze koji postoje među stanjima polaznih automata.
- Skup startnih stanja novog automata je unija skupova startnih stanja polaznih automata, u konkretnom slučaju radi se o skupu $\{A_1, A_2\}$ što čini novi automat nedeterminističkim.
- Skup stanja prihvatanja novog automata je unija skupova stanja prihvatanja polaznih automata.

Nije teško utvrditi da novi automat prihvata sve sekvence koje prihvata bilo koji od datih automata i nijednu drugu sekvencu, čime se dokazuje zatvorenost operacije unije nad skupovima jezika konačnih automata.

Rešenje

a)

Rezultujući nedeterministički automat prikazan je na Sl. 1.3.11.(a).

b)

Za dobijeni automat određujemo ekvivalentan deterministički automat (videti Zadatak 1.3.2), prikazan na Sl. 1.3.11.(b).

	0	1	
$\rightarrow A_1$	C_1	B_1	1
B_1	B_1	C_1	1
C_1	C_1	C_1	0
$\rightarrow A_2$	D_2	B_2	0
B_2	C_2	C_2	0
C_2	D_2	D_2	1
D_2	D_2	D_2	0

(a)

	0	1	
$\rightarrow \{A_1, A_2\}$	$\{C_1, D_2\}$	$\{B_1, B_2\}$	1
$\{C_1, D_2\}$	$\{C_1, D_2\}$	$\{C_1, D_2\}$	0
$\{B_1, B_2\}$	$\{B_1, C_2\}$	$\{C_1, C_2\}$	1
$\{B_1, C_2\}$	$\{B_1,D_2\}$	$\{C_1, D_2\}$	1
$\{C_1, C_2\}$	$\{C_1, D_2\}$	$\{C_1, D_2\}$	1
$\{B_1,D_2\}$	$\{B_1,D_2\}$	$\{C_1, D_2\}$	1

(b)

Sl. 1.3.11

Uz preimenovana stanja i uklonjeno mrtvo stanje $\{C_1, D_2\}$ dobija se automat na Sl. 1.3.12.(a), a njegovom minimizacijom (stanja D i F su ekvivalentna) automat na Sl. 1.3.12.(b).

	0	1	
→ A	B	C	1
B	B	B	0
C	D	E	1
D	F	B	1
E	B	B	1
F	F	B	1

(a)

	0	1	
→ A	B	C	1
B	B	B	0
C	D	E	1
D	D	B	1
E	B	B	1

(b)

Sl. 1.3.12

Zadatak 1.3.6

Konstruisati minimalni deterministički konačni automat koji prihvata sve i samo one sekvence oblika s_1s_2 gde je s_1 sekvenca koju prihvata automat sa Sl. 1.3.13.(a), a s_2 sekvenca koju prihvata automat sa Sl. 1.3.13.(b).

	a	b	
→ 0	1		1
1	2	1	0
2		0	1

(a)

	b	c	
→ A	B	C	0
B	C		1
C		A	0

(b)

Sl. 1.3.13

Analiza problema

Opštija formulacija problema glasi: dokazati da je skup regularnih jezika (odnosno jezika koji mogu biti opisani konačnim automatima) zatvoren u odnosu na operaciju konkatenacije. Drugim rečima, potrebno je dokazati da je jezik L_D koji se dobija operacijom konkatenacije proizvoljna dva regularna jezika L_1 i L_2, takođe regularan. Operacija konkatenacije nad jezicima, kao skupovima sekvenci, definiše se kao Dekartov proizvod tih skupova, kao što je u postavci zadatka navedeno.

Dokaz sprovodimo konstrukcijom, na osnovu zadatih automata, rezultujućeg automata čiji jezik predstavlja konkatenaciju jezika zadatih automata. Procedura konstrukcije je opšta i ne zavisi od konkretnih automata.

U cilju dobijanja traženog determinističkog automata formiraćemo prvo nedeterministički automat koji prihvata traženi skup sekvenci:

- Skup ulaznih simbola novog automata je unija skupa ulaznih simbola polaznih automata.

- Skup stanja novog automata je unija skupova stanja polaznih automata.
- Funkcija prelaza novog automata sadrži sve prelaze koji postoje među stanjima polaznih automata. Dodatno, od stanja u novom automatu koja odgovaraju stanjima prihvatanja automata sa Sl. 1.3.13.(a) postoje prazni prelazi, odnosno prelazi za ulaznu sekvencu ε dužine nula simbola, do stanja novog automata koje odgovara startnom stanju automata sa Sl. 1.3.13.(b). Zbog postojanja praznih prelaza novi automat je nedeterministički.
- Startno stanje novog automata odgovara startnom stanju automata sa Sl. 1.3.13.(a).
- Stanja prihvatanja novog automata odgovaraju stanjima prihvatanja automata sa Sl. 1.3.13.(b).

Nije teško utvrditi da ovako formirani automat prihvata sve i samo one sekvence koje pripadaju jeziku L_D čime se ujedno dokazuje zatvorenost skupa regularnih jezika u odnosu na operaciju konkatenacije. Dokaz je naveden u okviru rešenja zadatka.

Da bismo rešili zadatak, potrebno je generalnu proceduru konstrukcije nedeterminističkog automata primeniti na date automate, odrediti zatim ekvivalentan deterministički automat i na njega primeniti postupak minimizacije da bismo dobili konačno rešenje. Postupak dobijanja tabele prelaza determinističkog ekvivalenta za dobijeni nedeterministički automat (Zadatak 1.3.2) modifikuje se usled postojanja ε–prelaza kod nedeterminističkog automata tako da glasi:

1. Startno stanje determinističkog automata određuje se kao ε–zatvaranje skupa startnih stanja nedeterminističkog automata. Obeležiti prvu vrstu tabele ovim stanjem. Operacija ε–zatvaranja definisana je u nastavku.
2. Izabrati nepopunjenu vrstu tabele označenu stanjem S.
3. S je stanje prihvatanja ako sadrži bar jedno stanje prihvatanja nedeterminističkog automata, u suprotnom radi se o stanju odbijanja.
4. Svaki ulaz u izabranoj vrsti tabele popunjava se na sledeći način: za svaki ulazni simbol u odrediti skup $S' = \bigcup_{s \in S} \delta(s,u)$, gde je δ funkcija prelaza nedeterminističkog automata. Ulaz tabele u vrsti S i koloni u popuniti sa skupom koji se dobija ε–zatvaranjem skupa S'. Ako nijedna vrsta tabele prelaza nije označena sa S', označiti novu vrstu tabele prelaza sa S'.
5. Ako postoji nepopunjena vrsta tabele, preći na korak 2. U suprotnom, konstrukcija automata je završena.

Operacija ε–zatvaranja glasi:

```
Ulaz: skup stanja T, izlaz: skup stanja ε–closure(T);
for (∀ stanje t ∈ T )
        staviti t na stek;
end for;
ε–closure(T) := T;
while (stek nije prazan)
        Skinuti vršni element t sa steka;
        for (∀ stanje u za koje postoji ε–prelaz od t ka u)
                ε–closure(T) := ε–closure(T) ∪ {u};
```

```
                    staviti u na stek;
                end for;
            end while
```

Rešenje

Tabela prelaza rezultujućeg nedeterminističkog automata prikazana je na Sl. 1.3.14. Startno stanje novog automata je startno stanje prvog automata, a stanje prihvatanja je stanje prihvatanja drugog automata.

	a	b	c	ε	
→ 0	1			A	0
1	2	1			0
2		0		A	0
A		B	C		0
B		C			1
C			A		0

Sl. 1.3.14

Da bismo dokazali da ovako dobijen automat zadovoljava postavku zadatka, treba dokazati da rezultujući automat prihvata tačno skup sekvenci $L_D = L_1 \times L_2$ (Dekartov proizvod, u skladu sa definicijom operacije konkatenacije), gde je L_1 skup sekvenci koje prihvata prvi automat, a L_2 skup sekvenci koje prihvata drugi automat.

Dokažimo prvo da dobijeni automat prihvata sve sekvence iz skupa L_D. Pretpostavimo prvo da je $s \in L_D$ proizvoljna sekvenca. Tada postoje podsekvence s' i s'' tako da važi $s = s's''$ i $s' \in L_1$ i $s'' \in L_2$, što će reći da prvi automat prihvata sekvencu s', a drugi sekvencu s''. Ukoliko sekvencu s's'' dovedemo na ulaz novog automata, posle obrade prefiksa s' automat se može naći ili u stanju 0 ili u stanju 2, s obzirom da su prelazi jednako definisani kao kod prvog automata, a da su navedena stanja kod prvog automata stanja prihvatanja. Iz svakog od ovih stanja postoji prazan prelaz ka stanju A novog automata. Od ovog stanja obrada preostalog dela ulaza s'' odvija se na isti način kao kod drugog automata, tako da je završno stanje – stanje prihvatanja B, čime je dokazano da rezultujući automat prihvata sekvencu s.

Da bismo dokazali da automat ne prihvata sekvence koje nisu u skupu L_D, pretpostavimo suprotno, dakle da postoji sekvenca s koju automat prihvata i $s \notin L_D$. U procesu prepoznavanja ove sekvence, rezultujući automat mora proći kroz stanje A jer ne postoji drugi način da se stigne u stanje prihvatanja B. Ovo znači da sekvencu s možemo podeliti na dve podsekvence s' i s'', pri čemu se automat posle obrade s' prvi put zatiče u stanju A. U ovo stanje može se doći isključivo iz stanja 0 ili 2 primenom praznog prelaza, pa sledi da automat posle obrade s' dolazi u jedno od ta dva stanja. Ovo neposredno znači da bi prvi automat prihvatao sekvencu s', a drugi sekvencu s'', što će reći da sekvenca pripada skupu L_D, što protivreči polaznoj pretpostavci. Ovim je završen dokaz da rezultujući automat prihvata tačno skup L_D.

Deterministički ekvivalent dobijenog automata prikazan je na Sl. 1.3.15(a). Dobijeni automat je ujedno i minimalan. Isti automat sa preimenovanim stanjima, X = {0, A}, 1 = {1}, B = {B}, C = {C}, Y = {2, A}, A = {A}, Z = {0, A, B}, V = {B, C}, prikazan je na Sl. 1.3.15(b).

	a	b	c	
→ {0, A}	{1}	{B}	{C}	0
{1}	{2, A}	{1}		0
{B}		{C}		1
{C}			{A}	0
{2, A}		{0, A, B}	{C}	0
{A}		{B}	{C}	0
{0, A, B}	{1}	{B, C}	{C}	1
{B, C}		{C}	{A}	1

(a)

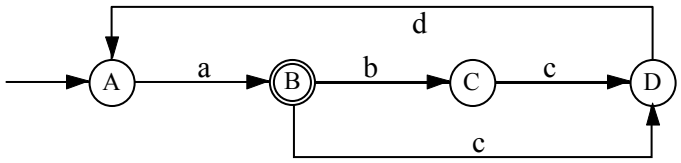
	a	b	c	
→ X	1	B	C	0
1	Y	1		0
B		C		1
C			A	0
Y		Z	C	0
A		B	C	0
Z	1	V	C	1
V		C	A	1

(b)

Sl. 1.3.15

Zadatak 1.3.7

Konstruisati minimalni deterministički konačni automat koji prihvata iste sekvence koje prihvata i automat sa Sl. 1.3.16, ali čitane s desna u levo.



Sl. 1.3.16

Analiza problema

Razmotrimo automat koji ima graf prelaza iste strukture kao i zadati automat, pri čemu sve grane imaju obrnut smer. U opštem slučaju, radi se o nedeterminističkom automatu koji ima istoimena stanja kao i zadati automat i pri tome:

1. Startna stanja novog automata odgovaraju stanjima prihvatanja datog automata.
2. Stanje prihvatanja novog automata odgovara startnom stanju datog automata.
3. Ako dati automat ima prelaz iz nekog stanja X za ulazni simbol y u neko stanje Z, tada i samo tada novi automat ima prelaz iz stanja Z u stanje X za ulazni simbol y.

Nije teško utvrditi da, ako neka ulazna sekvenca $u_1u_2..u_n$ u datom automatu inicira niz promena stanja $S_0 \xrightarrow{u_1} S_1 \xrightarrow{u_2} \ldots \xrightarrow{u_n} S_n$, tada i samo tada sekvenca $u_n...u_2u_1$ u novom automatu inicira niz promena stanja $S_n \xrightarrow{u_n} ... \xrightarrow{u_2} S_1 \xrightarrow{u_1} S_0$, iz čega se može izvesti zaključak da automat ispunjava uslove zadatka.

Rešenje

Ukoliko primenimo opisanu proceduru na zadati automat, dobija se nedeterministički automat čija je tabela prelaza prikazana na Sl. 1.3.17.

	a	b	c	d	
A				D	1
→ B	A				0
C		B			0
D			B, C		0

Sl. 1.3.17

Deterministički ekvivalent automata sa Sl. 1.3.17 dat je na Sl. 1.3.18(a). Dobijeni automat je minimalan. Isti automat sa preimenovanim stanjima, A = {A}, B = {B}, D = {D}, X = {B, C} prikazan je na Sl. 1.3.18(b).

	a	b	c	d	
→ {B}	{A}				0
{A}				{D}	1
{D}			{B, C}		0
{B, C}	{A}	{B}			0

(a)

	a	b	c	d	
→ B	A				0
A				D	1
D			X		0
X	A	B			0

(b)

Sl. 1.3.18

1.4. Regularni izrazi

Zadatak 1.4.1

Zadata su tri regularna izraza RI 1, RI 2 i RI 3:

RI 1: a* RI 2: a|b RI 3: abc

a) Konstruisati minimalni deterministički konačni automat koji prihvata one i samo one sekvence koje su opisane bar jednim od datih izraza.

b) Na pseudojeziku napisati algoritam zasnovan na automatu iz tačke a) koji prepoznaje najduži prefiks ulazne sekvence koji prihvata automat iz tačke a) i na izlazu daje broj regularnog izraza koji opisuje taj prefiks. (Ako isti prefiks opisuju dva regularna izraza, na izlazu se navodi regularni izraz sa manjim rednim brojem).

Analiza problema

Regularni izraz *s* nad ulaznom azbukom Σ opisuje regularni skup L(*s*) nizova simbola ulazne azbuke Σ. Regularni izrazi se definišu na sledeći način:

- ∅ je regularan izraz koji opisuje prazan skup L(∅)=∅.
- ε je regularan izraz koji opisuje skup {ε} L(ε)={ε}.
- ulazni simbol $n \in \Sigma$ je regularni izraz koji opisuje skup {*n*} L(*n*)={*n*}.
- složeniji regularni izrazi dobijaju se od regularnih izraza A i B primenom operacija:
  - unije, oznaka A|B, L(A|B) = L(A) ∪ L(B) = {x | x ∈ A ili x ∈ B}
  - konkatenacije (nadovezivanja) AB ili A·B
    L(AB)=L(A) x L(B)={xy | x ∈ A, y ∈ B} (dekartov proizvod skupova)
    Konvencija: $A^n \equiv \underbrace{A \cdot A \cdot \ldots \cdot A}_{n}$, $A^0 \equiv \{\varepsilon\}$
  - zvezdastog zatvaranja A^*
    $A^*=A^0|A^1|A^2|A^3|\ldots$ (beskonačan broj članova)
  - pozitivnog zatvaranja A^+
    $A^+ = A^1|A^2|A^3|\ldots$

Najviši prioritet imaju operacije $^+$ i *, zatim · pa |.

Primeri regularnih izraza:

L(abc)={abc} L(a | b | c)={a, b, c}

L[(a | b)(c | d)]={ac, ad, bc, bd} male zagrade su obavezne zbog prioriteta

$L(a^*)$={ε, a, aa, aaa, …}

$L[(a \mid b)^+]$={a, b} ∪ {a, b} x {a, b} ∪ {a, b} x {a, b} x {a, b} ∪ …

={a,b} ∪ {aa, ab, ba, bb} ∪ {aaa, aab, aba, abb, baa, bab, bba, bbb} ∪...

={a, b, aa, ab, ba, bb, …} = svi mogući nizove slova a i b dužine 1 ili više

Regularni izraz slovo =A | B |…| Z | a | b |…| z

Cifra = 0 | 1 |…|9

Identifikator = Slovo (Slovo | Cifra)*

Rešenje

Prvo ćemo konstruisati tri posebna automata (za svaki izraz po jedan), na osnovu njih konstruisati nedeterministički automat koji prihvata uniju jezika pojedinačnih automata (vidi Zadatak 1.3.5), pa njega pretvoriti u deterministički. Prvi automat (Sl. 1.4.1) prihvata praznu sekvencu, jedno ili više slova a, drugi automat (Sl. 1.4.2) prihvata slovo a ili slovo b, a treći (Sl. 1.4.3) prihvata sekvencu abc.

	a	b	c	
→ A	A			1

Sl. 1.4.1: Automat za izraz a*

	a	b	c	
→ B	C	C		0
C				1

Sl. 1.4.2: Automat za izraz a|b

	a	b	c	
→ E	F			0
F		G		0
G			H	0
H				1

Sl. 1.4.3: Automat za izraz abc

Nedeterministički automat za uniju jezika (vidi Zadatak 1.3.5) prikazan na slici Sl. 1.4.4. Stanje greške izostavljeno je iz tabele automata. Ekvivalentni deterministički automat prikazan je na slici Sl. 1.4.5.

		a	b	c	
→	A	A			1
→	B	C	C		0
	C				1
→	E	F			0
	F		G		0
	G			H	0
	H				1

Sl. 1.4.4

		a	b	c	
1.	{A,B,E}	{A,C,F}	{C}	{}	1
2.	{A,C,F}	{A}	{G}	{}	1
3.	{C}	{}	{}	{}	1
4.	{A}	{A}	{}	{}	1
5.	{G}	{}	{}	{H}	0
6.	{H}	{}	{}	{}	1
7.	{}	{}	{}	{}	0

Sl. 1.4.5

Sl. 1.4.6 prikazuje isti automat, sa preimenovanim stanjima. Ovaj automat nema nedostižnih stanja, ali ima ekvivalentnih. Očigledno je da su stanja 3. i 6. ekvivalentna. Radimo metodom particija:

P_0=({1, 2, 3, 4, 6}, {5, 7}), delimo za ulaz a,

P_1=({1, 2, 4}, {3, 6}, {5, 7}), delimo za ulaz c,

P_2=({1, 2, 4}, {3, 6}, {5}, {7}), delimo za ulaz b

P_3=({1}, {2}, {4}, {3, 6}, {5}, {7}), utvrđujemo da je podela konačna.

	a	b	c	
1	2	3	7	1
2	4	5	7	1
3	7	7	7	1
4	4	7	7	1
5	7	7	6	0
6	7	7	7	1
7	7	7	7	0

Sl. 1.4.6

Znači samo su stanja 3 i 6 ekvivalentna. Stanje 6 izbacujemo iz tabele prelaza, zamenjujući prelaze u to stanje prelazima u stanje 3 (Sl. 1.4.7). Ovo je minimalni deterministički automat koji prihvata one i samo one sekvence koje su opisane bar jednim od datih regularnih izraza.

	a	b	c	
1	2	3	7	1
2	4	5	7	1
3	7	7	7	1
4	4	7	7	1
5	7	7	3	0
7	7	7	7	0

Sl. 1.4.7

b)

Kao osnova za traženi algoritam može da posluži algoritam za određivanje najdužeg prefiksa ulazne sekvence koji se prihvata (videti Zadatak 1.1.6). Pomenuti algoritam modifikujemo tako da pored pozicije u ulaznom nizu pamti i poslednje stanje prihvatanja kroz koje je automat prošao tokom rada. Za utvrđivanje kojem regularnom izrazu odgovara određeno stanje prihvatanja, razmatramo odgovarajući skup stanja nedeterminističkog automata sa Sl. 1.4.5:

- stanju 1 odgovara skup {A, B, E}. Sada gledamo determinističke automate koji odgovaraju pojedinim izrazima. Stanje A je stanje prihvatanja automata za RI 1, dok su stanja B i E stanja odbijanja. Znači, stanje 1 odgovara prihvatanju sekvence opisane izrazom RI 1.
- stanju 2 odgovara skup {A, C, F}. Stanje A je stanje prihvatanja automata za RI 1, stanje C je stanje prihvatanja automata za RI 2, a F je stanje odbijanja. Znači, stanje 2 odgovara prihvatanju sekvenci opisanih kako sa RI 1 tako i sa RI 2, pa biramo RI 1 prema uslovu zadatka.
- stanju 3 (koje je ekvivalentirano sa stanjem 6 pri minimizaciji) odgovaraju skupovi {C} i {H}. Na osnovu stanja C sledi izbor izraza RI 2, a na osnovu H izraza RI 3. Ovde bismo se prema uslovu zadatka mogli odlučiti za RI 2, ali to ne bi bilo korektno jer na primer za ulaznu sekvencu abc dobili bismo da je opisana izrazom RI 2. Ovo je rezultat postupka minimizacije (koji spaja stanja na osnovu istog ponašanja automata u nastavku sekvence, a ne na osnovu scenarija dolaska iz startnog stanja). Ovaj problem rešavamo na taj način što za osnovu algoritma uzimamo neminimizovan automat sa Sl. 1.4.6, sa razdvojenim stanjima 3 i 6. Sada je korektno ustvrditi da stanju 3 odgovara izraz RI 2.
- stanju 6, u skladu sa prethodnim razmatranjem , odgovara izraz RI 3.
- stanju 4 odgovara skup {A} i izraz RI 1.

Sledi modifikovani algoritam.

```
tekuća_pozicija := 0;
pozicija_prihvatanja := -1;
stanje_prihvatanja := -1;
tekuće_stanje := S_t
tekući_ulaz := prvi simbol ulazne sekvence;
while not (kraj ulazne sekvence ili tekuće_stanje = stanje_greške)
```

```
            if ( tekuće_stanje ∈ P )
                    then  pozicija_prihvatanja := tekuća_pozicija;
                          stanje_prihvatanja := tekuće_stanje;
            end if;
            tekuće_stanje := δ( tekuće_stanje, tekući ulaz );
            tekući_ulaz := sledeći simbol ulazne sekvence;
            tekuća_pozicija := tekuća_pozicija + 1;
    end while;
    if ( pozicija_prihvatanja = -1 )
            then  ne postoji traženi prefiks jer nije bilo prolaska kroz stanje prihvatanja;
            else  if ( pozicija_prihvatanja = 0 )
                        then  traženi prefiks je prazna sekvenca;
                        else  traženi prefiks je deo ulazne sekvence od 1.
                              pozicije do pozicije_prihvatanja;
                  end if;
    end if;
    if ( stanje_prihvatanja ∈ {1, 2, 4} )
            then  prefiks je opisan sa RI 1;
            else  if ( stanje_prihvatanja = 3 )
                        then  prefiks je opisan sa RI 2;
                        else  if ( stanje_prihvatanja = 6 )
                                    then prefiks je opisan sa RI 3.
                              end if;
                  end if;
    end if;
```

Diskusija

Zadatak ilustruje koncept funkcionisanja programskog alata *lex* – generatora leksičkog analizatora, opisanog u Prilogu 5.1. Ulaz u *lex* je definiciona datoteka za nizom regularnih izraza i, za svaki od izraza, programskom akcijom koja se preduzima kada se u ulaznom nizu prepozna sekvenca koja je opisana izrazom. Prepoznavanje izraza podleže pravilima iz postavke ovoga zadatka. Alat *lex* generiše konačni automat za prepoznavanje unije izraza i na izlazu daje programsku implementaciju automata i gore navedenog algoritma.

Zadatak 1.4.2

a) Napisati regularni izraz koji opisuje označene decimalne konstante. Primeri dozvoljenih konstanti su: 1257, +0**.**0392, -12345**.**36, 2**.**0 a nedozvoljenih: **.**123, -23**.**, +, **.** Napomena: U regularnom izrazu koristiti simbole +, -, **.** i *d* (simbol *d* predstavlja proizvoljnu cifru).

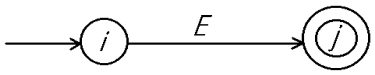
b) Konstruisati nedeterministički konačni automat koji odgovara regularnom izrazu dobijenom u tački a).

c) Konstruisati minimalni deterministički konačni automat ekvivalentan automatu dobijenom u tački b).

Analiza problema

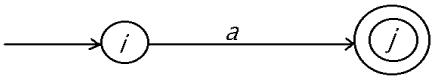
Tompsonov algoritam definiše sledeća pravila za konstrukciju nedeterminističkog konačnog automata (NKA) za dati regularni izraz:

- regularnom izrazu ε odgovara sledeći automat (Sl. 1.4.8):



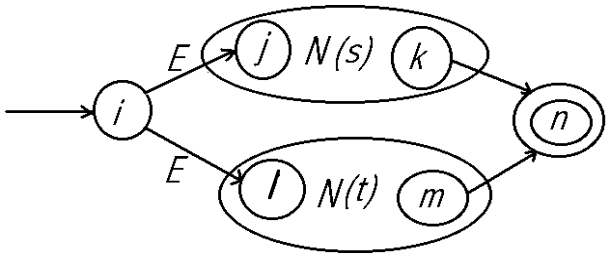
Sl. 1.4.8

- regularnom izrazu a, gde a ∈ Σ (to jest, a je simbol ulazne azbuke), Sl. 1.4.9:



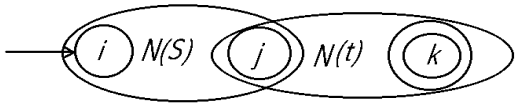
Sl. 1.4.9

- ako je *N(s)* NKA za regularni izraz *s*, *N(t)* NKA za regularni izraz *t*, onda se NKA za uniju *s|t* formira na sledeći način, Sl. 1.4.10.



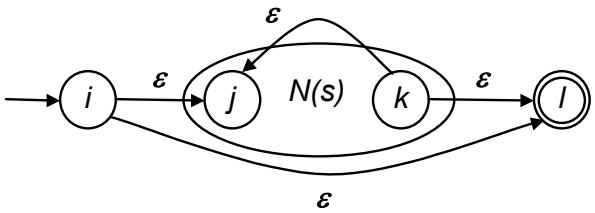
Sl. 1.4.10

- ako je *N(s)* NKA za regularni izraz *s*, *N(t)* NKA za regularni izraz *t*, onda se NKA za konkatenaciju *st* formira na sledeći način, Sl. 1.4.11:



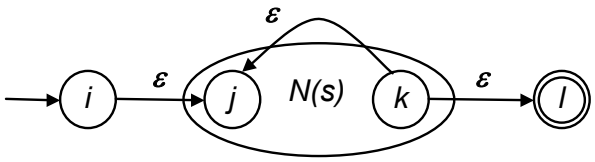
Sl. 1.4.11

- ako je *N(s)* NKA za regularni izraz *s*, onda se NKA za zvezdasto zatvaranje s^* formira na sledeći način, Sl. 1.4.12:



Sl. 1.4.12

- ako je *N(s)* NKA za regularni izraz *s*, onda se NKA za pozitivno zatvaranje s^+ formira na sledeći način, Sl. 1.4.13:



Sl. 1.4.13

Rešenje

a)

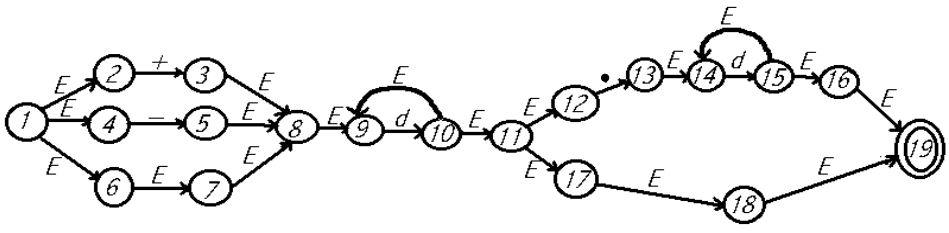
Sledeći regularni izraz zadovoljava uslove postavke:

$(+ \mid - \mid \varepsilon)\ d^+ (\ .d^+ \mid \varepsilon)$

Predznak je opcioni, decimale su takođe opcione, ali ako je decimalna tačka prisutna onda sa obe strane mora postojati bar po jedna cifra.

b)

Regularnom izrazu iz tačke a) odgovara nedeterministički automat prikazan na Sl. 1.4.14.



Sl. 1.4.14

c)

Nedeterminističkom automatu iz tačke b) odgovara deterministički automat prikazan na Sl. 1.4.15. Zadatak 1.3.6 definiše proceduru konstrukcije ovog automata, uzimajući u obzir i postojanje ε prelaza između pojedinih stanja nedeterminističkog automata.

	+	-	*	d	
A={1,2,4,6,7,8,9}	{3,8,9}	{5,8,9}	{}	{10,9,11,12,17,18,19}	0
B={3,8,9}	{}	{}	{}	{10,9,11,12,17,18,19}	0
C={5,8,9}	{}	{}	{}	{10,9,11,12,17,18,19}	0
D={}	{}	{}	{}	{}	0
E={9,10,11,12,17,18,19}	{}	{}	{13,14}	{10,9,11,12,17,18,19}	1
F={13,14}	{}	{}	{}	{15,14,16,19}	0
G={14,15,16,19}	{}	{}	{}	{15,14,16,19}	1

Sl. 1.4.15

Sl. 1.4.16 prikazuje isti automat sa preimenovanim stanjima. Stanja prihvatanja su E i G jer sadrže završno stanje 19 nedeterminističkog automata. Prazni ulazi odgovaraju prelazima u stanje greške D koje je po konvenciji izostavljeno.

	+	-	*	d	
A	B	C		E	0
B				E	0
C				E	0
E			F	E	1
F				G	0
G				G	1

Sl. 1.4.16

Eliminacija suvišnih (nedostižnih) stanja:

Dostižna stanja = {A,B,C,E,F,G}

Nema suvišnih stanja. Sledi minimizacija (traženje ekvivalentnih stanja) prema particionom algoritmu.

Početna particija ima sledeći izgled: $P_0=(\{A,B,C,F\},\{E,G\})$

Razmatranjem prelaza za d: $P_1=(\{A,B,C,F\},\{E,G\})$

Razmatranjem prelaza za +: $P_2=(\{A\},\{B,C,F\},\{E,G\})$

Razmatranjem prelaza za *: $P_3=(\{A\},\{B,C,F\},\{E\},\{G\})$

Razmatranjem prelaza za d: $P_4=(\{A\},\{B,C\},\{F\},\{E\},\{G\})$

Razmatranjem svih ulaznih simbola zaključujemo da je P_4 konačna particija. Stanja B i C su međusobno ekvivalentna. Minimalan automat prikazan je na slici Sl. 1.4.17. Stanje X reprezentuje stanja B i C.

	+	-	*	d	
A	X	X		E	0
X				E	0
E			F	E	1
F				G	0
G				G	1

Sl. 1.4.17

Zadatak 1.4.3

a) Nacrtati sintaksno stablo regularnog izraza (A | (BC)*)D⊣

b) Za dati izraz izračunati funkcije *prva pozicija, poslednja pozicija, poništiva pozicija* i *sledeća pozicija.*

c) Na osnovu funkcije *sledeća pozicija* konstruisati deterministički konačni automat.

d) Minimizovati dobijeni automat.

a)

Analiza problema

Čvorovi sintaksnog stabla predstavljaju operatore i operande koji se pojavljuju u izrazu. Kod izraza a OP b, gde je OP proizvoljan binarni operator, u stablo se unosi čvor OP sa levim naslednikom a i desnim naslednikom b. Unarni operatori imaju jednog naslednika – odgovarajući operand.

Rešenje

Stablo za dati izraz prikazano je na Sl. 1.4.18.

b)

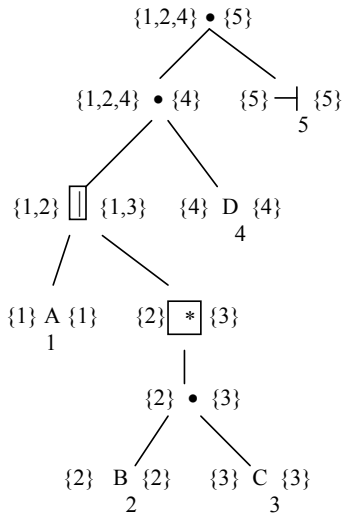
Analiza problema

Pozicijama se označavaju listovi sintaksnog stabla (osim ε) redom pri obilasku stabla sa leva u desno. Na Sl. 1.4.18 pozicije su naznačene ispod listova.

Čvor sintaksnog stabla je poništiv ako podizraz sa korenom u datom čvoru može generisati praznu sekvencu. Računanje poništivosti:

1. Listovi nisu poništivi, osim ako su označeni sa ε.

2. Čvor konkatenacije • je poništiv ako su oba njegova naslednika poništiva.
3. Čvor unije | je poništiv ako je bar jedan od njegovih naslednika poništiv.
4. Čvor zvezdastog zatvaranja * je uvek poništiv. Na Sl. 1.4.18 poništivi čvorovi su zaokruženi.



Sl. 1.4.18

Prva pozicija za svaki čvor stabla predstavlja skup pozicija listova koji se mogu pojaviti na prvom mestu sekvence koju generiše podizraz sa korenom u datom čvoru. Računanje prvih pozicija:

1. $prva\_pozicija(a \bullet b) = \begin{cases} prva\_pozicija(a), & a \text{ nije poništiv} \\ prva\_pozicija(a) \cup prva\_pozicija(b), & a \text{ je poništiv} \end{cases}$
2. $prva\_pozicija(a|b) = prva\_pozicija(a) \cup prva\_pozicija(b)$
3. $prva\_pozicija(a^*) = \text{prva\_pozicija}(a)$.

Na Sl. 1.4.18 prve pozicije čvorova stabla navedene su levo od čvorova.

Poslednja pozicija za svaki čvor stabla predstavlja skup pozicija listova koji se mogu pojaviti na poslednjem mestu sekvence koju generiše podizraz sa korenom u datom čvoru. Računanje poslednjih pozicija:

1. $poslednja\_pozicija(a \bullet b) = \begin{cases} poslednja\_poz.(b), & b \text{ nije poništiv} \\ poslednja\_poz.(b) \cup poslednja\_poz.(a), & b \text{ je poništiv} \end{cases}$
2. $poslednja\_pozicija(a|b) = poslednja\_pozicija(a) \cup poslednja\_pozicija(b)$
3. $poslednja\_pozicija(a^*) = \text{poslednja\_pozicija}(a)$.

Na Sl. 1.4.18 poslednje pozicije čvorova stabla navedene su desno od čvorova.

Sledeća pozicija(p) definiše se za svaku poziciju lista p kao skup pozicija koje mogu slediti posmatranu poziciju p u proizvoljnoj sekvenci znakova izvedenoj iz datog regularnog izraza. Pri računanju funkcije sledeća pozicija u sintaksnom stablu razmatraju se svi čvorovi tipa • i * na sledeći način:

- Čvor •: prva pozicija desnog naslednika je sledeća pozicija za sve iz poslednje pozicije levog naslednika.
- Čvor *: prva pozicija naslednika je sledeća pozicija za sve iz poslednje pozicije naslednika.

Rešenje

U konkretnom slučaju, dobijaju se vrednosti funkcije sledeća pozicija date sledećom tabelom.

poz.	sled. pozicija
1	{4}
2	{3}
3	{2,4}
4	{5}
5	/

c)

Analiza problema

Ulazna azbuka U traženog automata je skup znakova koji se pojavljuju u regularnom izrazu. Pojedinim stanjima automata pridruženi su skupovi pozicija listova sintaksnog stabla. Stanja se formiraju i evidentiraju u skupu States a pravi se i tabela prelaza δ po sledećem algoritmu:

```
startno stanje S := prva_pozicija(koren sintaksnog stabla);
inicijalizovati States sa S koje nije markirano;
while (∃ T ∈ States ∧ T nije markirano)
        markiraj T;
        if (pozicija markera kraja ∈ T )
                then označiti T kao stanje prihvatanja;
                else označiti T kao stanje odbijanja;
        end if;
        for (∀a ∈ U)
                N := ∅;
                for (∀p ∈ T)
                        if (simbol na poziciji p = a)
                                then N := N ∪ sledeća_pozicija(p);
                        end if;
                end for;
```

```
                        δ(T, a) := N;
                        if (N ≠ ∅ ∧ N ∉ States)
                                then dodati U kao nemarkirano stanje u States;
                        end if;
                end for;
        end while
```

Rešenje

Startnom stanju automata pridružen je skup *prva_pozicija* korena stabla, u konkretnom slučaju radi se o stanju {1, 2, 4}. Pošto navedeni skup ne sadrži poziciju 5 koja je pridružena markeru kraja ulaza, radi se o stanju odbijanja.

Vrsta tabele prelaza, koja je prikazana na Sl. 1.4.19(a), za određeno stanje popunjava se razmatrajući pojedine pozicije iz skupa pridruženog tom stanju. Na primer, za stanje {1, 2, 4}:

- razmatramo ulaz u koloni A: određujemo pozicije iz skupa {1, 2, 4} koje odgovaraju listovima označenim sa A. Radi se o poziciji 1 i pri tome je *sledeća_pozicija*(1) = {4}. U tabeli se u vrsti {1, 2, 4} i koloni A upisuje prelaz u stanje {4}. U slučaju da je u skupu {1, 2, 4} bilo više pozicija koje odgovaraju znaku A, skup pozicija novog stanja bio bi unija odgovarajućih skupova sledećih pozicija.
- razmatramo ulaz u koloni B: pozicija 2 iz skupa {1, 2, 4} odgovara slovu B, sledeća_pozicija(2) = {3}, znači da se iz stanja {1, 2, 4} za ulaz B prelazi u stanje {3}.
- razmatramo ulaz u koloni C: nema pozicija u skupu {1,2,4} koje odgovaraju slovu C, znači da se iz stanja {1, 2, 4} za ulaz C prelazi u stanje {}.
- razmatramo ulaz u koloni D: pozicija 4 iz skupa {1,2,4} odgovara slovu D, *sledeća_pozicija*(4)={5}, znači da se iz stanja {1, 2, 4} za ulaz D prelazi u stanje {5}.

Popunjavanje tabele nastavlja se za novodobijena stanja, sve dok se sva stanja ne pronađu i sve vrste tabele ne popune. Automat sa preimenovanim stanjima, M = {1, 2, 4}, N = {4}, O = {3}, P = {5} i Q = {2, 4} prikazan je na Sl. 1.4.19(b). Stanje greške {} po konvenciji je izostavljeno.

	A	B	C	D	
→ {1, 2, 4}	{4}	{3}	{}	{5}	0
{4}	{}	{}	{}	{5}	0
{3}	{}	{}	{2, 4}	{}	0
{}	{}	{}	{}	{}	0
{5}	{}	{}	{}	{}	1
{2, 4}	{}	{3}	{}	{5}	0

(a)

	A	B	C	D	
→ M	N	O		P	0
N				P	0
O			Q		0
P					1
Q		O		P	0

(b)

Sl. 1.4.19

d)

Prema načinu konstrukcije automat ne može imati nedostižnih stanja. Proverom utvrđujemo da je dobijeni automat u konkretnom slučaju i minimalan.

Zadatak 1.4.4

a) Napisati regularni izraz koji opisuje sve nizove jedinica i nula u proizvoljnom redosledu u kojem je broj jedinica paran i veći od nule.

b) Za nađeni regularni izraz konstruisati nedeterministički automat Tompsonovim algoritmom.

c) Odrediti minimalni deterministički automat ekvivalentan dobijenom nedeterminističkom automatu.

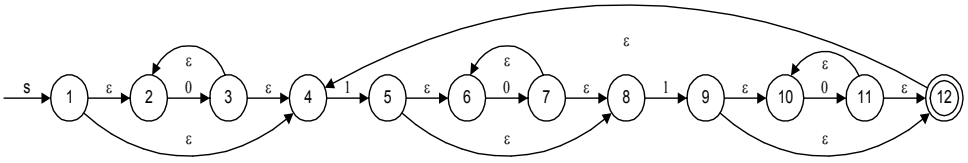
Rešenje

a)

Broj i pozicije nula su nebitni pa stoga jedan ovakav izraz zadovoljava: $0^*(10^*10^*)^+$

b)

Nedeterministički automat za dati izraz, dobijen Tompsonovim algoritmom (Zadatak 1.4.2), prikazan je na Sl. 1.4.20.



Sl. 1.4.20

c)

Odgovarajući deterministički automat prikazan je na Sl. 1.4.21(a). Startno stanje je ε_closure (1) = {1, 2, 4}. Kada preimenujemo stanja dobijemo automat prikazan na Sl. 1.4.21(b).

	0	1	
{1, 2, 4}	{2, 3, 4}	{5, 6, 8}	0
{2, 3, 4}	{2, 3, 4}	{5, 6, 8}	0
{5, 6, 8}	{6, 7, 8}	{4, 9, 10,12}	0
{6, 7, 8}	{6, 7, 8}	{4, 9, 10,12}	0
{4, 9, 10, 12}	{10, 11, 12}	{5, 6, 8}	1
{10, 11, 12}	{10, 11, 12}	{5, 6, 8}	1

(a)

	0	1	
A	B	C	0
B	B	C	0
C	D	E	0
D	D	E	0
E	F	C	1
F	F	C	1

(b)

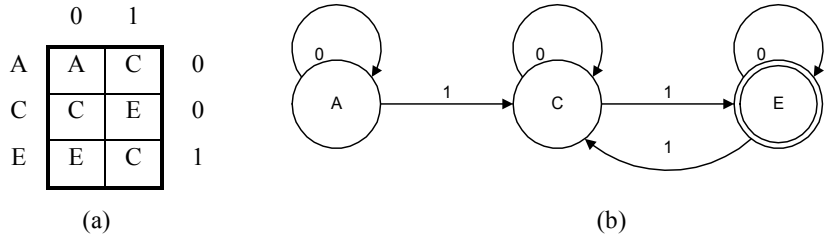
Sl. 1.4.21

Ovaj automat nema suvišnih stanja a minimizovaćemo particionom tehnikom:

$P_0 = (\{A, B, C, D\}, \{E, F\})$ inicijalna podela

$P_1 = (\{A, B\}, \{C, D\}, \{E, F\})$ delimo P_0 u odnosu na ulaz 1

Ova podela je konačna. Ostaju samo tri stanja: A=B, C=D, E=F, kao na Sl. 1.4.22.



Sl. 1.4.22

Zadatak 1.4.5

Koristeći algoritam zasnovan na pozicijama konstruisati minimalni deterministički automat koji prepoznaje sekvence:

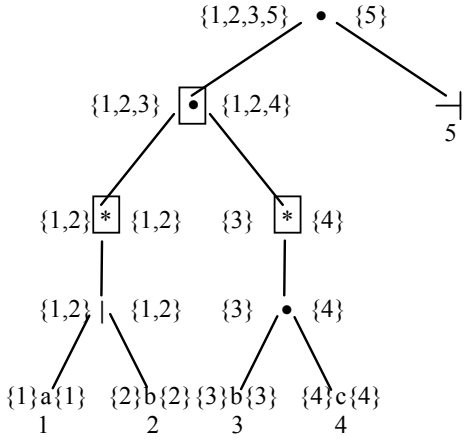
(a|b)* (bc)*.

Rešenje

Sintaksno stablo izraza

$(a|b)^*(b\ c)^*\dashv$

prikazano je na Sl. 1.4.23.



Sl. 1.4.23

Na osnovu stabla dobijaju se skupovi sledećih pozicija:

POZICIJA	SLEDEĆA POZICIJA
1	{1,2,3,5}
2	{1,2,3,5}
3	{4}
4	{5,3}
5	-

Startno stanje je $\{1_a\ 2_b\ 3_b\ 5_{\dashv}\}$ (indeksima su označeni ulazni simboli koji odgovaraju pojedinim pozicijama). Odredimo prelaze:

$\{1_a\ 2_b\ 3_b\ 5_{\dashv}\} \xrightarrow{a} \{1_a\ 2_b\ 3_b\ 5_{\dashv}\}$ $\{1_a\ 2_b\ 3_b\ 5_{\dashv}\} \xrightarrow{b} \{1_a,2_b\ 3_b\ 4_c\ 5_{\dashv}\}$

$\{1_a\ 2_b\ 3_b\ 5_{\dashv}\} \xrightarrow{c} \{\}$

$\{1_a,2_b\ 3_b\ 4_c\ 5_{\dashv}\} \xrightarrow{a} \{1_a\ 2_b\ 3_b\ 5_{\dashv}\}$ $\{1_a,2_b\ 3_b\ 4_c\ 5_{\dashv}\} \xrightarrow{b} \{1_a\ 2_b\ 3_b\ 4_c\ 5_{\dashv}\}$

$\{1_a,2_b\ 3_b\ 4_c\ 5_{\dashv}\} \xrightarrow{c} \{3_b\ 5_{\dashv}\}$

$\{3_b\ 5_{\dashv}\} \xrightarrow{a} \{\}$ $\{3_b\ 5_{\dashv}\} \xrightarrow{b} \{4_c\}$ $\{3_b\ 5_{\dashv}\} \xrightarrow{c} \{\}$

$\{4_c\} \xrightarrow{a} \{\}$ $\{4_c\} \xrightarrow{b} \{\}$ $\{4_c\} \xrightarrow{c} \{3_b\ 5_{\dashv}\}$

Dobijamo automat prikazan na Sl. 1.4.24(a). Isti automat, uz preimenovana stanja A = {1, 2, 3, 5}, B = {1, 2, 3, 4, 5}, C = {3, 5}, D = {4}, uz izostavljeno stanje greške {}, prikazan je na Sl. 1.4.24(b). Nema suvišnih niti ekvivalentnih stanja, pa je ovo ujedno i minimalni automat.

	a	b	c	
→ {1, 2, 3, 5}	{1, 2, 3, 5}	{1, 2, 3, 4, 5}	{}	1
{1,2, 3, 4, 5}	{1, 2, 3, 5}	{1, 2, 3, 4, 5}	{3, 5}	1
{3, 5}	{}	{4}	{}	1
{4}	{}	{}	{3, 5}	0
{}	{}	{}	{}	0

(a)

	a	b	c	
→ A	A	B		1
B	A	B	C	1
C		D		1
D			C	0

(b)

Sl. 1.4.24

1.5. Leksički procesori

Zadatak 1.5.1

U programskom jeziku ISPIT postoje dve vrste iskaza:

promenljiva = izraz

IF promenljiva THEN iskaz ENDIF

Identifikatori promenljivih su jednoslovni. Izrazi se sastoje od promenljivih i celobrojnih pozitivnih konstanti povezanih operatorima + i -. IF iskazi se mogu ugneždavati; iskaz iza THEN se izvršava ako je vrednost promenljive iza IF različita od nule. Sve leksičke jedinice moraju međusobno biti razdvojene prazninama (razmak ili novi red).

a) Konstruisati deterministički konačni procesor koji služi da prepozna jednu leksičku jedinicu (leksemu) i pretvori je u interni leksički kod. Kodovi su dati u sledećoj tabeli (P – promenljiva, C – konstanta):

Klasni deo	P	C	IF	THEN	ENDIF	=	+
Vrednosni deo	redni broj ulaza u tabeli simbola	vrednost	-	-	-	-	-

b) Koji niz kodova odgovara programu: A = X + 2 IF A THEN IF B THEN C = 3 ENDIF ENDIF?

c) Objasniti šta u ovom slučaju radi transliterator.

Analiza problema

Leksički analizator ima ulogu pripreme ulaza za sintaksno-semantičku analizu i prevođenje. Ulaz u leksički analizator je program u obliku niza znakova. Zadatak leksičkog analizatora je da u nizu znakova identifikuje pojavu logički povezanih grupa znakova (leksičkih jedinica, odnosno leksema) i na izlazu izda za svaku prepoznatu leksemu njoj odgovarajući interni leksički kod. Klasni deo internog koda služi da označi tip leksičke jedinice, a vrednosni deo daje eventualne dodatne informacije.

Algoritam funkcionisanja traženog leksičkog analizatora glasi:

1. izdvojiti leksemu omeđenu prazninama;
2. ako leksema počinje slovom odrediti da li se radi o rezervisanoj reči ili promenljivoj;
   - za promenljivu: ubaciti promenljivu u tabelu simbola i vratiti interni kod (p,broj ulaza);
   - za rezervisanu reč vratiti odgovarajući interni kod;
3. za specijalne znake vratiti odgovarajući interni kod;
4. za konstantu: niz znakova sakupiti, pretvoriti u binarnu vrednost i vratiti interni kod (c,binarna vrednost).

Rešenje

b)

Niz internih kodova u formi (klasa, vrednost) koji odgovara datom programu je:

(P,1) (=,-) (P,2) (+,-) (c,2) (IF,-) (P,1) (THEN,-) (IF,-) (P,3) (THEN,-) (P,4) (=,-) (c,3) (ENDIF,-) (ENDIF,-)

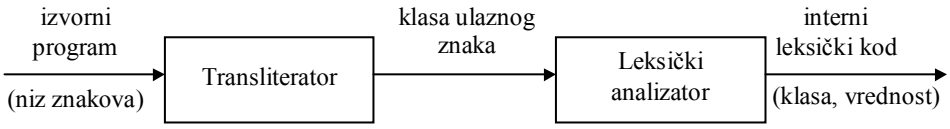
U okviru leksičke analize formira se i tabela simbola (Sl. 1.5.1).

broj ulaza	simbol
1	A
2	X
3	B
4	C

Sl. 1.5.1

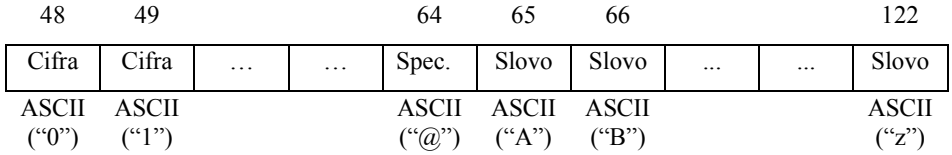
c)

Uloga transliteratora je da izvrši pripremu ulaza za leksički analizator (Sl. 1.5.2), u cilju njegovog pojednostavljivanja. Transliterator treba da na nivou pojedinačnog ulaznog znaka odredi njegovu klasu: slovo, cifra, specijalni znak, marker kraja i to prosledi leksičkom analizatoru (kome je naravno dostupan i sam ulazni znak za potrebe procesiranja). Pojedine klase ulaznih znakova definišu ulaznu azbuku automata kojim se realizuje leksički analizator.



Sl. 1.5.2

Transliterator se efikasno realizuje u vidu vektora (Sl. 1.5.3). Kod znaka sa ulaza služi za indeksiranje; vrednosti elemenata vektora su pojedine klase ulaznih znakova.



Sl. 1.5.3

a)

Leksički analizator realizovaćemo u vidu automata sa konačnim brojem stanja (Sl. 1.5.4). Automat služi da prepozna jednu leksičku jedinicu, tako da se nailazak na prazninu u ulaznom

nizu tretira kao nailazak kraja sekvence ⊣ što se može realizovati transliteracijom. Za narednu leksičku jedinicu ponovo se aktivira leksički analizator od mesta gde se stalo u ulaznom nizu.

Svakom ulazu tabele prelaza odgovara jedna akcija. Prazni ulazi označavaju greške u leksičkoj analizi. Akcije koje odgovaraju praznim ulazima mogu biti prijava greške uz moguć pokušaj oporavka – popravke greške.

	slovo	cifra	spec	⊣
startno stanje A	B1	C1	D	
nastavlja se ključna reč B	B2	B2		akcija1
nastavlja se konstanta C		C2		akcija2
došao specijalni znak D				akcija3

Sl. 1.5.4

Akcije:

B1: zn:= 1
bafer[zn]:= vrednosni deo leksičkog koda

B2: zn:=zn+1
bafer[zn]:= tekući znak sa ulaza

C1: BINVR:= ASCII(tekući znak sa ulaza) - ASCII("0")

C2: BINVR:= BINVR * 10 + ASCII(tekući znak sa ulaza) - ASCII("0")

D: SPEC:= klasa tekućeg znaka sa ulaza

akcija1:

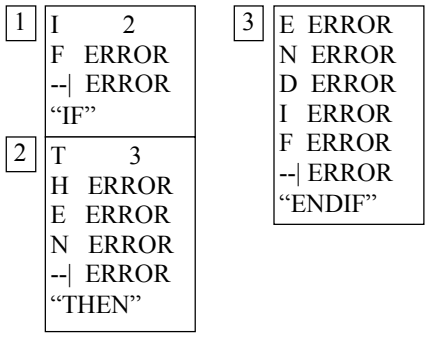
1. konsultovati prepoznavač ključnih reči (Sl. 1.5.5)
2. ako je prepoznavač vratio error onda
   ako je zn=1 onda ubaciti bafer[zn] u tabelu simbola u novi ulaz n i
   vratiti interni kod (p,n)
   inače error
   inače vratiti interni kod (ključna reč, –) koji je vratio prepoznavač ključnih reči

akcija2: vratiti interni kod (c,binarna vrednost)

akcija3: vratiti interni kod (spec,–)

Diskusija

Čitaocu se za vežbu preporučuje da realizuje opisani leksički procesor korišćenjem programskog jezika po sopstvenom izboru.



Sl. 1.5.5 Prepoznavač ključnih reči u vidu proširive liste prelaza

Zadatak 1.5.2

a) Konstruisati nedeterministički automat za prepoznavanje neoznačenih realnih konstanti. Primeri dozvoljenih konstanti su: 1257,.0392, 1E + 534, 1., 1.E5, 2.0E10 a nedozvoljenih - 21, .E, E, E10, 12E.

b) Proširiti dobijeni automat akcijama za izračunavanje vrednosti konstante.

Rešenje

a)

Ulaznu azbuku automata definišemo u vidu skupa {DIGIT, •, E, SIGN} gde se simbol DIGIT odnosi na proizvoljnu cifru, • predstavlja decimalnu tačku, E početak eksponenta a SIGN predznak eksponenta. Po metodu numeracije ulaznih simbola (Zadatak 1.1.2) razmatramo nekoliko primera ulaznih sekvenci i određujemo odgovarajuće prelaze stanja:

```
3 8 . 7 E - 3          . 9 E 2 1
0 1 1 2 3 4 5 6        0 7 3 4 6 6
```

Formiramo tabelu prelaza deterministočkog automata (Sl. 1.5.6).

		DIGIT	E	•	SIGN	
inicijalno stanje	0	1		7		0
cifra pre opcione tačke	1	1	4	2		1
opciona decimalna tačka	2	3	4			1
cifra nakon decimalne tačke	3	3	4			1
slovo E	4	6			5	0
predznak eksponenta	5	6				0
cifra eksponenta	6	6				1
tačka iza koje mora ići cifra	7	3				0

Sl. 1.5.6: Prepoznavač realnih konstanti

Postupkom minimizacije utvrđujemo da su stanja 2 i 3 ekvivalentna.

b)

Procesor izračunava binarnu vrednost konstante postavljajući sadržaje sledeća četiri registra:

- NR (number register) sadrži binarnu vrednost celog broja koji se dobija uklanjanjem decimalne tačke iz mantise;
- ER (exponent register) sadrži binarnu vrednost eksponenta koji je naveden u konstanti (ne vrši se normalizacija mantise);
- CR (count register) sadrži broj cifara iza decimalne tačke;
- SR (sign register) pamti predznak eksponenta (+1 ili -1)

Na primer, za konstantu 12.3E4 registri će imati sledeće vrednosti:

NR	ER	CR	SR
123	4	1	+1

Automat pretvaramo u procesor na taj način što svakom (nepraznom) ulazu u tabeli prelaza pridružujemo akciju koja ažurira vrednosti registara (Sl. 1.5.7). Praznim ulazima, po potrebi, možemo pridružiti akcije za detekciju i oporavak od leksičkih grešaka.

	DIGIT	E	•	SIGN	⊣
0	1a		7		
1	1b	4a	23c		YES1
23	23a	4b			YES2
4	6a			5	
5	6b				
6	6c				YES3
7	23b				

Sl. 1.5.7: Procesor za realne konstante

Akcije imaju sledeći izgled. Operator val(cifra) daje binarnu vrednost cifre.

1a: NR := val(DIGIT)

1b: NR := NR * 10 + val(DIGIT)

23a: CR := CR + 1
NR := NR * 10 + val(DIGIT)

23b: CR := 1
NR := val(DIGIT)

23c: CR := 0

4a: CR := 0

4b: ———

5: SR := val(SIGN)

6a: SR := +1
ER := val(DIGIT)

6b: ER := val(DIGIT)

6c: ER := ER * 10 + val(DIGIT)

7: ———

YES1: ER := 0
CR := 0

YES2: ER := 0

YES3: ———

Zadatak 1.5.3

Skup reči koje treba prepoznati je ARRAY, ARCTAN, B1, B2, B3, DOG, CAT, DOGFIGHT. Nacrtati dijagram strukture sledećih prepoznavača ovog skupa:

a) lista prelaza

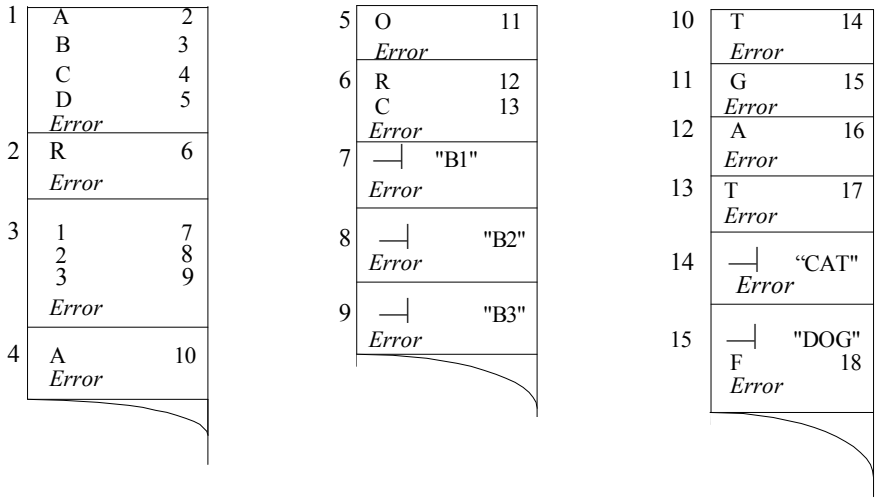
b) proširiva lista prelaza

c) uređena lista

Rešenje

a)

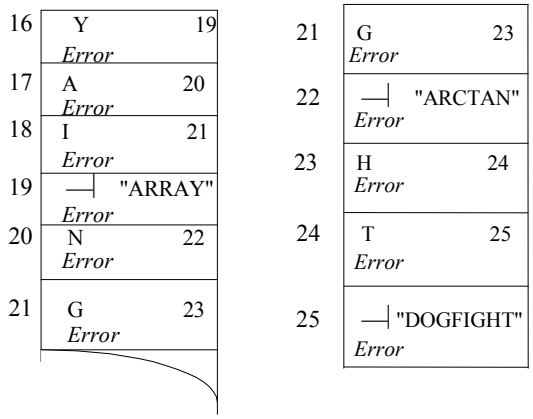
U listu 1 na Sl. 1.5.8 i Sl. 1.5.9 smeštamo sva različita prva slova i uz svako od njih adresu liste na koju prelazimo zbog dalje analize ukoliko se prvi znak sa ulaza poklapa sa slovom u listi (na primer, ako je prvo slovo A prelazi se na listu 2). Ukoliko se prvi ulazni znak ne poklapa ni sa jednim slovom navedenim u listi 1, preduzima se akcija Error, koja označava da ni jedna od zadatih reči nije prepoznata. Ukoliko se neka reč upari kompletno zaključno sa markerom kraja, odgovarajuća akcija označava prepoznavanje te reči.



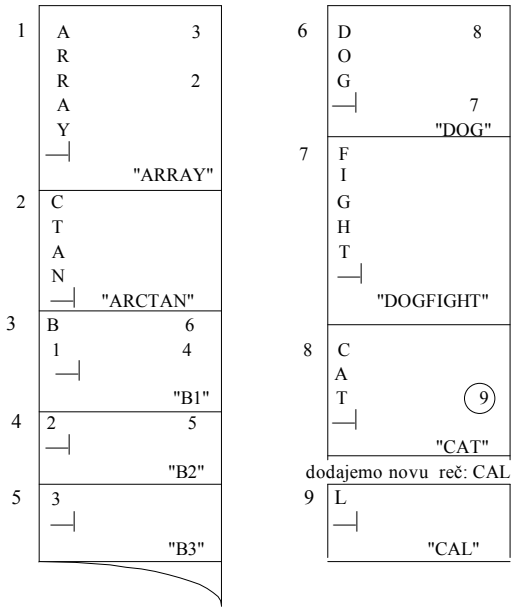
Sl. 1.5.8

b)

Jednom formiranu listu prelaza iz tačke a) rešenja je teško modifikovati da prepoznaje neku novu reč. U prethodnoj listi se ne vide, ali postoje, dva pokazivača: alternativni i sledeći (ovaj je implicitan). Ako bismo celu stvar obrnuli tako da sledeći pokazivač bude eksplicitno dat a alternativni implicitno, dobićemo listu prelaza koja se može proširivati (Sl. 1.5.10). Tamo gde pored slova nije eksplicitno navedena akcija, podrazumeva se Error.



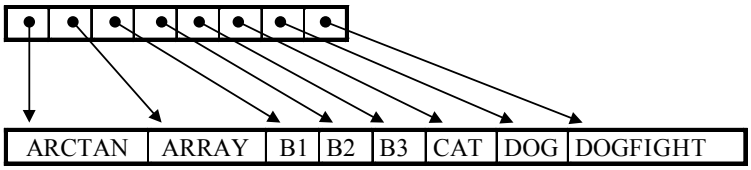
Sl. 1.5.9



Sl. 1.5.10

c)

Sortirana lista reči može se realizovati korišćenjem alfanumeričkih nizova fiksne dužine (jednake dužini najduže reči), ili alternativno, sa vektorom pokazivača na pojedine nizove kao na Sl. 1.5.11. Za nalaženje reči tada se može primeniti metod binarnog traženja koji ima složenost $\log_2 N$ gde je N broj reči.



Sl. 1.5.11

Zadatak 1.5.4

Skup reči koje treba prepoznati je ARRAY, ARCTAN, B1, B2, B3, DOG, CAT, DOGFIGHT. Realizovati prepoznavač na bazi savršene heš funkcije.

Analiza problema

Jedan od efikasnih metoda prepoznavanja reči iz fiksnog skupa je na bazi savršene heš funkcije. Princip rada takvog prepoznavača je sledeći: Definiše se funkcija Hash koja za svaku ulaznu reč, bez obzira da li je jednaka nekoj od zadatih ili ne, pridružuje numerički indeks iz određenog opsega. Ova funkcija se bira tako da rečima iz datog skupa dodeljuje jedinstvene indekse (zato se naziva "savršena"). U heš tabelu smeštaju se date reči prema svom heš indeksu (neki ulazi tabele mogu ostati nepopunjeni, ali ne može doći do kolizija).

Prepoznavanje ulazne reči vrši se na taj način što se izračuna njen heš indeks i koristeći ovu vrednost vrši čitanje iz heš tabele. Jednim poređenjem ulazne reči sa pročitanom reči iz tabele, utvrđuje se da li je ulazna reč jednaka toj pročitanoj reči, ili nije jednaka ni sa jednom od datih reči.

Jedna jednostavna heš funkcija za reč *id* dužine n znakova definisana je kao zbir po modulu veličine heš tabele HASHSIZE kodova svih znakova pomnoženih multiplikativnom konstantom HASHMULT, sledećom rekurentnom formulom:

Hash(0) := 0

Hash(t) := Hash(t - 1) * HASHMULT + id(t), t = 1, 2, ..., n

Hash := Hash(n) mod HASHSIZE

gde HASHSIZE predstavlja veličinu heš tabele, a HASHMULT multiplikativnu konstantu.

Za zadati skup reči, konstante HASHSIZE i HASHMULT mogu se odrediti na sledeći način (principijelno rešenje, moguća je racionalizacija):

```
        Ulaz: skup reči S; izlaz: celobrojne konstante HASHMULT i HASHSIZE
        for (i := broj_reči to MAX_INT)
            for (j := 1 to MAX_INT )
                kreirati praznu tabelu T veličine i;
                for (∀id ∈ S)
                    Izračunati Hash za reč id koristeći j kao multiplikativnu konstantu;
                    if (T(Hash) je nepopunjeno)
                        then T(Hash) := id;
                        else idi na izlaz;
                    end if;
                end for;
                HASHSIZE := i;
                HASHMULT := j;
                završetak rada;
            end for;
    izlaz:
        end for;
        konstante nisu određene; završetak rada;
```

Rešenje

Za gornji skup reči dobija se da je HASHMULT = 20, HASHSIZE = 9. Drugim rečima, samo jedan ulaz heš tabele ostaje nepopunjen. Sledi programska realizacija na C-u prepoznavača za dati skup reči koji koristi opisanu heš funkciju. Potprogram kljucna() vraća indikaciju da li ulazna reč pripada ili ne datom skupu reči.

```
        #include <string.h>

        static char *table[] ={ NULL, "B1", "B2", "B3", "ARCTAN", "CAT",
                                "DOG", "ARRAY", "DOGFIGHT" };

        int kljucna( const char *rec )
        {
                unsigned hash = 0; /* ne valja int */
                const char *tek = rec;
                while ( *tek ) hash = hash * 20 + *tek++;
                return !strcmp( table[ hash % 9 ], rec );
        }
```

Diskusija

Kada je skup reči za prepoznavanje veliki, nedostatak opisane heš funkcije je što ostavlja veliki procenat nepopunjenih ulaza u heš tabeli, jer je najmanja veličina HASHSIZE, za koju je moguće odrediti konstantu HASHMULT tako da svaka reč ima jedinstvenu vrednost heš funkcije, mnogo veća od broja reči u skupu. Mogući su i drugi, složeniji načini definisanja savršenih heš funkcija koji idu za tim da što više smanje procenat nepopunjenih ulaza. S druge strane, ako je heš funkcija definisana na složen način, troši se više vremena na njeno

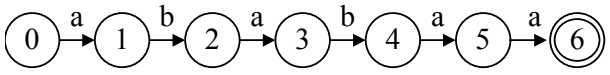
izračunavanje, a upravo je vreme prepoznavanja reči kritičan faktor. Na UNIX sistemima programski alat **gperf** služi za generisanje prepoznavača na bazi savršene heš funkcije.

Zadatak 1.5.5

Dat je algoritam koji koristi funkciju neuspeha *f* da bi se odredilo da li je ključna reč $b_1b_2...b_m$ podniz niza $a_1a_2...a_n$

```
S:=0
for (i:=1 to n)
        while (S > 0 and a_i ≠ b_{S+1})
                S := f (S);
        end while;
        if (a_i = b_{S+1})
                then S := S + 1;
        end if;
        if (S = m)
                then return "YES";
        end if;
end for;
return "NO";
```

Razmotrimo dijagram prelaza determinističkog automata koji prepoznaje zadatu ključnu reč. Na primer, za ključnu reč ababaa dobija se sledeći automat:



Funkcija neuspeha *f* preslikava stanje gore navedenog determinističkog konačnog automata na celobrojnu vrednost koja odgovara dužini najdužeg pravog sufiksa u datom stanju koji je istovremeno i prefiks ključne reči.

a) Primeniti dati algoritam da bi se odredilo da li je ababaa podniz od abababaab.

b) Dokazati da algoritam vraća logičku vrednost "YES" ako je $b_1b_2....b_m$ podniz niza $a_1a_2.....a_n$.

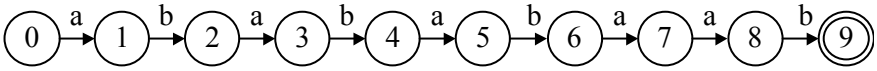
c) Pokazati da se algoritam izvršava za vreme O(m+n) ako se *f*(S) izračunava za vreme O(m).

d) Za zadatu ključnu reč y, pokazati da se funkcija može iskoristiti za konstruisanje determinističkog automata sa |y|+1 stanjem, za O(|y|) vreme, a za regularni izraz $.^*y.^*$, gde je **.** oznaka za bilo koji znak, a |y| označava dužinu niza y.

Analiza problema

Razmatramo nekoliko primera u svrhu ilustracije izračunavanja funkcije neuspeha:

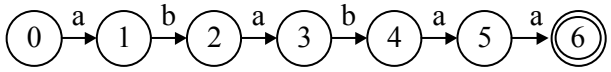
1) ključna reč = abababaab



stanja	1	2	3	4	5	6	7	8	9
funkcija neuspeha	0	0	1	2	3	2	2	1	2

Na primer, da bi se automat našao u stanju 5 mora procesirati sekvencu ababa. Posmatrajući redom sufikse ove sekvence, ustanovljavamo da su sufiksi a, aba i ababa istovremeno i prefiksi polazne sekvence. Pošto ababa upravo predstavlja procesiranu sekvencu, ovaj niz nije njen pravi sufiks koji mora biti kraći od same sekvence, tako da je najduži pravi sufiks aba koji ima dužinu 3, što je upravo vrednost funkcije neuspeha za stanje 5.

2) Ključna reč = ababaa



stanje	1	2	3	4	5	6
funkcija neuspeha	0	0	1	2	3	1

Rešenje

a)

Posmatrajmo nizove:

α = abababaab i β = ababaa

Odavde se vidi da je i = 1..9, i S = 0..6.

Polazeći od već izračunate funkcije neuspeha za ključnu reč β, posmatraćemo vrednosti za i i S pri prolasku kroz petlju, kao i logičke vrednosti uslova ① $S > 0$ and $a_i \neq b_{S+1}$, ② $a_i = b_{S+1}$, i ③ S = m. Izvršavajući ručno dati algoritam, dobija se:

1. S=0
   i=1
   uslov ① = false
   uslov ② ($a_1=b_1$) = true
   S=1
   povratak na početak petlje
2. i = 2
   uslov ① (S>0 and $a_2 \neq b_2$)=false
   uslov ② ($a_2=b_2$)=true
   S=2
   povratak na početak petlje

3. i=3
   uslov ① (S>0 and $a_3 \neq b_3$)=false
   uslov ② ($a_3=b_3$)=true
   S=3
   povratak na početak petlje
4. i=4
   uslov ① (S>0 and $a_4 \neq b_4$)=false
   uslov ② ($a_4=b_4$)=true
   S=4
   povratak na početak petlje
5. i=5
   uslov ① (S>0 and $a_5 \neq b_5$)=false
   uslov ② ($a_5=b_5$)=true
   S=5
   povratak na početak petlje
6. i=6
   uslov ① (S>0 and $a_6 \neq b_6$)=true
   izvršava se S:=f(5); iz tabele funkcije neuspeha za reč β dobija se da je S=3
7. uslov ① (S>0 and $a_6 \neq b_4$)=false
   uslov ② ($a_6=b_4$)=true
   S=4
   povratak na početak petlje
8. i=7
   uslov ① (S>0 and $a_7 \neq b_5$)=false
   uslov ② ($a_7=b_5$)=true
   S=5
   povratak na početak petlje
9. i=8
   uslov ① (S>0 and $a_8 \neq b_6$)=false
   uslov ② ($a_8=b_6$)=true
   S=6
   uslov ③ (S=m)
   "YES"

Dakle, niz β je podniz niza α.

b)

Posmatrajmo ovaj problem iz dva ugla:

1) $\beta=b_1b_2...b_m$ je podniz niza $\alpha=a_1a_2...a_n$

i

2) β nije podniz niza α.

1. Neka niz β nije prefiks niza α. S će dobiti vrednost različitu od nule za neku vrednost i=K, K>1, kada uslov $a_k=b_1$ ima vrednost true. Pretpostavimo sada da je $a_{k+1} = b_2$, $a_{k+2} = b_3$,..., $a_{k+j-1} = b_j$, za j < m, a da je $a_{k+j} \neq b_{j+1}$ i S = j. U tom slučaju se primenjuje funkcija neuspeha koja postavlja S na vrednost *f*(S), koja odgovara najdužem sufiksu u stanju S=j koji je ujedno i prefiks ključa ($b_1b_2...b_m$). Ako bismo išli na klasičan način, pa se pomerili za jednu poziciju u nizu a, to jest, od a_k na a_{k+1}, poređenje bi se nastavilo ponovo od pozicije 1 u nizu β, što usporava proces pretraživanja. Koncept funkcije neuspeha ubrzava ovaj proces. Drugim rečima, funkcija neuspeha daje poziciju u nizu β odakle treba nastaviti poređenje (vraćanje na neki od prethodnih elemenata niza β umesto od b_1) sa članovima niza α od pozicije povrede, to jest, od a_{k+j}.

   Dakle, funkcija neuspeha ubrzava nalaženje ključne reči. Do primene funkcije neuspeha može doći više puta, ali po pretpostavci, za neko l, moraće da važi $a_l=b_1$, $a_{l+1}=b_2$,...$a_{l+m-1}=b_m$, i biće ispunjen uslov S = m a kao rezultat će biti vraćeno "YES".

   Ako je niz β prefiks niza α, onda ćemo odmah na početku imati $a_l=b_1$, $_2=b_2...a_m=b_n$, pa se uslov 1 neće nijedanput izvršiti i kao rezultat će biti vraćeno "YES".

2. U slučaju kada β nije podniz od α, to znači da ne postoji takvo K (K=0...n-m) za koje važi skup uslova U: $a_{K+j}=b_j$, j=1, 2, ..., m, za bar jedno K iz skupa {0, 1, ..., n-m}.

   Drugim rečima, $a_{K+j}\neq b_j$ barem za jednu vrednost j iz navedenog skupa vrednosti i za bilo koje K.

   Pretpostavimo sada da algoritam vraća "YES" i kada β nije podniz od α. To znači da bi jednovremeno važili uslovi i≤n i S=m. S može postati jednako m jedino izvršavanjem iskaza S:=S+1. Tu postoje sledeće mogućnosti:

   a) Iskaz u liniji 3 se nikada ne izvršava, to jest, S=0 ostaje sve vreme izvršenja programa. Međutim, to znači da je za K=0 skup uslova U zadovoljen, što podrazumeva da je β podniz od α, to jest njegov prefiks, a to protivreči pretpostavci.

   b) Posle nekoliko iteracija u kojima važi uslov $a_i=b_{s+1}$ dolazi do $a_i\neq b_{s+1}$ i izvršava se S:=*f*(S). Treba uočiti da je vrednost funkcije neuspeha uvek manja od vrednosti nezavisno promenljive, to jest, S>*f*(S), što, uostalom, proizilazi iz njene definicije. Funkciju neuspeha možemo shvatiti i kao prelaz na neko prethodno stanje determinističkog konačnog automata, koje odgovara upotrebljivom, već viđenom prefiksu. Uslov S=m može da bude ispunjen tek kasnijom inkrementacijom S, ali će tom prilikom postojati i neko K za koje važi U. To dovodi do zaključka da je β podniz od α, što opet protivreči pretpostavci.

   c) Ako je i>n pre nego što je S=m, onda β nije podniz niza α, jer algoritam vraća u tom slučaju vrednost "NO".

   Dakle, u slučaju da podniz β nije podniz niza α, algoritam uvek vraća kao rezultat "NO".

c)

Ako se dva algoritma, A1 i A2, izvršavaju sekvencijalno, svaki sa vremenom O(P) i O(Q) respektivno, tada se A=A1 seq A2 izvršava u vremenu O(P+Q). Ako algoritam A1 odgovara izračunavanju funkcije neuspeha u vremenu O(m), a A2 uparivanju podniza, onda treba

pokazati da se A2 izračunava u vremenu O(n). Da bismo to utvrdili, treba pokazati da se iskaz S:=f(S) u while petlji u algoritmu A2 ne izvršava više od n puta u toku izvršenja celog algoritma A2. Stoga se mora pronaći najnepovoljnija situacija sa gledišta izvršenja. Najnepovoljnija situacija je ona koja omogućuje najveći broj izvršenja iskaza S:=f(S) bez izlaska iz petlje.

Pretpostavimo da prilikom prvog traženja ključne reči β u nizu α dolazi do povrede poređenja pri b_i. Neka tada algoritam A2 uđe u while petlju u kojoj nailazi na lanac primene funkcije neuspeha dužine i-1:

$f(i-1) = i-2$

$f(i-2) = i-3$

$f(i-3) = i-4$

.....

.....

$f(3) = 2$

$f(2) = 1$

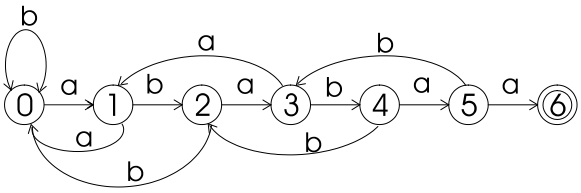
$f(1) = 0$

Odavde proizilazi da je potrošeno prvih i-1 simbola u nizu α. Sledeće poređenje polazi od simbola b_1 u ključnoj reči β i simbola a_i u nizu pretrage α. Neka je scenario pretrage u drugom traženju isti. To znači da će novih i-1 simbola u nizu α biti potrošeno i da će sledeća pretraga početi takođe od simbola b_1 ključne reči i simbola $a_{2*(i-1)+1}$ niza pretrage. Odavde sledi da je za k-tu neuspešnu pretragu potrošeno k*(i-1) simbola u nizu α uz k*(i-1) izvršenja while petlje, odnosno k*(i-1) primena funkcije neuspeha. Ako uzmemo najnepovoljniji slučaj da imamo l celih pretraga (ostaje samo poslednji simbol a_n u nizu α, koji takođe prouzrokuje povredu), onda je maksimalan ukupan broj izvršenja while petlje (odnosno, broj primena funkcije neuspeha) jednak n-1, čime se dokazuje da se algoritam uparivanja niza (A2) izvršava u vremenu O(n), a celokupan algoritam (A1 seq A2) u vremenu O(m+n).

d)

Svako stanje determinističkog automata koji prepoznaje ključnu reč β u nizu α može se predstaviti kao deo (prefiks) ključne reči koji je zaključno sa tim stanjem uparen. Ako se u datom stanju učita neupariv ulazni znak, onda se tekuće stanje mora promeniti u stanje koje odgovara najvećoj dužini niza koji je pravi sufiks učitanog niza i ujedno prefiks ključne reči (bez simbola povrede) jer uparivanje treba nastaviti od stanja koje odgovara maksimalnoj dužini prefiksa ključne reči koji se sadrži u učitanom nizu. Ovakvu primenu stanja može da obezbedi funkcija neuspeha. Broj stanja takvog automata odgovara dužini niza β uvećanom za jedan, da bi se uključilo i startno stanje.

Na primer, za prepoznavanje niza β=ababaa u ulaznoj sekvenci α, koja se sastoji od istih znakova kao i β, može se konstruisati deterministički automat prikazan na Sl. 1.5.12.



Sl. 1.5.12

Pretpostavimo sada da želimo da konstruišemo deterministički automat za regularni izraz

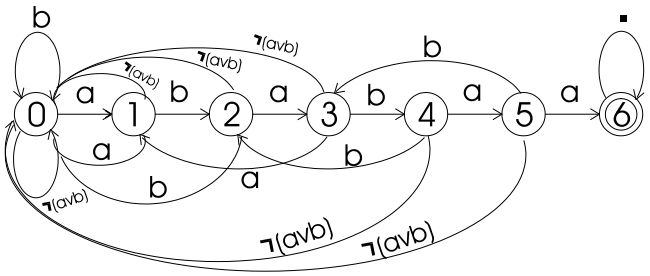
r = **.***y**.***

gde je **.** oznaka za proizvoljni znak koji može da pripada y ali i ne mora. Treba pokazati da se iz automata dobijenog prethodnom analizom može dobiti i automat za regularni izraz r bez uvodjenja dodatnih stanja, jer to proizilazi iz formulacije samog zadatka.

Neka je y=β, dakle |y|= |β|=m.

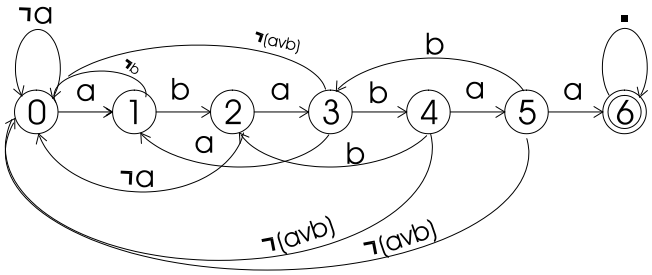
Posmatrajmo stanje K automata za prepoznavanje niza y (β). Ako se u tom stanju pojavi znak povrede p∈(znaci ključne reči y), tada se primenjuje funkcija neuspeha f(K). Ako se pojavi znak povrede p∉ (znaci ključne reči y), tada se ne može primeniti funkcija neuspeha jer znak p prekida sekvencu poređenja, što zahteva da se sa poređenjem počne ispočetka, a to je vraćanje u startno (nulto) stanje. To znači da u svakom stanju automata za prepoznavanje y treba dodati još jednu granu prelaza (izuzev za stanje prihvatanja) koji će taj automat vratiti u stanje nula ako je učitan znak povrede p koji ne pripada znacima ključne reči y. Kada se dođe u stanje prihvatanja automata za prepoznavanje y, tada se, dalje, za svaki ulazni znak, prelaz vrši u isto stanje (stanje prihvatanja) dok se ne iscrpi ulaz. Odavde sledi da je broj stanja za prepoznavanje regularnog izraza r isti kao i u slučaju prepoznavanja samo y. Osnovna dopuna grafa prelaza automata je da se za svaki simbol p koji ne pripada y dodaje u datom stanju (ako nije stanje prihvatanja) prelazi u nulto stanje, a u stanju prihvatanja prelaz u isto stanje za svaki ulazni simbol. Pošto za ove prelaze nije potrebno nikakvo izračunavanje funkcije neuspeha, ostaje da se izračunavaju samo stanja prelaza za prepoznavanje y, što se svodi na izračunavanje funkcije neuspeha. Pošto se izračunavanje funkcije neuspeha izvodi u vremenu O(m), a |y|=m, to se vreme izračunavanja determinističkog automata za izraz r izvodi u vremenu O(|y|).

Za dati primer na Sl. 1.5.12, deterministički automat imao bi oblik kao na Sl. 1.5.13.



Sl. 1.5.13

Jasno je da se graf sa Sl. 1.5.13 može uprostiti, tako da bi uprošćen automat izgledao kao na Sl. 1.5.14.



Sl. 1.5.14

Diskusija

Ostavlja se čitaocu da analizira primenu funkcije neuspeha u prepoznavanju ključnih reči u kompilatorima.

1.6. Oporavak od grešaka

Zadatak 1.6.1

Ako korak 1. funkcije SPELLING_REPAIR daje:

I = {BASK, THEN, THERE, THAN, HAND}

kao moguće popravke za pogrešan identifikator THANE, koji će identifikator biti izabran kao popravka.

```
Funkcija SPELLING_REPAIR(SUBJECT: niz znakova)

1. [generisati sve moguće popravke]
   I := { i | i je identifikator ili ključna reč, i i se može desiti u tekućem kontekstu; ovo
   poslednje ograničenje može da uključi ispitivanje tipa i opsega važenja identifikatora}

2. [naći blisku popravku]
   for (∀j ∈ I)
       if COMPARE(SUBJECT, j)
           then return(j)
       end if
   end for

3. [ne može da se nađe bliska popravka; vratiti 0 kao znak neuspeha]
   return(0)
```

```
Funkcija COMPARE(S,T: nizovi znakova)

1.  [naći dužine S i T i proveriti razlike u dužinama]
    LS := LENGTH(S)
    LT := LENGTH(T)
    if (LS < 3 or LT < 3 or ABS(LS - LT) > 1)
        then return(false)
    end if
    EQUAL := true

2.  [naći prvu poziciju gde se S i T razlikuju]
    for (j = 1,2,...MIN(LS,LT))
        if (SUB(S, j, 1) ≠ SUB(T, j, 1))
            then EQUAL := false;
                 goto izlaz;
        end if
    end for
    izlaz:

3.  [S i T su jednaki ili se razlikuju za jedno umetanje ili brisanje na kraju]
    if (EQUAL)
        then return(true)
    end if

4.  [proveriti zamenu mesta ili promenu jednog simbola]
    if (LS = LT)
        then    if (j < LS)
                    then    if  (SUB(S, j+1,1) = SUB(T, j, l) and
                                 SUB(T,j+l, l) = SUB(S,j, l))
                                then j := j + 1
                            end if
                end if
                if (SUB(S, j+1) = SUB(T, j+1))
                    then    return(true)
                    else    return(false)
                end if
        else [provera za umetanje ili brisanje]
                if(LS > LT)
                    then    if (SUB(T,j) = SUB(S,j+1))
                                then        return(true)
                                else        return(false)
                            end if
                    else    if (SUB(S,j) = SUB(T,j+1))
                                then        return(true)
                                else        return(false)
                            end if
                end if
    end if
```

Napomena: Funkcija SUB(S, i, j) vraća podniz niza S od počev od i-tog znaka, dužine j znakova. Ako se j izostavi, vraća se podniz od i-tog znaka do kraja niza S.

Analiza problema

Alfanumerički identifikator igra vrlo veliku ulogu u većini programskih jezika. Identifikatori se koriste za označavanje promenljivih, konstanti, operatora, naziva (labela) i tipova podataka. Programski jezici sadrže klasu specijalnih identifikatora nazvanih ključne reči, koji se koriste kao sintaksni delimiteri. Greške pri kucanju i zaboravnost su česti uzroci pogrešnog zadavanja identifikatora. Stoga, identifikator ili netačna ključna reč se pojavljuju tamo gde određena ključna reč treba da se nađe, ili se ključna reč ili nedefinisan identifikator pojavljuje tamo gde se očekuje identifikator određene promenljive. U takvim slučajevima leksička greška se popravlja pomoću Morganovog algoritma koji pokušava da odabere najbolji identifikator koji je blizak ilegalnom identifikatoru koji se pojavljuje u izvornom programu. Ako algoritam uspe, popravka se sastoji od zamene ilegalnog identifikatora legalnim.

Funkcija SPELLING_REPAIR(SUBJECT): Ako je dat netačan identifikator SUBJECT, ova funkcija poziva funkciju COMPARE koja treba da odredi da li su dva identifikatora dovoljno slična da bi jedan mogao da bude posledica greške drugog.

Morgan napominje da je neefikasno da se pokušava popravka imena promenljivih dužine manje od tri znaka. Ovo važi ne samo za imena promenljivih već i za ključne reči i ostale identifikatore.

Morganov algoritam koristi funkciju COMPARE da ispita da li je SUBJECT u nekom smislu blizak identifikatoru j. Pisac prevodioca može da izabere bilo koji razuman metod radi obavljanja ovog poređenja. Funkcija treba da vrati true onda i samo onda kada se SUBJECT može dobiti iz j pomoću nekih transformacija koje predstavljaju verovatne greške pri kucanju ili izgovoru. Ovaj poseban algoritam koji je koristio Morgan zasniva se na postupku Damera (1964). Ovaj algoritam vraća true u slučajevima kada se SUBJECT može dobiti iz j jednom od sledećih transformacija:

1. Izmena jednog simbola.
2. Brisanje jednog simbola.
3. Umetanje jednog simbola.
4. Zamena mesta dva susedna simbola.

Damero i Morgan izveštavaju da pojedinačni primeri jedne od ovih transformacija obuhvataju više od 80% grešaka. Polazeći od funkcije COMPARE koja je zasnovana samo na pomenutim transformacijama, korak 1. funkcije SPELLING_REPAIR može se modifikovati tako da odbaci bilo koji identifikator **i** čija se dužina razlikuje od dužine SUBJECT-a za više od jednog znaka. Ovaj test je uključen u prvi korak funkcije COMPARE.

Rešenje

Funkcija SPELLING_REPAIR inicira redom izvršavanje funkcije COMPARE za svaku od reči iz skupa I. Razmotrimo izvršavanje funkcije COMPARE za popravak BASK:

1. S = THANE, LS = 5

T = BASK, LT = 4

EQUAL = true

2. j = 1,

   SUB(THANE,1,1) ≠ SUB(BASK,1,1), T ≠ B

   EQUAL = false

3. ići na korak 4.

4. LS > LT, 5 > 4

   SUB(BASK, 1) ≠ SUB(THANE, 2), BASK ≠ HANE

   return (false)

Razmotrimo izvršavanje funkcije COMPARE za popravak THEN:

1. T = THEN, LT = 4

   EQUAL = true

2. j = 3,

   SUB(THANE,3,1) ≠ SUB(THEN,3,1), A ≠ E

   EQUAL = false

3. ići na korak 4.

4. LS > LT, 5 > 4

   SUB(THEN, 3) ≠ SUB(THANE, 4), EN ≠ NE

   return (false)

Razmotrimo izvršavanje funkcije COMPARE za popravak THERE:

1. T = THERE, LT = 5

   EQUAL = true

2. j = 3,

   SUB(THANE,3,1) ≠ SUB(THERE,3,1), A ≠ E

   EQUAL = false

3. ići na korak 4.

4. LS = LT, 5 = 5

   SUB(THANE, 4, 1) ≠ SUB(THERE, 3, 1), N ≠ E

   SUB(THANE, 4) ≠ SUB(THERE, 4), NE ≠ RE

   return (false)

Razmotrimo izvršavanje funkcije COMPARE za popravak THAN:

1. T = THAN, LT = 4

   EQUAL = true

2. j = 4,

   EQUAL = true

   return (true)

Funkcija SPELLING_REPAIR završava rad; za popravak je izabran identifikator THAN.

Diskusija

Opisani algoritam radi kada postoji samo jedna greška u identifikatoru. Moguće je da identifikator bude pogrešan tako da postoje dva identifikatora. Na primer, neki simbol u identifikatoru može da bude izmenjen u neki specijalni simbol, proizvodeći dva identifikatora (be#in). Na sličan način, prvi simbol može slučajno da bude otkucan kao cifra, menjajući identifikator u broj ili u broj iza kojeg sledi identifikator. Stoga funkcija SPELLING_REPAIR ne popravlja sve moguće leksičke greške identifikatora, čak i kada se one mogu popraviti jednom od pomenutih transformacija.

1.7. Programski primeri

Zadatak 1.7.1

Realizovati na C-u konačni automat sa Sl. 1.7.1 koristeći:

a) implicitno

b) eksplicitno

predstavljanje stanja.

	a	b	⊣
∏A	B	NO	YES
B	A	B	NO

Sl. 1.7.1

Rešenje

a)

Pod implicitnom predstavom stanja podrazumeva se da je stanje određeno mestom u programu koji se izvršava. Funkcija DKA() realizuje zadati automat. Deo koda označen labelom A odgovara vrsti A tabele prelaza, a deo koda označen labelom B odgovara vrsti B tabele prelaza.

Priložen je i glavni program kojim se učitava ulazni niz s, poziva funkcija DKA() kojoj se prosleđuje niz s i ispisuje rezultat koji vraća ova funkcija, 1 u slučaju da se niz s prihvata, 0 u suprotnom.

```
#include<stdio.h>
#define ENDM '\0'  /* marker kraja ulaza */
enum Boolean {NO, YES};

enum Boolean DKA(char *ulaz) {
   A: switch (*ulaz++) {
            case 'a': goto B;
            case 'b': return NO;
            case ENDM: return YES;
            default: return NO;
      }
   B: switch (*ulaz++) {
            case 'a': goto A;
            case 'b': goto B;
            case ENDM: return NO;
            default: return NO;
      }
}

main () {
            char s[80];
            scanf("%s",s);
            printf("%u\n",DKA(s));
}
```

b)

Funkcija DKA() realizuje zadati automat. U eksplicitnoj predstavi, postoji promenljiva (tekuce) koja pamti tekuće stanje pri radu automata. Tabela prelaza predstavljena je matricom tabela. U ovoj varijanti neophodna je transliteracija ulaznog niza znakova koja je realizovana switch strukturom (za tekući ulazni znak *ulaz određuje se odgovarajuća kolona tabele prelaza i upisuje u promenljivu znak).

Akcijama automata (NO1 i YES1) dodeljeni su kodovi viši od kodova pojedinih stanja da bi se po čitanju iz matrice tabela na jednostavan način donela odluka da li izvršiti prelaz u novo stanje ili preduzeti neku od akcija.

```
#include<stdio.h>
#define ENDM '\0'
enum Boolean {NO, YES};

enum Boolean DKA(char *ulaz) {
   enum Stanje {A,B,NO1,YES1};
   enum Stanje tabela[2][3] = {B,NO1,YES1,A,B,NO1};
   int znak, tekuce=A;
```

```
    while(1) {
      switch (*ulaz++) {
            case 'a': znak=0; break;
            case 'b': znak=1; break;
            case ENDM: znak=2; break;
            default: return NO;
      }
      tekuce=tabela[tekuce][znak];
      if(tekuce>B) return tekuce-NO1;
    }
}

main () {
            char s[80];
            scanf("%s",s);
            printf("%u\n",DKA(s));
}
```

Zadatak 1.7.2

Realizovati na C-u algoritam minimizacije konačnog automata metodom particija. Program treba u svakom koraku da ispisuje trenutne skupove stanja i na kraju tabelu prelaza minimalnog automata.

Rešenje

Razmotrimo najpre izlaz programa za jedan primer automata sa stanjima obeleženim brojevima od 0 do 7 i ulazima obeleženim sa 0, 1, 2. Startno stanje 1 je obeleženo zvezdicom. Najpre se ispisuje zadati automat, zatim, u svakom koraku rada, tekuća particija i ulazni simbol kojim se vrši razbijanje particije i na kraju se ispisuje minimalni automat.

```
      0  1  2
  0   0  0  0  0
* 1   2  3  4  1
  2   2  5  4  1
  3   6  3  4  1
  4   2  5  4  0
  5   0  3  0  1
  6   6  5  4  1

P:  0  1  1  1  0  1  1
Razmatra se ulaz 0.
P:  0  1  1  1  2  3  1
Razmatra se ulaz 0.
Nema promene.
Razmatra se ulaz 1.
```

```
P:  0  1  2  1  3  4  2
Razmatra se ulaz 0.
Nema promene.
Razmatra se ulaz 1.
Nema promene.
Razmatra se ulaz 2.
Nema promene.

      0  1  2
  0   0  0  0  0
* 1   2  1  3  1
  2   2  4  3  1
  3   2  4  3  0
  4   0  1  0  1
```

Particija se predstavlja na taj način što se za svako stanje polaznog automata ispisuje redni broj skupa kome pripada u okviru particije (skupovi se numerišu počev od nule). Na primer,

```
P:  0  1  1  1  0  1  1
```

označava inicijalnu podelu stanja na dva skupa, nulti i prvi:

```
P = ({0, 4}, {1, 2, 3, 5, 6})
```

dok

```
P:  0  1  1  1  2  3  1
```

označava sledeću podelu:

```
P = ({0}, {1,2,3,6}, {4}, {5})
```

Sledi listing programa. Struktura DKA predstavlja opis automata. Procedura minimizacija() realizuje iterativno formiranje particija. Procedura napravi_novu_particiju() realizuje ključni deo algoritma – formiranje nove particije na osnovu tekuće particije, zadatog ulaznog simbola i tabele prelaza automata. Ova procedura za svako stanje automata vodi evidenciju o rednom broju skupa u kome se ono nalazi u tekućoj particiji i o rednom broju skupa u koji se prelazi iz tog stanja pod zadatim ulazom. Sva stanja koja imaju ovaj par brojeva isti, u novoj particiji naći će se u istom skupu .

```
/* minimizacija konacnog automata particionom metodom */
#include <stdio.h>
#define MAX_STANJA 100
#define MAX_ULAZNIH_SIMBOLA   20

typedef struct DKA
{
   int br_stanja, br_ulaza, startno;
   int prihvata[MAX_STANJA];
   int prelaz[MAX_STANJA][MAX_ULAZNIH_SIMBOLA];
} DKA;
```

```
typedef int Particija[MAX_STANJA];

void ispisi_particiju( Particija part, int br_stanja )
{
   int i;

   printf( "P:" );
   for ( i = 0; i < br_stanja; i++ )
            printf( "%4d", part[i] );

   printf( "\n" );
}

void ispisi_dka( DKA *dka )
{
   int i,j;

   printf("    ");
   for ( j = 0; j < dka->br_ulaza; j++ )
      printf( "%4d", j );
   printf("\n\n");

   for ( i = 0; i < dka->br_stanja; i++ )
   {
     if ( i == dka->startno )
        printf( "*%3d", i );
     else
        printf( "%4d", i );
     for ( j = 0; j < dka->br_ulaza; j++ )
     {
        printf( "%4d", dka->prelaz[i][j] );
     }
     printf("%4d\n", dka->prihvata[i] );
   }
   printf("\n\n");
}

void inicijalna( Particija prva, DKA *dka )
{
    int i;

    for ( i = 0; i < dka->br_stanja; i++ )
    {
      if ( dka->prihvata[i] )
        prva[i] = 1;
      else
```

```
            prva[i] = 0;
        }
    }

    int jednake( Particija stara, Particija nova, int br_stanja)
    {
        int i;
        int jednake=1;

        for ( i = 0; i < br_stanja; i++ )
        {
          if ( stara[i] != nova[i] )
          {
            jednake = 0;
            stara[i] = nova[i];
          }
        }
        return jednake;
    }

void napravi_novu_particiju( DKA *dka, int ulazni_simbol, Particija stara, Particija nova)
    {
        int i,j;
        int br_skupova_u_novoj_particiji = 0;

        for ( i=0; i < dka->br_stanja; i++ )
        {
          for( j=0; j < i; j++ )
            if (       stara[i] == stara[j]
               && stara[dka->prelaz[i][ulazni_simbol]] == stara[dka->
                                    prelaz[j][ulazni_simbol]])
              break;

          if ( j == i )
            nova[i] = br_skupova_u_novoj_particiji++;
          else
            nova[i] = nova[j];
        }
    }

    void napravi_min_dka( Particija konacna_p, DKA *orig_dka, DKA *min_dka )
    {
      int orig_stanje,ulaz;
      int max_novo_stanje=-1;

      for ( orig_stanje = 0; orig_stanje < orig_dka->br_stanja; orig_stanje++ )
      {
```

```
        for ( ulaz = 0; ulaz < orig_dka->br_ulaza; ulaz++ )
          min_dka->prelaz[konacna_p[orig_stanje]][ulaz] =
                        konacna_p[orig_dka->prelaz[orig_stanje][ulaz]];

        min_dka->prihvata[konacna_p[orig_stanje]] = orig_dka->prihvata[orig_stanje];
        if ( konacna_p[orig_stanje] > max_novo_stanje )
            max_novo_stanje = konacna_p[orig_stanje];

    }
    min_dka->br_stanja = max_novo_stanje + 1;
    min_dka->br_ulaza = orig_dka->br_ulaza;
    min_dka->startno = konacna_p[orig_dka->startno];
}

void minimizacija(DKA *orig_dka, DKA *min_dka)
{
    Particija stara_particija;
    Particija nova_particija;
    int ulazni_simbol = 0;

    inicijalna( stara_particija, orig_dka );
    ispisi_particiju( stara_particija, orig_dka->br_stanja );

    while ( ulazni_simbol < orig_dka->br_ulaza )
    {
        printf( "Razmatra se ulaz %d.\n", ulazni_simbol );
        napravi_novu_particiju(orig_dka, ulazni_simbol, stara_particija, nova_particija );

        if ( jednake( stara_particija, nova_particija, orig_dka->br_stanja ) )
        {
            ulazni_simbol++;
            printf("Nema promene.\n");
        }
        else
        {
            ulazni_simbol=0;
            ispisi_particiju( nova_particija, orig_dka->br_stanja );
        }
    }

    napravi_min_dka( nova_particija, orig_dka, min_dka);
}

int main()
{
   DKA automat = { 7,  /* broj stanja */
                   3, /* broj ulaznih simbola */
```

```
                1, /* startno stanje */
                { 0, 1, 1, 1, 0, 1, 1 }, /* prihvatanje */
                { { 0, 0, 0 }, /* tabela prelaza */
                  { 2, 3, 4 },
                  { 2, 5, 4 },
                  { 6, 3, 4 },
                  { 2, 5, 4 },
                  { 0, 3, 0 },
                  { 6, 5, 4 }
                }
               };

    DKA min_automat;

    ispisi_dka( &automat);
    minimizacija( &automat, &min_automat );
    ispisi_dka( &min_automat );
    return 0;
}
```

Zadatak 1.7.3

Realizovati metodom rekurzivnog spusta, program koji određuje deterministički konačni automat za proizvoljan regularni izraz opisan sledećom gramatikom koju treba dopuniti atributima i akcionim simbolima.

1. <S> → <U> <U_list> ENDM
2. <U_list> → | <U> <U_list>
3. <U_list> → ε
4. <U> → <C> <C_list>
5. <C_list> → <C> <C_list>
6. <C_list> → ε
7. <C> → <I> <I_list>
8. <I_list> → * <I_list>
9. <I_list> → ε
10. <I> → I
11. <I> → =
12. <I> → (<U> <U_list>)

Operandi u regularnom izrazu su jednoslovni simboli ili simbol = koji označava praznu sekvencu u ulaznom izrazu. Terminal ENDM označava kraj izraza.

Rešenje

Data ulazna gramatika jeste LL(1); selekcioni skupovi smena glase:

SELECT(1)=FIRST(<U>)=FIRST(<C>)=FIRST(<I>)={I,=,(}
SELECT(2)={|}

SELECT(3)=FOLLOW(<U_list>)={ENDM,)}
SELECT(4)={I,=,(}
SELECT(5)=FIRST(<C>)=FIRST(<I>)={I,=,(}
SELECT(6)=FOLLOW(<C_list>)=FOLLOW(<U>)=
=FIRST(<U_list>)∪FOLLOW(<U_list>)={|,ENDM,)}
SELECT(7)={I,=,(}
SELECT(8)={*}
SELECT(9)=FOLLOW(<I_list>)=FOLLOW(<C>)=
=FIRST(<C_list>)∪FOLLOW(<C_list>)=
=FIRST(<C>)∪FOLLOW(<U>)=
=FIRST(<I>)∪FIRST(<U_list>)∪FOLLOW(<U_list>)=
={I,=,(,|,ENDM,)}
SELECT(10)={I}
SELECT(11)={=}
SELECT(12)={(}

U nastavku sledi L atributivno–translaciona gramatika prilagođena metodu rekurzivnog spusta. Atributi naziva null su logičkog tipa i imaju vrednost TRUE ako je čvor sintaksnog stabla koga predstavlja odgovarajući neterminal poništiv. Atributi Fpos i Lpos su tipa "skup pozicija" i predstavljaju skup prvih odnosno poslednjih pozicija za odgovarajući neterminal (koji predstavlja čvor sintaksnog stabla regularnog izraza). Ovo važi za sintetizovane atribute, dok se nasleđeni pojavljuju kod čvorova <U_list>, <C_list> i <I_list> i služe da prenesu odgovarajuće podatke o levom operandu kod gramatičkih pravila 2, 5 i 8. Atribut terminala I je skup pozicija sa jednim elementom – pozicijom odgovarajućeg simbola u izrazu. Akcioni simbol {NEXT_POS} predstavlja poziv procedure koja ažurira vektor sledećih pozicija Npos[] u skladu sa datim ulaznim parametrima; za svaku poziciju i iz Lpos:

Npos[i] = Npos[i] ∪ Fpos

Element vektora Npos[i] predstavlja skup pozicija koje su u sledećoj poziciji u odnosu na poziciju i. Za atribute važi:

$<S>_{Fpos}$	Fpos: sintetizovan
$<U>_{null,Fpos,Lpos}$	null,Fpos,Lpos: sintetizovani
$<U\_list>_{null1,Fpos1,Lpos1,null2,Fpos2,Lpos2}$	null1,Fpos1,Lpos1: nasleđeni null2,Fpos2,Lpos2: sintetizovani
$<C>_{null,Fpos,Lpos}$	null,Fpos,Lpos: sintetizovani
$<C\_list>_{null1,Fpos1,Lpos1,null2,Fpos2,Lpos2}$	null1,Fpos1,Lpos1: nasleđeni null2,Fpos2,Lpos2: sintetizovani
$<I>_{null,Fpos,Lpos}$	null,Fpos,Lpos: sintetizovani
$<I\_list>_{null1,Fpos1,Lpos1,null2,Fpos2,Lpos2}$	null1,Fpos1,Lpos1: nasleđeni null2,Fpos2,Lpos2: sintetizovani

Atributi akcionog simbola su nasleđeni.

1. $\langle S\rangle_{Fpos} \rightarrow \langle U\rangle_{null1,Fpos1,Lpos1} \langle U\_list\rangle_{null1,Fpos1,Lops1,null2,Fpos2,Lpos2}$
   $ENDM_{pos}\{NEXT\_POS\}_{Lpos2,\{pos\}}$

   $$Fpos \leftarrow \begin{cases} Fpos2, & null2 = FALSE \\ Fpos2 \cup \{pos\}, & null2 = TRUE \end{cases}$$

2. $\langle U\_list\rangle_{null1,Fpos1,Lpos1,null2,Fpos2,Lpos2} \rightarrow \mid \langle U\rangle_{null3,Fpos3,Lpos3} \langle U\_list\rangle_{null4,Fpos4,Lpos4,null2,Fpos2,Lpos2}$

   $null4 \leftarrow null1 \text{ OR } null3$
   $Fpos4 \leftarrow Fpos1 \cup Fpos3$
   $Lpos4 \leftarrow Lpos1 \cup Lpos3$

3. $\langle U\_list\rangle_{null1,Fpos1,Lpos1,null2,Fpos2,Lpos2} \rightarrow \varepsilon$

   $null2 \leftarrow null1 \quad Fpos2 \leftarrow Fpos1 \quad Lpos2 \leftarrow Lpos1$

4. $\langle U\rangle_{null,Fpos,Lpos} \rightarrow \langle C\rangle_{null1,Fpos1,Lpos1} \langle C\_list\rangle_{null1,Fpos1,Lpos1,null,Fpos,Lpos}$

5. $\langle C\_list\rangle_{null1,Fpos1,Lpos1,null2,Fpos2,Lpos2} \rightarrow \langle C\rangle_{null3,Fpos3,Lpos3} \{NEXT\_POS\}_{Lpos1,\ Fpos3}$
   $\langle C\_list\rangle_{null4,Fpos4,Lpos4,null2,Fpos2,Lpos2}$

   $null4 \leftarrow null1 \text{ AND } null3$

   $$Fpos4 \leftarrow \begin{cases} Fpos1, & null1 = FALSE \\ Fpos1 \cup Fpos3, & null1 = TRUE \end{cases}$$

   $$Lpos4 \leftarrow \begin{cases} Lpos1, & null1 = FALSE \\ Lpos1 \cup Lpos3, & null1 = TRUE \end{cases}$$

6. $\langle C\_list\rangle_{null1,Fpos1,Lpos1,null2,Fpos2,Lpos2} \rightarrow \varepsilon$

   $null2 \leftarrow null1 \quad Fpos2 \leftarrow Fpos1 \quad Lpos2 \leftarrow Lpos1$

7. $\langle C\rangle_{null,Fpos,Lpos} \rightarrow \langle I\rangle_{null1,Fpos1,Lpos1} \langle I\_list\rangle_{null1,Fpos1,Lpos1,null,Fpos,Lpos}$

8. $\langle I\_list\rangle_{null1,Fpos1,Lpos1,null2,Fpos2,Lpos2} \rightarrow \{NEXT\_POS\}_{Lpos1,Fpos1} * \langle I\_list\rangle_{TRUE,Fpos1,Lpos1,null2,Fpos2,Lpos2}$

9. $\langle I\_list\rangle_{null1,Fpos1,Lpos1,null2,Fpos2,Lpos2} \rightarrow \varepsilon$

   $null2 \leftarrow null1 \qquad Fpos2 \leftarrow Fpos1 \qquad Lpos2 \leftarrow Lpos1$

10. $\langle I\rangle_{null,Fpos,Lpos} \rightarrow I_p$

    $null \leftarrow FALSE \qquad (Fpos,Lpos) \leftarrow p$

11. $\langle I\rangle_{null,Fpos,Lpos} \rightarrow =$

    $null \leftarrow TRUE \qquad (Fpos,Lpos) \leftarrow \varnothing$

12. $\langle I\rangle_{null,Fpos,Lpos} \rightarrow (\langle U\rangle_{null1,Fpos1,Lpos1} \langle U\_list\rangle_{null1,Fpos1,Lpos1,null,Fpos,Lpos})$

Sledi procesor na bazi rekurzivnog spusta realizovan na VAX Pascalu. Leksički analizator realizuje procedura ADVANCE koja putem globalnih promenljivih INCLASS (ulazni simbol) i INATT (atribut ulaznog simbola) vraća podatke sintaksnom procesoru. Uzajamno rekurzivne procedure PROCS, PROCU, PROCUlist, PROCC, PROCClist, PROCI i PROCIlist realizuju sintaksni analizator. Procedura NEXTPOS odgovara istoimenom akcionom simbolu. Procedura ERR služi za prijavu greške u izrazu. U okviru glavnog programa inicira se analiza i vrši konstrukcija tabele prelaza automata u skladu sa odgovarajućim algoritmom (Zadatak 1.4.3).

```
program DetMach (input,output);

label 1; {an exit if an error occurs}

const MAXPOS    = 100; {Max. number of positions (number of  ALL
                  symbols in a regular expression); depends
                  on  implemented  max. set  size;  also
                  determines program memory(run-time stack)
                  requirements}
    ENDM     = '#'; {the end-marker for finite-state machine}
    EPS      = '='; {a symbol for empty sequence (epsilon)}

type  Positions = 0..MAXPOS;
    SetOfPos  = set of Positions;
    Inputs    = '0'..'Z';
    Ptr       = ^State;
    State     = record s: SetOfPos; next: Ptr end;
    String    = varying [30] of char;

var   INCLASS: char; {a class part of a current input symbol}
    INATT  : SetOfPos; {a value part of a current input}
    pos    : Positions; {positions counter}
    ptos   : array [Positions] of Inputs;
           {shows what symbol is on what pos.}
    inpSym : array [Inputs] of Boolean;
           {inpSym[c]=TRUE if c appears in reg. expr.}
    Npos   : array [Positions] of SetOfPos;
           { represents function Next Position }

    nrows  : integer; {a no.of computed transition-table rows}
    HEAD   : Ptr; {a pointer to a head of a state list}
    LAST   : Ptr; {a pointer to a last state on the list}
    cstate : Ptr; {a ptr. to a current state on the list}
    row    : array [Inputs] of SetOfPos;
           {a current row of a transition table }
    endyes : Boolean; {a flag if the machine accepts on ENDM
                  in a current transition-table row }

    i      : Inputs; {input symbols counter }
    j      : Positions; {positions counter }
    k      : integer; {states counter}
    sc     : Ptr; {the another kind of states counter}
    newSt  : Boolean; {a flag if a state already exists}

   procedure NextPos(Lpos,Fpos: SetOfPos);
   { for every position i in Lpos this procedure adds all the
    Fpos positions in a next position [i] set }
```

```
var i: Positions;
begin
   for i:=0 to pos do
      if i in Lpos then Npos[i]:=Npos[i]+Fpos
end;

procedure ERR(exp: String); {the error handler}
begin
   write ('^ Found ');
       if INCLASS in ['I',EPS] then write ('symbol')
       else if INCLASS=ENDM then write ('end of expr.')
       else write (INCLASS);
       writeln (' when expected ',exp);
       goto 1
end;

procedure ADVANCE;
{reads the regular expresion character by character and
 updates following variables according to their meanings:
 INCLASS,INATT,pos, ptos[],inpSym[],Npos[]}
begin {ADVANCE}
   if not EOLN then
   begin
      read(INCLASS);
      write (' '); {needed for the error pointer}
      if INCLASS in ['a'..'z'] {'A' and 'a' are the same}
      then INCLASS:=chr(ord('A')-ord('a')+ord(INCLASS));
      if (INCLASS in ['A'..'Z','0'..'9']) then
      begin
         INATT:=[pos];
         ptos[pos]:=INCLASS;
         inpSym[INCLASS]:=TRUE;
         Npos[pos]:=[];
         INCLASS:='I';
         if pos<MAXPOS then pos:=pos+1
         else
         begin
            writeln ('^ Reg. expr. too long.');
            goto 1
         end;
      end
      else if not (INCLASS in ['(',')','|','*',EPS]) then
      begin
         writeln ('^ Illegal symbol in reg. expr.');
         goto 1
      end;
   end
```

```
   else begin INCLASS:=ENDM; write (' '); Npos[pos]:=[] end
end; {advance}

procedure PROCUlist(null1: Boolean; Fpos1,Lpos1: SetOfPos; var null2: Boolean;
                    var Fpos2,Lpos2: SetOfPos); Forward;

procedure PROCU(var null: Boolean; var Fpos,Lpos: SetOfPos); Forward;

procedure PROCClist(null1: Boolean; Fpos1,Lpos1: SetOfPos; var null2: Boolean;
                    var Fpos2,Lpos2: SetOfPos); Forward;

procedure PROCC(var null: Boolean; var Fpos,Lpos  : SetOfPos); Forward;

procedure PROCIlist(null1: Boolean; Fpos1,Lpos1: SetOfPos; var null2: Boolean;
                    var Fpos2,Lpos2: SetOfPos); Forward;

procedure PROCI(var null: Boolean; var Fpos,Lpos  : SetOfPos); Forward;

procedure PROCS(var Fpos: SetOfPos);
var null1,null2: Boolean;
   Fpos1,Fpos2: SetOfPos;
   Lpos1,Lpos2: SetOfPos;
begin
   if INCLASS in ['I',EPS,'('] then
   begin
      PROCU(null1,Fpos1,Lpos1);
      PROCUlist(null1,Fpos1,Lpos1,null2,Fpos2,Lpos2);
      if INCLASS<>ENDM then ERR('end of expr.');
      if null2 then Fpos:=Fpos2+[pos] else Fpos:=Fpos2;
      { ADVANCE not needed }
      NextPos(Lpos2,[pos])
   end
   else ERR('symbol or (')
end;

procedure PROCUlist;
var null3,null4: Boolean;
   Fpos3,Fpos4: SetOfPos;
   Lpos3,Lpos4: SetOfPos;
begin
   if INCLASS='|' then
   begin
      ADVANCE;
      PROCU(null3,Fpos3,Lpos3);
      null4:=null1 or null3;
      Fpos4:=Fpos1+Fpos3;
      Lpos4:=Lpos1+Lpos3;
```

```
      PROCUlist(null4,Fpos4,Lpos4,null2,Fpos2,Lpos2)
   end
   else if INCLASS in [')',ENDM] then
   begin null2:=null1; Fpos2:=Fpos1; Lpos2:=Lpos1 end
   else ERR('|,end of expr. or )')
end;

procedure PROCU;
var null1: Boolean;
   Fpos1: SetOfPos;
   Lpos1: SetOfPos;
begin
   if INCLASS in ['I',EPS,'('] then
   begin
      PROCC(null1,Fpos1,Lpos1);
      PROCClist(null1,Fpos1,Lpos1,null,Fpos,Lpos);
   end
   else ERR('symbol or (')
end;

procedure PROCClist;
var null3,null4: Boolean;
   Fpos3,Fpos4: SetOfPos;
   Lpos3,Lpos4: SetOfPos;
begin
   if INCLASS in ['I',EPS,'('] then
   begin
      PROCC(null3,Fpos3,Lpos3);
      NextPos(Lpos1,Fpos3);
      null4:=null1 and null3;
      if null1 then Fpos4:=Fpos1+Fpos3 else Fpos4:=Fpos1;
      if null3 then Lpos4:=Lpos1+Lpos3 else Lpos4:=Lpos3;
      PROCClist(null4,Fpos4,Lpos4,null2,Fpos2,Lpos2)
   end
   else if INCLASS in ['|',')',ENDM] then
   begin null2:=null1; Fpos2:=Fpos1; Lpos2:=Lpos1 end
   else ERR('symbol,(,),| or end of expr.')
end;

procedure PROCC;
var null1: Boolean;
   Fpos1: SetOfPos;
   Lpos1: SetOfPos;
begin
   if INCLASS in ['I',EPS,'('] then
   begin
      PROCI(null1,Fpos1,Lpos1);
```

```
            PROCIlist(null1,Fpos1,Lpos1,null,Fpos,Lpos);
        end
        else ERR('symbol or (')
    end;

    procedure PROCIlist;
    begin
        if INCLASS='*' then
        begin
            NextPos(Lpos1,Fpos1);
            ADVANCE;
            PROCIlist(TRUE,Fpos1,Lpos1,null2,Fpos2,Lpos2)
        end
        else if INCLASS in ['I',EPS,'(',')','|',ENDM] then
        begin null2:=null1; Fpos2:=Fpos1; Lpos2:=Lpos1 end
        else begin writeln ('FATAL ERROR.'); goto 1 end
    end;

    procedure PROCI;
    var null1: Boolean;
        Fpos1: SetOfPos;
        Lpos1: SetOfPos;
    begin
        if INCLASS='I' then
        begin null:=FALSE; Fpos:=INATT; Lpos:=INATT; ADVANCE end
        else if INCLASS=EPS then
        begin null:=TRUE; Fpos:=[]; Lpos:=[]; ADVANCE end
        else if INCLASS='(' then
        begin
            ADVANCE;
            PROCU(null1,Fpos1,Lpos1);
            PROCUlist(null1,Fpos1,Lpos1,null,Fpos,Lpos);
            if INCLASS<>')' then ERR(')');
            ADVANCE
        end
        else ERR('symbol or (')
    end;

begin { main }
    pos:=0;
    for i:='0' to 'Z' do inpSym[i]:=FALSE;
    writeln; write (' '); ADVANCE;
    new(cstate);PROCS(cstate^.s);
    {if not EOLN then begin writeln ('}
    cstate^.next:=NIL; HEAD:=cstate; LAST:=cstate;
    writeln; write (' ':5);
    for i:='0' to 'Z' do if inpSym[i] then write (i:4);
```

```
    writeln ('  -|'); write (' ':5);
    for i:='0' to 'Z' do if inpSym[i] then write ('----');
    writeln ('------');
    nrows:=0;
    repeat
       write (nrows:5,'|');
        nrows:=nrows+1;
       for i:='0' to 'Z' do row[i]:=[];
       endyes:=FALSE;
       for j:=0 to pos-1 do
          if j in cstate^.s
          then row[ptos[j]]:=row[ptos[j]]+Npos[j];
       if pos in cstate^.s then endyes:=TRUE;
       { the next is to mark states with numbers and add new
         states to the list }
       for i:='0' to 'Z' do
          if inpSym[i] and (row[i]<>[]) then
          begin
             k:=0; sc:=HEAD; newSt:=TRUE;
             while (sc<>NIL) and newSt do
                if sc^.s=row[i]
                then begin newSt:=FALSE; write(k:4) end
                else begin k:=k+1; sc:=sc^.next end;
             if newSt then
             begin
                new(LAST^.next);
                LAST:=LAST^.next;
                LAST^.s:=row[i]; LAST^.next:=NIL;
                write (k:4)
             end
          end
          else
             if inpSym[i] and (row[i]=[]) then write('  NO');
       {end for}
       if endyes then write (' YES') else write ('  NO');
       writeln ('|');
       cstate:=cstate^.next
    until cstate=NIL;
    write (' ':5);
    for i:='0' to 'Z' do if inpSym[i] then write ('----');
    writeln ('------');
1:  writeln
end.
```

Sledi nekoliko primera rezultata izvršavanja programa za različite ulazne podatke.

Primer 1

```
abcd|=

       A   B   C   D  -|
    ----------------------
   0|   1  NO  NO  NO YES|
   1|  NO   2  NO  NO  NO|
   2|  NO  NO   3  NO  NO|
   3|  NO  NO  NO   4  NO|
   4|  NO  NO  NO  NO YES|
    ----------------------
```

Primer 2

```
a*b*c*d*

       A   B   C   D  -|
    ----------------------
   0|   0   1   2   3 YES|
   1|  NO   1   2   3 YES|
   2|  NO  NO   2   3 YES|
   3|  NO  NO  NO   3 YES|
    ----------------------
```

Primer 3

```
(ab))
    ^ Found ) when expected end of expr.
```

2. Gramatike

2.1. Krajnje levo i krajnje desno izvođenje u gramatici

Zadatak 2.1.1

Koja od sledećih sekvenci pripada jeziku opisanom zadatom gramatikom sa startnim neterminalom <S>? Za svaki od slučajeva dati levo izvođenje, desno izvođenje i stablo izvođenja.

a) aacb

b) aacbccb

1. <S> $\rightarrow$ a <A> c <B>

2. <S> $\rightarrow$ <B> d <S>

3. <B> $\rightarrow$ a <S> c <A>

4. <B> $\rightarrow$ c <A> <B>

5. <A> $\rightarrow$ <B> a <B>

6. <A> $\rightarrow$ a <B> c

7. <A> $\rightarrow$ a

8. <B> $\rightarrow$ b

Analiza problema

Bezkontektsna gramatika G opisana je uređenom četvorkom (V_N, V_T, P, <S>) gde:

- V_N predstavlja skup neterminalnih simbola, u konkretnom slučaju V_N = {<S>, <A>, <B>}.
- V_T predstavlja skup terminalnih simbola, u konkretnom slučaju V_T = {a, b, c, d}. Terminalni i neterminalni simboli se jednim imenom nazivaju gramatički simboli. $V = V_T \cup V_N$ predstavlja skup gramatičkih simbola.

- P predstavlja skup smena (produkcija) oblika <X> $\rightarrow \alpha$, gde <X> na levoj strani smene predstavlja proizvoljan neterminalni simbol, a α na desnoj strani smene predstavlja proizvoljan niz od nula ili više gramatičkih simbola.
- <S> $\in V_N$ predstavlja startni neterminal.

Gramatičko izvođenje, u oznaci

$$\beta \langle X \rangle \gamma \Rightarrow \beta \alpha \gamma$$

predstavlja primenu neke gramatičke smene <X> $\rightarrow \alpha$ da bi se u nekom nizu gramatičkih simbola (takozvanoj sentencijalnoj formi) β <X> γ zamenilo pojavljivanje leve strane smene odgovarajućom desnom stranom. Ako se izvođenje posmatra kao relacija na skupu sentencijalnih formi V^* (zvezdastom zatvaranju skupa V, odnosno skupu svih sekvenci simbola iz V dužine 0 ili više), onda se tranzitivno zatvaranje ove relacije, u oznaci $\overset{+}{\Rightarrow}$ naziva izvođenje u jednom ili više koraka, a refleksivno-tranzitivno zatvaranje relacije izvođenja, u oznaci $\overset{*}{\Rightarrow}$, naziva se izvođenje u nula ili više koraka.

Jezik opisan gramatikom G, u oznaci L(G) predstavlja skup svih sentenci (nizova terminalnih simbola) koje se mogu izvesti iz startnog neterminala <S> u nula ili više koraka izvođenja:

$$L(G) = \{x \mid \langle S \rangle \overset{*}{\Rightarrow} x, x \in V_T^*\}$$

Rešenje

a)

Sekvenca aacb pripada jeziku date gramatike ako i samo ako je moguće izvođenje sekvence aacb iz startnog neterminala <S>. U prvom koraku izvođenja, u obzir za zamenu <S> dolaze:

- smena 1, čija desna strana počinje terminalom a, kao i
- smena 2. čija desna strana počinje neterminalom <B>, koji se može kasnije zameniti primenom smene 3. čija desna strana takođe počinje terminalom a.

Uočimo međutim da desna strana smene 2. sadrži terminal d koji se ne pojavljuje u datoj gramatici, pa prema tome otpada iz daljeg razmatranja. U izvođenju, prema tome, primenjujemo smenu 1:

$$\underset{\uparrow}{\langle S \rangle} \Rightarrow a \langle A \rangle c \langle B \rangle$$
$$1$$

Sada se vidi da neterminal <A> treba zameniti primenom smene 7, a neterminal <B> primenom smene 8 (ostale smene eliminišu se iz razmatranja jer sadrže i druge terminale osim potrebnih):

$$\underset{\uparrow}{\langle S \rangle} \Rightarrow a \underset{\uparrow}{\langle A \rangle} c \underset{\uparrow}{\langle B \rangle} \Rightarrow a\,a\,c \underset{\uparrow}{\langle B \rangle} \Rightarrow a\,a\,c\,b$$
$$1 \qquad 7 \qquad 8$$

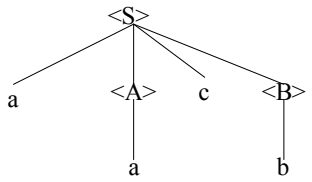
Pošto je u svakom koraku izvođenja zamenjivan prvi neterminal s leve strane u svakoj sentencijalnoj formi, dobijeno izvođenje naziva se levim izvođenjem (engl. leftmost derivation), što se može posebno naznačiti na sledeći način:

$$\langle S \rangle \Rightarrow_{lm} a \langle A \rangle c \langle B \rangle \Rightarrow_{lm} a\,a\,c \langle B \rangle \Rightarrow_{lm} a\,a\,c\,b$$

Ukoliko se u svakom koraku izvođenja zamenjuje krajnje desni neterminal u sentencijalnoj formi, dobija se desno izvođenje (engl. rightmost derivation):

$$\langle S\rangle \Rightarrow_{rm} a \langle A\rangle c \langle B\rangle \Rightarrow_{rm} a \langle A\rangle c\ b \Rightarrow_{rm} a\ a\ c\ b$$

Stablo izvođenja je grafička predstava izvođenja. Čvorovi stabla odgovaraju gramatičkim simbolima. Izvođenje $\langle X\rangle \Rightarrow Y\ Z\ V$ se u stablu izvođenja predstavlja na taj način što čvor $\langle X\rangle$ ima čvorove naslednike Y, Z i V navedenim redosledom. U konkretnom slučaju, navedenim izvođenjima sekvence aacb iz neterminala $\langle S\rangle$ odgovara stablo izvođenja na Sl. 2.1.1.



Sl. 2.1.1

b)

Razmotrimo izvođenje sekvence aacbccb iz startnog neterminala $\langle S\rangle$. U prvom koraku mora se primeniti 1. smena (jer 2. smena sadrži terminal d koji se ne pojavljuje u ulaznoj sekvenci):

$$\langle S\rangle \Rightarrow a \langle A\rangle c \langle B\rangle$$
$$\uparrow$$
$$1$$

Sada moramo razmotriti sve varijante uparivanja terminala c iz sentencijalne forme $a \langle A\rangle c \langle B\rangle$ sa ulazom aacbccb. Ukoliko se uparuje sa prvom pojavom simbola c s leve strane u ulazu, treba da važi:

1. $\langle A\rangle \overset{*}{\Rightarrow} a$, i
2. $\langle B\rangle \overset{*}{\Rightarrow} bccb$.

Tačka 1. sledi na osnovu 7. smene. U razmatranju tačke 2. otpadaju 3. smena jer sadrži a, 4. smena jer počinje sa c i 8. smena jer sadrži samo b. Znači tačka 2. nije ispunjena.

Razmotrimo sada varijantu kada se c u sekvenci $a \langle A\rangle c \langle B\rangle$ uparuje sa drugom pojavom simbola c u ulaznom nizu aacbccb. Tada treba da važi:

1'. $\langle A\rangle \overset{*}{\Rightarrow} acb$, i

2'. $\langle B\rangle \overset{*}{\Rightarrow} cb$.

Nijedna od smena 5, 6 i 7 ne može se primeniti u izvođenju 1', prema tome ova varijanta otpada.

Poslednja varijanta je uparivanje simbola c u sekvenci a <A> c <B> sa poslednjom pojavom simbola c u ulaznom nizu aacbccb.

Tada treba da važi:

1". $\langle A \rangle \overset{*}{\Rightarrow}$ acbc, i

2". $\langle B \rangle \overset{*}{\Rightarrow}$ b.

U izvođenju 1" u prvom koraku može se primeniti isključivo smena 6. $\langle A \rangle \rightarrow$ a <B> c, što znači da treba da važi $\langle B \rangle \rightarrow$ cb. Međutim, neterminal <B> ne generiše nijednu sekvencu dužine 2 simbola, tako da i ova varijanta otpada.

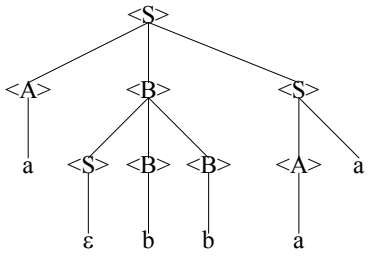
Zaključujemo da ne važi $\langle S \rangle \overset{*}{\Rightarrow}$ aacbccb, odnosno da pomenuta sekvenca ne pripada jeziku date gramatike.

Diskusija

Postoje formalne metode za utvrđivanje pripadnosti sentence nekom jeziku. Parser je procedura koja na osnovu date gramatike utvrđuje, za zadatu sekvencu, pripadnost sekvence jeziku opisanom datom gramatikom i njeno izvođenje iz startnog neterminala. U sledećim poglavljima razmatraju se različiti metodi konstrukcije parsera.

Zadatak 2.1.2

Jedna bezkontekstna gramatika generiše stablo izvođenja prikazano na Sl. 2.1.2.



Sl. 2.1.2

a) Odrediti levo izvođenje koje odgovara datom stablu.

b) Koliko različitih izvođenja odgovara ovom stablu?

Rešenje

a)

Dato je stablo izvođenja za sentencu abbaa što se jednostavno dobija čitanjem listova pri obilasku stabla. Za jedno stablo izvođenja postoji jedinstvena sentenca koji mu odgovara, ali obrnuto ne mora da važi kao u slučaju dvosmislenih gramatika koje nisu pogodne za programske jezike.

Moguće je identifikovati sledeće gramatičke smene korišćene u konstrukciji datog stabla:

1. <S> → <A> <B> <S>
2. <S> → <A> a
3. <S> → ε
4. <A> → a
5. <B> → <S> <B> <B>
6. <B> → b

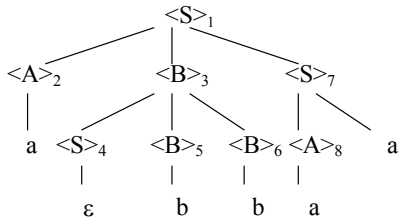
Traženo levo izvođenje glasi:

$$\underset{1}{\Rightarrow_{lm}} \underset{2}{\Rightarrow_{lm}} \underset{3}{\Rightarrow_{lm}} \underset{4}{\Rightarrow_{lm}} \underset{5}{\Rightarrow_{lm}} \Rightarrow_{lm} \underset{6}{\Rightarrow_{lm}} \underset{7}{\Rightarrow_{lm}} \underset{8}{\Rightarrow_{lm}}$$

<S> ⇒ lm <A><B><S> ⇒ lm a<B><S> ⇒ lm a<S><B><B><S> ⇒ lm a<B><B><S>
⇒ lm ab<B><S> ⇒ lm abb<S> ⇒ lm abb<A>a ⇒ lm abbaa

b)

U izvođenju sentence abbaa postoji ukupno 8 primena smena (na Sl. 2.1.3 svakom unutrašnjem čvoru stabla odgovara po jedna primena smene; primene su numerisane indeksima uz čvorove). Kada bi redosled primena smena bio proizvoljan, broj različitih izvođenja bio bi jednak broju permutacija 8 elemenata, a to je 8!. Međutim, prvo se uvek mora primeniti smena koja odgovara čvoru stabla, a takođe primene 4., 5. i 6. ne mogu pre primene 3. kao što ni primena 8. ne može pre primene 7. Kada se iz razmatranja eliminišu permutacije koje ne zadovoljavaju ova ograničenja, ostaje 630 permutacija, što ujedno predstavlja broj različitih izvođenja koje odgovaraju datom stablu.



Sl. 2.1.3

Zadatak 2.1.3

Date su dve ekvivalentne gramatike G_1 i G_2:

$G_1 = (\{<S>, <A>, <B>\}, \{a, b\}, P_1, <S>)$ i

$G_2 = (\{<S>, <A>, <B>\}, \{a, b\}, P_2, <S>)$

gde su P_1 i P_2 dati sa:

P_1	P_2
1. <S> → b<A>	9. <S> → a<B><S>
2. <S> → a<B>	10. <S> → a<B>
3. <A> → a	11. <S> → b<A><S>
4. <B> → b	12. <S> → b<A>
5. <A> → a<S>	13. <A> → b<A><A>
6. <B> → b<S>	14. <A> → a
7. <A> → b<A><A>	15. <B> → a<B><B>
8. <B> → a<B><B>	16. <B> → b

Koja je od ovih gramatika dvosmislena?

Analiza problema

Bezkontekstna gramatika $G = (V_N, V_T, P, <S>)$ je dvosmislena ako postoji bar jedna reč u L(G) sa dva ili više krajnje levih izvođenja. Gramatika čije su smene date skupom P_1 se analizira i sa drugog aspekta (Zadatak 2.2.4).

Rešenje

Testirajmo najpre gramatiku G_1. Pođimo od smene 1. i sprovedimo krajnje levo izvođenje (redni brojevi primenjenih smena navedeni su iznad oznake izvođenja):

$$<S> \underset{G_1}{\overset{1}{\Rightarrow}} b\,<A> \underset{G_1}{\overset{7}{\Rightarrow}} bb\,<A><A> \underset{G_1}{\overset{3}{\Rightarrow}} bba\,<A> \underset{G_1}{\overset{7}{\Rightarrow}} bbab\,<A><A>$$

Izvršimo drugačije krajnje levo izvođenje:

$$<S> \underset{G_1}{\overset{1}{\Rightarrow}} b\,<A> \underset{G_1}{\overset{7}{\Rightarrow}} bb\,<A><A> \underset{G_1}{\overset{5}{\Rightarrow}} bba\,<S><A> \underset{G_1}{\overset{1}{\Rightarrow}} bbab\,<A><A>$$

Prvo i drugo krajnje levo izvođenje daju istu sentencijalnu formu iz koje se može dobiti niz istih sekvenci. Prvo izvođenje je različito od drugog jer u prvom imamo sentencu 1–7–3–7 a u drugom 1–7–5–1. Dakle: gramatika G_1 je dvosmislena.

Testirajmo sada gramatiku G_2. Dovoljno je da ispitamo smene 9. i 10. jer su smene 11. i 12. u pogledu postavljenog cilja slične. Prvo izvođenje daje:

$$<S> \underset{G_2}{\overset{9}{\Rightarrow}} a\,<B><S> \underset{G_2}{\overset{15}{\Rightarrow}} aa\,<B><B><S> \underset{G_2}{\overset{16}{\Rightarrow}} aab\,<B><S> \underset{G_2}{\overset{16}{\Rightarrow}} aabb\,<S> = w\,<S>$$

Drugo izvođenje daje:

$$<S> \underset{G_2}{\overset{9}{\Rightarrow}} a\,<B><S> \underset{G_2}{\overset{15}{\Rightarrow}} aa\,<B><B><S> \underset{G_2}{\overset{16}{\Rightarrow}} aab\,<B><S> \underset{G_2}{\overset{15}{\Rightarrow}} aaba\,<B><B><S>$$

Odavde sledi da se polazeći od smene 9. ne mogu dobiti iste sentencijalne forme sa različitim krajnje levim izvođenjem zato što se moraju da primene samo dve smene za <B> (15. i 16.) koje se razlikuju po početnim neterminalima. Pretpostavimo krajnje levo izvođenje:

$$<S> \overset{*}{\underset{G}{\Rightarrow}} w\,<S>$$

Ovim se stvar vraća na početak. Ako stavimo $<S> \rightarrow a<B><S>$, taj slučaj je već razmatran. Ako stavimo $<S> \rightarrow a<B>$, analiza je ista jer se i ovde mogu primeniti samo smene 15. i 16. Ako bismo stavili $<S> \rightarrow b<A><S>$ ili $<S> \rightarrow b<A>$, dobija se isto po principu simetrije. Dakle, zaključak je da gramatika G_2 nije dvosmislena.

Zadatak 2.1.4

Ako gramatika G sadrži smene koje su levo i desno rekurzivne sa istim simbolima na levoj strani, onda ona mora da bude dvosmislena. Dokazati.

Analiza problema

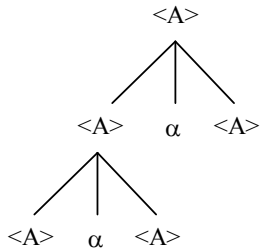
Gramatika je dvosmislena ako za datu sekvencu $w \in L(G)$ ima dva ili više stabla izvođenja. S obzirom da nije data konkretna gramatika, utvrđivanje dvosmislenosti se može formulisati i na sledeći način: gramatika je dvosmislena ako postoje dva ili više stabla izvođenja koje proizvode istu sentencijalnu formu. Jasno je da se iz istih sentencijalnih formi može dobiti ista sekvenca $w \in L(G)$.

Rešenje

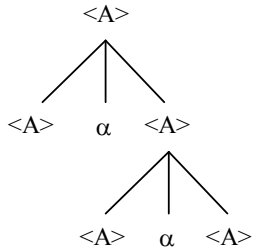
Prema postavci zadatka, u gramatici mora da postoji smena oblika

$$<A> \rightarrow <A>\alpha<A>.$$

Očigledno je da od ove smene mogu da se proizvedu dva ili više podstabala istih sentencijalnih formi, Sl. 2.1.4 i Sl. 2.1.5.



Sl. 2.1.4: Sentencijalna forma: <A>α<A>α<A>



Sl. 2.1.5: Sentencijalna forma: <A>α<A>α<A>

Zadatak 2.1.5

Dati primer gramatike kod koje svaka sekvenca u jeziku ima beskonačan broj stabala izvođenja.

Analiza problema

Jedan mogući pristup rešavanju ovog problema je da se dobije sentencijalna forma čija dužina može da se menja od 0 do ∞ i koja bi se sastojala samo od jednog neterminala, startnog simbola <S>. Uvođenjem prazne smene <S> → ε moglo bi se kontrolisati dobijanje svake sekvence preko beskonačnog broja stabala izvođenja.

Rešenje

Polazeći od ideje izložene u analizi problema, jedna gramatika G koja ima beskonačan broj stabala izvođenja za svaku sekvencu mogla bi da bude oblika

1. <S>→ <S><S>
2. <S>→ a
3. <S>→ ε

Prvom i trećom smenom se obezbeđuje generisanje sentencijalne forme čija dužina može da se menja od 0 do ∞, a drugom i trećom smenom se može kontrolisati dobijanje neke sekvence na beskonačan broj načina. Jezik generisan gramatikom G je

$$L(G) = a^*.$$

Zadatak 2.1.6

Dati postupak kojim se utvrđuje da data gramatika generiše beskonačan broj sekvenci.

Analiza problema

Kao osnova rešenja zadatka može da posluži sledeća teorema (videti takođe Zadatak 2.2.7):

Neka je L bezkontekstni jezik. Postoje konstante r i s koje zavise samo od L, takve da za neku reč z u L čija je dužina $|z| > r$, z se može napisati kao z = uvwxy, gde je $|vwx| \leq s$ i v i x nisu oboje prazni, tako da je za svaki ceo broj $i \geq 0$, uv^iwx^iy u L.

Rešenje

Neka su r i s konstante iz navedene teoreme. Ako je z u L i $|z| > r$, tada se z može napisati kao uvwxy, gde za svako $i \geq 0$, $uv^iwx^iy \in L$. S obzirom da **i** nije ograničeno s gornje strane, tada ako postoji reč dužine veće ili jednake r, jezik L je beskonačan.

Pretpostavimo da je L beskonačan. Tada postoje proizvoljno duge reči u L, posebno reči dužine veće od r + s. Ova reč se može napisati kao:

$$uv^iwx^iy,$$

gde je $|vwx| \leq s$, $|v| + |x| > 0$, a uv^iwx^iy je u L za svako $i \geq 0$. Posebno, $uwy \in L$, i $|uwy| < |uvwxy|$. Takođe, važe $|uwy| > r$ jer je:

$$|uvwxy| > r + s$$

$$|vwx| + |uwy| = |uvwxy| + |w|$$

$$|uwy| = |uvwxy| - |vwx| + |w|$$

odakle sledi da je:

$$|uwy| > r,$$

Ako je $|uwy| > r + s$, postupak treba ponavljati dok se ne nađe eventualno reč u L dužine l, $r < l \leq r + s$. Stoga je L jezik sa beskonačnim brojem reči onda i samo onda ako sadrži reč dužine l.

Primer: Data je bezkontekstna gramatika G u normalnoj formi Čomskog:

$$\langle A \rangle \rightarrow \langle B \rangle\langle C \rangle$$

$$\langle B \rangle \rightarrow \langle B \rangle\langle A \rangle$$

$$\langle C \rangle \rightarrow \langle B \rangle\langle A \rangle$$

<A> → a

<B> → b

Proveriti da li je jezik generisan ovom gramatikom beskonačan.

Pošto gramatika ima tri neterminala, neka je $r = 2^{k-1}$, i $s = 2^k$, gde je k jednako broju neterminala. U ovom primeru k=3. Dakle, r = 4 i s = 8. Pronađimo reč u datoj gramatici čija je dužina veća od 12. Neka je to reč

Z = ba␣bbabba␣ba␣␣bba

odnosno u = ba, v = bbabba, w = ba, x = ε, y = bba.

|Z| = 13 > r + s = 12

|vwx| = 8 ≤ s = 8

|uwy| = 7

4 < 7 < 12

Dakle, jezik L(G) sadrži beskonačan broj sekvenci.

2.2. Nalaženje formalnog i neformalnog opisa jezika na osnovu gramatike i obrnuto

Zadatak 2.2.1

Naći bezkontekstnu gramatiku za izraze λ-računa koji su opisani sledećom definicijom:

Izraz λ-računa je:

- promenljiva, ili
- simbol λ iza čega sledi promenljiva iza koje sledi izraz, ili
- dva izraza odvojena zarezom unutar zagrada

Rešenje

Izraz će biti opisan neterminalom <E>, koji je ujedno i startni neterminal. Skup terminalnih simbola je V_T = {λ variable ( ) , }. Gramatičke smene pišemo na osnovu neformalnog opisa:

1. <E> → variable
2. <E> → λ variable <E>
3. <E> → (<E> , <E>)

Zadatak 2.2.2

Opisati jezik koji se generiše pomoću sledeće gramatike sa startnim simbolom <S>.

1. $<S> \rightarrow 10<S>0$

2. $<S> \rightarrow a<A>$

3. $<A> \rightarrow b<A>$

4. $<A> \rightarrow a$

Rešenje

Razmotrimo najpre skup sentenci koje se mogu izvesti iz neterminala <A>. Višestrukim primenama 3., pa zatim 4. smene dobija se:

$$<A> \overset{*}{\Rightarrow} b^m a, \qquad m \geq 0$$

Višestrukim primenama 1. smene, pa zatim primenom 2. smene dobija se:

$$<S> \overset{*}{\Rightarrow} (10)^n a <A> 0^n, \qquad n \geq 0$$

Kada u obzir uzmemo sentence koje je moguće generisati iz <A>, dobijamo konačno rešenje:

$$L(G) = \{(10)^n a\, b^m a\, 0^n \mid m, n \geq 0\}$$

Zadatak 2.2.3

Pronaći bezkontekstne gramatike koje generišu sledeće jezike:

a) $\{1^n 0^m\}$ $\quad n > m > 0$

b) $\{1^n 0^n 1^m 0^m\}$ $\quad m, n \geq 0$

c) $\{1^n 0^m 1^m 0^n\}$ $\quad m, n \geq 0$

d) $\{1^{3n+2} 0^n\}$ $\quad n \geq 0$

e) $\{1^n 0^m 1^p\}$ $\quad n + p > m \geq 0$

Rešenje

a)

Najkraća sentenca u skupu $\{1^n 0^m\}$, $n > m > 0$ je 110. Duže sentence mogu se dobiti repetitivnim dodavanjem jedne jedinice na početak sentence, ili istovremeno jedne jedinice na početak i jedne nule na kraj:

1. $<S> \rightarrow 110$

2. $<S> \rightarrow 1<S>0$

3. $<S> \rightarrow 1<S>$

b)

Sentence oblika $1^n\,0^n\,1^m\,0^m$, m, n ≥ 0 sastoje se iz dva podniza istog oblika $1^k\,0^k$, k ≥ 0 koji se opisuju posebno uvedenim neterminalom $<S_1>$:

1. $<S> \rightarrow <S_1><S_1>$
2. $<S_1> \rightarrow 1<S_1>0$
3. $<S_1> \rightarrow \varepsilon$

c)

Posebnim neterminalom $<S_1>$ opisujemo podnizove oblika $0^m\,1^m$, m ≥ 0. Sentence dobijamo repetitivnim dodavanjem jedinice na početak i nule na kraj, što je definisano pravilima za startni neterminal <S>:

1. $<S> \rightarrow 1<S>0$
2. $<S> \rightarrow <S_1>$
3. $<S_1> \rightarrow 0<S_1>1$
4. $<S_1> \rightarrow \varepsilon$

d)

Tražena gramatika glasi:

1. $<S> \rightarrow 11$
2. $<S> \rightarrow 111<S>0$

e)

Zadati skup sekvenci $\{1^n\,0^m\,1^p\}$ n + p > m ≥ 0, sastoji se iz dva podskupa:

- sekvence koje ne sadrže 0 i koje predstavljaju niz od jedne ili više jedinica; ove sekvence opisane su smenama 2., 14. i 15. u gramatici koja sledi;
- sekvence oblika $\{1^n\,0^m\,1^p\}$, n + p > m > 0. Da bismo opisali ove poslednje, razmotrimo najpre sekvence oblika $\{1^n\,0^k\,1^p\}$, n + p = k > 0. Ovaj skup sekvenci možemo predstaviti unijom:

  $$\{1^n\,0^n\} \cup \{0^p\,1^p\} \cup \{1^n\,0^n\} \times \{0^p\,1^p\}, \qquad n > 0, p > 0$$

  i opisan je smenama 9., 10, 11, 12. i 13. pri čemu smene 10. i 11. opisuju skup $\{1^n\,0^n\}$, smene 12. i 13. skup $\{0^p\,1^p\}$, a smene 7., 8. i 9. napred navedenu uniju.

  Konačno, svaka sekvenca skupa $\{1^n\,0^m\,1^p\}$, n + p > m > 0 dobija se dodavanjem jedne ili većeg broja jedinica na početak ili na kraj ili sa obe strane neke sekvence skupa $\{1^n\,0^k\,1^p\}$, n + p = k > 0. Dodavanje jedinica na početak sekvence opisano je smenama 3. i 5., dodavanje na kraj smenama 4. i 6. a kombinacijom ovih smena postiže se dodavanje i na početak i na kraj. Smenom 1. dodaje se opisani skup u jezik tražene gramatike.

1. $<S> \rightarrow <S_0>$
9. $<S_1> \rightarrow <S_2><S_3>$

2. $\langle S\rangle \rightarrow \langle S_4\rangle$
3. $\langle S_0\rangle \rightarrow 1\langle S_0\rangle$
4. $\langle S_0\rangle \rightarrow \langle S_0\rangle 1$
5. $\langle S_0\rangle \rightarrow 1\ \langle S_1\rangle$
6. $\langle S_0\rangle \rightarrow \langle S_1\rangle 1$
7. $\langle S_1\rangle \rightarrow \langle S_2\rangle$
8. $\langle S_1\rangle \rightarrow \langle S_3\rangle$
10. $\langle S_2\rangle \rightarrow 10$
11. $\langle S_2\rangle \rightarrow 1\langle S_2\rangle 0$
12. $\langle S_3\rangle \rightarrow 01$
13. $\langle S_3\rangle \rightarrow 0\langle S_3\rangle 1$
14. $\langle S_4\rangle \rightarrow 1\langle S_4\rangle$
15. $\langle S_4\rangle \rightarrow 1$

Zadatak 2.2.4

Data je bezkontekstna gramatika $G = (V_N, V_T, P, \langle S\rangle)$, gde je $V_N = \{\langle S\rangle, \langle A\rangle, \langle B\rangle\}$, $V_T=\{a, b\}$, a P se sastoji od sledećih smena:

1. $\langle S\rangle \rightarrow a\langle B\rangle$
2. $\langle S\rangle \rightarrow b\langle A\rangle$
3. $\langle A\rangle \rightarrow a$
4. $\langle A\rangle \rightarrow a\langle S\rangle$
5. $\langle A\rangle \rightarrow b\langle A\rangle\langle A\rangle$
6. $\langle B\rangle \rightarrow b$
7. $\langle B\rangle \rightarrow b\langle S\rangle$
8. $\langle B\rangle \rightarrow a\langle B\rangle\langle B\rangle$

Treba dokazati da je $L(G) = \{w \mid w \in V_T^+$, i w se sastoji od jednakog broja terminala a i b$\}$.

Analiza problema

Dokazivanje prethodnog stava treba zasnovati na primeni indukcije na dužinu sentence generisane u gramatici G. Iz analize gramatike sledi formulacija sledeće induktivne hipoteze:

1. $\langle S\rangle \overset{*}{\underset{G}{\Rightarrow}}$ w akko w sadrži isti broj terminala a i b.
2. $\langle A\rangle \overset{*}{\underset{G}{\Rightarrow}}$ w akko w sadrži za jedan više terminala a od terminala b.
3. $\langle B\rangle \overset{*}{\underset{G}{\Rightarrow}}$ w akko w sadrži za jedan više terminala b nego terminala a

Rešenje

Induktivna hipoteza

Ova hipoteza je sigurno tačna za $|w| = 1$, jer je $\langle A\rangle \overset{*}{\Rightarrow}$ a, $\langle B\rangle \overset{*}{\Rightarrow}$ b, i iz $\langle S\rangle$ se ne može izvesti sentenca dužine 1. Takođe, iz smena 4. i 5. mogu se izvesti samo sentence a i b dužine jedan.

Pretpostavimo da je induktivna hipoteza tačna za sve sentence w čija je dužina n–1 ili manja. Treba pokazati da je ona tačna i za $|w| = n$. Ako je $\langle S\rangle \overset{*}{\Rightarrow}$ w, tada izvođenje mora da započne sa $\langle S\rangle \rightarrow a\langle B\rangle$ ili $\langle S\rangle \rightarrow b\langle A\rangle$. U prvom slučaju w je oblika aw_1, gde je $|w_1| = n–1$ i $\langle B\rangle \overset{*}{\Rightarrow} w_1$. Prema induktivnoj hipotezi, broj terminala b u w_1 je za jedan veći od broja terminala a, te se w sastoji od jednakog broja neterminala a i b. Na sličan način se razmišlja i u slučaju da izvođenje započinje sa $\langle S\rangle \rightarrow b\langle A\rangle$.

Induktivna hipoteza – stav 1.

Sada treba dokazati akko deo prvog stava induktivne hipoteze, to jest, ako je $|w| = n$, w se sastoji od jednakog broja terminala a i b, i tada je $<S> \overset{*}{\Rightarrow} w$. a ili b su prvi simboli u w. Pretpostavimo da je $w = aw_1$. S obzirom da je $|w_1| = n-1$, w_1 ima za jedan veći broj terminala b od broja terminala a. Prema induktivnoj hipotezi, $<B> \overset{*}{\Rightarrow} w_1$. U tom slučaju, $<S> \Rightarrow a <B> \overset{*}{\Rightarrow} aw_1 = w$. Na sličan način se vrši dokaz i ako w počinje sa b.

Induktivna hipoteza – stav 2.

Sada treba dokazati akko deo drugog dela induktivne hipoteze, to jest, ako je $|w| = n = 2k + 1$ i w sadrži za jedan više broj terminala a od terminala b, tada je $<A> \overset{*}{\underset{G}{\Rightarrow}} w$. a ili b su prvi simboli u w. Pretpostavimo da je $w = aw_1$. S obzirom da je $|w_1| = n-1$, w_1 ima isti broj terminala a i b. Prema induktivnoj hipotezi, $<S> \overset{*}{\underset{G}{\Rightarrow}} w_1$. Tada je $<A> \underset{G}{\Rightarrow} a <S> \Rightarrow aw_1 = w$, odnosno za $w = baaw_1$, $<A> \underset{G}{\Rightarrow} b<A><A> \Rightarrow ba<A> \Rightarrow baa<S> \overset{*}{\Rightarrow} baaw_1$, gde je $<S> \overset{*}{\Rightarrow} w_1$. Po hipotezi, w_1 sadrži jednak broj terminala a i b, $|w_1| = 2k+1-3 = 2k-2$, te je time stav 2. dokazan.

Induktivna hipoteza – stav 3.

Ostavlja se čitaocu da sam dokaže ovaj deo hipoteze. S obzirom na simetriju smena za <A> i <B>, dokaz je u potpunosti sličan dokazu za stav 2. induktivne hipoteze.

Zadatak 2.2.5

Naći kontekstno-osetljivu gramatiku G koja generiše jezik

$$L=\{ww \mid w \in \{0, 1\}^+\}$$

U neformalnoj interpretaciji L je skup reči u $\{0, 1\}^+$ čije su prva i druga polovina jednake.

Analiza problema

Kao i kod rešavanja mnogih problema, moguće su različite putanje do cilja. Mogući koraci do cilja postavljenog ovim zadatkom bili bi:

a) Generisati sentencijalne forme oblika
$$w<W>w^r<R><E> \underset{G}{\Rightarrow} w<T>w^r<R><E>$$
gde je w^r slika u ogledalu sentenci w u odnosu na neterminal <W>, a <T>, <R> i <E> neterminali sa specijalnom ulogom.

b) Transformisati w^r u w.

Za generisanje sentencijalne forme pod a) potrebno je definisati poseban neterminal <W>, a za kontrolu završetka njenog generisanja neterminal <T>. Za proces transformisanja w^r, korak po korak, prebacivati simbole na početak druge polovine sentence u L koristeći neterminal <R>. Kontrola završetka dobijanja sentenci u L, ujedno i preslikavanja w^r u w bila bi izvedena neterminalom <E>.

Rešenje

Na osnovu analize problema, gramatika G koja generiše jezik L imala bi sledeći oblik:

1. <S> → <W><R><E>
2. <W> → 0<W>0 ; Generisanje sentencijalne
3. <W> → 1<W>1 ; forme oblika w<S>w^r<R><E>
4. <W> → <T> ; Kontrola završetka generisanja w i w^r
5. 00<R><E> → 0<R>0<R><E> ; Uvodi se neterminal <R> kako bi se
6. 01<R><E> → 1<R>0<R><E> ; obezbedilo pomeranje svakog znaka
7. 10<R><E> → 0<R>1<R><E> ; s desnog kraja w^r ka početku druge
8. 11<R><E> → 1<R>1<R><E> ; polovine sentenci u L.
9. 00<R> → 0<R>0 ; obezbeđuje pomeranje znaka s
10. 01<R> →1<R>0 ; desnog kraja w^r ka odgovarajućoj
11. 10<R> → 0<R>1 ; poziciji u drugoj polovini
12. 11<R> → 1<R>1 ; sentence u L.
13. <T>0<R> → 0<R><T> ; Izolovanje poziciranog znaka u w^r
14. <T>1<R> → 1<R><T> ; od preostalih znakova koje treba pozicionirati
15. <R><T> → <L><T> ; obezbeđivanje uslova za poništavanje neterminala <R>
16. <T><E> →<L><L> ; završetak preslikavanja w^r i obezbeđivanje uslova za ; završetak generisanja sentence u <L>.
17. <L> → ε ; generisanje sentence u <L>.

Primer:

Generisati sentencu 0011100111. Imamo da je

w = 00111

Sentencijalna forma 00111<T>11100<R><E> se dobija na jednostavan način primenom smena 1., 2., 3. i 4., zatim se izvođenje nastavlja primenom 5. smene itd.

$$<S> \overset{*}{\underset{G}{\Rightarrow}} 00111<T>111\underline{00<R><E>} \overset{5}{\underset{G}{\Rightarrow}} 00111<T>11\underline{10<R>}0<R><E>$$
$$\overset{11}{\underset{G}{\Rightarrow}} 00111<T>110<R>\underline{10<R><E>} \overset{7}{\underset{G}{\Rightarrow}} 00111<T>1\underline{10<R>}0<R>1<R><E>$$
$$\overset{11}{\underset{G}{\Rightarrow}} 00111<T>10<R>\underline{10<R>}1<R><E> \overset{11}{\underset{G}{\Rightarrow}} 00111<T>10<R>0<R>\underline{11<R><E>}$$
$$\overset{8}{\underset{G}{\Rightarrow}} 00111<T>\underline{10<R>}0<R>1<R>1<R><E> \overset{11}{\underset{G}{\Rightarrow}} 00111<T>0<R>\underline{10<R>}1<R>1<R><E>$$
$$\overset{11}{\underset{G}{\Rightarrow}} 00111<T>0<R>0<R>\underline{11<R>}1<R><E> \overset{12}{\underset{G}{\Rightarrow}} 00111<T>0<R>0<R>1<R>\underline{11<R><E>}$$
$$\overset{8}{\underset{G}{\Rightarrow}} 00111\underline{<T>0<R>}0<R>1<R>1<R>1<R><E> \overset{13}{\underset{G}{\Rightarrow}} 001110\underline{<R><T>}0<R>1<R>1<R>1<R><E>$$
$$\overset{15}{\underset{G}{\Rightarrow}} 001110<L>\underline{<T>0<R>}1<R>1<R>1<R><E> \overset{13}{\underset{G}{\Rightarrow}} 001110<L>0\underline{<R><T>}1<R>1<R>1<R><E>$$
$$\overset{15}{\underset{G}{\Rightarrow}} 001110<L>0<L>\underline{<T>1<R>}1<R>1<R><E> \overset{14}{\underset{G}{\Rightarrow}} 001110<L>0<L>1\underline{<R><T>}1<R>1<R><E>$$

$\overset{15}{\underset{G}{\Rightarrow}}$ 001110<L>0<L>1<L><u><T>1<R></u>1<R><E> $\overset{14}{\underset{G}{\Rightarrow}}$ 001110<L>0<L>1<L>1<u><R><T></u>1<R><E>
$\overset{15}{\underset{G}{\Rightarrow}}$ 001110<L>0<L>1<L>1<L><u><T>1<R></u><E> $\overset{14}{\underset{G}{\Rightarrow}}$ 001110<L>0<L>1<L>1<L>1<u><R><T></u><E>
$\overset{15}{\underset{G}{\Rightarrow}}$ 001110<L>0<L>1<L>1<L>1<L><u><T><E></u> $\overset{16}{\underset{G}{\Rightarrow}}$ 001110<u><L></u>0<u><L></u>1<u><L></u>1<u><L></u>1<u><L><L><L></u>
$\overset{17}{\underset{G}{\Rightarrow}}\ldots\overset{17}{\underset{G}{\Rightarrow}}$ 0011100111

Diskusija

Ostavlja se čitaocu da nađe eventualno bolje rešenje, to jest, gramatiku koja ima manje smena. U tom smislu, treba definisati druge globalne korake, ili pak izvršiti analizu datih polaznih koraka u cilju nalaženja efikasnije gramatike.

Posmatrajmo jezik $L = \{wcw \mid w \in (a|b)^+\}$. Iz prethodnog zadatka sledi da ovaj jezik nije bezkontekstni. U stvari, ovaj jezik daje apstrakciju problema provere da su identifikatori deklarisani pre njihovog korišćenja u programima. Prvo w predstavlja deklaraciju identifikatora, a drugo, njegovo korišćenje. Ovaj problem je značajan kod programskih jezika kao što je Pascal, koji zahteva da identifikatori budu deklarisani pre njihovog korišćenja. Stoga, gramatika za Pascal ne specificira znakove u identifikatoru. Umesto toga, svi identifikatori su u gramatici predstavljeni kao lekseme sa <u>id</u>. Kompilatori za takve jezike vrše proveru korišćenja identifikatora posle njegove deklaracije u toku semantičke analize.

Posmatrajmo takođe jezik $L = \{a^n b^m c^n d^m, n \geq 1, m \geq 1\}$. Ni ovaj jezik se ne može dobiti iz bezkontekstne gramatike. Ovaj jezik daje apstrakciju problema provere usaglašenosti broja formalnih parametara u deklaraciji procedure sa brojem stvarnih parametara u pozivu procedure, a^n i b^m predstavljaju liste formalnih parametara u dve procedure koje imaju n i m argumenata, respektivno. c^n i d^m su liste stvarnih parametara u pozivima ovih dveju procedura. I ovde, sintaksa definicija i poziva procedure se ne bavi brojanjem parametara, (jer bi to bio ulazak na područje parsiranja kontekstnog jezika) već se to takođe radi u fazi semantičke analize.

Zadatak 2.2.6

Naći kontekstno-osetljivu gramatiku G koja generiše jezik

$$L=\{w \mid w \in \{a, b, c\}^* \text{ i } w \text{ sadrži jednak broj terminala a, b i c}\}$$

Analiza problema

Gramatika mora da bude potpuna, to jest da generiše sve sentence date osobine, a ne podskup jezika tih osobina. Zbog tih uslova gramatika mora da bude kontekstno-osetljiva. Da je broj terminala samo dva (na primer, a i b), problem bi se mogao rešiti i bezkontekstnom gramatikom.

Rešenje

Pre svega, mora se obezbediti da se u procesu izvođenja obezbede uslovi da sentencijalna forma u svakom trenutku mora da sadrži jednak broj terminala a, b i c, i da može da počinje bilo kojim od njih. Ovaj uslov se lako ostvaruje sledećim smenama:

<S> → a<S><B><C> <A> → a

<S> → b<S><A><C> <B> → b

<S> → c<S><A><B> <C> → c

<S> → ε

S obzirom da svaka smena za <S> (izuzev <S> → ε) zadovoljava uslov da generiše jednak broj terminala a, b i c, tada u bilo kom izvođenju sentencijalna forma će imati strukturu iz koje se dobija takođe jednak broj terminala a, b i c.

Drugi uslov koji se mora ugraditi u smene je da je raspored terminala a, b i c u sentenci proizvoljan. Da bi se to obezbedilo moraju se uvesti kontekstno-osetljive smene:

<A><B> → <B><A>

<A><C> → <C><A>

<B><C> → <C><B>

<B><A> → <A><B>

<C><A> → <A><C>

<C><B> → <B><C>

Ako posmatramo krajnje levo izvođenje, lako je zaključiti da se bilo koji niz neterminala može prevesti u bilo koji drugi njihov raspored ne menjajući pritom njihov broj. Na primer, iz

<A><B><B><C><C><A><A><A><C><B><B>

u <A><A><A><A><B><B><B><B><C><C><C>

imamo sledeće izvođenje:

<A><B><B><C><C><A><A><A><C><B><B> ⇒
⇒ <A><B><B><C><C><A><A><A><B><C><B> ⇒

⇒ <A><B><B><C><C><A><A><A><B><B><C> ⇒
⇒ <A><B><B><C><A><C><A><A><B><B><C> ⇒

⇒ <A><B><B><A><C><C><A><A><B><B><C> ⇒
⇒ <A><B><B><A><C><A><C><A><B><B><C> ⇒

⇒ <A><B><B><A><A><C><C><A><B><B><C> ⇒
⇒ <A><B><B><A><A><C><A><C><B><B><C> ⇒

⇒ <A><B><B><A><A><C><A><B><C><B><C> ⇒
⇒ <A><B><B><A><A><C><A><B><B><C><C> ⇒

⇒ <A><B><B><A><A><A><C><B><B><C><C> ⇒
⇒ <A><B><B><A><A><A><B><C><B><C><C> ⇒

⇒ <A><B><B><A><A><A><B><B><C><C><C> ⇒
⇒ <A><B><A><B><A><A><B><B><C><C><C> ⇒

$\Rightarrow$ <A><A><B><B><A><A><B><B><C><C><C> $\Rightarrow$
$\Rightarrow$ <A><A><B><A><B><A><B><B><C><C><C> $\Rightarrow$

$\Rightarrow$ <A><A><A><B><B><A><B><B><C><C><C> $\Rightarrow$
$\Rightarrow$ <A><A><A><B><A><B><B><B><C><C><C> $\Rightarrow$

$\Rightarrow$ <A><A><A><A><B><B><B><B><C><C><C>

Dakle, gramatika ima sledeći oblik:

1. <S> $\rightarrow$ a<S><B><C>
2. <S> $\rightarrow$ b<S><A><C>
3. <S> $\rightarrow$ c<S><A><B>
4. <A><B> $\rightarrow$ <B><A>
5. <A><C> $\rightarrow$ <C><A>
6. <B><C> $\rightarrow$ <C><B>
7. <B><A> $\rightarrow$ <A><B>
8. <C><A> $\rightarrow$ <A><C>
9. <C><B> $\rightarrow$ <B><C>
10. <S> $\rightarrow \varepsilon$
11. <A> $\rightarrow$ a
12. <B> $\rightarrow$ b
13. <C> $\rightarrow$ c

Zadatak 2.2.7

Pokazati da

a) $\{a^i b^i \mid i \geq 1\}$ pripada bezkontekstnom jeziku.

b) $\{a^i b^i c^i \mid i \geq 1\}$ ne pripada bezkontekstnom jeziku.

c) Naći kontekstno-osetljivu gramatiku koja generiše jezik dat pod b).

Analiza problema

Neka je L proizvoljan bezkontekstni jezik. Postoje konstante p i q zavisno od L, takve da ako postoji reč $z \in L$, i $|z| > p$, tada se z može napisati kao z = uvwxy, gde je $|vwx| \leq q$ a v i x nisu istovremeno ε, tako da za svaki ceo broj $i \geq 0$, $uv^i wx^i y$ pripada jeziku L. Primenjujući ovu teoremu, može se dokazati tačnost stavova a) i b) postavljenih zadatkom.

Rešenje

a)

Polazeći od formalnog opisa sentenci u jeziku $uv^i wx^i y$, i uopštavajući odnos

$$v = a, \quad w = \varepsilon, \quad y = \varepsilon, \quad x = b, \quad u = \varepsilon$$

dobija se z = ab. Odavde sledi da je

$$|z| > 1 \text{ i } |vwx| \leq 2$$

čime su zadovoljeni uslovi postavljeni teoremom: postoje vrednosti p(1) i q(2) i z pripada jeziku L. Sledi zaključak da se jezik $\{a^ib^i \mid i \geq 1\}$ može generisati bezkontekstnom gramatikom.

b)

Ako se u ovom slučaju uspostavi preslikavanje $u = a^i$, $v = b$, $w = \varepsilon$, $x = c$, $y = \varepsilon$

dobija se:

$$|z| = |a^ibc| > p$$

S obzirom da je p konstanta, uslov $|z| > p$ je ispunjen za $p = 2$ i $\mathbf{i} \geq 1$, kao i uslov

$$|vwx| = |vx| = |bc| \leq 2$$

Međutim, $z = a^ibc$ ne pripada jeziku izuzev za $i = 1$.

Lako se može utvrditi da ni preslikavanje

$$v = a, \quad w = b^i, \quad x = c, \quad u = \varepsilon, \quad y = \varepsilon$$

ne zadovoljava postavljene uslove, kao ni ostala preslikavanja. Dakle, jezik $\{a^ib^ic^i \mid i \geq 1\}$ ne može se opisati bezkontekstnom gramatikom.

c)

S obzirom da sve sentence počinju sa a, kao i da je svaka sentenca uređena po terminalnim gramatičkim simbolima, za pretpostaviti je da je startna smena:

1. <S> $\rightarrow$ a<S><B><C>

Primenimo ovu smenu n–1 puta:

$$\text{<S>} \overset{1}{\underset{G}{\Rightarrow}} \text{a<S><B><C>} \overset{1}{\underset{G}{\Rightarrow}} \text{aa<S><B><C><B><C>} \overset{1}{\underset{G}{\Rightarrow}} \ldots \overset{1}{\underset{G}{\Rightarrow}} a^{n-1}\text{<S>(<B><C>)}^{n-1}$$

Kako u sentenci imamo jednak broj a, b i c druga smena bi bila

2. <S> $\rightarrow$ a<B><C>

Ako sada primenimo smenu 2. na dobijenu sentencijalnu formu, dolazi se do sledeće sentencijalne forme:

$$\text{<S>} \overset{*}{\underset{G}{\Rightarrow}} a^n\text{(<B><C>)}^n$$

Pošto se neterminali <B> i <C> naizmenično smenjuju, treba definisati kontekstno osetljivu smenu koja bi dovela do toga da prvo dođe n neterminala <B> pa zatim n neterminala <C>:

3. <C><B> $\rightarrow$ <B><C>

Primenom smene 3. na poslednju sentencijalnu formu, dobija se

$$\text{<S>} \overset{*}{\underset{G}{\Rightarrow}} a^n\,\text{<B>}^n\,\text{<C>}^n$$

Ne možemo sada da stavimo direktno smene <B> $\rightarrow$ b i <C> $\rightarrow$ c zbog prve dve smene. Stoga sledeće smene koje treba da zaokruže gramatički opis jezika mora da vode računa o prve dve smene. Dakle, da bi dobili abc, koristićemo drugu smenu i smene:

4. a<B> $\rightarrow$ ab

5. b<C> $\rightarrow$ bc

Ovo se lako može proveriti izvođenjem:

$$<S> \overset{*}{\underset{G}{\Rightarrow}} a<B><C> \underset{G}{\Rightarrow} ab<C> \underset{G}{\Rightarrow} abc$$

Međutim, ovo nije dovoljno. Posmatrajmo slučaj n = 2:

$$<S> \overset{*}{\underset{G}{\Rightarrow}} a^2 <B>^2 <C>^2 = aa<B><B><C><C> \underset{G}{\Rightarrow} aab<B><C><C>$$

Potrebna nam je sada smena:

6. b<B> $\rightarrow$ bb

Primenom smena 5. i 6. dobija se

$$aab<B><C><C> \underset{G}{\Rightarrow} aabb<C><C> \underset{G}{\Rightarrow} aabbc<C>$$

Lako je zaključiti da nam je potrebna još smena:

7. c<C> $\rightarrow$ cc

Na osnovu prethodne analize dobija se gramatika

1. <S> $\rightarrow$ a<S><B><C>
2. <S> $\rightarrow$ a<B><C>
3. <C><B> $\rightarrow$ <B><C>
4. a<B> $\rightarrow$ ab
5. b<C> $\rightarrow$ bc
6. b<B> $\rightarrow$ b<C>
7. c<C> $\rightarrow$ cc

Sada ćemo i formalno pokazati da data gramatika predstavlja jezik

$$L(G) = \{a^n b^n c^n \mid n \geq 1\}$$

U bilo kojem izvođenju koje počinje sa <S> bez korišćenja smene 2., nije moguće primeniti smene od 3–7 jer svaka od njih zahteva kontekst koji se ne pojavljuje u sentencijalnoj formi dobijenoj polazeći od smene 1. Posle primene smene 2., sentencijalna forma se sastoji od simbola a, iza kojih sledi <S> koje je praćeno naizmeničnim neterminalima <B> i <C>.

Posle primene smene 3., sentencijalna forma se sastoji od n simbola a, n ≥ 1, iza kojih slede n neterminala <B> i n neterminala <C> po istom redu. Pošto se <S> više ne pojavljuje, dalja primena smena 1. i 2. je isključena. Oblik sentencijalne forme je da iza svih terminala slede svi neterminali. Posle primene bilo koje od smena 3.–7., sentencijalna forma završava ovakvu strukturu. Dalje, lako se dolazi do zaključka da su smene 4.–7. primenljive jedino na granici između terminala i neterminala. Svaka od ovih smena ima za posledicu konverziju jednog neterminala <B> u b ili <C> u c. Smena 3. prouzrokuje pomeranje <B> u levo i <C> u desno.

Pretpostavimo da je izvršena konverzija nekog od neterminala c pre konverzije svih <B> u b. U tom slučaju, sentencijalna forma se može predstaviti kao $a^i b^i c\ \alpha$, gde je i < n, a α je niz

neterminala <B> i <C>, ali ne svih neterminala <C>. Sada se mogu primeniti samo smene 3. i 7.; smena 7. se primenjuje na spoju terminala i neterminala, a smena 3. na spoju dva neterminala. Može se sada koristiti smena 3. da se preurede neterminali <B> i <C> u α, ali ne i da se ukloni bilo koji neterminal <B>. Smena 7. se može iskoristiti da konvertuje <C> u c na spoju terminala i neterminala, ali će <B> biti, eventualno, krajnje levi neterminal. Međutim, ne postoji nijedna smena koja će da izmeni <B>, tako da se sentencijalna forma ne može nikada prevesti u sentencu.

Može se zaključiti da se svi neterminali moraju konvertovati na spoju terminala i neterminala pre konverzije bilo kojeg neterminala <C> u c. Stoga, sentencijalna forma sastavljena od a^n iza kojih sledi n neterminala <B> i n neterminala <C> u bilo kojem redosledu, može se svesti jedino na sentencu $a^nb^nc^n$. Stoga je $L(G) = \{a^nb^nc^n \mid n \geq 1\}$.

Zadatak 2.2.8

Naći bezkontekstnu gramatiku koja generiše jezik $L(G) = \{w \mid w \in \{a, b\}^*$ gde w sadrži dvostruko više terminala a od terminala b$\}$.

Analiza problema

Skup terminala je dat sa $V_T = \{a, b\}$. Ako je broj članova b u sentenci w jednak n, broj članova a iznosi tada 2n. Dakle, može se zaključiti da je $|w| = 3n$. Dakle, dozvoljene dužine su 3, 6, 9, 12, 15, ... Prema zadatku, prazna sentenca ne pripada jeziku L(G). Takođe, prema uslovu zadatka, svi mogući rasporedi a i b u dogovorenom odnosu moraju biti zastupljeni u jeziku. Na primer, w = aab, ali i w = baa, odnosno w = aba..

Rešenje

Pođimo od činjenice da sentenca w može da počinje sa b ili sa a. Stoga važi:

1. <S> $\rightarrow$ a<B>
2. <S> $\rightarrow$ b<A>

Za n =1, uvode se još tri smene

3. <A> $\rightarrow$ aa
4. <B> $\rightarrow$ ab
5. <B> $\rightarrow$ ba.

Postavlja se pitanje kolika je dužina nizova generisanih

$$<B> \overset{*}{\underset{G}{\Rightarrow}} w_B \text{ i } <A> \overset{*}{\underset{G}{\Rightarrow}} w_A$$

<S> obzirom da je

$$<S> \overset{*}{\underset{G}{\Rightarrow}} w \text{ i } |w| = 3n$$

dobija se

$|w_B| = 3n - 1$, $|w_A| = 3n - 1$

Sledeće pitanje koje traži odgovor je odnos broja članova a i b u sentenci w_B i w_A. Imajući u vidu da u <S> $\rightarrow$ a<B> postoji već jedno a, broj članova a u w_B treba da je $2n - 1$, a broj članova b ostaje nepromenjen, dakle, n. Obrnuto, u <S> $\rightarrow$ b<A> se nalazi već jedno b, pa broj članova b u w_A je $n - 1$ dok broj članova a ostaje nepromenjen (2n). Sada je potrebno naći smene koje će generisati sentence w_A i w_B na osnovu postavljenih uslova uz dodatnu napomenu, da one mogu da započinju takođe sa a ili b.

Posmatrajmo najpre sentencu w_A. Pored već navedene smene, <A> $\rightarrow$ aa, moguće su i sledeće smene pod navedenim uslovima:

6. <A> $\rightarrow$ b<A><A>
7. <A> $\rightarrow$ <A><A>b
8. <A> $\rightarrow$ <A>b<A>
9. <A> $\rightarrow$ a<A><B>
10. <A> $\rightarrow$ a<B><A>
11. <A> $\rightarrow$ <A><B>a
12. <A> $\rightarrow$ <B><A>a
13. <A> $\rightarrow$ <B>a<A>
14. <A> $\rightarrow$ <A>a<B>

Slično, i za sentence w_B se mogu formulisati smene, uz već postojeće, <B> $\rightarrow$ ab i <B> $\rightarrow$ ba.

10. <B> $\rightarrow$ a<B><B>
11. <B> $\rightarrow$ <B>a<B>
12. <B> $\rightarrow$ <B><B>a
13. <B> $\rightarrow$ b<A><B>
14. <B> $\rightarrow$ <A><B>b
15. <B> $\rightarrow$ <B><A>b
16. <B> $\rightarrow$ b<B><A>
17. <B> $\rightarrow$ <A>b<B>
18. <B> $\rightarrow$ <B>b<A>

Lako je pokazati da za svaku smenu važe postavljeni uslovi.

Na primer, za smenu 6.:

$$<A> \overset{6}{\underset{G}{\Rightarrow}} b<A><A> \overset{6}{\underset{G}{\Rightarrow}} \ldots \overset{6}{\underset{G}{\Rightarrow}} b^{n-1}<A>^n \overset{3}{\underset{G}{\Rightarrow}} \ldots \overset{3}{\underset{G}{\Rightarrow}} b^{n-1}(aa)^n = b^{n-1}a^{2n}$$

Smene 7. i 8. su permutacije gramatičkih simbola smene 6., tako da i one ispunjavaju uslove. Smena 9. daje sledeću sentencijalnu formu i sentencu:

$$<A> \overset{9}{\underset{G}{\Rightarrow}} a<A><B> \overset{9}{\underset{G}{\Rightarrow}} \ldots \overset{9}{\underset{G}{\Rightarrow}} a^{n-1}<A><B>^{n-1} \overset{4}{\underset{G}{\Rightarrow}} \ldots \overset{4}{\underset{G}{\Rightarrow}} a^{n-1}aa(ab)^{n-1}$$

Broj terminala a je 2n, a broj terminala b je n–1. Smene 10.–14. su samo permutacije gramatičkih simbola smene 9. tako da i one ispunjavaju postavljene uslove. Za smenu 15. se dobija:

$$<B> \overset{15}{\underset{G}{\Rightarrow}} a<B><B> \overset{15}{\underset{G}{\Rightarrow}} \ldots \overset{15}{\underset{G}{\Rightarrow}} a^{n-1}<B>^{n} \overset{4}{\underset{G}{\Rightarrow}} \ldots \overset{4}{\underset{G}{\Rightarrow}} a^{n-1}(ab)^{n}$$

Broj terminala a je 2n–1, a broj terminala b je n. Smene 16. i 17. su permutacije gramatičkih simbola smene 15. dok se za smenu 19. dobija:

$$<B> \overset{19}{\underset{G}{\Rightarrow}} <A><B>b \overset{19}{\underset{G}{\Rightarrow}} \ldots \overset{19}{\underset{G}{\Rightarrow}} <A>^{n-1}<B>b^{n-1} \overset{4}{\underset{G}{\Rightarrow}} <A>^{n-1}abb^{n-1} \overset{3}{\underset{G}{\Rightarrow}} \ldots$$
$$\ldots \overset{3}{\underset{G}{\Rightarrow}} (aa)^{n-1}abb^{n-1} = a^{2n-1}b^{n}$$

Smena 18. i smene 20.–23. su permutacije gramatičkih simbola smene 19., tako da za njih nije potrebno izvoditi posebno dokaz.

Videli smo iz ovih dokaza da su zadovoljeni uslovi na nivou svake od smena. Isto važi i za slučaj da se sentencijalna forma dobija primenom više različitih smena – odnos ostaje isti. Na primer, primenom smene 21. u sentencijalnu formu dobijenu iz smene 9., dolazi se do:

$$<A> \overset{9}{\underset{G}{\Rightarrow}} \ldots \overset{9}{\underset{G}{\Rightarrow}} a^{n-1} <A><B>^{n-1} \overset{21}{\underset{G}{\Rightarrow}} \ldots \overset{21}{\underset{G}{\Rightarrow}} a^{n-1} <A> \left(b^{n-1} <B><A>^{n-1}\right)^{n-1} \overset{3}{\underset{G}{\Rightarrow}}$$
$$\overset{3}{\underset{G}{\Rightarrow}} a^{n-1}aa\left(b^{n-1} <B><A>^{n-1}\right)^{n-1} \overset{5}{\underset{G}{\Rightarrow}} a^{n-1}aa\left(b^{n-1}ba <A>^{n-1}\right)^{n-1} \overset{3}{\underset{G}{\Rightarrow}} \ldots$$
$$\overset{3}{\underset{G}{\Rightarrow}} a^{n-1}aa\left(b^{n-1}ba(aa)^{n-1}\right)^{n-1}$$

Broj članova a u sentenci je

$$n_a = 2(n-1)^2 + 2(n-1) + 2$$

a broj članova b:

$$n_b = (n-1)^2 + (n-1)$$

čime je ispunjen uslov:

$$n_a = 2(n_b + 1)$$

Time je dokazano da u ovom slučaju važe postavljeni odnosi. Isto važi i za ostale moguće kombinacije smena.

Ovim je utvrđeno da dobijena gramatika generiše jezik datih karakteristika. Sada se postavlja pitanje da li se ovakva gramatika može redukovati, to jest, smanjiti broj smena. Imajući na umu da su smene 7. i 8. permutacije smene 6., smene 10.–14. permutacije smene 9., smene 16. i 17. permutacije smene 15. i smene 19.–23. permutacije smene 18., za pretpostaviti je da se neke smene mogu izostaviti a da jezik ostane nepromenjen. Dakle, može se dobiti ekvivalentna gramatika s manjim brojem smena.

Za svaku od smena 6.–23. formiraćemo njene osnovne etalone koji se dobijaju određivanjem svih nizova terminala koji se mogu izvesti iz njene desne strane (Tabela 2.2.1).

Idući unazad od smene 14. ka smeni 6., i od smene 23. ka smeni 15., eliminišu se one smene čiji se etaloni nalaze u etalonima prethodnih smena. Elipsasto uokvirene smene su one koje se odbacuju, a pravougaono uokvirene ostaju.

Tabela 2.2.1: Etaloni gramatičkih smena

neterminal <A>		neterminal <B>	
etalon	smena	etalon	smena
baaaa	6. <A> → b<A><A>	aabab abaab aabba ababa	15. <B> → a<B><B>
aaaab	7. <A> → <A><A>b	abaab ababa baaab baaba	16. <B> → <B>a<B>
aabaa	8. <A> → <A>b<A>	ababa baaba babaa abbaa	17. <B> → <B><B>a
aaaab aaaba	9. <A> → a<A><B>	baaab baaba	18. <B> → b<A><B>
aabaa abaaa	10. <A> → a<B><A>	aaabb aabab	19. <B> → <A><B>b
aaaba aabaa	11. <A> → <A><B>a	baaab abaab	20. <B> → <B><A>b
abaaa baaaa	12. <A> → <B><A>a	bbaaa babaa	21. <B> → b<B><A>
abaaa baaaa	13. <A> → <B>a<A>	aabab aabba	22. <B> → <A>b<B>
aaaab aaaba	14. <A> → <A>a<B>	abbaa babaa	23. <B> → <B>b<A>

Konačno, naša gramatika dobija oblik:

<S> → a<B> <S> → b<A>	<A> → b<A><A> <A> → a<A><B> <A> → a<B><A> <A> → aa	<B> → a<B><B> <B> → <B>a<B> <B> → <B><B>a <B> → <A><B>b <B> → b<B><A> <B> → ab <B> → ba

Diskusija

Ostavlja se čitaocu da pokuša da nađe drugačije rešenje. Da li je data gramatika dvosmislena? Ako jeste, da li se može naći ekvivalentna nedvosmislena gramatika? Treba imati na umu da postoje dvosmislene gramatike kod kojih ne postoji ekvivalentna nedvosmislena bezkontekstna gramatika koja generiše isti jezik.

Zadatak 2.2.9

a) Pokazati da se jezik $a^* + b^*$ ne može generisati bilo kojom bezkontekstnom gramatikom G koja ima samo jedan neterminal.

b) Pronaći, ako je moguće, regularnu gramatiku za jezik $a^* + b^*$.

Rešenje

a)

Pretpostavimo suprotno, to jest, da postoji bezkontekstna gramatika G sa terminalima a i b i jedinstvenim neterminalom <S> koja opisuje skup $L = \{a^m \mid m \geq 0\} \cup \{b^n \mid n \geq 0\}$.

S obzirom da je broj sekvenci jezika L beskonačan, a broj smena u gramatici G konačan, za neku dovoljno veliku konstantnu vrednost $K > 0$ sentenca a^K ne može se neposredno izvesti iz neterminala <S> primenom samo jedne smene. Drugim rečima, u izvođenju se mora koristiti smena na čijoj desnoj strani postoji bar jedna pojava <S> kao jedinog gramatičkog neterminala. Pretpostavimo da se radi o smeni oblika:

$$<S> \rightarrow \alpha<S>\beta$$

gde su α i β proizvoljni nizovi gramatičkih simbola dužine 0 ili više, pri čemu ne mogu istovremeno i α i β biti dužine 0 jer bi se tada ova smena mogla ukloniti iz gramatike bez uticaja na jezik koji opisuje. Bez smanjenja opštosti tvrđenja možemo pretpostaviti da je ova smena upotrebljena u prvom koraku izvođenja:

$$<S> \underset{G}{\Rightarrow} \alpha<S>\beta \overset{*}{\underset{G}{\Rightarrow}} a^P a^Q a^R$$

pri čemu je:

$$\alpha \overset{*}{\underset{G}{\Rightarrow}} a^P, <S> \overset{*}{\underset{G}{\Rightarrow}} a^Q \text{ i } \beta \overset{*}{\underset{G}{\Rightarrow}} a^R$$

P, Q i R su proizvoljne konstante za koje važi P, Q, R ≥ 0, P+Q+R=K i nisu istovremeno i P i R jednaki nuli. S druge strane, u skladu sa pretpostavkom imamo

$$<S> \overset{*}{\underset{G}{\Rightarrow}} b^L$$

za proizvoljno L > 0. Na osnovu gornjeg razmatranja zaključujemo da mora postojati izvođenje:

$$<S> \underset{G}{\Rightarrow} \alpha<S>\beta \overset{*}{\underset{G}{\Rightarrow}} a^P b^L a^R$$

odnosno u jeziku L se nalazi i sentenca koja se sastoji i od terminala a i od terminala b, što protivreči pretpostavci. Prema tome, ne postoji gramatika sa jednim neterminalom za opis skupa L, čime je dokaz završen.

b)

Analiza problema

Jezik a* + b* generisan gramatikom G se može razložiti na dva jezika $L(G_a)$ i $L(G_b)$, gde je

$$L(G_a) = \{a^*\} \quad \text{i} \quad L(G_b) = \{b^*\}$$

to jest,

$$L(G) = L(G_a) \cup L(G_b) = L(G_a) + L(G_b)$$

Rešenje

Jezik $L(G_a) = \{a^*\}$ može se generisati regularnom gramatikom

1. $<S> \rightarrow a<S>$
2. $S \rightarrow \varepsilon$

a jezik $L(G_b) = \{b^*\}$ regularnom gramatikom

1. $<S> \rightarrow b<S>$
2. $<S> \rightarrow \varepsilon$

Polazeći od teoreme da je regularna gramatika zatvorena u odnosu na operaciju unije, rezultantna gramatika treba da generiše takođe regularne sentence. Gramatika koja generiše uniju dva regularna skupa imala bi sledeći oblik:

1. $<S> \rightarrow <S_1>$
2. $<S> \rightarrow <S_2>$
3. $<S_1> \rightarrow a<S_1>$
4. $<S_1> \rightarrow \varepsilon$
5. $<S_2> \rightarrow b<S_2>$
6. $<S_2> \rightarrow \varepsilon$

Diskusija

Sledeći definiciju dobijena gramatika je bezkontekstna i pripada klasi desno-linearnih gramatika. Međutim, desno-linearna gramatika generiše regularne sentence i može se transformisati u regularnu gramatiku u skladu sa njenom definicijom.

2.3. Gramatičke transformacije

Zadatak 2.3.1

Pronaći suvišne neterminale u sledećoj gramatici sa startnim simbolom <S>.

1. <S> → a<A><B><C>
2. <S> → b<C><E><S>
3. <S> → a<E>
4. <A> → b<E>
5. <A> → <S><C><D>
6. <A> → d
7. <B> → d<F><S>
8. <B> → a<B><C>
9. <C> → a<E><S>
10. <C> → b<E>
11. <D> → a<A><C>
12. <D> → d
13. <E> → a<C><E>
14. <E> → ε
15. <F> → <A><B>
16. <F> → a<F>

Analiza problema

Suvišni neterminali dele se na mrtve i nedostižne. Neterminal <A> je mrtav ako i samo ako se iz <A> ne može izvesti nijedna sekvenca terminalnih simbola, to jest:

$$\{w \mid <A> \overset{*}{\Rightarrow} w\} = \varnothing.$$

U suprotnom slučaju neterminal <A> je živ.

Neterminal <A> je nedostižan ako i samo ako se <A> ne pojavljuje ni u jednoj sentencijalnoj formi izvedenoj iz startnog neterminala <S>, odnosno ako ne važi:

$$<S> \overset{*}{\Rightarrow} \alpha<A>\beta$$

gde su α i β proizvoljne sekvence gramatičkih simbola dužine nula ili više. U suprotnom slučaju neterminal <A> je dostižan.

Suvišne neterminale i smene u kojima se oni pojavljuju moguće je ukloniti iz gramatike bez uticaja na jezik koji ona opisuje.

Rešenje

Prvo treba pronaći i ukloniti mrtve neterminale, pa onda nedostižne, jer uklanjanjem mrtvih neterminala neki od prethodno dostižnih neterminala mogu postati nedostižni.

Odredićemo sve žive neterminale date gramatike, preostali su mrtvi. Skup živih neterminala formiramo iterativno na sledeći način:

1. Inicijalno je skup živih neterminala Ž prazan.
2. Razmatramo redom smene date gramatike: Ako je razmatrana smena oblika <A> $\rightarrow \alpha$ pri čemu je desna strana prazna sekvenca ili sekvenca sastavljena isključivo od terminala i samo onih neterminala koji se pojavljuju u skupu Ž (ako takvi postoje), dodajemo neterminal <A> u skup Ž.
3. Postupak nastavljamo sve dok postoji promena u skupu Ž, razmatrajući po potrebi svaku gramatičku smenu više puta.
4. Neterminali koji nisu živi pripadaju skupu mrtvih neterminala $M = V_N - Ž$.

U zadatoj gramatici,

- <A>, <D>, <E> su sigurno živi jer se mogu zameniti terminalom
- <S> je živ na osnovu 3. smene
- <C> je živ na osnovu 9. smene

Mrtvi neterminali su dakle <B> i <F>. Sve smene u kojima se pojavljuju <B> i <F> su nepotrebne pa iz gramatike uklanjamo smene 1, 7, 8, 15 i 16, pa ostaju smene:

2. <S> $\rightarrow$ b <C> <E> <S>
3. <S> $\rightarrow$ a <E>
4. <A> $\rightarrow$ b <E>
5. <A> $\rightarrow$ <S> <C> <D>
6. <A> $\rightarrow$ d
9. <C> $\rightarrow$ a <E> <S>
10. <C> $\rightarrow$ b <E>
11. <D> $\rightarrow$ a <A> <C>
12. <D> $\rightarrow$ d
13. <E> $\rightarrow$ a <C> <E>
14. <E> $\rightarrow \varepsilon$

Sada treba odrediti sve dostižne neterminale date gramatike, preostali su nedostižni. Skup dostižnih neterminala formiramo iterativno na sledeći način:

1. Inicijalno, skup dostižnih neterminala D sadrži samo startni neterminal.
2. Iz skupa D biramo neterminal <A> koji nije prethodno razmatran. Razmatramo sve smene sa levom stranom <A>. U skup D dodajemo sve neterminale koji se pojavljuju na desnim stranama ovih smena.
3. Prethodni korak ponavljamo dok ne razmotrimo sve neterminale iz skupa D.
4. Neterminali koji nisu dostižni pripadaju skupu nedostižnih neterminala $N = V_N - D$.

U zadatoj gramatici:

- dostižan je <S>
- iz <S> možemo izvesti sekvence sa <C> i ><E> pa su i oni dostižni;
- iz smena za <C> i <E> ne možemo u skup D dodati nove neterminale; postupak se okončava.

Dakle nedostižni su <A> i <D>. Iz gramatike uklanjamo sve smene u kojima se oni nalaze. Tako se gramatika svodi na:

7. <S> → b <C> <E> <S>
8. <S> → a <E>
15. <C> → a <E> <S>
16. <C> → b <E>
19. <E> → a <C> <E>
20. <E> → ε

Zadatak 2.3.2

Data je bezkontekstna gramatika G koja sadrži, između ostalog, smene oblika <A> → <B>. Naći ekvivalentnu gramatiku G_1 koja neće sadržavati smene ovog oblika.

Analiza problema

Eliminisanje smena ovog tipa ima praktičan smisao u eliminisanju kruženja pri izvođenju terminalnog niza u nekoj gramatici. Kruženje ili cikliranje je opisano sa $<A> \overset{+}{\underset{G}{\Rightarrow}} <A>$. Na primer, <A> → <B>, <B> → <C>, <C> → <D>, <D> → <A>: $<A> \overset{+}{\underset{G}{\Rightarrow}} <A>$.

Rešenje

Neka je gramatika $G = (V_N, V_T, P, <S>)$. Smene oblika <A> → <B> nazivamo smenama tipa E a ostale smenama tipa NE. Formirajmo novi skup smena P_1 polazeći od P, uključivanjem svih smena tipa NE koje se nalaze u P. Neka je $<A> \overset{+}{\underset{G}{\Rightarrow}} <B>$, za <A>, <B> ∈ V_N. U P_1 dodajemo sve smene oblika <A> → α, gde je <B> → α smena tipa NE u P. Ovim pristupom dobili smo gramatiku $G_1 = (V_N, V_T, P_1, <S>)$. Stoga, ako postoji izvođenje sekvence w u G_1, tada postoji i izvođenje w u G. Neka je sada w ∈ L(G) i posmatrajmo krajnje levo izvođenje w u G, na primer $<S> = \alpha_0 \underset{G}{\Rightarrow} \alpha_1 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} \alpha_n = w$. Ako je, za $0 \le i < n$, $\alpha_i \underset{G}{\Rightarrow} \alpha_{i+1}$ dobijeno primenom smene tipa NE, tada je $\alpha_i \underset{G1}{\Rightarrow} \alpha_{i+1}$. Pretpostavimo da je $\alpha_i \underset{G}{\Rightarrow} \alpha_{i+1}$ dobijeno primenom smene tipa E, ali je $\alpha_{i-1} \underset{G}{\Rightarrow} \alpha_i$ izvedeno smenom tipa NE izuzev ako je i = 0. Pretpostavimo takođe da je $\alpha_{i+1} \underset{G}{\Rightarrow} \alpha_{i+2} \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} \alpha_j$ dobijeno primenom niza smena tipa E, a $\alpha_j \underset{G}{\Rightarrow} \alpha_{j+1}$ smene tipa NE. Odavde sledi da su nizovi $\alpha_i, \alpha_{i+1}, \ldots, \alpha_j$ iste dužine, a pošto je izvođenje krajnje levo, simbol koji je zamenjen u svakom od njih mora da se nalazi na istom položaju. Tada je $\alpha_i \underset{G1}{\Rightarrow} \alpha_{j+1}$ dobijeno primenom jedne od smena u $P_1 - P$. Otuda je $L(G_1) = L(G)$.

Zadatak 2.3.3

Posmatrajmo gramatiku

$$G = (\{<S>, <A>, <B>\}, \{0, 1\}, P, <S>),$$

gde se P sastoji od sledećih smena:

1. $\langle S\rangle \rightarrow \langle A\rangle\langle B\rangle$
2. $\langle A\rangle \rightarrow \langle B\rangle\langle S\rangle\langle B\rangle$
3. $\langle A\rangle \rightarrow \langle B\rangle\langle B\rangle$
4. $\langle B\rangle \rightarrow 0\langle A\rangle 1$
5. $\langle B\rangle \rightarrow \varepsilon$
6. $\langle B\rangle \rightarrow 0$
7. $\langle A\rangle \rightarrow 1$

Naći ekvivalentnu gramatiku u kojoj se <S> ne pojavljuje na desnoj strani bilo koje smene i $\langle S\rangle \rightarrow \varepsilon$ je jedina prazna smena.

Analiza problema

Problem postavljen zadatkom znači eliminisanje praznih smena. Eliminisanje praznih smena je jedan od neophodnih preduslova za primenu algoritama za eliminisanje leve rekurzije. Ako se u nekoj gramatici G, dobija $\langle S\rangle \overset{*}{\underset{G}{\Rightarrow}} \varepsilon$ onda je $\langle S_1\rangle \rightarrow \varepsilon$ jedina dozvoljena prazna smena u novoj gramatici G_1. U tom slučaju, takođe, neophodno je najpre gramatiku G transformisati u oblik u kojem se startni simbol ne pojavljuje na desnoj strani bilo koje smene. Ako se to ne bi uradilo, tada bi smena oblika $\langle A\rangle \rightarrow \langle S\rangle$ prouzrokovala nova poništavanja neterminala, što se u ovom postupku želi izbeći. Nakon ove transformacije, prišlo bi se uklanjanju praznih smena u skladu s algoritmom u Prilogu 5.2.6.

Rešenje

Posmatrajmo sledeće izvođenje:

$$\langle S\rangle \overset{1}{\underset{G}{\Rightarrow}} \langle A\rangle\langle B\rangle \overset{3}{\underset{G}{\Rightarrow}} \langle B\rangle\langle B\rangle\langle B\rangle \overset{5}{\underset{G}{\Rightarrow}} \langle B\rangle\langle B\rangle \overset{5}{\underset{G}{\Rightarrow}} \langle B\rangle \overset{5}{\underset{G}{\Rightarrow}} \varepsilon$$

S obzirom da je izvođenjem dobijeno da je $\langle S\rangle \overset{*}{\underset{G}{\Rightarrow}} \varepsilon$, mora se pristupiti modifikaciji gramatike kako ona ne bi imala ni u jednoj smeni na desnoj strani startni simbol <S>. Postupak koji se sprovodi pri ovoj modifikaciji je sledeći.

Neka je data gramatika $G = (V_N, V_T, P, \langle S\rangle)$ i neka je novodobijena gramatika $G_1 = (V_N \cup \{\langle S_1\rangle\}, V_T, P_1, \langle S_1\rangle)$, pri čemu važi $\langle S_1\rangle \notin V_N$ i $\langle S_1\rangle \notin V_T$. P_1 se sastoji od svih smena gramatike G, uz dodavanje smena oblika $\langle S_1\rangle \rightarrow \alpha$ gde je $\langle S\rangle \rightarrow \alpha$ smena u P. Očigledno je da se $\langle S_1\rangle$ kao novi startni simbol ne pojavljuje ni u jednoj smeni P_1 na desnoj strani. Sada treba utvrditi da važi $L(G_1) = L(G)$. Pretpostavimo da je $\langle S\rangle \overset{*}{\underset{G}{\Rightarrow}} w$. Neka je prva smena primenjena u izvođenju $\langle S\rangle \rightarrow \alpha$. Tada možemo da pišemo da

$$\langle S\rangle \underset{G}{\Rightarrow} \alpha \overset{*}{\Rightarrow} w$$

Prema definiciji za P_1

$$\langle S_1\rangle \rightarrow \alpha \in P_1$$

te važi

$$\langle S_1\rangle \underset{G1}{\Rightarrow} \alpha$$

S obzirom da P_1 sadrži sve smene u P, može se napisati

$$\alpha \overset{*}{\underset{G1}{\Rightarrow}} w$$

te je

$$<S_1> \overset{*}{\underset{G1}{\Rightarrow}} w$$

Iz ovoga se može zaključiti da je $L(G) \subseteq L(G_1)$. Ako pokažemo da $L(G_1) \subseteq L(G)$ onda smo dokazali da je $L(G) = L(G_1)$. Pretpostavimo da je

$$<S_1> \overset{*}{\underset{G1}{\Rightarrow}} w$$

Neka je prva primenjena smena $<S_1> \rightarrow \alpha<S>$ obzirom da

$$<S> \rightarrow \alpha \in P$$

važi

$$<S> \underset{G}{\Rightarrow} \alpha$$

Sada je

$$\alpha \overset{*}{\underset{G1}{\Rightarrow}} w$$

ali α ne može da ima $<S_1>$ među svojim simbolima. Budući da se $<S_1>$ ne pojavljuje na desnoj strani ni u jednoj smeni P_1, nijedna sentencijalna forma u izvođenju $\alpha \overset{*}{\underset{G1}{\Rightarrow}} w$ ne uključuje $<S_1>$. Stoga je ovo izvođenje takođe izvođenje u gramatici G; to jest

$$\alpha \overset{*}{\underset{G}{\Rightarrow}} w$$

Na osnovu ovog razmatranja zaključujemo da je

$$<S> \overset{*}{\underset{G}{\Rightarrow}} w$$

to jest,

$$L(G) = L(G_1)$$

Polazeći od ove analize, gramatika G se lako transformiše u G_1; P_1 postaje:

1. $<S_1> \rightarrow <A><B>$
2. $<S> \rightarrow <A><B>$
3. $<A> \rightarrow <B><S><B>$
4. $<A> \rightarrow <B><B>$
5. $<B> \rightarrow 0<A>1$
6. $<B> \rightarrow \varepsilon$
7. $<B> \rightarrow 0$
8. $<A> \rightarrow 1$

Koristeći algoritam iz Priloga 5.2.6, sprovodi se sledeći niz koraka da bi se dobila gramatika G_2 u kojoj P_2 nema praznih smena izuzev za startni simbol.

1. $<S_1> \overset{1}{\underset{G1}{\Rightarrow}} <A><B> \overset{4}{\underset{G1}{\Rightarrow}} <B><B><B> \overset{6}{\underset{G1}{\Rightarrow}} <B><B> \overset{6}{\underset{G1}{\Rightarrow}} <B> \overset{6}{\underset{G1}{\Rightarrow}} \varepsilon$

Dakle, jedna smena u P_2 je

$$<S_1> \rightarrow \varepsilon$$

2. Iz P_1 se uklanja smena $<B> \rightarrow \varepsilon$.

3. $<S>_1$,<S>,<A>,<B> predstavlja skup neterminala iz kojih se mogu izvesti prazne sentence. Poništive smene su 1., 2., 3. i 4. obzirom da se na desnoj strani ovih smena nalaze neterminali koji su svi poništivi, onda treba napisati sve moguće kombinacije neterminala na desnoj strani.

1. smena	2. smena	3. smena	4. smena
$<S_1> \to <A><B>$ $<S_1> \to <A>$ $<S_1> \to <B>$	$<S> \to <A><B>$ $<S> \to <A>$ $<S> \to <B>$	$<A> \to <B><S><B>$ $<A> \to <S><B>$ $<A> \to <B>$ $<A> \to <B><B>$ $<A> \to <B>$ $<A> \to <B><S>$ $<A> \to <S>$	$<A> \to <B><B>$ $<A> \to <B>$ $<A> \to <B>$

Odavde se vidi da se eliminacijom praznih smena dobijaju četiri redundantne smene $<A> \to <B>$ i dve smene $<A> \to <B><B>$ koje se na jednostavan način eliminišu. Nakon njihovog uklanjanja, skup smena P_2 u gramatici G_2 dobija sledeći oblik:

$<S_1> \to <A><B>$	$<S> \to <A><B>$	$<A> \to <B><S><B>$	$<B> \to 0<A>1$
$<S_1> \to <A>$	$<S> \to <A>$	$<A> \to <S><B>$	$<B> \to 0$
$<S_1> \to <B>$	$<S> \to <B>$	$<A> \to <B><B>$	
$<S_1> \to \varepsilon$		$<A> \to <B><S>$	
		$<A> \to <B>$	
		$<A> \to <S>$	
		$<A> \to 1$	

Diskusija

Broj smena u gramatici G_2 se više nego udvostručio uz uvođenje samo jednog neterminala. Ostavlja se čitaocu da automatizuje ovaj postupak. U tom slučaju mogla bi se sprovesti prethodna analiza kojom bi se utvrdilo da li $\varepsilon \in L(G)$, odnosno da li je $<S> \overset{*}{\underset{G}{\Rightarrow}} \varepsilon$. Ako ovo nije slučaj, onda prva modifikacija s uvođenjem novog startnog neterminala nije potrebna. Dakle, programsko rešenje bi sadržavalo četiri celine: 1) Ispitivanje, 2) Uvođenje novog startnog neterminala po potrebi, 3) Eliminisanje praznih smena, 4) Eliminisanje redundantnih smena.

Zadatak 2.3.4

Eliminisati kružna izvođenja u bezkontekstnim gramatikama.

Analiza problema

Kružno izvođenje (ili kruženje) u bezkontekstnoj gramatici G je opisano sa $<A> \overset{+}{\underset{G}{\Rightarrow}} <A>$. Ono je nepoželjno kod nekih vrsta parsiranja pa je potrebno eliminisati ga. Do kružnog izvođenja ne dolazi samo zbog postojanja jediničnih smena (na primer, $<A> \to <B>$, $<B> \to <C>$, $<C> \to <D>$, $<D> \to <A>$) već i zbog postojanja poništivih neterminala i smena. Na primer,

niz smena <A> $\rightarrow$ <B><C>, <B> $\rightarrow$ <D>, <D> $\rightarrow$ <A>, <C> $\rightarrow \varepsilon$ dovodi do kružnog izvođenja:

<A> $\Rightarrow$ <B><C> $\Rightarrow$ <B> $\Rightarrow$ <D> $\Rightarrow$ <A>.

Dakle, do kružnog izvođenja može da dođe i indirektno, preko poništivih neterminala i smena. Stoga, kao prvi korak u eliminisanju kružnih izvođenja u datoj gramatici treba iz gramatike ukloniti prazne smene na način opisan u Prilogu 5.2.6.

Rešenje

Kao sledeći korak u uklanjanju kružnih izvođenja, treba ustanoviti da li takva izvođenja postoje u datoj gramatici. Da bismo to ustanovili potrebno je izdvojiti jedinične smene i na osnovu njih formirati relaciju BDW (Begins-Directly-With). Očigledno je da će se tranzitivnim zatvaranjem relacije BDW(BDW$^+$) dobiti uvid u to da li postoji kružno izvođenje. Ako se na glavnoj dijagonali relacije BDW$^+$ pojavi jedinica, onda to znači da postoji kružno izvođenje za neterminal koji označava red relacije BDW$^+$ u kojoj se pojavila jedinica na glavnoj dijagonali. Uzmimo primer sledećih jediničnih smena:

<A> $\rightarrow$ <B>

<B> $\rightarrow$ <C>

<C> $\rightarrow$ <D>

<D> $\rightarrow$ <A>

Relacije BDW i BDW$^+$ prikazane su na Sl. 2.3.1. Iz BDW$^+$ sledi da postoji kružno izvođenje za sve neterminale.

	<A>	<B>	<C>	<D>
<A>		1		
<B>			1	
<C>				1
<D>	1			

(a) Relacija BDW

	<A>	<B>	<C>	<D>
<A>	1	1	1	1
<B>	1	1	1	1
<C>	1	1	1	1
<D>	1	1	1	1

(b) Relacija BDW$^+$

Sl. 2.3.1

Eliminisanje kružnog izvođenja se sada lako izvodi polazeći od relacije BDW$^+$. Formira se, na osnovu relacije BDW$^+$, skup svih neterminala koji u svojoj vrsti imaju jedinicu na glavnoj dijagonali. Da bi se eliminisalo kružno izvođenje, potrebno je iz tog skupa uzeti jedan od neterminala i ukloniti iz gramatike jediničnu smenu koja mu odgovara (moguće je da ih bude i više). Treba napomenuti da isključenje bilo koje jedinične smene treba da sledi ranije opisano rešenje (Zadatak 2.3.2). Dakle, u ovom primeru, dovoljno je eliminisati jediničnu smenu <B> $\rightarrow$ <C>, da bi se uklonilo kružno izvođenje. Međutim, da bismo se u to uverili, potrebno je ponovo odrediti relacije BDW i BDW$^+$ (Sl. 2.3.2) za posmatranu gramatiku iz koje smo uklonili smenu <B> $\rightarrow$ <C>.

	<A>	<B>	<C>	<D>

	<A>	<B>	<C>	<D>

<A>		1		
<B>				
<C>				1
<D>	1			

(a) Relacija BDW

<A>		1		
<B>				
<C>	1			1
<D>	1	1		

(b) Relacija BDW^+

Sl. 2.3.2

Iz BDW^+ vidi se da na glavnoj dijagonali nema nijedne jedinice, što znači da je uklanjanjem smene <B> $\rightarrow$ <C> eliminisano kružno izvođenje. U potrebu ovakve provere može da nas uveri sledeći primer:

Dodajmo na dati skup jediničnih smena smenu <B> $\rightarrow$ <D>, tako da sada imamo:

<A> $\rightarrow$ <B>

<B> $\rightarrow$ <C>

<B> $\rightarrow$ <D>

<C> $\rightarrow$ <D>

<D> $\rightarrow$ <A>.

Nađimo sada relacije BDW i BDW^+ (Sl. 2.3.3).

	<A>	<B>	<C>	<D>
<A>		1		
<B>			1	1
<C>				1
<D>	1			

(a) Relacija BDW

	<A>	<B>	<C>	<D>
<A>	1	1	1	1
<B>	1	1	1	1
<C>	1	1	1	1
<D>	1	1	1	1

(b) Relacija BDW^+

Sl. 2.3.3

Vidi se da BDW^+ ostaje nepromenjeno. Uklonimo sada opet smenu <B> $\rightarrow$ <C>. Relacije BDW i BDW+ u ovom slučaju prikazane su na Sl. 2.3.4.

	<A>	<B>	<C>	<D>
<A>		1		
<B>				1
<C>				1
<D>	1			

(a) Relacija BDW

	<A>	<B>	<C>	<D>
<A>	1	1		1
<B>	1	1		1
<C>	1	1		1
<D>	1	1		1

(b) Relacija BDW^+

Sl. 2.3.4

Iz BDW^+ se vidi da su u kružno izvođenje uključeni neterminali <A>, <B> i <D>. Ako sada iz gramatike izbacimo, na primer, smenu <B> → <D>, relacije BDW i BDW+ su iste kao u slučaju kada je iz polazne gramatike od četiri smene bila uklonjena smena <B> → <C>. To znači da je ovim potpuno eliminisano kružno izvođenje iz gramatike.

Međutim, u smislu efikasnosti algoritma, nije svejedno koju jediničnu smenu uklanjamo. Vratimo se na primer sa pet jediničnih smena:

<A> → <B>

<B> → <C>

<B> → <D>

<C> → <D>

<D> → <A>.

Ako uklonimo smenu <D> → <A>, relacije BDW i BDW+ imaju izgled prikazan na Sl. 2.3.5.

	<A>	<B>	<C>	<D>
<A>		1		
<B>			1	1
<C>				1
<D>				

(a) Relacija BDW

	<A>	<B>	<C>	<D>
<A>		1	1	1
<B>			1	1
<C>				1
<D>				

(b) Relacija BDW^+

Sl. 2.3.5

Očigledno je da su uklanjanjem smene <D> → <A> eliminisana oba kružna izvođenja u gramatici.

Diskusija

Eliminisanje potencijalnog kružnog izvođenja može se izvesti uklanjanjem svih jediničnih smena na osnovu ranije opisanog rešenja (Zadatak 2.3.2). U tom slučaju nije potrebno formiranje relacija BDW i BDW^+. Ostavlja se čitaocu da eventualno nađe, drugačije, jednostavnije rešenje eliminacije kružnih izvođenja, kao i algoritam za izbor jedinične smene za uklanjanje koji daje eliminisanje maksimalnog broja kružnih izvođenja.

Zadatak 2.3.5

Posmatrajmo gramatiku G_d

1. <R> → <R> | <R>
2. <R> → <R><R>
3. <R> → <R>*

4. <R> → (<R>)
5. <R> → a
6. <R> → b

a) Pokazati da ova gramatika generiše sve regularne izraze nad simbolima a i b.

b) Pokazati da je ova gramatika dvosmislena.

c) Konstruisati ekvivalentnu nedvosmislenu gramatiku koja definiše prvenstvo i asocijativnost operatora *, konkatenacije i unije (|) na sledeći način:

   1. unarni operator zvezdastog zatvaranja ima najviše prvenstvo i levo je asocijativan,
   2. operator konkatenacije ima drugo najviše prvenstvo i levo je asocijativan,
   3. operator unije (|) ima najviše prvenstvo i levo je asocijativan.

Rešenje

a)

Smene 5. i 6. generišu elementarne regularne izraze sastavljene od simbola a i b respektivno. Ostale smene generišu složenije regularne izraze.

Smena 1. generiše regularne izraze sastavljene od proizvoljnog broja unija regularnih izraza.

Smena 2. generiše regularne izraze sastvaljene od proizvoljnog broja konkatenacija regularnih izraza.

Smena 3. generiše regularne izraze na koje je primenjena operacija zvezdastog zatvaranja proizvoljan broj puta.

Smena 4. generiše regularne izraze zatvorene u proizvoljan broj zagrada.

Gramatika u celini obezbeđuje generisanje proizvoljnih regularnih izraza u kojima su na simbole a i b primenjeni operatori unije, konkatenacije, zvezdastog zatvaranja u proizvoljnom redosledu i proizvoljnom broju, pri čemu je dozvoljeno korišćenje zagrada.

Ovo se može pokazati i analogijom s aritmetičkim izrazima:

1. <E> → <E> + <E>
2. <E> → <E> * <E>
3. <E> → – <E>
4. <E> → (<E>)
5. <E> → a
6. <E> → b

koja generiše sve aritmetičke izraze sa operandima a i b koji sadrže operatore sabiranja, množenja i promene znaka uz korišćenje zagrada.

b)

Iz ranijeg razmatranja (Zadatak 2.1.4) odmah sledi da je gramatika G dvosmislena.

c)

Koristeći se analogijom sa nedvosmislenom gramatikom za aritmetičke izraze, lako se dolazi do nedvosmislene gramatike G_{nd}, sa datim karakteristikama asocijativnosti i prioriteta operatora.

1. <R> → <R> | <T>
2. <R> → <T>
3. <R> → <T><P>
4. <T> → <P>
5. <P> → <R>*
6. <P> → (<R>)
7. <P> → a
8. <P> → b

2.4. Specijalne forme bezkontekstnih gramatika

Zadatak 2.4.1

Naći desno-linearnu gramatiku bez suvišnih neterminala koja odgovara konačnom automatu sa slike:

	0	1	
→A	A, B	E	0
B	E	C	1
C	C	B, C	0
D	E	C, D	1
E	E	E	0

Sl. 2.4.1

Analiza problema

Bezkontekstna gramatika naziva se desno-linearnom ako i samo ako svaka gramatička smena sadrži najviše jedan neterminal na desnoj strani koji, ako je prisutan, mora biti poslednji simbol desne strane. Drugim rečima, svaka smena desno-linearne gramatike je oblika:

<A> → w<B> ili <A> → w

gde je w proizvoljna sekvenca od nula ili više terminala.

Za proizvoljan konačni automat moguće je odrediti desno-linearnu gramatiku koja opisuje isti jezik kao i zadati automat, i obrnuto, za datu desno-linearnu gramatiku moguće je konstruisati automat koji prepoznaje isti jezik.

Za zadati konačni automat moguće je odrediti odgovarajuću desno-linearnu gramatiku na sledeći način:

- Terminalnim simbolima odgovaraju ulazni simboli automata.
- Neterminalnim simbolima odgovaraju stanja automata. Startnom neterminalu odgovara startno stanje automata.
- Proizvoljnom prelazu automata iz stanja A u stanje B pod ulazom x, odgovara smena $\langle A \rangle \rightarrow x\langle B \rangle$.
- Proizvoljno stanje prihvatanja A indukuje dodavanje smene: $\langle A \rangle \rightarrow \varepsilon$.

Rešenje

Zadatom automatu odgovara sledeća gramatika:

1. $\langle A \rangle \rightarrow 0\langle A \rangle$
2. $\langle A \rangle \rightarrow 0\langle B \rangle$
3. $\langle A \rangle \rightarrow 1\langle E \rangle$
4. $\langle B \rangle \rightarrow 0\langle E \rangle$
5. $\langle B \rangle \rightarrow 1\langle C \rangle$
6. $\langle B \rangle \rightarrow \varepsilon$
7. $\langle C \rangle \rightarrow 0\langle C \rangle$
8. $\langle C \rangle \rightarrow 1\langle B \rangle$
9. $\langle C \rangle \rightarrow 1\langle C \rangle$
10. $\langle D \rangle \rightarrow 0\langle E \rangle$
11. $\langle D \rangle \rightarrow 1\langle C \rangle$
12. $\langle D \rangle \rightarrow 1\langle D \rangle$
13. $\langle D \rangle \rightarrow \varepsilon$
14. $\langle E \rangle \rightarrow 0\ \langle E \rangle$
15. $\langle E \rangle \rightarrow 1\ \langle E \rangle$

Lako ustanovljavamo da je neterminal <E>, koji odgovara stanju greške mrtav, pa možemo ukloniti smene 3., 4., 10., 14. i 15. Neterminal <D> nije dostižan, pa možemo ukloniti i smene 11., 12. i 13. Konačni izgled gramatike je:

1. $\langle A \rangle \rightarrow 0\langle A \rangle$
2. $\langle A \rangle \rightarrow 0\langle B \rangle$
5. $\langle B \rangle \rightarrow 1\langle C \rangle$
6. $\langle B \rangle \rightarrow \varepsilon$
7. $\langle C \rangle \rightarrow 0\langle C \rangle$
8. $\langle C \rangle \rightarrow 1\langle B \rangle$
9. $\langle C \rangle \rightarrow 1\langle C \rangle$

Diskusija

Dobijena gramatika pripada klasi regularnih gramatika; radi se o podskupu desno-linearnih gramatika u kojima svaka smena ima oblik $\langle A \rangle \rightarrow a\langle B \rangle$, gde je a proizvoljan terminal ili $\langle A \rangle \rightarrow \varepsilon$.

Za ilustraciju da gramatika opisuje sekvence koje dati automat prihvata, navodimo gramatičko izvođenje sentence 00000, koju prihvata dati automat:

$$\begin{array}{ccccccccccc} <A> & \Rightarrow & 0<A> & \Rightarrow & 00<A> & \Rightarrow & 000<A> & \Rightarrow & 0000<A> & \Rightarrow & 00000<B> \Rightarrow 00000 \\ \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\ 1 & & 1 & & 1 & & 1 & & 2 & & 6 \end{array}$$

Zadatak 2.4.2

Data je desno-linearna gramatika sa startnim simbolom <A>:

1. $<A> \rightarrow ab<B>$
2. $<A> \rightarrow ac$
3. $<B> \rightarrow <A>$
4. $<B> \rightarrow \varepsilon$

a) Transformisati datu gramatiku u regularnu gramatiku.

b) Konstruisati nedeterministički konačni automat koji odgovara regularnoj gramatici.

Analiza problema

a)

Za definiciju desno-linearnih i regularnih gramatika videti Zadatak 2.4.1. Regularna gramatika koja opisuje isti jezik kao zadata desno-linearna gramatika može se dobiti transformacijom zadate gramatike po sledećim pravilima:

1. Smenu oblika

   $<A> \rightarrow w$

   gde w predstavlja niz od jednog ili više terminalnih simbola treba zameniti smenama

   $<A> \rightarrow w<\varepsilon>$

   $<\varepsilon> \rightarrow \varepsilon$

   gde je $<\varepsilon>$ novo uvedeni neterminalni simbol.

2. Smenu oblika

   $<A> \rightarrow w<B>$

   gde je $w = a_1a_2...a_n$ i $n > 1$, zameniti smenama

   $<A> \rightarrow a_1 <a_2...a_nB>$

   $<a_2...a_nB> \rightarrow a_2 <a_3...a_nB>$

   ...

   $<a_{n-1}a_nB> \rightarrow a_{n-1} <a_nB>$

$<a_nB> \rightarrow a_n <B>$

gde $<a_2...a_nB>$, $<a_3...a_nB>$, ..., $<a_{n-1}a_nB>$, $<a_nB>$ predstavljaju novouvedene neterminale.

3. Smene oblika

$<A> \rightarrow <B>$

$<B> \rightarrow \alpha_1$		$<A> \rightarrow \alpha_1$
$<B> \rightarrow \alpha_2$	zameniti smenama	$<A> \rightarrow \alpha_2$
...		...
$<B> \rightarrow \alpha_n$		$<A> \rightarrow \alpha_n$

pri čemu prethodno iz gramatike treba ukloniti smenu $<B> \rightarrow <B>$ ako eventualno postoji.

Rešenje

Primenjujući pravilo 1. na drugu smenu zadate gramatike, dobijamo:

1. $<A> \rightarrow ab<B>$
2. $<A> \rightarrow ac<\varepsilon>$
3. $<\varepsilon> \rightarrow \varepsilon$
4. $<B> \rightarrow <A>$
5. $<B> \rightarrow \varepsilon$

Primenjujući pravilo 2. na smene 1. i 2. dobijamo:

1. $<A> \rightarrow a <bB>$
2. $<bB> \rightarrow b <B>$
3. $<A> \rightarrow a <C\varepsilon>$
4. $<C\varepsilon> \rightarrow c <\varepsilon>$
5. $<\varepsilon> \rightarrow \varepsilon$
6. $<B> \rightarrow <A>$
7. $<B> \rightarrow \varepsilon$

Konačnu regularnu gramatiku dobijamo primenom pravila 3. na smenu 6:

1. $<A> \rightarrow a <bB>$
2. $<bB> \rightarrow b <B>$
3. $<A> \rightarrow a <C\varepsilon>$
4. $<C\varepsilon> \rightarrow c <\varepsilon>$
5. $<\varepsilon> \rightarrow \varepsilon$
6. $<B> \rightarrow a <bB>$
7. $<B> \rightarrow a <C\varepsilon>$
8. $<B> \rightarrow \varepsilon$

b)

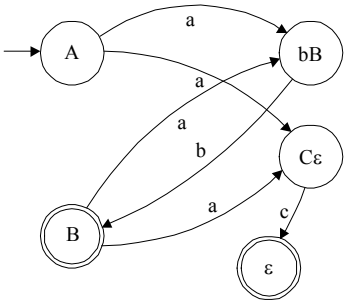
Analiza problema

Konačni, u opštem slučaju nedeterministički, automat koji prihvata jezik regularne gramatike dobijamo prema sledećim pravilima:

- Ulaznim simbolima automata odgovaraju terminalni gramatički simboli.
- Stanjima automata odgovaraju neterminali. Startnom stanju odgovara startni neterminal.
- Svakoj smeni oblika <A> → a<B> odgovara u automatu prelaz iz stanja A u stanje B za ulazni simbol x.
- Smena oblika <A> → ε implicira da je A stanje prihvatanja automata.

Rešenje

Na Sl. 2.4.2 prikazani su graf i tabela prelaza nedeterminističkog automata koji odgovara gramatici dobijenoj u tački a) rešenja.



	a	b	c	
→ A	bB, Cε			0
bB		B		0
B	bB, Cε			1
Cε			ε	0
ε				1

Sl. 2.4.2

Zadatak 2.4.3

Pokazati da nijedna sekvenca u jeziku generisana S–gramatikom ne može da bude prefiks bilo koje druge sekvence istog jezika.

Analiza problema

Za bezkontekstnu gramatiku tipa S važe sledeći uslovi:

a) desna strana svake smene počinje terminalnim simbolima;

b) ako dve smene imaju istu levu stranu, tada njihove desne strane počinju različitim terminalnim simbolima.

Iz ove definicije gramatike tipa S proizilaze sledeći zaključci:

c) nijedna smena nema na desnoj strani prazan niz;

d) nijedna smena ili terminal nisu poništivi;

e) ne može se dobiti sentencijalna forma u kojoj je bar jedan deo poništiv.

Rešenje

Neka su date dve sekvence, v i w, koje pripadaju jeziku L generisanom gramatikom G tipa S. Pretpostavimo da je sekvenca v prefiks sentence w.

Neka je <S> startni simbol gramatike G. Primenjujući krajnje levo izvođenje u gramatici G, pokušajmo da dobijemo sekvencu

$$Z = vw.$$

Primenom više smena u G, dolazimo do sledeće sentencijalne forme:

$$<S> \overset{+}{\underset{G}{\Rightarrow}} v\beta$$

Znamo da sekvenca v pripada jeziku L; s obzirom na jedinstvenost krajnjeg levog izvođenja i nedvosmislenosti gramatike G tipa S, niz β bi trebalo da bude prazan ili poništiv. Kako iz analize problema sledi da niz β ne može da bude poništiv, onda on mora da bude prazan niz. To dovodi do zaključka da se iz niza β ne može generisati sentenca w, pa prema tome sentenca v ne može da bude prefiks sentence w, što je i trebalo pokazati.

Zadatak 2.4.4

Pokazati da sledeće dve regularne gramatike generišu isti jezik.

G1

1.	$<X> \to 0$
2.	$<X> \to 0<Y>$
3.	$<X> \to 1<Z>$
4.	$<Y> \to 0<X>$
5.	$<Y> \to 1<Y>$
6.	$<Y> \to 1$
7.	$<Z> \to 0<Z>$
8.	$<Z> \to 1<X>$

G2

1.	$<A> \to 0<B>$	9.	$<D> \to 1<D>$
2.	$<A> \to 1<E>$	10.	$<D> \to \varepsilon$
3.	$<B> \to 0<A>$	11.	$<E> \to 0<C>$
4.	$<B> \to 1<F>$	12.	$<E> \to 1<A>$
5.	$<B> \to \varepsilon$	13.	$<F> \to 0<A>$
6.	$<C> \to 0<C>$	14.	$<F> \to 1<B>$
7.	$<C> \to 1<A>$	15.	$<F> \to \varepsilon$
8.	$<D> \to 0<A>$		

Analiza problema

Prema postavci zadatka, potrebno je pokazati da je $L(G_1) = L(G_2)$

Jednostavan način da se ovo pokaže je da se regularne gramatike G_1 i G_2 prikažu redukovanim mašinama, a da se zatim preimenovanjem stanja jedne od njih, utvrdi identičnost tih mašina.

Predstavljanje gramatike konačnim automatom je jednostavno i direktno. Međutim, to nije slučaj sa gramatikom G_1 jer ona sadrži smene oblika $<A> \to a$ koje nisu pogodne za direktnu transformaciju u konačan automat. Dakle, smene $<X> \to 0$ i $<Y> \to 1$ se moraju ukloniti i zameniti smenom (ili smenama) koje neće menjati jezik ali će omogućiti da se odrede stanja prihvatanja odgovarajućeg konačnog automata.

Rešenje

Da bi neko stanje moglo da se proglasi stanjem prihvatanja, mora da postoji smena oblika $\langle A\rangle \to \varepsilon$. Dakle, umesto $\langle X\rangle \to 0$ i $\langle Y\rangle \to 1$, mogu se uvesti smene

$$\langle X\rangle \to \varepsilon \text{ i } \langle Y\rangle \to \varepsilon$$

i videti da li uvođenje ovih smena menja generisani jezik. Ispitivanje može da utvrdi da su obe smene dozvoljene, ili pak da je dozvoljena samo jedna od njih. Pre svega je lako utvrditi da $\langle Z\rangle$ ne može da bude stanje prihvatanja jer za $\langle Z\rangle$ nema smene oblika $\langle A\rangle \to a$.

Pretpostavimo da su dozvoljene obe smene, $\langle X\rangle \to \varepsilon$ i $\langle Y\rangle \to \varepsilon$. Lako se dolazi do zaključka da smena $\langle X\rangle \to \varepsilon$ menja jezik jer se dobija sekvenca X = 11 koja ne postoji u jeziku $L(G_1)$. Dakle, $\langle Y\rangle \to \varepsilon$ mora da je važeće (da bi se odredilo stanje prihvatanja), što se utvrđuje na jednostavan način. Naime smene

$$\langle Y\rangle \to 1\langle Y\rangle$$

$$\langle Y\rangle \to \varepsilon$$

obezbeđuju ekvivalentnu zamenu za $\langle Y\rangle \to 1\langle Y\rangle$, $\langle Y\rangle \to 1$ i $\langle X\rangle \to 0$, što znači da ne menjaju jezik. Modifikovana gramatika G_1' glasi:

1. $\langle X\rangle \to 0\langle Y\rangle$
2. $\langle X\rangle \to 1\langle Z\rangle$
3. $\langle Y\rangle \to 0\langle X\rangle$
4. $\langle Y\rangle \to 1\langle Y\rangle$
5. $\langle Y\rangle \to \varepsilon$
6. $\langle Z\rangle \to 0\langle Z\rangle$
7. $\langle Z\rangle \to 1\langle X\rangle$

Prema tome, stanje Y je stanje prihvatanja, te konačni automati koji odgovaraju modifikovanoj gramatici G_1' i gramatici G_2 imaju sledeći oblik, Sl. 2.4.3.

	0	1	
→ X	Y	Z	0
Y	X	Y	1
Z	Z	X	0

$KA(G_1')$

	0	1	
→ A	B	E	0
B	A	F	1
C	C	A	0
D	A	D	1
E	C	A	0
F	A	B	1

$KA(G_2)$

Sl. 2.4.3:

Lako se utvrđuje da konačni automat $KA(G_1')$ predstavlja redukovanu mašinu, a $KA(G_2)$ ne. Minimizacija automata se izvodi na sledeći način. Početna particija je:

$$P_0 = (\{B,D,F\},\{A,C,E\})$$

Deoba particije P_0 na nove grupe u odnosu na ulaze 0 i 1 daje P_1:

$$P_1 = (\{B,D,F\},\{A\},\{C,E\})$$

Deoba particije P_1 u odnosu na ulaze 0 i 1 nije moguća jer se dobija:

$$P_2 = (\{B,D,F\},\{A\},\{C,E\})$$

Dakle, $P_2=P_1$. Ekvivalentna stanja su {B,D,F} i {C,E}. Usvojićemo u prvoj grupi za nosioca imena ekvivalentnih stanja B, a u drugoj C. Time redukovana mašina $KA_r(G_2)$ dobija oblik prikazan na Sl. 2.4.4.

	0	1	
A	B	C	0
B	A	B	1
C	C	A	0

Sl. 2.4.4: $KA_r(G_2)$

Preimenovanjem stanja A u X, B u Y i C u Z, dobija se automat prikazan na Sl. 2.4.5, koji je identičan automatu $KA(G_1')$. Time je pokazano da je $L(G_1) = L(G_2)$.

	0	1	
X	Y	Z	0
Y	X	Y	1
Z	Z	X	0

Sl. 2.4.5: $KA_r(G_2)$

Zadatak 2.4.5

Pokazati da je skup jezika koje je moguće opisati regularnom gramatikom zatvoren u odnosu na operaciju komplementiranja.

Analiza problema

Neka je data regularna gramatika G koja generiše jezik L(G), i neka je skup neterminala gramatike G, V_T. Tada je komplement jezika L(G) u odnosu na terminalni skup V_T dat sa

$$\overline{L(G)} = V_T^* - L(G)$$

Rešenje

Dokaz se može izvesti na jednostavan način konstruišući konačni automat koji simulira izvođenje jezika $\overline{L(G)}$. Neka je taj automat dat sa $\overline{F}$:

$$\overline{F} = (V_N \cup \{X\}, V_T, \delta, S, SP)$$

gde je

V_N – skup stanja automata koji odgovara skupu neterminala regularne gramatike G,

X – dodatno stanje automata $\overline{F}$ u odnosu na stanja automata F; njime se hvataju (označavaju) prazni ulazi (ako ih ima) u F koji predstavljaju stanje greške koje se katkada označava sa E,

V_T – ulazna azbuka automata $\overline{F}$ koja je ista kao i kod F,

δ – funkcije prelaza automata $\overline{F}$ koje su iste kao kod F uz dodavanje prelaza $\delta(V_N, a) \rightarrow X$ i $\delta(X, a) \rightarrow X$, $a \in V_T$, ako automat ima prazne ulaze za V_N i a, to jest, ulaze greške,

S – startno stanje automata $\overline{F}$,

SP – skup stanja prihvatanja automata $\overline{F}$ koji odgovara skupu stanja odbijanja automata F.

Očigledno je da ovako definisan automat simulira izvođenje jezika $\overline{L(G)}$, čime je pokazano da je skup regularnih jezika zatvoren u odnosu na operaciju komplementiranja.

Primer

Data je gramatika G

1. $\langle A \rangle \rightarrow 0\langle B \rangle$
2. $\langle A \rangle \rightarrow 1\langle A \rangle$
3. $\langle B \rangle \rightarrow 0\langle A \rangle$
4. $\langle B \rangle \rightarrow 1\langle B \rangle$
5. $\langle B \rangle \rightarrow \varepsilon$

Naći gramatiku koja generiše jezik $\overline{L(G)}$.

S obzirom da F nema prazne ulaze, $\overline{F}$ je dat sa

$$\overline{F} = (\{\langle A \rangle, \langle B \rangle\}, \{0, 1\}, \delta, \langle A \rangle, \{\langle B \rangle\})$$

te odgovarajuća gramatika dobija sledeći oblik

1. $\langle A \rangle \rightarrow 0\langle B \rangle$
2. $\langle A \rangle \rightarrow 1\langle A \rangle$
3. $\langle A \rangle \rightarrow \varepsilon$
4. $\langle B \rangle \rightarrow 0\langle A \rangle$
5. $\langle B \rangle \rightarrow 1\langle B \rangle$

Diskusija

Ostavlja se čitaocu da utvrdi da li je regularna gramatika zatvorena u odnosu na operaciju obrtanja, u kojoj se svaka sekvenca jezika obrne (kao lik u ogledalu), to jest, $a_1a_2\ldots a_n$ u polaznom jeziku L(G) postaje $a_na_{n-1}\ldots a_2a_1$ u novom jeziku $L(G_R)$. Koristiti pristup preko konačnih automata.

Zadatak 2.4.6

Pokazati da su regularni jezici zatvoreni u odnosu na operaciju preseka.

Analiza problema

Neka su date dve regularne gramatike G_1 i G_2 koje generišu jezike $L(G_1)$ i $L(G_2)$. Da bi se dokazao gornji stav, potrebno je utvrditi da je jezik L(G) dat sa

$$L(G) = L(G_1) \cap L(G_2)$$

regularan skup, te se može predstaviti regularnom gramatikom.

Rešenje

Dokaz stava o zatvorenosti regularnih gramatika u odnosu na operaciju preseka je jednostavan ako se pođe od toga da su regularne gramatike zatvorene u odnosu na operacije unije i komplementiranja, što se može videti iz prethodnih zadataka. Naime, prema zakonima Bulove algebre važi

$$L(G) = L(G_1) \cap L(G_2) = \overline{\overline{L(G_1)} \cup \overline{L(G_2)}}$$

S obzirom da su jezici $\overline{L(G_1)}$ i $\overline{L(G_2)}$ regularni, da je jezik $\overline{L(G_1)} \cup \overline{L(G_2)}$ takođe regularan, kao i da je jezik $\overline{\overline{L(G_1)} \cup \overline{L(G_2)}}$ regularan, sledi da je L(G) regularan skup, što je i trebalo pokazati.

Zadatak 2.4.7

Neka je data nesamoutiskujuća gramatika G u normalnoj formi Greibach-ove. Ako G ima m neterminala i l je dužina najduže desne strane bilo koje smene, tada nijedna sentencijalna forma ne može da ima više od ml neterminala koji se u njoj pojavljuju. Dokazati ovaj stav.

Rešenje

Pretpostavimo krajnje levo izvođenje u gramatici G. Usvojimo da se u sentencijalnoj formi α pojavljuje više od ml neterminala. U stablu izvođenja za α, posmatrajmo čvorove putanja od korena do krajnje levog neterminala sentencijalne forme α. Posebno, posmatrajmo one čvorove u kojima se uvode novi neterminali desno od putanje. S obzirom da je maksimalan broj neterminala koji se mogu izvesti iz bilo kog čvora l–1 (zbog normalne forme Greibach-ove), i imajući u vidu da se nijedan neterminal ne pojavljuje ponovo desno od posmatrane putanje (zbog nesamoutiskujućeg svojstva gramatike), mora da bude najmanje m+1 takvih čvorova da bismo dobili više od ml neterminala u sentencijalnoj formi. Stoga se neki neterminal <X> mora da pojavi dva puta u čvorovima koji čine sentencijalnu formu. Pošto posmatramo samo one čvorove kod kojih se uvode novi neterminali desno od putanje, a s obzirom da smena uvodi terminal kao krajnje desni, neterminal <X> mora da bude samoutiskujući, što protivreči polaznoj pretpostavci da je gramatika G nesamoutiskujuća.

Ovim je pokazano da gramatika G ne može da ima u bilo kojoj sentencijalnoj formi više od ml neterminala.

Zadatak 2.4.8

Koja je od sledećih gramatika samoutiskujuća?

a) $G_1 = (\{<A>, <B>, <C>\}, \{a, b\}, P_1, <A>)$ gde P_1 sadrži smene:

   1. $<A> \rightarrow <C><B>$
   2. $<A> \rightarrow b$
   3. $<B> \rightarrow <C><A>$
   4. $<C> \rightarrow <A><B>$
   5. $<C> \rightarrow a$

b) $G_2 = (\{<A>, <B>, <C>\}, \{a, b\}, P_2, <A>)$ gde P_2 sadrži:

   1. $<A> \rightarrow <C><B>$
   2. $<C> \rightarrow a<B>$
   3. $<B> \rightarrow b<C>$
   4. $<A> \rightarrow <C>a$
   5. $<C> \rightarrow b$

Naći konačni automat koji prepoznaje jezik generisan iz nesamoutiskujuće gramatike.

Analiza problema

Za bezkontekstnu gramatiku G se kaže da je samoutiskujuća ako postoji neki neterminal <A> s osobinom da je $<A> \overset{+}{\underset{G}{\Rightarrow}} \alpha_1<A>\alpha_2$, gde su α_1 i α_2 neprazni nizovi. Ako gramatika G nema ovo svojstvo, onda je L(G) regularni jezik. Ukoliko ovakvu gramatiku transformišemo u neku od normalnih formi (forma Čomskog i Greibach-ove), tada će i nova gramatika zadržati isto svojstvo. Najpovoljnija forma za transformaciju sa gledišta postavljenog zadatka je forma Greibach-ove.

Rešenje

Lako se zaključuje da je gramatika G_1 samoutiskujuća:

$$<A> \overset{1}{\underset{G_1}{\Rightarrow}} <C><B> \overset{4}{\underset{G_1}{\Rightarrow}} <A><B><B> \overset{3}{\underset{G_1}{\Rightarrow}} <A><C><A><B> = \alpha_1<A>\alpha_2$$

$\alpha_1 = <A><C>$, $\alpha_2 = <B>$

Nije potrebno vršiti dalju proveru i za ostale neterminale. Dovoljno je da barem jedan od neterminala bude samoutiskujući.

Izvršimo sličnu proveru u G_2 za neterminal <A>.

$$<A> \overset{1}{\underset{G_2}{\Rightarrow}} <C><B> \overset{2}{\underset{G_2}{\Rightarrow}} a<B><B> \overset{3}{\underset{G_2}{\Rightarrow}} ab<C><B>$$

U stvari, odmah je jasno da neterminal <A> nije samoutiskujući jer se ni u jednoj smeni za <C> i <B> ne pojavljuje <A>.

Za neterminal <C> dobijamo:

$$<C> \overset{2}{\underset{G_2}{\Rightarrow}} a<B> \overset{3}{\underset{G_2}{\Rightarrow}} ab<C> \overset{2}{\underset{G_2}{\Rightarrow}} aba<B>$$

I ovde se može zaključiti da neterminal <C> nije samoutiskujući jer se <C> uvek pojavljuje na kraju sentencijalne forme. Sličan zaključak važi i za <B>. Dakle, G_2 nije samoutiskujuća.

Transformišimo G_2 u normalnu formu Greibach-ove. Usvajamo sledeće zamene:

$<A> = <A_1>$
$<C> = <A_2>$
$<B> = <A_3>$

Smene dobijaju sada sledeći oblik:

1. $<A_1> \rightarrow <A_2><A_3>$
2. $<A_2> \rightarrow a<A_3>$
3. $<A_3> \rightarrow b<A_2>$
4. $<A_1> \rightarrow <A_2>a$
5. $<A_2> \rightarrow b$

Prostom zamenom svih smena za $<A_2>$ u smene za $<A_1>$, i odbacivanjem smena 1. i 4. dobija se sledeći skup smena (P_3) u novoj gramatici G_3:

1. $\langle A_1\rangle \rightarrow a\langle A_3\rangle\langle A_3\rangle$
2. $\langle A_1\rangle \rightarrow b\langle A_3\rangle$
3. $\langle A_1\rangle \rightarrow a\langle A_3\rangle a$
4. $\langle A_1\rangle \rightarrow ba$
5. $\langle A_2\rangle \rightarrow a\langle A_3\rangle$
6. $\langle A_2\rangle \rightarrow b$
7. $\langle A_3\rangle \rightarrow b\langle A_2\rangle$

Sada bi trebalo projektovati regularnu gramatiku $G_4 = (V_N, V_T, P_4, \langle S_4\rangle)$ polazeći od gramatike G_3. Postupak se odvija na sledeći način: Neterminali u G_4 odgovaraju nizovima neterminala u G_3 dužine manje ili jednake m*l*, gde je m broj neterminala, a *l* dužina najduže desne strane bilo koje smene u P_3. Stoga je $V_N' = \{[\alpha] \mid |\alpha| \leq ml \text{ i } \alpha \in V_N^+\}$. $\langle S_4\rangle$ je $[\langle A_1\rangle]$. Ako je $\langle A\rangle \rightarrow b\alpha \in P_3$, tada za sve neterminale u V_N' koji odgovaraju nizovima koji počinju sa $\langle A\rangle$ uvodimo smene $[\langle A\rangle\beta] \rightarrow b[\alpha\beta]$ u P_4, uz uslov da je $|\alpha\beta| \leq ml$. Očigledno je iz konstrukcije da G_4 simulira krajnje levo izvođenje u G_3, pa je $L(G_4) = L(G_3)$. Stoga je $L(G_2)$ regularan jezik.

Posmatrajući smene u P_3 vidimo da ne ispunjavaju sve smene uslov $\langle A\rangle \rightarrow b\alpha$ gde je $\alpha \in V_N^+$. Smene koje odstupaju od ovoga su 3. $\langle A_1\rangle \rightarrow a\langle A_3\rangle a$ i 4. $\langle A_1\rangle \rightarrow ba$. Da bismo G_3 doveli u pogodnu formu, uvedimo novi neterminal $\langle A_4\rangle$ i zamenimo smene 3. i 4. smenama:

$\langle A_1\rangle \rightarrow a\langle A_3\rangle\langle A_4\rangle$

$\langle A_1\rangle \rightarrow b\langle A_4\rangle$

$\langle A_4\rangle \rightarrow a$

Na osnovu ove transformacije koja ne menja jezik, dobija se skup smena P_3':

1. $\langle A_1\rangle \rightarrow a\langle A_3\rangle\langle A_3\rangle$
2. $\langle A_1\rangle \rightarrow b\langle A_3\rangle$
3. $\langle A_1\rangle \rightarrow a\langle A_3\rangle\langle A_4\rangle$
4. $\langle A_1\rangle \rightarrow b\langle A_4\rangle$
5. $\langle A_2\rangle \rightarrow a\langle A_3\rangle$
6. $\langle A_2\rangle \rightarrow b$
7. $\langle A_4\rangle \rightarrow b\langle A_2\rangle$
8. $\langle A_4\rangle \rightarrow a$

Skup sekvenci sukcesivnih neterminala koje se pojavljuju na desnim stranama smena je dat sa:

$$\{\langle A_2\rangle, \langle A_3\rangle, \langle A_4\rangle, \langle A_3\rangle\langle A_3\rangle, \langle A_3\rangle\langle A_4\rangle\}$$

Odavde sledi da je:

$$V_N' = \{[\langle A_1\rangle]. [\langle A_2\rangle], [\langle A_3\rangle], [\langle A_4\rangle], [\langle A_3\rangle\langle A_3\rangle], [\langle A_3\rangle\langle A_4\rangle]\}$$

Primenjujući prethodno izloženi postupak, dobijamo:

$[<A_1>] \rightarrow a[<A_3><A_3>]$ $[<A_2>] \rightarrow b$

$[<A_1>] \rightarrow b[<A_3>]$ $[<A_3>] \rightarrow b[<A_2>]$

$[<A_1>] \rightarrow a[<A_3><A_4>]$ $[<A_3><A_3>] \rightarrow b[<A_2><A_3>]$

$[<A_1>] \rightarrow b[<A_4>]$ $[<A_3><A_4>] \rightarrow b[<A_2><A_4>]$

$[<A_2>] \rightarrow a[<A_3>]$ $[<A_4>] \rightarrow a$

Međutim, u navedenim smenama se pojavljuju novi neterminali $[<A_2><A_3>]$ i $[<A_2><A_4>]$, tako da se V_N' proširuje i postaje V_N'':

$V_N'' = \{[<A_1>], [<A_2>], [<A_3>], [<A_4>], [<A_3><A_3>], [<A_3><A_4>], [<A_2><A_3>], [<A_2><A_4>]\}$

Novonastala situacija zahteva reinspekciju za smene $<A_2> \rightarrow a<A_3>$ i $<A_2> \rightarrow b$, tako da se za njih dobija:

$[<A_2>] \rightarrow a[<A_3>]$ $[<A_2>] \rightarrow b$

$[<A_2><A_3>] \rightarrow a[<A_3><A_3>]$ $[<A_2><A_3>] \rightarrow b[<A_3>]$

$[<A_2><A_4>] \rightarrow a[<A_3><A_4>]$ $[<A_2><A_4>] \rightarrow b\ [<A_4>]$

Nakon ove reinspekcije ne dobija se nijedan novi neterminal, te se za P_4 konačno dobija:

1. $[<A_1>] \rightarrow a[<A_3><A_3>]$
2. $[<A_1>] \rightarrow b[<A_3>]$
3. $[<A_1>] \rightarrow a[<A_3><A_4>]$
4. $[<A_1>] \rightarrow b[<A_4>]$
5. $[<A_2>] \rightarrow a[<A_3>]$
6. $[<A_2><A_3>] \rightarrow a[<A_3><A_3>]$
7. $[<A_2><A_4>] \rightarrow a[<A_3><A_4>]$
8. $[<A_2>] \rightarrow b$
9. $[<A_2><A_3>] \rightarrow b[<A_3>]$
10. $[<A_2><A_4>] \rightarrow b\ [<A_4>]$
11. $[<A_3>] \rightarrow b[<A_2>]$
12. $[<A_3><A_3>] \rightarrow b[<A_2><A_3>]$
13. $[<A_3><A_4>] \rightarrow b[<A_2><A_4>]$
14. $[<A_4>] \rightarrow a$

Ako sada izvršimo preimenovanje neterminala, dobija se P_4 u čitljivijoj formi:

1. $<A> \rightarrow a<E>$
2. $<A> \rightarrow b<C>$
3. $<A> \rightarrow a<F>$
4. $<A> \rightarrow b<D>$
5. $<B> \rightarrow a<C>$
6. $<G> \rightarrow a<E>$
7. $<H> \rightarrow a<F>$
8. $<B> \rightarrow b$
9. $<G> \rightarrow b<C>$
10. $<H> \rightarrow b<D>$
11. $<C> \rightarrow b<B>$
12. $<E> \rightarrow b<G>$
13. $<F> \rightarrow b<H>$
14. $<D> \rightarrow a$

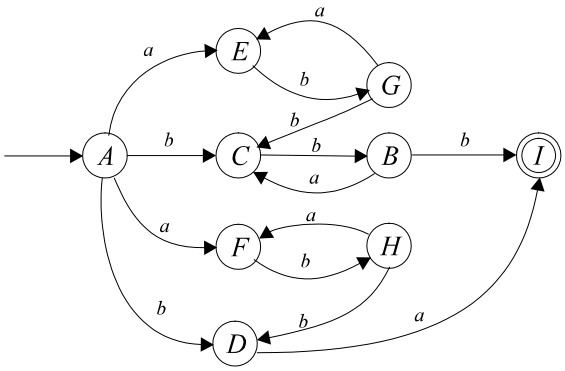
Za dobijanje konačnog automata u potpunoj formi za realizaciju, potrebno je uvesti dodatni neterminal <I> i povezati ga praznom smenom. Smene 8. i 12. treba da budu izmenjene dodavanjem novouvedenog neterminala:

8. <B> → b<I>

14. <D> → a<I>

15. <I> → ε

Nakon ovog konačnog doterivanja regularne gramatike G_4, konstrukcija konačnog automata je trivijalna. Sl. 2.4.6 prikazuje graf prelaza, a Sl. 2.4.7 tabelu prelaza rezultujućeg automata. Kao što se sa Sl. 2.4.6 vidi, automat je nedeterministički.



Sl. 2.4.6

	a	b	⊣	
→ A	E,F	C,D		0
B	C	I		0
C		B		0
D	I			0
E		G		0
F		H		0
G	E	C		0
H	F	D		0
I			Accept	1

Sl. 2.4.7

Diskusija

Iz dobijene forme Greibach-ove se odmah vidi da gramatika nije LL(1) te da bi parsiranje trebalo da se odvija od dna ka vrhu. S obzirom da gramatika G_2 nije samoutiskujuća, dobija se ekvivalentna regularna gramatika G_4. Dakle, ako ne bismo utvrdili prethodnim postupkom da gramatika nije samoutiskujuća, primenili bismo parsiranje nekom od metoda od dna ka vrhu. Doduše, broj smena u odnosu na polaznu gramatiku se utrostručio. Međutim, treba takođe upozoriti na sledeće: ako je neka gramatika samoutiskujuća to ne znači da generisani jezik ne može da bude regularan. Na primer, gramatika:

$$G = (\ \{<A>\},\ \{a, b\},\ P,\ <A>\ )$$

gde je P:

1. $<A> \rightarrow a<A>a$
2. $<A> \rightarrow a<A>$
3. $<A> \rightarrow b<A>$
4. $<A> \rightarrow a$
5. $<A> \rightarrow b$

generiše regularni jezik $L(G) = \{a, b\}^+$.

Ostavlja se čitaocu da za automat sa Sl. 2.4.6 nadje ekvivalentni deterministički automat i izvrši njegovu redukciju, ako je ona moguća.

Zadatak 2.4.9

Dat je konačni automat

	0	1	
$\rightarrow$ A	B	A	0
B	C	B	1
C	B	A	0

Sl. 2.4.8

koji prepoznaje jezik L generisan gramatikom G. Naći desno-linearne gramatike za jezike:

a) $L + \varepsilon$;

b) $L \bullet L$;

c) L^*;

d) L^+.

Analiza problema

Prvi korak u rešenju problema bi bio prevođenje automata u bezkontekstnu gramatiku, a drugi korak transformacija dobijene gramatike u desno-linearnu formu. Dobijanje jezika b), c) i d) se zasniva na teoremi da je regularna gramatika zatvorena u odnosu na konkatenaciju, zvezdasto i pozitivno zatvaranje. To znači da za dva jezika L_1 i L_2, dobijenih iz regularnih gramatika G_1 i G_2, jezici $L_1 \bullet L_2$, L_1^*, L_2^*, L_1^+, L_2^+ se takođe mogu dobiti regularnim gramatikama, polazeći od gramatika G_1 i G_2. Pri spajanju jezika iz dvaju gramatika, G_1 i G_2, potrebno je izvršiti u jednoj od njih preimenovanje neterminala tako da je

$$V_N^{G1} \cap V_N^{G2} = \varnothing$$

gde su V_N^{G1}, V_N^{G2} – skupovi neterminala u G_1 i G_2 i uvesti novi startni simbol <S> takav da je

$$<S> \neq <S_1> \neq <S_2>$$

gde su $<S_1>$, $<S_2>$ – startni neterminali u G_1 i G_2.

S obzirom da se radi o konačnom automatu, gramatika mora da generiše regularne sekvence. To znači da se može dobiti desno-linearna gramatika jer ona takođe generiše regularne skupove.

Rešenje

Pre nego što se pristupi transformaciji automata u regularnu gramatiku, potrebno je proveriti da li konačni automat predstavlja redukovanu mašinu. Prvo, lako je utvrditi da mašina nema suvišna stanja. Drugo, utvrđivanje da li je konačan automat minimalan može se lako sprovesti primenom algoritma deobe stanja automata na particije. Početna particija P_0 je data sa:

$$P_0 = (\{A, C\}, \{B\})$$

Dalja deoba se može izvesti samo na grupi {A,C}. Izvršimo prvo deobu u odnosu na ulaz 0. Dobija se:

$$P_1 = (\{A,C\}, \{B\})$$

Dakle, u odnosu na ulaz 0 nije dobijena nova particija, jer je $P_0 = P_1$. Izvršimo sledeću deobu u odnosu na ulaz 1. Dobija se:

$$P_2 = (\{A,C\}, \{B\})$$

Dakle, ni P_2 nije nova particija. Odavde se zaključuje da su stanja A i C ekvivalentna. Stoga se stanje C i odgovarajući red u konačnom automatu može izbaciti, a stanje C preimenovati u A. Konačni automat, posle minimizacije, dobija sledeći oblik:

	0	1	
→ A	B	A	0
B	A	B	1

Sl. 2.4.9

a odgovarajuća regularna gramatika G_1 je

1. $<A> \rightarrow 0<B>$

2. <A> → 1<A>

3. <B> → 0<A>

4. <B> → 1<B>

5. <B> → ε

Ovde je pretpostavljeno da je <A> startni simbol. Regularna gramatika je uvek **desno-linearna gramatika**.

a)

Gramatika G_a koja generiše jezik L+ε lako se nalazi ako se definiše novi startni neterminal <S> i uvedu dve nove smene:

<S> → <A>

<S> → ε

Gramatika G_a, dakle, ima sledeći oblik:

1. <S> → <A>
2. <S> → ε
3. <A> → 0<B>
4. <A> → 1<A>
5. <B> → 0<A>
6. <B> → 1<B>
7. <B> → ε

G_a zadovoljava uslove koje treba da ispunjava desno-linearna gramatika. Transformacija u regularnu gramatiku se može obaviti na jednostavan način, zamenjujući desnu stranu smene 1. desnim stranama smena za <A>. Dakle, G_a^r je

1. <S> → 0<B>
2. <S> → 1<A>
3. <S> → ε
4. <A> → 0<B>
5. <A> → 1<A>
6. <B> → 0<A>
7. <B> → 1<B>
8. <B> → ε

b)

Gramatika G_b koja generiše jezik L•L se nalazi tako što se definiše smena

<S> → <A><A>

i zatim dodaju sve smene gramatike G. <S> je ovde startni neterminal gramatike G_b:

1. <S> → <A><A>
2. <A> → 0<B>
3. <A> → 1<A>
4. <B> → 0<A>
5. <B> → 1<B>
6. <B> → ε

Dobijena gramatika je bezkontekstna.

Ovde smena 1. odstupa od uslova desno-linearne gramatike. S obzirom da je regularna gramatika zatvorena u odnosu na konkatenaciju, gramatika G_b se može transformisati u desno-linearnu ili regularnu gramatiku imajući u vidu da je gramatika G_b bez samoutiskivanja. Odgovarajuća regularna gramatika se može naći najbolje dovođenjem G_b u normalnu formu Greibach-ove, G_b^g:

1. <S> → 0<B><A>
2. <S> → 1<A><A>
3. <A> → 0<B>
4. <A> → 1<A>
5. <B> → 0<A>
6. <B> → 1<B>
7. <B> → ε

Postupak za dobijanje desno-linearne (ili regularne) gramatike za bezkontekstnu nesamoutiskujuću gramatiku u formi Greibach-ove već je pokazan (Zadatak 2.4.8). Ovde ćemo dati samo konačno rešenje; G_b^{dl}:

1. [<S>] → 0[<B><A>]
2. [<S>] → 1[<A><A>]
3. [<B><A>] → 0[<A><A>]
4. [<B><A>] → 1[<B><A>]
5. [<B><A>] → [<A>]
6. [<B>] → 0[<A>]
7. [<B>] → 1[<B>]
8. [<B>] → ε
9. [<A><A>] → 0[<B><A>]
10. [<A><A>] → 1[<A><A>]
11. [<A>] → 0[<B>]
12. [<A>] → 1[<A>]

Zbog smene 5., gramatika G_b^{dl} je desno-linearna.

c)

Generisanje jezika L^* se izvodi na jednostavan način uvođenjem smena oblika

<S> → <A><S>

<S> → ε

Novi startni neterminal je <S>. Na ove dve smene dodaju se smene gramatike G, te se dobija gramatika G_c:

1. <S> → <A><S>
2. <S> → ε
3. <A> → 0<B>
4. <A> → 1<A>
5. <B> → 0<A>
6. <B> → 1<B>
7. <B> → ε

Dovođenje gramatike G_c u normalnu formu Greibach-ove G_c^g izvodi se vrlo jednostavno zamenom krajnjeg levog neterminala u smeni 1. desnim stranama smena 3. i 4.:

1. $\langle S\rangle \to 0\langle B\rangle\langle S\rangle$
2. $\langle S\rangle \to 1\langle A\rangle\langle S\rangle$
3. $\langle S\rangle \to \varepsilon$
4. $\langle A\rangle \to 0\langle B\rangle$
5. $\langle A\rangle \to 1\langle A\rangle$
6. $\langle B\rangle \to 0\langle A\rangle$
7. $\langle B\rangle \to 1\langle B\rangle$
8. $\langle B\rangle \to \varepsilon$

Prevođenje G_c^g, koja je nesamoutiskujuća, u desno-linearnu gramatiku G_c^{dl} izvodi se analogno postupku primenjenom u tački b). Rezultat je:

1. $[\langle S\rangle] \to 0[\langle B\rangle\langle S\rangle]$
2. $[\langle S\rangle] \to 1[\langle A\rangle\langle S\rangle]$
3. $[\langle S\rangle] \to \varepsilon$
4. $[\langle B\rangle\langle S\rangle] \to 0[\langle A\rangle\langle S\rangle]$
5. $[\langle B\rangle\langle S\rangle] \to 1[\langle B\rangle\langle S\rangle]$
6. $[\langle B\rangle\langle S\rangle] \to [\langle S\rangle]$
7. $[\langle B\rangle] \to 0[\langle A\rangle]$
8. $[\langle B\rangle] \to 1[\langle B\rangle]$
9. $[\langle B\rangle] \to \varepsilon$
10. $[\langle A\rangle\langle S\rangle] \to 0[\langle B\rangle\langle S\rangle]$
11. $[\langle A\rangle\langle S\rangle] \to 1[\langle A\rangle\langle S\rangle]$
12. $[\langle A\rangle] \to 0[\langle B\rangle]$
13. $[\langle A\rangle] \to 1[\langle A\rangle]$

d)

Generisanje gramatike L^+ se takođe može izvesti na jednostavan način, uvođenjem sledećih smena:

$$\langle S\rangle \to \langle A\rangle\langle S_1\rangle$$

$$\langle S_1\rangle \to \langle A\rangle\langle S_1\rangle$$

$$\langle S_1\rangle \to \varepsilon$$

Novi startni neterminal je $\langle S\rangle$. Na prethodne tri smene treba dodati sve smene gramatike G_1 tako da se dobija gramatika G_d:

1. $\langle S\rangle \to \langle A\rangle\langle S_1\rangle$
2. $\langle S_1\rangle \to \langle A\rangle\langle S_1\rangle$
3. $\langle S_1\rangle \to \varepsilon$
4. $\langle A\rangle \to 0\langle B\rangle$
5. $\langle A\rangle \to 1\langle A\rangle$
6. $\langle B\rangle \to 0\langle A\rangle$
7. $\langle B\rangle \to 1\langle B\rangle$
8. $\langle B\rangle \to \varepsilon$

Dovođenje gramatike G_d u normalnu formu Greibach-ove, G_d^g, izvodi se zamenom krajnje levih neterminala u smenama 1. i 2. desnim stranama smena 3. i 4.:

1. $\langle S\rangle \to 0\langle B\rangle\langle S_1\rangle$
2. $\langle S\rangle \to 1\langle A\rangle\langle S_1\rangle$
3. $\langle S_1\rangle \to 0\langle B\rangle\langle S_1\rangle$
4. $\langle S_1\rangle \to 1\langle A\rangle\langle S_1\rangle$
5. $\langle S_1\rangle \to \varepsilon$
6. $\langle A\rangle \to 0\langle B\rangle$
7. $\langle A\rangle \to 1\langle A\rangle$
8. $\langle B\rangle \to 0\langle A\rangle$
9. $\langle B\rangle \to 1\langle B\rangle$
10. $\langle B\rangle \to \varepsilon$

S obzirom da je i gramatika G_d^g nesamoutiskujuća, prevođenje G_d^g u desno-linearnu gramatiku G_{dl} izvodi se kao u tački c). Rezultat je:

1. $[\langle S\rangle] \to 0[\langle B\rangle\langle S_1\rangle]$
2. $[\langle S\rangle] \to 1[\langle A\rangle\langle S_1\rangle]$
3. $[\langle S_1\rangle] \to 0[\langle B\rangle\langle S_1\rangle]$
4. $[\langle S_1\rangle] \to 1[\langle A\rangle\langle S_1\rangle]$
5. $[\langle S_1\rangle] \to \varepsilon$
6. $[\langle B\rangle\langle S_1\rangle] \to 0[\langle A\rangle\langle S_1\rangle]$
7. $[\langle B\rangle\langle S_1\rangle] \to 1[\langle B\rangle\langle S_1\rangle]$
8. $[\langle B\rangle\langle S_1\rangle] \to [\langle S_1\rangle]$
9. $[\langle B\rangle] \to 0[\langle A\rangle]$
10. $[\langle B\rangle] \to 1[\langle B\rangle]$
11. $[\langle B\rangle] \to \varepsilon$
12. $[\langle A\rangle\langle S_1\rangle] \to 0[\langle B\rangle\langle S_1\rangle]$
13. $[\langle A\rangle\langle S_1\rangle] \to 1[\langle A\rangle\langle S_1\rangle]$
14. $[\langle A\rangle] \to 0[\langle B\rangle]$
15. $[\langle A\rangle] \to 1[\langle A\rangle]$

Diskusija

Ostavlja se čitaocu da desno-linearne gramatike dobijene u tačkama b), c), i d) prevede u regularne gramatike.

Zadatak 2.4.10

Date su sledeće gramatike:

$$G_1 = (\ \{\langle S\rangle\},\ \{0,\ 1\},\ \{\langle S\rangle \to 0\langle S_1\rangle,\ \langle S\rangle \to \varepsilon\},\ \langle S\rangle)$$

i

$$G_2 = (\ \{\langle A_1\rangle,\ \langle B\rangle\},\ \{0,\ 1\},\ \{\langle A\rangle \to \langle B\rangle\langle A\rangle, \langle A\rangle \to \langle B\rangle,\ \langle B\rangle \to 0\langle B\rangle 1,\ \langle B\rangle \to \varepsilon\},\ \langle A\rangle)$$

Odrediti koja je od ovih gramatika sekvencijalna, a koja linearna.

Analiza problema

Gramatika $G = (V_N, V_T, P, \langle S\rangle)$ je sekvencijalna ako se neterminali u V_N mogu urediti u niz $\langle A_1\rangle, \langle A_2\rangle, \ldots, \langle A_n\rangle$ takav da ako je $\langle A_i\rangle \to \alpha$ smena u P, tada α ne sadrži $\langle A_j\rangle$ takvo da je $j < i$. Jezici generisani sekvencijalnim gramatikama nazivaju se sekvencijalni jezici.

Ako svaka smena u gramatici ima oblik <A> $\rightarrow$ u<B>v ili <A> $\rightarrow$ u, gde su <A> i <B> neterminali, u i v terminalni nizovi, takva se gramatika naziva linearna. Jezici generisani linearnim gramatikama nazivaju se linearnim jezicima.

Rešenje

Gramatika G_1 je linearna, jer obe smene ispunjavaju uslove za nju:

$$<S> \rightarrow 0<S_1>, \quad u = 0, v = 1,$$

ili

$$<S> \rightarrow \varepsilon, \qquad u = \varepsilon.$$

G_1 je takođe i sekvencijalna gramatika.

Gramatika G_2 je sekvencijalna, jer se neterminali mogu urediti po postavljenim uslovima:

$$<A> = <A_1>, \qquad <B> = <A_2>$$

$$<A_1> \rightarrow <A_2><A_1>, \qquad <A_1> \rightarrow < A_2>, \quad <A_2> \rightarrow 0<A_2>1, \qquad <A_2> \rightarrow \varepsilon$$

Diskusija

Ostavlja se čitaocu da nađe neke složenije primere linearnih i sekvencijalnih gramatika.

Zadatak 2.4.11

Data je bezkontekstna gramatika

1. <S> $\rightarrow$ <T> + <S>
2. <S> $\rightarrow$ <T> – <S>
3. <S> $\rightarrow$ <T>
4. <T> $\rightarrow$ <L> * <T>
5. <T> $\rightarrow$ <L> / <T>
6. <T> $\rightarrow$ <L>
7. <L> $\rightarrow$ (<S>)
8. <L> $\rightarrow$ a
9. <L> $\rightarrow$ b

Prevesti je u normalnu formu Čomskog. Pri tome se ne smeju pojaviti prazne smene.

Analiza problema

Kod transformacije bezkontekstne gramatike u specijalnu formu Čomskog, može se postupiti slično prevođenju desno-linearne gramatike u regularnu gramatiku. Međutim, smene tipa <A> $\rightarrow$ <B> predstavljaju problem, pa se one moraju prevesti najpre u oblik <A> $\rightarrow \alpha$, gde je $|\alpha| \geq 2$, ili <A> $\rightarrow$ a.

Rešenje

Izvršimo najpre transformaciju kojom se eliminišu smene oblika <A> $\rightarrow$ <B>. To su smene 3. i 6. Njihovo prevođenje u oblik <A> $\rightarrow \alpha$ se izvodi tako što se <B> u smeni <A> $\rightarrow$ <B> zamenjuje desnim stranama smena tipa <B> $\rightarrow \alpha$ ili <B> $\rightarrow$ a što u konkretnom slučaju daje sledeći skup smena:

Smena 3.

<S> → <L> * <T>

<S> → <L> / <T>

<S> → (<S>)

<S> → a

<S> → b

Smena 6.

<T> → (<S>)

<T> → a

<T> → b

Nakon prvog koraka ka konačnom obliku, gramatika postaje:

1. <S> → <T>+<S>	8. <T> → <L> * <T>	13. <L> → (<S>)
2. <S> → <T> – <S>	9. <T> → <L> / <T>	14. <L> → a
3. <S> → <L> * <T>	10. <T> → (<S>)	15. <L> → b
4. <S> → <L> / <T>	11. <T> → a	
5. <S> → (<S>)	12. <T> → b	
6. <S> → a		
7. <S> → b		

U drugom koraku transformacije, na svaku smenu oblika <A> → α, gde je |α| ≥ 2, primenjujemo sledeći postupak:

1. ako je α = <B_1><B_2>, gde su <B_1> i <B_2> neterminali u datoj gramatici, ne raditi ništa.
2. ako je α = a<B>, gde je a terminal a <B> neterminal u datoj gramatici, uvesti novi neterminal <X> i staviti

   <A> → <X><B>
   <X> → a

3. ako je α = <B>a, staviti

   <A> → <B><X>
   <X> → a

4. ako je α = aα_1 , |α_1| ≥ 2, staviti

   <A> → <X_1><X_2>
   <X_1> → a
   <X_2> → α_1

   gde su <X_1> i <X_2> novouvedeni neterminali. Nastavljamo isti postupak (slučajevi 1.–6.) na smeni <X_2> → α, dok se ne dobiju oblici <A> → <B>C ili <A> → a.

5. ako je α = <B>α_1, |α_1| ≥ 2, staviti

$\langle A \rangle \rightarrow \langle B \rangle\langle X \rangle$
$\langle X \rangle \rightarrow \alpha_1$

pa primeniti isti postupak (slučajevi 1.–6.) na smenu $\langle X \rangle \rightarrow \alpha$.

6. ako je $\alpha = ab$, gde su a i b terminali u datoj gramatici, staviti

$\langle A \rangle \rightarrow \langle X_1 \rangle\langle X_2 \rangle$
$\langle X_1 \rangle \rightarrow a$
$\langle X_2 \rangle \rightarrow b$

gde su $\langle X_1 \rangle$ i $\langle X_2 \rangle$ novouvedeni terminali.

Smene koje podležu ovoj transformaciji su 1., 2., 3., 4., 5., 8., 9., 10., 13. Smene 1., 2., 3., 4., 8. i 9. su istog tipa u pogledu transformacije, te se na njih primenjuje korak 5. Pokažimo ovu transformaciju na primeru smene 1.

Primenjujući postupak 5. dobijamo:

$\langle S \rangle \rightarrow \langle T \rangle\langle X_1 \rangle$

$\langle X_1 \rangle \rightarrow +\langle S \rangle$

Sada primenjujemo postupak 2. na X_1:

$\langle X_1 \rangle \rightarrow \langle X_2 \rangle\langle S \rangle$

$\langle X_2 \rangle \rightarrow +$

Konačno, dobija se:

Smena 1.	Smena 2.	Smena 3.
$\langle S \rangle \rightarrow \langle T \rangle\langle X_1 \rangle$ $\langle X_1 \rangle \rightarrow \langle X_2 \rangle\langle S \rangle$ $\langle X_2 \rangle \rightarrow +$	$\langle S \rangle \rightarrow \langle T \rangle\langle X_3 \rangle$ $\langle X_3 \rangle \rightarrow \langle X_4 \rangle\langle S \rangle$ $\langle X_4 \rangle \rightarrow -$	$\langle S \rangle \rightarrow \langle L \rangle\langle X_5 \rangle$ $\langle X_5 \rangle \rightarrow \langle X_6 \rangle\langle T \rangle$ $\langle X_6 \rangle \rightarrow *$
Smena 4.	**Smena 8.**	**Smena 9.**
$\langle S \rangle \rightarrow \langle L \rangle\langle X_7 \rangle$ $\langle X_7 \rangle \rightarrow \langle X_8 \rangle\langle T \rangle$ $\langle X_8 \rangle \rightarrow /$	$\langle T \rangle \rightarrow \langle L \rangle\langle X_9 \rangle$ $\langle X_9 \rangle \rightarrow \langle X_{10} \rangle\langle T \rangle$ $\langle X_{10} \rangle \rightarrow *$	$\langle T \rangle \rightarrow \langle L \rangle\langle X_{11} \rangle$ $\langle X_{11} \rangle \rightarrow \langle X_{12} \rangle\langle T \rangle$ $\langle X_{12} \rangle \rightarrow /$

Takođe, i smene 5., 10. i 13. su istog transformacionog tipa. Pokažimo ovu transformaciju na primeru smene 5.

Primenjujući korak 4 dobijamo:

$\langle S \rangle \rightarrow \langle X_{13} \rangle\langle X_{14} \rangle$

$\langle X_{13}\rangle \rightarrow ($

$\langle X_{14}\rangle \rightarrow \langle S\rangle)$

Sada se primenjuje korak 3. na X_{14}:

$\langle X_{14}\rangle \rightarrow \langle S\rangle\langle X_{15}\rangle$

$\langle X_{15}\rangle \rightarrow)$

Konačno, dobija se:

Smena 5.	Smena 10.	Smena 13.
$\langle S\rangle \rightarrow \langle X_{13}\rangle\langle X_{14}\rangle$ $\langle X_{14}\rangle \rightarrow \langle S\rangle\langle X_{15}\rangle$ $\langle X_{13}\rangle \rightarrow ($ $\langle X_{15}\rangle \rightarrow)$	$\langle T\rangle \rightarrow \langle X_{16}\rangle\langle X_{17}\rangle$ $\langle X_{17}\rangle \rightarrow \langle S\rangle\langle X_{18}\rangle$ $\langle X_{16}\rangle \rightarrow ($ $\langle X_{18}\rangle \rightarrow)$	$\langle L\rangle \rightarrow \langle X_{19}\rangle\langle X_{20}\rangle$ $\langle X_{20}\rangle \rightarrow \langle S\rangle\langle X_{21}\rangle$ $\langle X_{19}\rangle \rightarrow ($ $\langle X_{21}\rangle \rightarrow)$

Dakle, polazna gramatika transformisana u normalnu formu Čomskog dobija sledeći oblik:

$\langle S\rangle \rightarrow\langle T\rangle\langle X_1\rangle$	$\langle X_1\rangle \rightarrow \langle X_2\rangle\langle S\rangle$	$\langle X_2\rangle \rightarrow +$	$\langle T\rangle \rightarrow \langle L\rangle\langle X_9\rangle$	$\langle L\rangle \rightarrow \langle X_{19}\rangle\langle X_{20}\rangle$
$\langle S\rangle \rightarrow \langle T\rangle\langle X_3\rangle$	$\langle X_3\rangle \rightarrow \langle X_4\rangle\langle S\rangle$	$\langle X_4\rangle \rightarrow -$	$\langle T\rangle \rightarrow \langle L\rangle\langle X_{11}\rangle$	$\langle L\rangle \rightarrow a$
$\langle S\rangle \rightarrow \langle L\rangle\langle X_5\rangle$	$\langle X_5\rangle \rightarrow \langle X_6\rangle\langle T\rangle$	$\langle X_6\rangle \rightarrow +$	$\langle T\rangle \rightarrow \langle X_{16}\rangle\langle X_{17}\rangle$	$\langle L\rangle \rightarrow b$
$\langle S\rangle \rightarrow \langle L\rangle\langle X_7\rangle$	$\langle X_7\rangle \rightarrow \langle X_8\rangle\langle T\rangle$	$\langle X_8\rangle \rightarrow /$	$\langle T\rangle \rightarrow a$	
$\langle S\rangle \rightarrow\langle X_{13}\rangle\langle X_{14}\rangle$	$\langle X_9\rangle \rightarrow \langle X_{10}\rangle\langle T\rangle$	$\langle X_{10}\rangle \rightarrow *$	$\langle T\rangle \rightarrow b$	
$\langle S\rangle \rightarrow a$	$\langle X_{11}\rangle \rightarrow \langle X_{12}\rangle\langle T\rangle$	$\langle X_{12}\rangle \rightarrow /$		
$\langle S\rangle \rightarrow b$	$\langle X_{14}\rangle \rightarrow \langle S\rangle\langle X_{15}\rangle$	$\langle X_{13}\rangle \rightarrow)$		
	$\langle X_{17}\rangle \rightarrow \langle S\rangle\langle X_{18}\rangle$	$\langle X_{15}\rangle \rightarrow)$		
	$\langle X_{20}\rangle \rightarrow \langle S\rangle\langle X_{21}\rangle$	$\langle X_{16}\rangle \rightarrow ($		
		$\langle X_{18}\rangle \rightarrow)$		
		$\langle X_{19}\rangle \rightarrow ($		
		$\langle X_{21}\rangle \rightarrow)$		

Diskusija rešenja

Broj smena ovom transformacijom se povećao četiri puta. Dok neterminali u polaznoj gramatici imaju uglavnom svoju vrlo jasnu semantičku interpretaciju, za novouvedene se to ne može reći. To je naročito neugodno pri dijagnostici grešaka kada je nemoguće dati njihovo razumljivo tumačenje. Ostavlja se čitaocu da sprovede transformaciju ako su u gramatici dozvoljene prazne smene.

Zadatak 2.4.12

Data je sledeća gramatika:

1. $\langle S\rangle \rightarrow p$
2. $\langle S\rangle \rightarrow \sim\langle S\rangle$
3. $\langle S\rangle \rightarrow (\langle S\rangle \supset \langle S\rangle)$

Naći normalnu formu Greibach-ove za ovu gramatiku.

Analiza problema

Za svaku bezkontekstnu gramatiku G može se naći ekvivalentna gramatika G_e u normalnoj formi Greibach-ove u kojoj je svaka smena oblika

$$\langle A\rangle \rightarrow a\alpha$$

gde je <A> neterminal, a terminal, a α niz neterminala koji može da bude prazan. Da bi se našla gramatika G_e potrebno je najpre transformisati G u normalnu formu Čomskog (G_c), a zatim numerisati neterminale (na primer, $\langle T_1\rangle$, $\langle T_2\rangle$, ..., $\langle T_m\rangle$) i onda primeniti određene transformacije. Svrha zadatka je da se pronađu te transformacije.

Rešenje

Da bi smo dobili normalnu formu Čomskog, treba transformisati smene 2. i 3. Njihovom obradom dobija se

Smena 2.	Smena 3.	
$\langle S\rangle \rightarrow \langle T_2\rangle\langle S\rangle$	$\langle S\rangle \rightarrow \langle T_3\rangle\langle T_4\rangle$	$\langle T_6\rangle \rightarrow \supset$
$\langle T_2\rangle \rightarrow \sim$	$\langle T_3\rangle \rightarrow ($	$\langle T_7\rangle \rightarrow \langle S\rangle\langle T_8\rangle$
	$\langle T_4\rangle \rightarrow \langle S\rangle\langle T_5\rangle$	$\langle T_8\rangle \rightarrow)$
	$\langle T_5\rangle \rightarrow \langle T_6\rangle\langle T_7\rangle$	

Ovde je već izvršena numeracija neterminala.Ostalo je još da <S> zamenimo sa $\langle T_1\rangle$, pa G_c dobija oblik:

1. $\langle T_1\rangle \rightarrow \langle T_2\rangle\langle T_1\rangle$	6. $\langle T_7\rangle \rightarrow \langle T_1\rangle\langle T_8\rangle$
2. $\langle T_1\rangle \rightarrow \langle T_3\rangle\langle T_4\rangle$	7. $\langle T_2\rangle \rightarrow \sim$
3. $\langle T_1\rangle \rightarrow p$	8. $\langle T_3\rangle \rightarrow ($
4. $\langle T_4\rangle \rightarrow \langle T_1\rangle\langle T_5\rangle$	9. $\langle T_6\rangle \rightarrow \supset$
5. $\langle T_5\rangle \rightarrow \langle T_6\rangle\langle T_7\rangle$	10. $\langle T_8\rangle \rightarrow)$

Modifikujmo G_c tako da ako je smena oblika $\langle T_i\rangle \rightarrow \langle T_j\rangle\gamma$, onda je $j > i$. Ovo se može sprovesti polazeći od $\langle T_1\rangle$ pa idući sve do $\langle T_8\rangle$ na sledeći način. Usvojimo da su smene

promenjene tako da za $1 \le i \le k$, $\langle T_i \rangle \to \langle T_j \rangle\gamma$ je važeća smena samo ako je $j > i$. Transformišimo sada smene za $\langle T_{k+1} \rangle$.

Ako je $\langle T_{k+1} \rangle \to \langle T_j \rangle\gamma$ smena sa $j < k+1$, generisati novi skup smena zamenjujući $\langle T_j \rangle$ desnim stranama svake smene za $\langle T_j \rangle$. Ponavljajući ovaj postupak najviše k–1 puta dobijaju se smene oblika

$$\langle T_{k+1} \rangle \to \langle T_l \rangle\gamma,\ l \ge k+1 \quad \text{ili} \qquad \langle T_{k+1} \rangle \to a\alpha, \text{ gde je } \alpha \text{ niz neterminala}$$

Prva smena koja ne zadovoljava postavljeni uslov uređenja je smena 4.

$$\langle T_4 \rangle \to \langle T_1 \rangle\langle T_5 \rangle$$

Primenom navedene transformacije dobija se:

$$\langle T_4 \rangle \to \langle T_2 \rangle\langle T_1 \rangle\langle T_5 \rangle \quad \Rightarrow \quad \langle T_4 \rangle \to \sim\langle T_1 \rangle\langle T_5 \rangle$$

$$\langle T_4 \rangle \to \langle T_3 \rangle\langle T_4 \rangle\langle T_5 \rangle \quad \Rightarrow \quad \langle T_4 \rangle \to (\langle T_4 \rangle\langle T_5 \rangle$$

$$\langle T_4 \rangle \to p\langle T_5 \rangle \quad \Rightarrow \quad \langle T_4 \rangle \to p\langle T_5 \rangle$$

Nakon ovoga smena $\langle T_4 \rangle \to \langle T_1 \rangle\langle T_5 \rangle$ može se ukloniti. Sledeća smena koja podleže istoj transformaciji je smena 6.

$$T_7 \to \langle T_1 \rangle\langle T_8 \rangle.$$

Na sličan način dobija se:

$$\langle T_7 \rangle \to \langle T_2 \rangle\langle T_1 \rangle\langle T_8 \rangle \quad \Rightarrow \quad \langle T_7 \rangle \to \sim\langle T_1 \rangle\langle T_8 \rangle$$

$$\langle T_7 \rangle \to \langle T_3 \rangle\langle T_4 \rangle\langle T_8 \rangle \quad \Rightarrow \quad \langle T_7 \rangle \to (\langle T_4 \rangle\langle T_8 \rangle$$

$$\langle T_7 \rangle \to p\langle T_8 \rangle \quad \Rightarrow \quad \langle T_7 \rangle \to p\langle T_8 \rangle$$

Transformisana gramatika dobija oblik:

1. $\langle T_1 \rangle \to \langle T_2 \rangle\langle T_1 \rangle$
2. $\langle T_1 \rangle \to \langle T_3 \rangle\langle T_4 \rangle$
3. $\langle T_1 \rangle \to p$
4. $\langle T_4 \rangle \to \sim\langle T_1 \rangle\langle T_5 \rangle$
5. $\langle T_4 \rangle \to (\langle T_4 \rangle\langle T_5 \rangle$
6. $\langle T_4 \rangle \to p\langle T_5 \rangle$
7. $\langle T_5 \rangle \to \langle T_6 \rangle\langle T_7 \rangle$
8. $\langle T_7 \rangle \to \sim\langle T_1 \rangle\langle T_8 \rangle$
9. $\langle T_7 \rangle \to (\langle T_4 \rangle\langle T_8 \rangle$
10. $\langle T_7 \rangle \to p\langle T_8 \rangle$
11. $\langle T_2 \rangle \to \sim$
12. $\langle T_3 \rangle \to ($
13. $\langle T_6 \rangle \to \supset$
14. $\langle T_8 \rangle \to)$

Uočava se da transformacijama nije dobijena nijedna smena oblika $\langle T_k \rangle \to \langle T_k \rangle\gamma$, čime se proces transformisanja uprošćava.

Gledajući smene u dobijenoj gramatici vidimo da samo smene za $\langle T_1 \rangle$ i $\langle T_5 \rangle$ imaju na krajnjoj levoj strani neterminale. Idući unazad kroz gramatiku, to jest, od smena sa najvišim rednim brojem neterminala i zamenjujući njihove desne strane u smene sa nižim rednim brojem dobija se gramatika u normalnoj formi Greibach-ove. Dakle, naša gramatika postaje:

1. $<T_1> \to \sim<T_1>$	8. $<T_7> \to \sim<T_1><T_8>$
2. $<T_1> \to (<T_4>$	9. $<T_7> \to (<T_4><T_8>$
3. $<T_1> \to p$	10. $<T_7> \to p<T_8>$
4. $<T_4> \to \sim<T_1><T_5>$	11. $<T_2> \to \sim$
5. $<T_4> \to (<T_4><T_5>$	12. $<T_3> \to ($
6. $<T_4> \to p<T_5>$	13. $<T_6> \to \supset$
7. $<T_5> \to \supset<T_7>$	14. $<T_8> \to)$

Diskusija rešenja

Prevođenje gramatike u normalnu formu Greibach-ove omogućuje da se odredi da li je gramatika tipa LL(1). Ako su selekcioni skupovi disjunktni, gramatika je LL(1). U tom smislu je i data polazna gramatika tipa LL(1). U ovom posebnom slučaju, ovakva transformacija i nije bila potrebna da bi se utvrdio tip gramatike jer se to prostom inspekcijom može odmah ustanoviti.

Za slučaj prethodnih transformacija nismo imali smene oblika:

$$<T_k> \to <T_k>\gamma_1,\ <T_k> \to <T_k>\gamma_2,\ldots,\ <T_k> \to <T_k>\gamma_r$$

Ako se takve smene pojave, onda treba primeniti određenu transformaciju uvodeći novi neterminal $<Z_k>$. Neka postoje još smene oblika:

$$<T_k> \to \beta_1,\ <T_k> \to \beta_2,\ldots,\ <T_k> \to \beta_s$$

gde β_i za $1 \le i \le s$ ne počinje neterminalom $<T_k>$. Tada navedene smene zamenjujemo smenama:

$$\left.\begin{array}{l} <T_k> \to \beta_i \\ <T_k> \to \beta_i<Z_k> \end{array}\right\} \ 1 \le i \le s$$

$$\left.\begin{array}{l} <Z_k> \to \gamma_i \\ <Z_k> \to \gamma_i<Z_k> \end{array}\right\} \ 1 \le i \le r$$

Ponavljajući celokupan proces transformacije za svaki polazni neterminal, dobijaju se smene oblika:

1. $<T_k> \to <T_l>\mu,$ $\quad l > k$

2. $<T_k> \to a\mu,$ $\quad a \in V_T$

3. $<Z_k> \to \mu,$ $\quad \mu \in (V_N \cup \{<Z_1>, <Z>, \ldots, <Z_m>\})^*$

Krajnje levi simbol na desnoj strani svake smene za $<T_k>$ mora da bude terminal pošto je $<T_k>$ neterminal sa najvećim rednim brojem. Krajnje levi simbol na desnoj strani bilo koje smene za $<T_{k-1}>$ mora da bude $<T_k>$ ili terminal.Ako je $<T_k>$, generišu se nove smene zamenom $<T_k>$ njegovim desnim stranama. Desne strane tako dobijenih smena mora da otpočinju terminalima. Ovaj postupak se nastavlja nad smenama $<T_{k-2}>, \ldots, <T_2>, <T_1>$ sve dok svaka smena za $<T_i>$ ne počinje terminalom.

U poslednjem koraku, ispituju se smene novih neterminala $<Z_1>$, $<Z_2>$, ... , $<Z_m>$. Ove smene počinju bilo terminalom bilo neterminalom polazne gramatike. Ako smena za $<Z_i>$ počinje neterminalom $<T_j>$, onda njega zamenjujemo svim smenama oblika

$$<T_j> \rightarrow a\gamma$$

jer su sve smene za neterminale $<T_j>$ već dovedene u formu Greibach-ove. Nakon toga i smene za $<Z_i>$ dobijaju istu formu.

Iz zadatka se vidi da se broj smena skoro udvostručuje u odnosu na originalnu gramatiku. To povećanje može da bude i veće ako se pojave u procesu transformacije i smene oblika $<T_k> \rightarrow <T_k>\gamma$. Ovako veliko povećanje je posledica samog načina transformacije koji zahteva najpre transformaciju gramatike u normalnu formu Čomskog a zatim u normalnu formu Greibach-ove.

Zadatak 2.4.13

Ako je gramatika G data u normalnoj formi Čomskog, i za $w \in L(G)$ postoji izvođenje od p koraka, koliko w ima terminala? Pretpostaviti da ne postoje smene oblika $<A> \rightarrow \varepsilon$.

Analiza problema

Prema teoriji koju je dao Čomski, svaki bezkontekstni jezik se može generisati gramatikom u kojoj su smene oblika $<A> \rightarrow <B><C>$ ili $<A> \rightarrow a$; drugim rečima, svaka se bezkontekstna gramatika može transformisati u oblik u kojem svaka smena ima ili dva neterminala na desnoj strani ili jedan terminal, takođe, na desnoj strani. Ograničavanjem oblika i dužine smena omogućuje da se na relativno jednostavan način odredi dužina proizvedene sentence u zavisnosti od broja koraka u izvođenju u datoj gramatici; dakle

$$|w| = f(p)$$

gde je

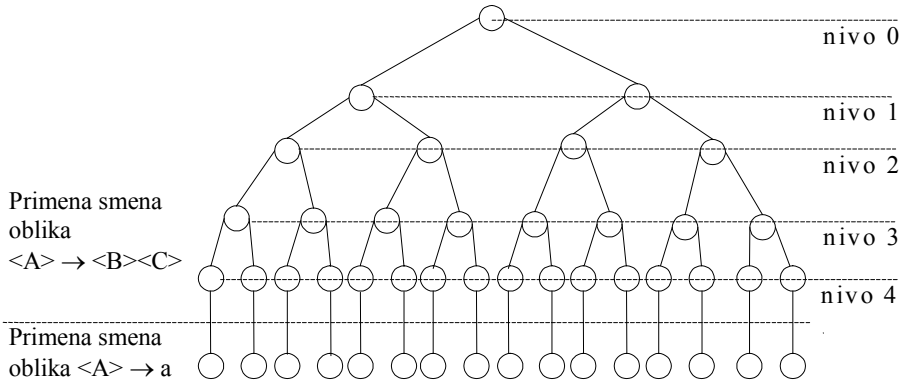
$|w|$ – dužina sentence

p – broj koraka.

Korak ćemo definisati kao primenu jedne smene u izvođenju sentence.

Rešenje

Izvođenje u ovakvom tipu bezkontekstne gramatike se može prikazati binarnim stablom. Definišemo puno binarno stablo izvođenja kao ono stablo u kojem se u svim čvorovima primenjuju smene oblika $<A> \rightarrow <B><C>$ a na nivou stabla d smene oblika $<A> \rightarrow a$, gde je d unapred zadata veličina. Nivo stabla je određen brojem čvorova od korena stabla (ne računajući njega ako ne postoji smena tipa $<S> \rightarrow a$) do poslednjeg neterminalnog čvora, Sl. 2.4.10.



Sl. 2.4.10: Puno binarno stablo izvođenja

Na Sl. 2.4.10 je dat primer punog binarnog stabla izvođenja za d = 4. Odavde se lako uočava da je maksimalna dužina sentence definisane na osnovu punog binarnog stabla i za proizvoljnu dužinu d data sa

$$|w|_{max} = 2^d \left\{ \begin{array}{l} d > 0, \text{ za } <S> \not\rightarrow a \\ d \geq 0, \text{ za } <S> \rightarrow a \end{array} \right\}$$

Jasno, ovde je i pretpostavljeno da se na nivou d nalaze samo oni neterminali za koje postoje smene oblika $<A> \rightarrow a$.

Broj koraka p_{max} koji odgovara dužini $|w|_{max}$ se jednostavno nalazi i iznosi:

$$p_{max} = 2^{d+1} - 1$$

Dovoljno je sabrati čvorove po svim nivoima, polazeći od nivoa 0 (jer se primenjuje prva smena) pa sve do nivoa d:

$$1 + 2 + 4 + 8 + 16 + 32 + ... =$$

$$2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + ... + 2^d = \frac{2^{d+1} - 1}{2 - 1} = 2^{d+1} - 1 = p_{max}$$

Tako se, na primer, za slučaj sa Sl. 2.4.10, dobija:

$$\boxed{p_{max} = 31, \ |w|_{max} = 16}$$

Pretpostavimo sada da broj koraka nije maksimalan, već iznosi p. To znači da sada nemamo puno binarno stablo izvođenja, već se na nekim nivoima stabla izvođenja manjim od d primenjuju smene oblika $<A> \rightarrow a$. Nađimo razliku p_{dif} između maksimalnog mogućeg i stvarnog broja koraka u izvođenju sentence w i za nivo stabla d koji sada predstavlja maksimalnu putanju u stablu:

$$\boxed{p_{dif} = 2^{d+1} - 1 - p}$$

Veza između p_{max} i $|w|_{max}$ se lako nalazi i iznosi:

$$\log_2|w|_{max} = \log_2(p_{max} + 1) - 1$$

to jest,

$$\log_2|w|_{max} = \log_2(p_{max} + 1) / 2$$

ili

$$\boxed{|w|_{max} = (p_{max} + 1) / 2}$$

Dužina maksimalne sentence $|w|_{max}$ biće umanjena za broj terminala u zavisnosti od p_{dif}. Označimo to umanjenje sa $|w|_u$, i nađimo

$$|w|_u = f(p)$$

Očigledno je da ako na nekom nivou i u nekom čvoru i primenimo smenu oblika <A> → a, da se čitavo podstablo koje polazi od tog čvora eliminiše dalje iz izvođenja. Ako je time broj koraka izvođenja smanjen za p_{dif}^i, smanjenje dužine sentence koja potiče od datog čvora $|w|_u^i$ iznosi

$$|w|_u^i = p_{dif}^i / 2$$

jer se u čvoru i primenjuje ipak jedno izvođenje oblika <A> → a.

Odavde sledi da je

$$|w|_u = \sum|w|_u^i = \sum p_{dif}^i \ / \ 2 = \frac{p_{dif}}{2}$$

Dakle, dobijamo

$$|w| = |w|_{max} - |w|_u = \frac{p_{max} + 1}{2} - \frac{p_{dif}}{2} = \frac{p^{d+1} - 1 + 1}{2} - \frac{p^{d+1} - 1 - p}{2}$$

$$\boxed{|w| = \frac{p + 1}{2}}$$

Diskusija rešenja

Očigledno je da p ne može da ima proizvoljne, već određene vrednosti koje moraju da budu neparne; koje sve vrednosti p može da ima zavisi od gramatike.

Zadatak 2.4.14

Neka je G nedvosmislena bezkontekstna gramatika bez praznih smena. Ako $w \in L(G)$, pokazati da je broj koraka potreban da se izvede sentenca w linearno zavisan od dužine w.

Analiza problema

Da bi se problem rešio, pogodno je gramatiku G transformisati u normalnu formu Čomskog gde su smene oblika <A> → <B> <C> i <A> → a.

Rešenje

Zadatak 2.4.13 rešava problem zavisnosti dužine sekvence $w \in L(G)$ od broja koraka za gramatiku datu u normalnoj formi Čomskog i bez praznih smena, odakle se lako izvodi obratna zavisnost

$p = 2|w| - 1$

gde je

p – broj koraka izvođenja sekvence w u gramatici Čomskog i

|w| – dužina sekvence w.

2.5. Programski primeri

Zadatak 2.5.1

Realizovati na C-u algoritam eliminacije suvišnih neterminala i smena iz zadate gramatike. Program na izlazu treba da ispiše rezultujuću gramatiku, kao i da ispiše koji neterminali su mrtvi, a koji su nedostižni. Gramatika se zadaje u tekstualnoj datoteci. Svaki red odgovara jednoj smeni. Svako slovo odgovara jednom gramatičkom simbolu (neterminalima velika slova, terminalima mala). Prvo slovo u redu odgovara simbolu leve strane smene, zatim ide razmak pa niz simbola desne strane smene, npr. smena <X> → <Y> <Z> v w se zadaje sa

X Yvw

Iz praktičnih razloga može se uzeti da dužina desnih strana smena ne prelazi dvadeset simbola.

Rešenje

Priloženi program za ulaz na Sl. 2.5.1(a) daje izlaz na Sl. 2.5.1(b).

```
S Ab
S cC
A DB
A a
B
C cDb
D aAC
```

(a)

```
DEAD: C D

S -> Ab
A -> a
B ->

UNREACHABLE: B

S -> Ab
A -> a
```

(b)

Sl. 2.5.1

U programu koji sledi, leve strane gramatičkih smena pamte se u znakovnom vektoru LHS[], a desne strane u vektoru znakovnih nizova RHS[]. Vektor alive[], za proizvoljan neterminal <X> čuva informaciju da li je <X> živ, ukoliko je alive['X'] jednako 1, ili mrtav, ukoliko je

alive[‘X’] jednako 0. Na sličan način vektor reachable[] čuva informaciju o dostižnim i nedostižnim neterminalima.

Gramatika se čita sa standardnog ulaza procedurom read_grammar(). Zatim se procedurom calc_alive() određuje koji su neterminali živi a koji mrtvi (vidi Zadatak 2.3.1). Procedura print_dead() na izlazu ispisuje mrtve neterminale. Procedura eliminate_dead() izbacuje iz gramatike sve smene u kojima se pojavljuju mrtvi neterminali. Procedura print_grammar() ispisuje izmenjenu gramatiku. Procedura calc_reachable() u ovoj gramatici određuje dostižne i nedostižne neterminale (vidi Zadatak 2.3.1), ovi poslednji se ispisuju procedurom print_unreachable() i zatim iz gramatike eliminišu smene u kojima se oni pojavljuju procedurom eliminate_unreachable(). Izmenjena gramatika se ispisuje procedurom print_grammar().

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAX_PROD 100
#define MAX_RHS   20

char LHS[MAX_PROD];
char RHS[MAX_PROD][MAX_RHS];
int alive[128];
int reachable[128];

int num_prod;

void read_grammar(void){
   char line[80];
   num_prod=0;
   while ( gets(line) != NULL ){
      LHS[num_prod] = line[0];
      strcpy( RHS[num_prod++], ( ( strlen( line ) > 1 ) ? line + 2 : "" ) );
   }
}

void print_grammar(void){
   int i;
   for ( i = 0; i < num_prod; i++ )
      printf("%c -> %s\n", LHS[i], RHS[i]);
   printf("\n");
}

void calc_alive(void){
   int i,j;
   int change;
   for ( i = 'A'; i <= 'Z'; i++ )
      alive[i] = 0;
   for ( i = 'a'; i <= 'z'; i++ )
```

```
        alive[i] = 1;
    do{
        change = 0;
        for ( i = 0; i < num_prod; i++ )
            if ( !alive[LHS[i]] ){
                int alive_rhs = 1;
                for ( j = 0; j < strlen( RHS[i] ); j++ )
                    if ( !alive[RHS[i][j]] )
                        alive_rhs = 0;
                    if (alive_rhs){
                        alive[LHS[i]] = 1;
                        change = 1;
                    }
            }
    } while ( change );
}

void eliminate_dead(void){
    int i,j;
    int k = 0;
    for ( i = 0; i < num_prod; i++ )
        if ( alive[LHS[i]] ){
            int alive_rhs = 1;
            for ( j = 0; j < strlen( RHS[i] ); j++ )
                if ( !alive[RHS[i][j]] )
                    alive_rhs = 0;
                if (alive_rhs){
                    LHS[k] = LHS[i];
                    strcpy( RHS[k++], RHS[i] );
                }
        }
        num_prod = k;
}

void print_dead(void){
    int i,j;
    int printed[128];
    for ( i = 0; i < 128; i++ )
        printed[i] = 0;
    printf("\n\nDEAD:");
    for ( i = 0; i < num_prod; i++ ){
        if ( !alive[LHS[i]] && !printed[LHS[i]] ){
            printf(" %c", LHS[i]);
            printed[LHS[i]] = 1;
        }
        for ( j = 0; j < strlen( RHS[i]) ; j++ )
            if ( !alive[RHS[i][j]] && !printed[RHS[i][j]] ){
```

```
            printf(" %c", RHS[i][j]);
            printed[RHS[i][j]] = 1;
        }
    }
    printf("\n\n");
}

void calc_reachable(void){
    int i,j;
    int change;
    for ( i = 'A'; i <= 'Z'; i++ )
        reachable[i] = 0;
    reachable[LHS[0]] = 1;
    do{
        change = 0;
        for ( i = 0; i < num_prod; i++ )
            if ( reachable[LHS[i]] )
                for ( j = 0; j < strlen( RHS[i] ); j++ )
                    if ( isupper( RHS[i][j]) && !reachable[RHS[i][j]] ){
                        reachable[RHS[i][j]] = 1;
                        change =1;
                    }
    } while ( change );
}

void eliminate_unreachable(void){
    int i;
    int k = 0;
    for ( i = 0; i < num_prod; i++ )
        if ( reachable[LHS[i]] ){
            LHS[k] = LHS[i];
            strcpy( RHS[k++], RHS[i] );
        }
        num_prod = k;
}

void print_unreachable(void){
    int i;
    int printed[128];
    for ( i = 0; i < 128; i++ )
        printed[i] = 0;
    printf("\n\nUNREACHABLE:");
    for ( i = 0; i < num_prod; i++ )
        if ( !reachable[LHS[i]] && !printed[LHS[i]] ){
            printf(" %c", LHS[i]);
            printed[LHS[i]] = 1;
        }
```

```
        printf("\n\n");
}

main(){
    read_grammar();
    calc_alive();
    print_dead();
    eliminate_dead();
    print_grammar();
    calc_reachable();
    print_unreachable();
    eliminate_unreachable();
    print_grammar();
}
```

3. Sintaksna analiza od vrha ka dnu

3.1. Potisni automati

Zadatak 3.1.1

a) Izložiti tri sekvence iz skupa sekvenci koje prepoznaje potisni automat sa jednim stanjem prikazan na Sl. 3.1.1. Startni stek je ∇. Prikazati rad automata za svaku od ovih sekvenci.

b) Opisati skup sekvenci koje dati automat prihvata.

S_0:

	a	b	c	⊣
A	PUSH(A) ADVANCE	REJECT	POP ADVANCE	REJECT
B	PUSH(C) ADVANCE	POP RETAIN	PUSH(A) ADVANCE	REJECT
C	PUSH(B) ADVANCE	PUSH(C) ADVANCE	POP ADVANCE	REJECT
∇	PUSH(A) ADVANCE	PUSH(B) ADVANCE	REJECT	ACCEPT

Sl. 3.1.1

Analiza problema

Potisni automat definisan je uređenom šestorkom (U, Y, Y_s, S, S_t, δ_c) gde:

- U predstavlja skup ulaznih simbola. U konkretnom slučaju, ulazni simboli označavaju pojedine kolone tabele (osim poslednje) na Sl. 3.1.1, pa je U = {a, b, c}.

- Y predstavlja skup simbola steka. U konkretnom slučaju, stek simboli obeležavaju pojedine vrste tabele na Sl. 3.1.1, pa je Y = {A, B, C, ∇}.
- Y_s je početna konfiguracija steka i definisana je sekvencom simbola steka koja počinje simbolom ∇. Simbol ∇ naziva se dnom steka.
- S predstavlja skup stanja potisnog automata. U konkretnom slučaju, S = $\{S_0\}$.
- $S_t \in S$ predstavlja startno stanje automata, u konkretnom slučaju to je jedino stanje automata S_0.
- δ_c: $(U \cup \{\dashv\}) \times Y \times S \rightarrow O$ predstavlja kontrolnu funkciju koja za svaki ulazni simbol (kojima se, iz praktičnih razloga, dodaje marker kraja ulazne sekvence ⊣), svaki simbol steka i svako stanje definiše određeni element skupa svih mogućih operacija potisnog automata $O = O_y \times S \times O_u \cup \{ACCEPT, REJECT\}$, pri čemu:
  - O_y = {POP, PUSH(x), nepromenjen stek} je skup mogućih operacija nad stekom, POP označava skidanje jednog simbola sa steka, PUSH(x) označava stavljanje simbola steka x na vrh steka.
  - O_u = {ADVANCE, RETAIN} je skup mogućih operacija nad ulaznom sekvencom. ADVANCE označava da sledeći ulazni simbol iza tekućeg postaje novi tekući simbol, a RETAIN označava da se tekući simbol ulaza ne menja.

  Prema tome, svaka operacija automata sastoji se iz određene operacije nad stekom, promene stanja u određeno stanje $s \in S$ i određene operacije nad ulazom. Izuzetak od ovoga su operacije ACCEPT koja označava kraj rada automata pri čemu se ulazna sekvenca prihvata i REJECT koja označava kraj rada pri čemu se ulazna sekvenca odbija. Kontrolna funkcija δ_c je u konkretnom slučaju zadata kontrolnom tabelom na Sl. 3.1.1.

Rad automata odvija se po sledećem algoritmu:

```
tekuće_stanje := St;
tekući_ulaz := prvi simbol ulazne sekvence;
tekuća_konfiguracija_steka := Ys;
loop
    if δc( tekući_ulaz, tekuća_konfiguracija_steka, tekuće_stanje ) = ACCEPT
        then kraj rada uz prihvatanje ulazne sekvence;
    else if δc( tekući_ulaz, tekuća_konfiguracija_steka, tekuće_stanje ) = REJECT
        then kraj rada uz odbijanje ulazne sekvence
    else
        δc( tekući_ulaz, tekuća_konfiguracija_steka, tekuće_stanje ) je oblika (y, s, u);
        primeniti na tekuću_konfiguraciju_steka operaciju y;
        tekuće_stanje := s;
        primeniti na tekući_ulaz operaciju u;
    end if
end loop;
```

Skup svih sekvenci ulaznih simbola koje automat prihvata naziva se jezik tog automata.

Rešenje

a)

Primeri sekvenci koje prihvata gore dati potisni automat:

1) prazna ulazna sekvenca: ⊣

	stek:	ulaz:	akcija:
1.	∇	⊣	ACCEPT

2) ulazna sekvenca: ac⊣

	stek:	ulaz:	akcija:
1.	∇	ac ⊣	PUSH(A), ADVANCE
2.	∇ A	c ⊣	POP, ADVANCE
3.	∇	⊣	ACCEPT

3) ulazna sekvenca: aaccac⊣

	stek:	ulaz:	akcija:
1.	∇	aaccac ⊣	PUSH(A), ADVANCE
2.	∇ A	accac ⊣	PUSH(A), ADVANCE
3.	∇AA	ccac ⊣	POP, ADVANCE
4.	∇ A	cac ⊣	POP, ADVANCE
5.	∇	ac ⊣	PUSH(A), ADVANCE
6.	∇ A	c ⊣	POP, ADVANCE
7.	∇	⊣	ACCEPT

b)

U cilju određivanja skupa sekvenci koje prihvata automat sa Sl. 3.1.1, prvo ćemo pokazati da A predstavlja jedini simbol steka (osim ∇) koji se, za prihvatljive ulazne sekvence, može pojaviti na steku u toku rada automata.

Primetimo najpre da se simbol C može pojaviti na steku jedino ako je simbol B već prisutan na steku, kao rezultat akcije u ulazu (B, a) kontrolne tabele. Jedina preostala akcija koja potiskuje C na stek je akcija u ulazu (C, b) kontrolne tabele, međutim kod nje se podrazumeva da se jedno C već nalazi na vrhu steka. Dakle, ako utvrdimo da se za prihvatljive sekvence B ne može pojaviti na steku, iz toga neposredno sledi da se ni C ne može pojaviti na steku.

Razmotrimo načine na koje se simbol B može pojaviti na steku. To se može desiti kao rezultat akcija u ulazima (∇, b) i (C, a) kontrolne tabele. Jedini način da se simbol B prvi put potisne na stek je izvršavanje akcije u ulazu (∇, b), s obzirom da pojava C na steku kao preduslov za izvršavanje akcije u ulazu (C, a) implicira da se na steku već nalazi simbol B u skladu sa ranijim zaključcima.

Sada ćemo pokazati da bilo koja ulazna sekvenca za koju se konsultuje ulaz (∇, b) kontrolne tabele ne može biti prihvaćena. Konfiguracija steka posle izvršavanja ove akcije je ∇B. Da bi sekvenca bila prihvaćena, stek mora biti prazan kada automat konzumira celokupan ulazni niz. Dakle, simbol B mora biti uklonjen sa steka. Jedini način da se ovaj simbol ukloni iz konfiguracije steka ∇B je da se izvrši akcija POP kada se B nalazi na vrhu steka. Ovaj preduslov zadovoljava jedino ulaz (B, b) kontrolne tabele. Međutim, akcije POP, RETAIN uslovljavaju obavezno konsultovanje ulaza (∇, b) u narednom koraku rada automata i izvršavanje akcija PUSH(B), ADVANCE čime se B vraća na stek. S obzirom da nema načina da se stek isprazni jednom kada se simbol B nađe na njemu, zaključujemo da automat ne može da završi rad sa ACCEPT.

Prema tome, pri određivanju skupa prihvatljivih ulaznih sekvenci mogu se zanemariti vrste B i C i ulaz (∇, b) kontrolne tabele. S obzirom da ulaz (A, b) sadrži REJECT, znači da se ulazni simbol b ne može pojaviti u prihvatljivoj sekvenci. Automat prema tome možemo uprostiti bez uticaja na skup sekvenci koji prepoznaje izostavljanjem vrsta B i C i kolone b (Sl. 3.1.2).

	a	c	⊣
A	PUSH(A) ADVANCE	POP ADVANCE	REJECT
∇	PUSH(A) ADVANCE	REJECT	ACCEPT

Sl. 3.1.2

Prazna ulazna sekvenca se prihvata. Neprazna ulazna sekvenca mora počinjati simbolom a da bi bila prihvatljiva . U nastavku mogu slediti simboli a i c. Simbol a na ulazu izaziva potiskivanje jednog simbola A na stek. Simbol c na ulazu izaziva skidanje jednog simbola A sa steka. Da bi sekvenca bila prihvatljiva, ni u jednom trenutku rada automata broj procesiranih simbola c ne sme preći broj procesiranih simbola a, jer bi u tom slučaju parser konsultovao ulaz (∇, c). U celokupnoj sekvenci, broj simbola a mora biti jednak broju simbola c da bi sekvenca bila prihvaćena.

Sekvenca se dakle prihvata ako i samo ako je prazna, ili se sastoji od simbola a i c, pri čemu broj simbola c nikada ne prelazi broj simbola a u bilo kojoj podsekvenci kojom počinje posmatrana sekvenca, a ukupan broj simbola a u posmatranoj sekvenci jednak je ukupnom broju simbola c u posmatranoj sekvenci.

Zadatak 3.1.2

Izložiti tri sekvence koje potisni automat sa Sl. 3.1.3 prihvata i prikazati promene na steku prilikom rada automata.

Startni stek je ∇A.

	0	1	⊣
A	REPLACE (AA) ADVANCE	POP ADVANCE	REJECT
∇	REJECT	REJECT	ACCEPT

Sl. 3.1.3

Analiza problema

Zadati automat koristi operaciju REPLACE, koja predstavlja proširenje primitivnih operacija za rad sa stekom. Operacija REPLACE(XYZ) ima efekat kao sledeće primitivne operacije: POP, PUSH(X), PUSH(Y), PUSH(Z). Dakle, prvo skida vršni simbol sa steka, a zatim na stek stavlja simbole zadate u argumentu redom kojim su napisani.

Rešenje

Akcija REPLACE(AA) u ulazu (A, 0) kontrolne tabele po svom efektu jednaka je akciji PUSH(A). Prefiksom neke sekvence nazivamo podsekvencu kojom počinje posmatrana sekvenca; dužina prefiksa može biti od nula znakova do broja znakova u sekvenci. Pravi prefiksi su svi oni prefiksi čija je dužina strogo manja od dužine celokupne sekvence.

Automat počinje rad u stanju A. Svaka 0 na ulazu uvećava broj simbola A na steku za jedan, a svaka jedinica na ulazu smanjuje ovaj broj za jedan. Sekvenca se odbija ako se stek isprazni pre kraja ulazne sekvence, ili ako na kraju ulazne sekvence stek nije prazan.

Prema tome, automat prihvata sve i samo one sekvence kod kojih je broj znakova 0 u svakom njihovom pravom prefiksu veći ili jednak broju znakova 1, a u kompletnoj sekvenci znakova 1 ima tačno za jedan više od znakova 0.

Primer rada parsera za ulaznu sekvencu 011:

stek	ulaz	akcija
∇A	011⊣	REPLACE(AA), ADVANCE
∇AA	11⊣	POP, ADVANCE
∇A	1⊣	POP, ADVANCE
∇	⊣	ACCEPT

Primer rada parsera za ulaznu sekvencu 01011:

stek	ulaz	akcija
∇A	01011⊣	REPLACE(AA), ADVANCE
∇AA	1011⊣	POP, ADVANCE
∇A	011⊣	REPLACE(AA), ADVANCE
∇AA	11⊣	POP, ADVANCE
∇A	1⊣	POP, ADVANCE
∇	⊣	ACCEPT

Primer rada parsera za ulaznu sekvencu 01000101111:

stek	ulaz	akcija
∇A	01000101111⊣	REPLACE(AA), ADVANCE
∇AA	1000101111⊣	POP, ADVANCE
∇A	000101111⊣	REPLACE(AA), ADVANCE
∇AA	00101111⊣	REPLACE(AA), ADVANCE
∇AAA	0101111⊣	REPLACE(AA), ADVANCE
∇AAAA	101111⊣	POP, ADVANCE
∇AAA	01111⊣	REPLACE(AA), ADVANCE
∇AAAA	1111⊣	POP, ADVANCE
∇AAA	111⊣	POP, ADVANCE
∇AA	11⊣	POP, ADVANCE
∇A	1⊣	POP, ADVANCE
∇	⊣	ACCEPT

Zadatak 3.1.3

Kakve izmene treba načiniti u kontrolnoj tabeli potisnog automata sa Sl. 3.1.4 da bi ovaj automat proveravao korektnu uparenost otvorenih i zatvorenih zagrada? Startni stek je ∇.

	(	)	⊣
A	PUSH(A) ADVANCE	POP ADVANCE	POP RETAIN
∇	PUSH(A) ADVANCE	REJECT	ACCEPT

Sl. 3.1.4

Rešenje

Razmotrimo primer procesiranja ulazne sekvence ((( ) ⊣.

	stek:	ulaz:	akcija:
1.	∇	((( )⊣	PUSH(A), ADVANCE
2.	∇ A	(( )⊣	PUSH(A), ADVANCE
3.	∇ AA	( )⊣	PUSH(A), ADVANCE
4.	∇ AAA	)⊣	POP, ADVANCE
5.	∇ AA	⊣	POP, RETAIN
6.	∇ A	⊣	POP, RETAIN
7.	∇	⊣	ACCEPT

Za svaki ulazni znak (na stek ide simbol A, dok se za svaki ulazni znak) sa steka skida jedan simbol. Ukoliko se celokupna ulazna sekvenca obradi, a stek nije prazan, dolazi do pražnjenja steka i akcije ACCEPT. Nije teško uočiti da automat prihvata sekvencu ako je u toj sekvenci, kao i u svakom njenom prefiksu broj otvorenih zagrada veći do jednak broju zatvorenih.

Da bi automat proveravao uparenost zagrada, u svakom prefiksu sekvence broj otvorenih zagrada mora biti veći do jednak broju zatvorenih zagrada, a u celokupnoj sekvenci mora biti jednak broj jednih i drugih. Izmena automata svodi se na upisivanje akcije REJECT u vrstu A i kolonu ⊣ kontrolne tabele.

Zadatak 3.1.4

Projektovati primitivan potisni automat koji prepoznaje sledeće skupove sekvenci.

a) $\{1^n\ 0^m\}$ $n > m > 0$

b) $\{1^n\ 0^n\} \cup \{0^m\ 1^{2m}\}$ $m, n > 0$.

Iz zadatih skupova izabrati po jednu sekvencu dužine veće od tri i prikazati proces njenog prepoznavanja.

Rešenje

a)

Ideja konstrukcije automata je sledeća: mada se svaki potisni automat može napraviti sa jednim stanjem, primera radi ćemo ovaj realizovati sa dva stanja. U početnom stanju S_1 automat će obrađivati vodeće jedinice na ulazu tako što će za svaku jedinicu na stek, koji je inicijalno prazan, biti stavljen simbol A. U trenutku kada se na ulazu pojavi nula, automat prelazi u stanje S_2 koje označava da se u nastavku prihvataju isključivo nule. Pri obradi nule, sa steka se skida jedan simbol A. Sekvenca se prihvata jedino ako se stek ne isprazni do pojave markera kraja ulaza, jer to označava da je broj jedinica na ulazu bio veći od broja nula. Konstrukciju sprovodimo po koracima:

1. Uočavamo ulazne simbole: {0, 1}
2. Uočavamo simbole steka: {∇, A}
3. Uočavamo simbole stanja: $\{S_1, S_2\}$
4. Definišemo početnu konfiguraciju steka ∇.
5. Pravimo kontrolne tabele (Sl. 3.1.5).

Startni stek: ∇ S_1

	0	1	⊣
A	POP ADVANCE STATE (S_2)	PUSH(A) ADVANCE STATE(S_1)	REJECT
∇	REJECT	PUSH(A) ADVANCE STATE(S_1)	REJECT

S_2

	0	1	⊣
A	POP ADVANCE STATE (S_2)	REJECT	ACCEPT
∇	REJECT	REJECT	REJECT

Sl. 3.1.5

b)

Za sekvence oblika $\{1^n 0^n\}$, $n > 0$ postupićemo analogno prethodnom slučaju.

1. Uočavamo ulazne simbole: {0, 1}
2. Uočavamo simbole steka: {∇, O, Z}
3. Uočavamo simbole stanja: $\{S_1, S_2, S_3, S_4\}$
4. Početna konfiguracija steka je ∇.
5. Pravimo kontrolne tabele na osnovu razmatranja koje sledi.

U stanju S_1 automat obrađuje vodeće jedinice, stavljajući na stek simbol O za svaku jedinicu na ulazu. Po dolasku prve nule, prelazi se u stanje S_2 gde se za svaku nulu sa ulaza sa steka skida jedno O. Sekvenca se prihvata ako stek ostane prazan kada se ceo ulaz obradi.

Za sekvence oblika $\{0^m 1^{2m}\}$ m, $n > 0$, na prvi pogled, obrada bi se mogla rešiti na isti način, s tim što bi se za svaku nulu na ulazu na stek potiskivala dva simbola da bi ih kasnije uparili sa dve jedinice sa ulaza. Međutim, istovremeno potiskivanje dva simbola na stek ne spada u operacije primitivnog potisnog automata. Zbog toga će biti primenjeno sledeće rešenje: automat obrađuje vodeće nule u stanju S_1 potiskujući za svaku nulu na ulazu jedno Z na stek (uvodimo novi stek simbol da bismo u nastavku ulaza razlikovali ovaj tip sekvence). Po dolasku prve jedinice, automat prelazi u stanje S_3 gde se za svaku 1 sa ulaza skida jedno Z sa steka i potom prelazi u stanje S_4, gde se obrađuje sledeća jedinica za ulaza, bez menjanja sadržaja steka i vrši povratak u stanje S_3. Na taj način obezbeđeno je uparivanje dve jedinice sa ulaza sa jednim Z sa steka.

Na osnovu prethodnog razmatranja dobija se potisni automat sa Sl. 3.1.6 (a) i (b).

Startni stek: ∇

S_1:

	0	1	⊣
O	POP ADVANCE STATE(S_2)	PUSH(O) ADVANCE STATE (S_1)	REJECT
Z	PUSH(Z) ADVANCE STATE (S_1)	ADVANCE STATE(S_3)	REJECT
∇	PUSH(Z) ADVANCE STATE (S_1)	PUSH(O) ADVANCE STATE (S_1)	REJECT

S_2:

	0	1	⊣
O	POP ADVANCE STATE(S_2)	REJECT	REJECT
Z	REJECT	REJECT	REJECT
∇	REJECT	REJECT	ACCEPT

Sl. 3.1.6 (a)

S_3:

	0	1	⊣
O	REJECT	REJECT	REJECT
Z	REJECT	POP ADVANCE STATE (S_4)	REJECT
∇	REJECT	REJECT	REJECT

S_4:

	0	1	⊣
O	REJECT	REJECT	REJECT
Z	REJECT	ADVANCE STATE (S_3)	REJECT
∇	REJECT	REJECT	ACCEPT

Sl. 3.1.6 (b)

Diskusija

Primetimo da su pojedini ulazi kontrolne tabele automata suvišni jer se njima ne može pristupiti tokom rada automata. Ovo se odnosi na vrstu 0 u stanju S_2 i vrstu 1 u stanjima S_3 i S_4. Čitaocu se ostavlja da, za vežbu, konstruiše potisni automat za isti jezik sa jednim stanjem i potrebnim brojem simbola steka.

Zadatak 3.1.5

Rešiti tačku b) prethodnog zadatka koristeći operaciju REPLACE. Pokušati sa automatom koji ima samo jedno stanje.

Analiza problema

Operacija REPLACE (vidi Zadatak 3.1.2) proširuje skup primitivnih operacija nad stekom. Ta operacija omogućava skidanje jednog elementa sa steka, a umesto njega smeštanje na stek proizvoljne sekvence simbola steka.

Rešenje

Automat (Sl. 3.1.7) treba da prepoznaje sekvence tipa $\{1^n 0^n\}$ i $\{0^m 1^{2m}\}$ m, n > 0. Startni stek je ∇A. Simbol A posle prvog ulaznog simbola skidamo sa steka i više ne koristimo, ali povrh dva simbola Z, za sekvence tipa $\{0^m 1^{2m}\}$ odnosno O, za sekvence tipa $\{1^n 0^n\}$, na vrh steka stavljamo kontrolni simbol B, odnosno C zavisno da li je prvi ulazni simbol bio 0 ili 1 i zadržavaju se na vrhu steka sve dok ne naiđe drugačiji ulazni simbol, kada se kontrolni simboli skidaju sa steka, a ostatak ulaza uparuje po broju sa simbolima na steku.

Diskusija

U konkretnom slučaju, za oba tipa prihvatljivih ulaznih sekvenci (1^n0^n ili 0^m1^{2m}), završetak je isti – prazan stek na kraju ulazne sekvence. Da se obezbedi drugačiji završetak obrade za različite tipove prihvatljivih ulaznih sekvenci, bilo bi neophodno da se uvedu novi simboli steka, na primer D i E koji bi se na kraju ulaza pojavili na vrhu steka, pa bi procesiranje moglo

da se razlikuje. Tada bi u ulazu (A,0) umesto REPLACE(ZZB) stajalo REPLACE(DZZB), a za (C,1) umesto REPLACE(OC) stajalo bi REPLACE(EOC).

	0	1	⊣
A	REPLACE (ZZB) ADVANCE	REPLACE (OC) ADVANCE	REJECT
C	POP RETAIN	REPLACE (OC) ADVANCE	REJECT
O	POP ADVANCE	REJECT	REJECT
∇	REJECT	REJECT	ACCEPT
B	REPLACE (ZZB) ADVANCE	POP RETAIN	REJECT
Z	REJECT	POP ADVANCE	REJECT

Sl. 3.1.7

Zadatak 3.1.6

Projektovati potisni automat koji obavlja sledeće prevođenje:

$$1^n\,0^m$$

prevodi u

$$1^n\,0^{2n}\ n > 0,\ m > 0.$$

Analiza problema

Traženi potisni automat (Sl. 3.1.8) spada u klasu procesora jer pored prepoznavanja ulaza vrši prevođenje generisanjem odgovarajućeg izlaznog niza. Radi generisanja izlaza skup operacija automata proširujemo operacijom OUT(XYZ) koja označava da se na izlaz šalje sekvenca izlaznih simbola XYZ.

Rešenje

Startno stanje steka je ∇C. Simbol C uklanja se sa steka čim naiđe prva jedinica na ulazu. Za svaku jedinicu na ulazu stavljamo po jedno O na stek (na vrhu se uvek zadržava simbol A koji označava da još nije stigla prva nula na ulazu) i na izlaz šaljemo po jednu jedinicu. Kada se pojavi prva 0 na ulazu, sa steka se skida simbol A. Nule se dalje čitaju ne menjajući stek. Kada se stigne do kraja ulazne sekvence, sa steka se redom skidaju simboli O i za svaki od njih izdaju dve nule na izlaz. Primetiti da će i neispravne ulazne sekvence generisati neki izlaz. Dva REJECT ulaza u vrsti ∇ nisu dostižni ni za jednu ulaznu sekvencu pri radu automata.

	0	1	⊣
C	REJECT	OUT (1) REPLACE (OA) ADVANCE	REJECT
A	POP ADVANCE	OUT (1) REPLACE (OA) ADVANCE	REJECT
O	ADVANCE	REJECT	OUT (00) POP RETAIN
∇	REJECT	REJECT	ACCEPT

Sl. 3.1.8

Zadatak 3.1.7

Pokazati da potisni automati sa Sl. 3.1.9 i Sl. 3.1.10 nikada ne ulaze u beskonačne cikluse, odnosno da se procesiranje proizvoljne ulazne sekvence završava u konačnom broju koraka. Početna konfiguracija steka oba automata je ∇.

a)

	0	1	2	3	⊣
A	PUSH(A) ADVANCE	PUSH(A) ADVANCE	REJECT	POP ADVANCE	ACCEPT
B	PUSH(B) ADVANCE	ACCEPT	POP ADVANCE	ACCEPT	REJECT
∇	PUSH(A) ADVANCE	PUSH(B) ADVANCE	REJECT	PUSH(A) ADVANCE	ACCEPT

Sl. 3.1.9

b)

	a	b	c	⊣
C	PUSH(C) ADVANCE	ACCEPT	PUSH(D) ADVANCE	ACCEPT
D	POP ADVANCE	PUSH(C) RETAIN	POP RETAIN	POP RETAIN
∇	PUSH(C) RETAIN	REJECT	PUSH(D) ADVANCE	ACCEPT

Sl. 3.1.10

Rešenje

a)

Svaki ulaz tabele, izuzev onih sa akcijama ACCEPT i REJECT, sadrži akciju ADVANCE. To znači da se rad automata završava u maksimalno onoliko koraka koliko u ulaznom nizu ima znakova.

b)

Ulazi kontrolne tabele koji sadrže akciju RETAIN potencijalno mogu izazvati beskonačni ciklus, pa moramo razmotriti rad automata za svaku takvu situaciju ponaosob.

Razmotrimo ulaz u vrsti D i koloni b – skraćena oznaka (D, b). Po stavljanju simbola C na vrh steka, s obzirom da se ulaz ne menja, biće konsultovan ulaz (C, b) i izvršena akcija ACCEPT.

Posle akcija (∇, a) u sledećem koraku sigurno ide akcija (C, a) gde postoji ADVANCE.

Posle akcija (D, c) sa steka se skida simbol D, što znači da će posle konačnog broja primena ove akcije biti konsultovan jedan od ulaza (C, c) ili (∇, c) koji oba sadrže akciju ADVANCE.

Slično rezonovanje važi i za (D, ⊣). Stek se prazni, a ulaz troši i u jednom trenutku će biti konsultovan jedan od ulaza (C, ⊣) ili (∇, ⊣), odnosno preduzeta akcija ACCEPT.

Na ovaj način smo utvrdili da će automat u konačnom broju koraka potrošiti svaki ulazni simbol, što znači da će u konačnom broju koraka okončati rad.

Zadatak 3.1.8

Pronaći dve ulazne sekvence, jednu koja počinje sa 0 i drugu koja počinje sa 1 za koje automat sa Sl. 3.1.11 upada u beskonačan ciklus. Startna konfiguracija steka je ∇.

	0	1	2	3	⊣
A	ACCEPT	PUSH(A) RETAIN	PUSH(B) ADVANCE	PUSH(A) ADVANCE	REJECT
B	POP ADVANCE	ACCEPT	REJECT	PUSH(C) RETAIN	POP RETAIN
C	PUSH(E) RETAIN	PUSH(A) ADVANCE	PUSH(D) RETAIN	POP RETAIN	REJECT
D	ACCEPT	PUSH(D) ADVANCE	PUSH(E) ADVANCE	ACCEPT	ACCEPT
E	PUSH(C) RETAIN	REJECT	ACCEPT	PUSH(E) ADVANCE	PUSH(D) RETAIN
∇	PUSH(A) ADVANCE	PUSH(D) ADVANCE	PUSH(A) ADVANCE	PUSH(B) ADVANCE	ACCEPT

Sl. 3.1.11

Rešenje

Uočimo najpre ulaz (A, 1) u kontrolnoj tabeli. Posle akcija (PUSH(A), RETAIN), automat ostaje u istom stanju sa nepromenjenim ulazom, pa će opet ponavljati iste akcije uz stalno punjenje steka. Da bismo odredili ulaznu sekvencu koja dovodi do ovoga razmatramo ulaze tabele koji imaju akciju PUSH(A), da bi se stanje A našlo na vrhu inicijalno praznog steka. Tražene ulazne sekvence su, na primer, sve koje počinju sa 01 ili sa 21. Rad automata za ulaznu sekvencu 01 je:

stek	ulaz	akcija
∇	01⊣	PUSH(A), ADVANCE
∇A	1⊣	PUSH(A), RETAIN
∇AA	1⊣	PUSH(A), RETAIN
∇AAA	1⊣	PUSH(A), RETAIN
...	...	...

Cikliranje takođe može da nastane naizmeničnim prebacivanjem kontrole iz (C, 0) u (E, 0) i obrnuto. Ulazna sekvenca koja dovodi do ovoga počinje sa 120:

stek	ulaz	akcija
∇	120⊣	PUSH(D), ADVANCE
∇D	20⊣	PUSH(E), ADVANCE
∇DE	0⊣	PUSH(C), RETAIN
∇DEC	0⊣	PUSH(E), RETAIN
∇DECE	0⊣	PUSH(C), RETAIN
∇DECEC	0⊣	PUSH(E), RETAIN
...	...	...

Prebacivanje kontrole iz (B, 3) u (C, 3) i obrnuto, do čega dovode ulazne sekvence koje počinju sa 33, isto tako dovodi do cikliranja:

stek	ulaz	akcija
∇	33⊣	PUSH(B), ADVANCE
∇D	3⊣	PUSH(C), RETAIN
∇DC	3⊣	POP, RETAIN
∇D	3⊣	PUSH(C), RETAIN
∇DC	3⊣	POP, RETAIN
∇D	3⊣	PUSH(C), RETAIN
...	...	...

3.2. Analiza od vrha ka dnu

Zadatak 3.2.1

Jedan prepoznavač od vrha ka dnu, na osnovu sledeće gramatike, prepoznaje smene za jednu ulaznu sekvencu po sledećem redosledu: 1, 4, 3, 2, 4, 2, 1, 4, 2, 4, 4.

O kojoj ulaznoj sekvenci je reč?

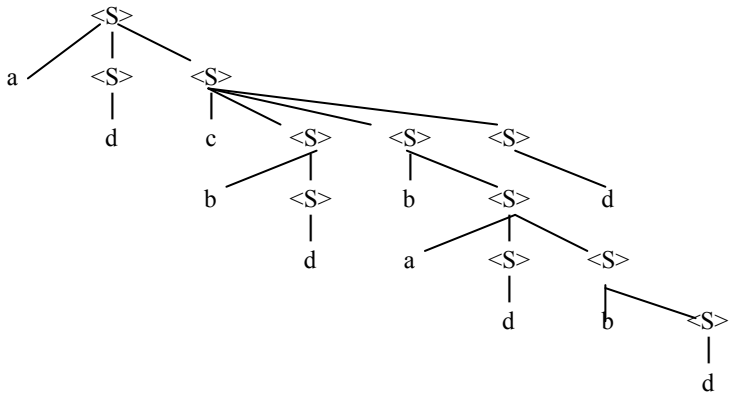
1. <S> → a <S><S>
2. <S> → b <S>
3. <S> → c <S><S><S>
4. <S> → d

Rešenje

Prepoznavač od vrha ka dnu prepoznaje smene u redosledu koji odgovara levom izvođenju tražene ulazne sekvence počev od startnog neterminala.

<S> ⇒ lm a<S><S> ⇒ lm ad<S> ⇒ lm adc<S><S><S> ⇒ lm adcb<S><S><S> ⇒ lm
↑ ↑ ↑ ↑ ↑
1 4 3 2 4
⇒ lm adcbd<S><S> ⇒ lm adcbdb<S><S> ⇒ lm adcbdba<S><S><S> ⇒ lm
↑ ↑ ↑
2 1 4
⇒ lm adcbdbad<S><S> ⇒ lm adcbdbadb<S><S> ⇒ lm
↑ ↑
2 4
⇒ lm adcbdbadbd<S> ⇒ lm adcbdbadbdd
↑
4

Traženi ulazni niz je adcbdbadbdd. Odgovarajuće stablo izvođenja prikazano je na Sl. 3.2.1.



Sl. 3.2.1

Zadatak 3.2.2

Sledeći scenario odgovara potisnom prepoznavaču od vrha ka dnu za jednu S-gramatiku. Nacrtati stablo izvođenja za zadatu ulaznu sekvencu tog scenarija.

stek	ulazni niz
1. ∇ <S>	a b b a b a b ⊣
2. ∇ b <D>	d b a b a b ⊣
3. ∇ b <D> a <S>	b a b a b ⊣
4. ∇ b <D> a	a b a b ⊣
5. ∇ b <D>	b a b ⊣
6. ∇ b a	a b ⊣
7. ∇ b	b ⊣
8. ∇	⊣

Analiza problema

Potrebno je na osnovu datog scenarija odrediti gramatičke smene upotrebljene u izvođenju:

$$<S> \Rightarrow_{lm} \text{abbabab}.$$

Ulazni simboli potisnog prepoznavača su terminalni simboli tražene gramatike, a simboli steka predstavljaju sve gramatičke simbole. Potisni prepoznavač od vrha ka dnu funkcioniše po sledećim principima:

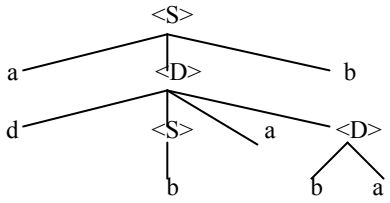
- U svakom trenutku rada, u slučaju prihvatljive ulazne sekvence, iz sentencijalne forme koja je opisana sadržajem steka (simboli se posmatraju počev od vrha steka ka dnu) moguće je izvesti preostali ulazni niz koristeći gramatiku na kojoj je prepoznavač zasnovan. U skladu sa tim, inicijalno je na steku samo startni neterminal koji treba da upari kompletnu ulaznu sekvencu. U završnom koraku stek je prazan a na ulazu je preostao samo marker kraja ulaza ⊣.
- Ukoliko je na vrhu steka terminalni simbol x, potrebno je da preostali ulaz počinje terminalom x. Akcije parsera svode se na skidanje simbola x sa steka i uklanjanje simbola x sa ulaza, dakle POP, ADVANCE.
- Ukoliko je na vrhu steka neterminalni simbol <X>, a tekući ulazni simbol je y, u slučaju prihvatljive ulazne sekvence neophodno je da postoji jedinstvena gramatička smena $<X> \rightarrow y\,\alpha$ (za definiciju S-gramatika, videti Zadatak 3.2.3). Parser preduzima akcije kojima se na vrhu steka zamenjuje leva strana smene $<X> \rightarrow y\,\alpha$ desnom, odnosno REPLACE(α^r), ADVANCE. U argumentu REPLACE akcije simboli sekvence α navode se obrnutim redosledom. Primetiti da je istovremeno izvršeno uparivanje početnog simbola desne strane y sa tekućim ulaznim simbolom, tako da se taj simbol ne smešta na stek, a istovremeno se uklanja sa ulaza. Pri ovome kažemo da je parser prepoznao (primenio) smenu $<X> \rightarrow y\,\alpha$.

Rešenje

Posmatrajući korake 1, 2, 3 i 5 scenarija uočavamo sve primenjene smene:

1. <S> → a <D> b
2. <D> → d <S> a <D>
3. <S> → b
4. <D> → b a

Stablo izvođenja za posmatranu ulaznu sekvencu, na osnovu date gramatike prikazano je na Sl. 3.2.2. Potisni automati od vrha ka dnu prepoznaju smene u redosledu koji odgovara konstrukciji stabla od korena ka listovima.



Sl. 3.2.2

Zadatak 3.2.3

Pronaći S-gramatiku i odgovarajući prepoznavač od vrha ka dnu za sledeće jezike:

a) $\{1^n\ m\ 0^n\}\ n \geq 0$

b) $\{1^n\ m\ 0^n\} \cup \{m\ 1^n\ m\ 0^{2n}\}\ n > 0$

Rešenje

a)

Najkraća sekvenca u skupu $\{1^n\ m\ 0^n\}$, $n \geq 0$, je m. Duže sekvence dobijaju se dodavanjem 1 na početku sekvence i nule na kraju. Kada ova zapažanja pretočimo u gramatička pravila dobija se:

1. <S> → 1<S>0
2. <S> → m

Dobijena gramatika ispunjava uslove za S-gramatiku:
1. Svaka desna strana smene mora da započne terminalnim simbolom.
2. Za smene sa istom levom stranom, desne strane moraju da započnu različitim terminalnim simbolima.

Ulazni simboli traženog automata su terminali 0, 1 i m. Simboli steka su dno steka ∇, svi neterminali, u konkretnom slučaju <S>, a od terminalnih simbola samo 0. Terminali 1 i m ne

idu na stek jer se sva njihova dešavanja u gramatici nalaze isključivo na počecima desnih strana smena. Startno stanje steka je: ∇<S>. Kontrolna tabela automata (Sl. 3.2.3) popunjava se po sledećim pravilima, proizašlim na osnovu razmatranja iz prethodnog zadatka:

- Ulaz tabele u vrsti ∇ i koloni ⊣ popunjava se sa ACCEPT, ostali ulazi te vrste sa REJECT.
- Ulaz tabele u vrsti označenoj nekim terminalom x i koloni x popunjava se sa POP, ADVANCE, a ostali ulazi te vrste sa REJECT.
- Ulaz tabele u vrsti označenoj nekim neterminalom <X> i koloni označenoj nekim terminalom y popunjava se sa REPLACE(α^r), ADVANCE ako postoji smena <X> → y α. Ako takva smena ne postoji ulaz se popunjava sa REJECT.
- Svi ulazi u koloni ⊣ osim onog u vrsti ∇ popunjavaju se sa REJECT.

	0	1	m	⊣
<S>	REJECT	#1	#2	REJECT
0	POP ADVANCE	REJECT	REJECT	REJECT
∇	REJECT	REJECT	REJECT	ACCEPT

Akcija #1: REPLACE (0<S>) Na stek prvo ide 0 pa <S>.
ADVANCE
Akcija #2: POP #1 i #2 se odnosi na redni broj smene koju procesiramo
ADVANCE

Sl. 3.2.3

b)
Zadati skup sekvenci {1^n m 0^n} U {m1^n m 0^{2n}}, n>0, moguće je opisati sledećom gramatikom. Smene 1, 2 i 3 opisuju prvi podskup, a preostale smene drugi. Radi se o S-gramatici.

1. <S> → 1<S_1>0
2. <S_1> → m
3. <S_1> → 1<S_1>0
4. <S> → m1<S_2>00
5. <S_2> → 1<S_2>00
6. <S_2> → m

Potisni automat za ovu gramatiku, konstruisan prema pravilima iz tačke a) rešenja, prikazan je na slici Sl. 3.2.4.

	1	0	m	⊣
<S>	#1	REJECT	#4	REJECT
<S1>	#3	REJECT	#2	REJECT
<S2>	#5	REJECT	#6	REJECT
0	REJECT	POP ADVANCE	REJECT	REJECT
1	POP ADVANCE	REJECT	REJECT	REJECT
∇	REJECT	REJECT	REJECT	ACCEPT

Startno stanje steka:∇ <S>

Akcije #1 i #3:	REPLACE (0,<S1>) ADVANCE	Akcija #4:	REPLACE (00 <S2>1) ADVANCE
Akcije #2 i #6:	POP ADVANCE	Akcija #5:	REPLACE (00 <S2>) ADVANCE

Sl. 3.2.4

Diskusija

S-gramatike predstavljaju najužu klasu gramatika od vrha ka dnu. Za ove gramatike najjednostavnije je konstruisati prepoznavač. Širu klasa gramatika, koja obuhvata S-gramatike, predstavljaju q-gramatike. Ova klasa obuhvaćena je LL(1) gramatikama, najširom klasom bezkontekstnih gramatika koje je moguće deterministički (bez pogrešnih pretpostavki o primenjenim smenama) prepoznavati od vrha ka dnu sa jednim predikcionim simbolom (to jest, gledajući po jedan ulazni simbol u svakom koraku rada prepoznavača – parsera).

Zadatak 3.2.4

Za svaki od sledećih jezika pronaći q gramatiku i odgovarajući potisni prepoznavač od vrha ka dnu.

a) $\{1^n\}$ $n > 0$

b) $\{1^n\ 0^n\}$ $n \geq 0$

Analiza problema

a) Jedna od mogućih gramatika za opis skupa $\{1^n\}$, $n > 0$ je:

1. $<S> \rightarrow 1<S_1>$
2. $<S_1> \rightarrow 1<S_1>$
3. $<S_1> \rightarrow \varepsilon$

Pojava prazne smene u gramatici dozvoljena je u klasi q-gramatika. Pre nego što definišemo q-gramatike i ispitamo da li navedena gramatika pripada ovoj klasi, razmotrimo levo izvođenje sekvence 11:

$$\underset{\uparrow}{<S>} \Rightarrow_{lm} 1\underset{\uparrow}{<S_1>} \Rightarrow_{lm} 11\underset{\uparrow}{<S_1>} \Rightarrow_{lm} 11$$
$$1 \qquad 2 \qquad 3$$

kome odgovara sledeći scenario rada traženog prepoznavača:

stek	ulazni niz	akcija
1. $\nabla<S>$	$11\dashv$	REPLACE($<S_1>$), ADVANCE
2. $\nabla<S_1>$	$1\dashv$	REPLACE($<S_1>$), ADVANCE
3. $\nabla<S_1>$	$\dashv$	POP, RETAIN
4. ∇	$\dashv$	ACCEPT

Vidimo da prepoznavanju prazne smene u 3. koraku rada prepoznavača odgovaraju akcije POP, RETAIN, to jest, sa steka se skida leva strana smene a na stek se potom ne stavlja ništa, s obzirom da je desna strana smene prazna sekvenca, a ulaz takođe ostaje nepromenjen.

Prazna smena $<X> \rightarrow \varepsilon$ prepoznaje se u trenutku kada je na vrhu steka simbol $<X>$ koji odgovara levoj strani smene. Da bismo odredili koji simbol može u tom trenutku predstavljati tekući ulaz, podsetimo se sledećeg principa rada prepoznavača od vrha ka dnu (vidi Zadatak 3.2.2):

U svakom trenutku rada, u slučaju prihvatljive ulazne sekvence, iz sentencijalne forme koja je opisana sadržajem steka (simboli se posmatraju počev od vrha steka ka dnu) moguće je izvesti preostali ulazni niz koristeći gramatiku na kojoj je prepoznavač zasnovan.

Po ovom principu, u trenutku rada kada je stanje obrade sledeće:

stek	ulaz
$\nabla\beta^R<X>$	$v\dashv$

odnosno kada je $<X>$ na vrhu steka, niz simbola β^R ispod njega (sa R je označena operacija obrtanja sekvence, što znači da se poslednji simbol u nizu β nalazi neposredno iznad markera dna steka), a v preostali ulaz sekvence uv, važi da je:

$$<S> \overset{*}{\Rightarrow}_{lm} \alpha<X>\beta \overset{*}{\Rightarrow}_{lm} uv$$

pri čemu je α već upareno sa u, a $<X>\beta$ je potrebno upariti sa v:

$$\alpha \overset{*}{\Rightarrow}_{lm} u$$

$$<X>\beta \overset{*}{\Rightarrow}_{lm} v$$

Iz ovoga sledi da se po prepoznavanju prazne smene za $<X>$ nepromenjeni preostali ulaz v uparuje sa ostatkom steka β posle skidanja simbola leve strane smene:

$$\beta \overset{*}{\Rightarrow}_{lm} v$$

odnosno da tekući ulazni simbol u trenutku prepoznavanja prazne smene može biti svaki ulazni simbol kojim počinje sekvenca β, to jest, koji se pojavljuje neposredno iza <X> u bilo kojoj sentencijalnoj formi izvedenoj iz <S>⊣ (marker kraja je dodat jer se svaki ulazni niz njime završava). Ovaj skup naziva se FOLLOW skup posmatranog neterminala <X>, formalno:

$$\text{FOLLOW}(<X>) = \{t \mid t \in V_t \cup \{\dashv\} \ \wedge \ <S>\dashv \overset{*}{\Rightarrow} \alpha<X>t\,\gamma\}$$

Skup ulaznih simbola koji mogu predstavljati tekući ulaz u trenutku prepoznavanja i-te smene od strane potisnog automata, naziva se selekcioni skup smene, u oznaci SELECT(i). Za klasu q-gramatika od interesa su sledeći slučajevi:

- ako desna strana i-te smene počinje terminalom t, važi da je
  SELECT(i) = {t}.
- ako je i-ta smena oblika <X>→ ε, važi da je
  SELECT(i) = FOLLOW(<X>).

Klasa q-gramatika obuhvata one i samo one bezkontekstne gramatike kod kojih:

1) desna strana svake smene počinje terminalom ili je prazna i
2) selekcioni skupovi smena sa istom levom stranom su međusobno disjunktni (njihov presek je prazan skup).

Drugi uslov obezbeđuje da parser u svakom koraku rada jednoznačno može da prepozna smenu za dati neterminal na vrhu steka.

Rešenje

a)

Razmatrana gramatika zadovoljava 1. preduslov za q-gramatike. Selekcioni skupovi razmatrane gramatike su:

$$\text{SELECT}(1) = \{1\}$$

$$\text{SELECT}(2) = \{1\}$$

$$\text{SELECT}(3) = \text{FOLLOW}(<S_1>) = \{\dashv\}$$

pri čemu je skup FOLLOW($<S_1>$) određen na osnovu zapažanja da se neterminal $<S_1>$ može pojaviti isključivo kao krajnje desni simbol bilo koje sentencijalne forme izvedene iz startnog neterminala <S>. Za računanje FOLLOW skupova u opštem slučaju videti Zadatak 3.2.5.

S obzirom da su selekcioni skupovi za drugu i treću smenu disjunktni, data gramatika jeste q-gramatika.

Sada možemo da konstruišemo potisni prepoznavač po pravilima iz prethodnog zadatka, uz dodatno pravilo:

- Ulaz tabele u vrsti označenoj nekim neterminalom <X> i koloni označenoj nekim ulaznim simbolom y popunjava se sa POP, RETAIN ako postoji smena <X> → ε i y pripada selekcionom skupu te smene.

U konkretnom slučaju, skup ulaznih simbola je {1}, skup simbola steka: {<S>, ∇, <S_1>}, startni stek je ∇ <S>. Kontrolna tabela parsera prikazana je na Sl. 3.2.5.

	1	⊣
<S>	#1	REJECT
<S_1>	#2	#3
∇	REJECT	ACCEPT

#1: REPLACE (<S>), ADVANCE
#2: REPLACE (<S_1>), ADVANCE
#3: POP, RETAIN

Sl. 3.2.5

b)

Za dati skup sekvenci {1^n0^n }, n ≥ 0 moguće je napisati sledeću gramatiku:

1. <S> → 1 <S> 0
2. <S> → ε

Da li je ovo q-gramatika? Prvi uslov (desne strane počinju terminalom ili su prazne smene) je ispunjen, a za drugi uslov moramo naći selekcione skupove za prvu i drugu smenu, pa počinjemo tražeći skup FOLLOW(<S>). Iza <S> može da dođe 0 na osnovu smene 1 i ⊣ na osnovu definicije FOLLOW skupa:

FOLLOW (<S>) = {0, ⊣ },

pa su

SELECT (1) = {1}

SELECT (2) = FOLLOW (<S>) = {0, ⊣ }

Pošto su selekcioni skupovi disjunktni ovo je q-gramatika i možemo napraviti potisni prepoznavač (Sl. 3.2.6). Ulazni simboli su {1,0}, simboli steka {<S>,0,∇}, a početni stek: ∇<S>

	1	0	⊣
<S>	REPLACE (0<S>) ADVANCE	POP RETAIN	POP RETAIN
0	REJECT	POP ADVANCE	REJECT
∇	REJECT	REJECT	ACCEPT

Sl. 3.2.6

Zadatak 3.2.5

Za sledeću gramatiku sa startnim simbolom <A>:

1. <A> → a <B> <C>
2. <A> → b <B>
5. <B> → ε
6. <C> → b <C>

3. <A> → ε 7. <C> → c

4. <B> → a <B> b

a) Odrediti FOLLOW skupove za svaki neterminal.

b) Odrediti SELECT skupove za svaku smenu.

c) Ispitati da li se radi o q-gramatici.

d) Konstruisati potisni automat koji prepoznaje ovu gramatiku od vrha ka dnu.

Analiza problema

U ovom zadatku ilustrovaćemo iterativni metod izračunavanja FOLLOW skupova. Za to je neophodno odrediti poništive neterminale u gramatici i FIRST skupove neterminala.

Neterminal <X> je poništiv ako i samo ako se iz njega može izvesti prazna sekvenca:

$$<X> \stackrel{*}{\Rightarrow} \varepsilon$$

Sekvenca gramatičkih simbola je poništiva ako i samo ako se sastoji od nula ili više poništivih neterminala. Skup poništivih neterminala P formiramo na sledeći način:

1. U skup P staviti sve neterminale koji se pojavljuju na levoj strani neke prazne smene.
2. Razmatrati redom smene čije se desne strane sastoje isključivo od neterminala, a leva strana se ne nalazi u skupu P. Ako se svi neterminali desne strane pojavljuju u skupu P, dodati levu stranu smene u P.
3. Sve dok postoji promena u skupu P ponavljati korak 2, u suprotnom završiti postupak.

Skup FIRST(<X>) za proizvoljan neterminal <X> predstavlja skup svih terminalnih simbola kojima može početi sentencijalna forma izvedena iz <X>:

$$\text{FIRST}(<X>) = \{ t \mid t \in V_t \wedge \alpha \in V^* \wedge <X> \Rightarrow^* t\,\alpha\}$$

Za q gramatike FIRST(<X>) računamo kao uniju početnih simbola desnih strana svih smena koje na levoj strani imaju <X>, a ne radi se o praznim smenama.

Moguće je definisati FIRST skupove sentencijalnih formi: Neka je $\alpha = A_1A_2...A_iA_{i+1}...A_n$ proizvoljna sentencijalna forma u kojoj prvih i simbola predstavlja poništive neterminale. Tada skup FIRST(α) predstavlja uniju FIRST skupova svih simbola sa levog kraja sekvence zaključno sa prvim koji nije poništiv:

$$\text{FIRST}(\alpha) = \text{FIRST}(A_1) \cup \text{FIRST}(A_2) \cup ... \cup \text{FIRST}(A_i) \cup \text{FIRST}(A_{i+1})$$

Ukoliko je simbol A terminal, onda je FIRST(A) = {A}.

FOLLOW skupove računamo prema sledećim pravilima:

1. Skup FOLLOW startnog neterminala sadrži marker kraja ulaza ⊣.
2. Za određivanje FOLLOW skupa neterminala <X> posmatramo sve gramatičke smene na čijoj se desnoj strani nalazi <X>. Neka je <Y> → α <X> β jedna od tih smena. Tada:

   FOLLOW(<X>) ⊃ FIRST(β), ako β nije poništivo, ili

FOLLOW(<X>) $\supset$ FIRST(β) $\cup$ FOLLOW(<Y>), ako je β poništivo.

U opštem slučaju, na osnovu datih pravila formira se sistem skupovnih jednakosti koji se rešava iterativno:

1. Inicijalno se usvoji da su svi FOLLOW skupovi prazni.
2. Usvojene vrednosti se uvrste u jednakosti čime se dobijaju nove vrednosti FOLLOW skupova.
3. Korak 2. ponavlja se sve dok postoji promena bar u jednom skupu. Krajnje vrednosti predstavljaju konačno rešenje.

Rešenje

a)

Odredimo prvo poništive neterminale u datoj gramatici. Inicijalno,

P = { <A>, <B>}

na osnovu 3. i 5. smene. Ovo je ujedno i konačan skup, jer neterminal <C> ne poseduje smene sa poništivom desnom stranom.

Na osnovu 1. i 2. smene je:

FIRST(<A>)={a, b}

Na osnovu 4. smene je:

FIRST(<B>)={a}

Konačno, na osnovu 6. i 7. smene:

FIRST(<C>)={b, c}

Odredimo sada FOLLOW skupove neterminala:

- Startni neterminal ne pojavljuje se na desnim stranama smena, pa je na osnovu pravila 1. :

  FOLLOW(<A> = {$\dashv$ }

- Razmotrimo neterminal <B> koji se pojavljuje na desnim stranama 1. , 2. i 4. smene. Na osnovu prve smene i pravila 2. gde je α = a, β = <C>:

  FOLLOW(<B>) $\supset$ FIRST(<C>)

  Na osnovu 2. smene i istog pravila, pri čemu je α = a, β = ε, a leva strana smene je <A>:

  FOLLOW(<B>) $\supset$ FIRST(ε) $\cup$ FOLLOW(<A>), odnosno
  FOLLOW(<B>) $\supset$ FOLLOW(<A>)

  pošto je FIRST(ε) = $\varnothing$.

  Na osnovu 4. smene i pravila 2. , pri čemu je α = a, β = b:

  FOLLOW(<B>) $\supset$ {b}

  jer je FIRST(b) = {b}. S obzirom da nema više pojava neterminala <B> u drugim smenama, važi da je:

FOLLOW(<B>) = FIRST(<C>) ∪ FOLLOW(<A>) ∪ {b}

S obzirom da su svi članovi na desnoj strani poznati možemo odrediti konačnu vrednost:

FOLLOW(<B>) = {b, c, ⊣ }

- Razmotrimo sada neterminal <C> pojavljuje se u 1. i 6. smeni, pa na osnovu pravila 2. za obe smene:

  1. FOLLOW(<C>) ⊃ FIRST(ε) ∪ FOLLOW(<C>)

  2. FOLLOW(<C>) ⊃ FIRST(ε) ∪ FOLLOW(<A>)

  Izraz 1. ne doprinosi konačnom rešenju pa ga možemo ukloniti, čime se dobija:

  FOLLOW(<C>) = ∅ ∪ FOLLOW(<A>) = {⊣ }

Konačno rešenje je, prema tome:

FOLLOW(<A> = {⊣ }

FOLLOW(<B>) = {b, c, ⊣ }

FOLLOW(<C>) = {⊣ }

b)

Zadatak 3.2.4 definiše pravila računanja selekcionih skupova. Po tim pravilima:

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}

c)

Ovo jeste q gramatika jer ispunjava dva uslova:

1. da su sve smene oblika <A> → t α (t je terminal,α je proizvoljan niz), ili oblika <A> → ε;
2. da smene koje imaju identične leve strane, imaju disjunktne selekcione skupove.

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

SELECT(1) ∩ SELECT(2) = SELECT(1) ∩ SELECT(3) =

= SELECT(2) ∩ SELECT(3) = ∅

Provera za smene 4. i 5. :

SELECT(4) ∩ SELECT(5)= ∅

Provera za smene 6. i 7. :

SELECT(6) ∩ SELECT(7)= ∅

d)

Startno stanje steka je ∇<A>. Posmatrajmo desne strane smena: pošto prvi terminal ne ide na stek, simboli steka su <A>, <B>, <C>, b,∇. Kontrolna tabela traženog automata prikazana je na Sl. 3.2.7.

	a	b	c	⊣
<A>	REPLACE (<C> <B>) ADVANCE	REPLACE (<B>) ADVANCE	REJECT	POP RETAIN
<B>	REPLACE (b <B>) ADVANCE	POP RETAIN	POP RETAIN	POP RETAIN
<C>	REJECT	REPLACE (<C>)* ADVANCE	POP ADVANCE	REJECT
b	REJECT	POP ADVANCE	REJECT	REJECT
∇	REJECT	REJECT	REJECT	ACCEPT

Sl. 3.2.7

* U ulazu tabele (<C>, b) može da stoji samo ADVANCE, jer je stanje <C> već na vrhu steka, pa akcija REPLACE(<C>) nema efekta.

Zadatak 3.2.6

Sledeći potisni automat konstruisan je primenom metodologije od vrha ka dnu.

a) Pronaći gramatiku koja je ovde korišćena.

b) Koji ulazi se mogu promeniti u REJECT bez uticaja na jezik koji se prepoznaje?

	a	b	c	d	⊣
<S>	#1	#2	#2	#2	#2
<A>	#2	#3	#2	#2	#2
<B>	#4	#4	#5	#6	#6
∇	#4	#4	#4	#4	#7

Sl. 3.2.8

Akcije:

#1: REPLACE (<B><A>), ADVANCE

#2: POP, RETAIN

#3: REPLACE (<B><A><S>), ADVANCE

#4: REJECT

#5: REPLACE (<B>), ADVANCE

#6: POP, ADVANCE

#7: ACCEPT

Rešenje

a)

Poznajući način konstrukcije potisnog prepoznavača za q-gramatike (Zadatak 3.2.4) možemo utvrditi sledeće:

Akciju #1 uzrokuje smena: 1. <S> → a<A><B>

Akciju #2 u vrsti <S > uzrokuje smena: 2. <S> → ε

Akciju #2 u vrsti <A> uzrokuje smena: 3. <A> → ε

Akciju #3 uzrokuje smena:. 4. <A> → b<S><A><B>

Akciju #5 uzrokuje smena:. 5. <B> → c<B>

Akciju #6 uzrokuje smena:. 6. <B> → d

b)

Da bismo proverili akcije navedene u kontrolnoj tabeli, treba odrediti selekcione skupove za gornje smene. Poništivi neterminali su samo <S> i <A>. Imamo da je:

FIRST(<S>) = {a}

FIRST(<A>) = {b}

FIRST(<B>) = {c, d}

Sada računamo FOLLOW skupove poništivih neterminala:

FOLLOW(<S>) = {┤} ∪ FIRST(<A><B>) =
= {┤} ∪ FIRST(<A>) ∪ FIRST(<B>) =
= { b, c, d, ┤}

FOLLOW(<A>) = FIRST(<B>) = {c, d}

Na kraju dobijamo selekcione skupove smena:

SELECT (1) = {a}

SELECT (2) = FOLLOW(<S>) = { b, c, d, ┤}

SELECT (3) = FOLLOW(<A>) = {c,d}

SELECT (4) = {b}

SELECT (5) = {c}

SELECT (6) = {d}

Možemo utvrditi da gramatika jeste klase q.

Razmotrimo sada ulaze kontrolne tabele po vrstama:

- U vrsti <S> kolona a odnosi se na akcije za prepoznavanje 1. smene (akcije su ispravno navedene u postavci), a ostale kolone na prepoznavanje 2. smene kao što je i navedeno u postavci.
- U vrsti <A> prepoznaju se smene 3. (kolone c i d) i smena 4. (kolona b). Preostale kolone a i ⊣ treba da sadrže REJECT, a ne akciju #2 kako je navedeno u postavci.
- U vrsti <B> prepoznaju se smene 5. (kolona c) i smena 6. (kolona d). Prema tome, kolona ⊣ treba da sadrži REJECT a ne akciju #6.

Ispravljena kontrolna tabela prikazana je na Sl. 3.2.9..

	a	b	c	d	⊣
<S>	#1	#2	#2	#2	#2
<A>	#4	#3	#2	#2	#4
<B>	#4	#4	#5	#6	#4
∇	#4	#4	#4	#4	#7

Sl. 3.2.9

Zadatak 3.2.7

Za sledeću gramatiku sa startnim simbolom <S>, pronaći FOLLOW skupove svih neterminala; pronaći SELECT skupove svih produkcija i odgovoriti na pitanje da li se radi o LL(1) gramatici.

1. <S> → a <A> <B> b <C> <D>
2. <S> → ε
3. <A> → <A> <S> d
4. <A> → ε
5. <B> → <S> <A> c
6. <B> → e <C>
7. <B> → ε
8. <C> → <S> f
9. <C> → <C> g
10. <C> → ε
11. <D> → a <B> <D>
12. <D> → ε

Analiza problema

Bezkontekstna gramatika pripada klasi LL(1) ako i samo ako smene te gramatike sa istom levom stranom imaju disjunktne selekcione skupove.

Može se zapaziti da klasa LL(1) predstavlja nadskup klase q gramatika jer je uklonjen preduslov da neprazne smene moraju počinjati neterminalom. Iz tog razloga selekcioni skupovi za LL(1) gramatike definišu se na nešto generalniji način nego u slučaju q gramatika:

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.

U rešenju ovog zadatka biće ilustrovan relacioni metod određivanja selekcionih skupova smena, koji obuhvata sledeće aktivnosti:

1. Pronalaženje poništivih neterminala i smena.
2. Konstrukcija relacije BEGINS_DIRECTLY_WITH.
3. Konstrukcija relacije BEGINS_WITH.
4. Određivanje FIRST skupova za svaki neterminal.
5. Određivanje FIRST skupova za svaku smenu.
6. Konstrukcija relacije IS_FOLLOWED_DIRECTLY_BY.
7. Konstrukcija relacije IS_DIRECT_END_OF.
8. Konstrukcija relacije IS_END_OF.
9. Konstrukcija relacije IS_FOLLOWED_BY.
10. Proširivanje domena relacije IS_FOLLOWED_BY uključivanjem markera kraja ulaza ⊣.
11. Određivanje FOLLOW skupova za sve poništive neterminale.
12. Određivanje selekcionih skupova smena.

U nastavku će detaljnije biti opisani pojedini koraci.

Rešenje

Algoritam:

1. Pronalaženje poništivih neterminala i smena (videti Zadatak 3.2.5).

   Poništivi neterminali: <S>, <A>, <B>, <C>, <D>

   Poništive smene: 2, 4, 7, 10, 12

2. Konstrukcija relacije BEGINS_DIRECTLY_WITH.

   Za proizvoljne gramatičke simbole A i B važi:

   A BEGINS_DIRECTLY_WITH B

   ako i samo ako postoji smena oblika A $\rightarrow \alpha$ B β i sekvenca α je poništiva.

   Ova relacija određuje se sistematskim razmatranjem početaka smena. Na primer, iz treće smene vidi se da:

<A> BEGINS_DIRECTLY_WITH <A>,

<A> BEGINS_DIRECTLY_WITH <S> i

<A> BEGINS_DIRECTLY_WITH d

jer su neterminali <A> i <S> poništivi.

Za datu gramatiku ova relacija prikazana je na Sl. 3.2.10 jedinicama. Na primer, jedinica u vrsti <S> i koloni a označava da važi <S> BEGINS_DIRECTLY_WITH a, dok na primer ne važi <S> BEGINS_DIRECTLY_WITH b jer je odgovarajući ulaz matrice nepopunjen.

	<S>	<A>	<B>	<C>	<D>	a	b	c	d	e	f	g
<S>	#					1						
<A>	1	1				#			1			
<B>	1	1	#			#		#	#	1		
<C>	1			1		#					1	1
<D>					#	1						
a						#						
b							#					
c								#				
d									#			
e										#		
f											#	
g												#

Sl. 3.2.10: Relacije BEGINS_DIRECTLY_WITH (1) i BEGINS_WITH (1,#)

3. Konstrukcija relacije BEGINS_WITH.

Za proizvoljne gramatičke simbole A i B važi:

A BEGINS_WITH B

ako i samo ako $A \stackrel{*}{\Rightarrow} B$. Ova relacija dobija se refleksivno-tranzitivnim zatvaranjem relacije BEGINS_DIRECTLY_WITH primenom Warshall-ovog algoritma. Relacija je predstavljena kvadratnom matricom dimenzije n:

```
for (j := 1 to n)
        for (i := 1 to n)
                for (k := 1 to n)
                        if a[i,j] = 1 and a[j,k] = 1
                                then  a[i,k] := 1;
                        end if;
                end for;
```

```
            end for;
        end for;
```

Na Sl. 3.2.10 ova relacija prikazana je upotrebom simbola 1 i # (ovi poslednji su dodati zatvaranjem).

4. Određivanje FIRST skupova za svaki neterminal na osnovu relacije BEGINS_WITH po sledećoj formuli:

   FIRST(<X>) ={ t | $t \in V_t \wedge$ <X> BEGINS_WITH t }

   U konkretnom slučaju:

   FIRST(<S>) = {a} FIRST(<B>) = {a,c,d,e} FIRST(<D>) = {a}
   FIRST(<A>) = {a,d} FIRST(<C>) = {a,f,g}

5. Određivanje FIRST skupova za svaku smenu (videti Zadatak 3.2.5).

   Na primer, FIRST(3) = FIRST(<A>) $\cup$ FIRST(<S>) $\cup$ {d} jer su <A> i <S> poništivi.

   FIRST(1) = {a} FIRST(5) = {a,d,c} FIRST(9) = {a,f,g}
   FIRST(2) = $\varnothing$ FIRST(6) = {e} FIRST(10) = $\varnothing$
   FIRST(3) = {a,d} FIRST(7) = $\varnothing$ FIRST(11) = {a}
   FIRST(4) = $\varnothing$ FIRST(8) = {a,f} FIRST(12) = $\varnothing$

6. Konstrukcija relacije IS_FOLLOWED_DIRECTLY_BY.

   Za proizvoljne gramatičke simbole A i B važi:

   A IS_FOLLOWED_DIRECTLY_BY B

   ako i samo ako postoji smena oblika $D \rightarrow \alpha A \beta B \gamma$ i sekvenca β je poništiva. Ova relacija određuje se sistematskim razmatranjem svaka dva simbola sa desne strane iste smene za svaku smenu. Za datu gramatiku relacija IS_FOLLOWED_DIRECLTY_BY prikazana je na Sl. 3.2.11.

	<S>	<A>	<B>	<C>	<D>	a	b	c	d	e	f	g
<S>		1						1	1		1	
<A>	1		1				1	1	1			
<B>					1		1					
<C>					1							1
<D>												
a		1	1		1		1					
b				1	1							
c												
d												
e				1								

f												
g												

Sl. 3.2.11: Relacija IS_FOLLOWED_DIRECTLY_BY

7. Konstrukcija relacije IS_DIRECT_END_OF.

   Za proizvoljne gramatičke simbole A i B važi:

   A IS_DIRECT_END_OF B

   ako i samo ako postoji smena oblika B $\rightarrow\alpha$ A β i sekvenca β je poništiva. Ova relacija određuje se sistematskim razmatranjem završetaka smena. Na primer, iz prve smene sledi:

   <D> IS_DIRECT_END_OF <S>

   <C> IS_DIRECT_END_OF <S>

   b IS_DIRECT_END_OF <S>

   jer su neterminali <C> i <D> poništivi. Relacija je prikazana na Sl. 3.2.12.

	<S>	<A>	<B>	<C>	<D>	a	b	c	d	e	f	g
<S>	#											
<A>		#										
<B>	#		#		1							
<C>	1		1	#	#							
<D>	1				1							
a	#				1	#						
b	1						#					
c	#		1		#			#				
d		1							#			
e	#		1		#					#		
f	#		#	1	#						#	
g	#		#	1	#							#

Sl. 3.2.12: Relacije IS_DIRECT_END_OF (samo 1) i IS_END_OF (1 i #)

8. Konstrukcija relacije IS_END_OF.

   Za proizvoljne gramatičke simbole A i B važi:

   A IS_DIRECT_END_OF B

   ako i samo ako B $\stackrel{*}{\Rightarrow} \alpha$ A.

Ova relacija predstavlja refleksivno-tranzitivno zatvaranje relacije IS_DIRECT_END_OF i za datu gramatiku prikazana je na Sl. 3.2.12.

9. Konstrukcija relacije IS_FOLLOWED_BY.

Za proizvoljne gramatičke simbole A i B važi:

A IS_FOLLOWED_BY B

ako i samo ako <S> $\stackrel{*}{\Rightarrow}$ α A B β, gde je <S> startni neterminal.

Ova relacija se izračunava kao relacioni proizvod:

IS_FOLLOWED_BY = IS_END_OF * IS_FOLLOWED_DIRECTLY_BY *
* BEGINS_WITH

Na primer, za datu gramatiku važi:

<C> IS_END_OF <B>

<B> IS_FOLLOWED_DIRECTLY_BY <D>

<D> BEGINS_WITH a

iz čega sledi:

<C> IS_FOLLOWED_BY a

Relacija IS_FOLLOWED_BY data je na Sl. 3.2.13. (bez poslednje kolone).

	<S>	<A>	<B>	<C>	<D>	a	b	c	d	e	f	g	⊣
<S>	1	1				1		1	1		1		1
<A>	1	1	1			1	1	1	1	1			
<B>	1	1			1	1	1	1	1		1		1
<C>	1	1			1	1	1	1	1		1	1	1
<D>	1	1				1		1	1		1		1
a	1	1	1		1	1	1	1	1	1	1		1
b	1	1		1	1	1		1	1		1		1
c	1	1			1	1	1	1	1		1	1	1
d	1	1	1			1	1	1	1	1			
e	1	1		1	1	1	1	1	1		1	1	1
f	1	1			1	1	1	1	1		1	1	1
g	1	1			1	1	1	1	1		1	1	1

Sl. 3.2.13: Relacija IS_FOLLOWED_BY proširena markerom ⊣

10. Proširivanje domena relacije IS_FOLLOWED_BY uključivanjem markera kraja ulaza ⊣.

    Za proizvoljan gramatički simbol A važi:

    A IS_FOLLOWED_BY ⊣

    ako i samo ako važi:

    A IS_END_OF <S>

    gde je <S> startni neterminal.

    Relaciji IS_FOLLOWED_BY dodaje se kolona za marker kraja ulaza ⊣ u koji se upisuje kolona <S> iz relacije IS_END_OF odnosno svi simboli koji mogu da završavaju <S>. Za datu gramatiku ova relacija prikazana je na Sl. 3.2.13.

11. Određivanje FOLLOW skupova za sve poništive neterminale.

    FOLLOW(<X>) = {t | $t \in V_t \wedge$ <X> IS_FOLLOWED_BY t}

    Za datu gramatiku:

    FOLLOW(<S>) = {a, c, d, f, ⊣}

    FOLLOW(<A>) = {a, b, c, d, e}

    FOLLOW(<B>) = {a, b, c, d, f, ⊣}

    FOLLOW(<C>) = {a, b, c, d, f, g, ⊣}

    FOLLOW(<D>) = {a, c, d, f, ⊣}

12. Određivanje selekcionih skupova na način opisan u analizi problema.

    SELECT(1) = {a}

    SELECT(2) = {a, c, d, f, ⊣}

    SELECT(3) = {a, d}

    SELECT(4) = {a, b, c, d, e}

    SELECT(5) = {a, d, c}

    SELECT(6) = {e}

    SELECT(7) = {a, b, c, d, f, ⊣}

    SELECT(8) = {a, f}

    SELECT(9) = {a, f, g}

    SELECT(10) = {a, b, c, d, f, g, ⊣}

    SELECT(11) = {a}

    SELECT(12) = {a, c, d, f, ⊣}

Gramatika nije LL(1), jer, na primer, selekcioni skupovi 1. i 2. smene nisu disjunktni.

Zadatak 3.2.8

a) Napisati LL(1) gramatiku na osnovu koje je konstruisan potisni automat sa Sl. 3.2.14.

	a	b	c	⊣
<S>	REPLACE (<B>a<A>) RETAIN	REPLACE (<B>a<A>) RETAIN	POP ADVANCE	REPLACE (<B>a<A>) RETAIN
<A>	POP RETAIN	REPLACE (<S>c<B>) RETAIN	POP RETAIN	POP RETAIN
<B>	REJECT	REPLACE(<A>) ADVANCE	REJECT	REJECT
a	POP ADVANCE	REJECT	REJECT	REJECT
c	REJECT	REJECT	POP ADVANCE	REJECT
∇	REJECT	REJECT	REJECT	ACCEPT

Sl. 3.2.14

b) Na osnovu gramatike nađene pod a) odgovoriti na pitanje: koje ulaze u tabeli datog automata je moguće promeniti u REJECT a da se pri tome ne promeni jezik koji automat prepoznaje?

Rešenje

a)

1. <S>→<A> a <B> na osnovu (<S>, a), (<S>, b) i (<S>, ⊣)
2. <S>→c na osnovu (<S>, c)
3. <A>→ε na osnovu (<A>, a), (<A>, c), (<A>, ⊣)
4. <A>→<B> c <S> na osnovu (<A>, b)
5. <B>→b <A> na osnovu (<B>, b)

b)

FIRST (<S>) = {c, a, b} FOLLOW (<S>) = a, c, ⊣ }

FIRST (<A>) = {b} FOLLOW (<A>) = {a, c, ⊣ }

FIRST (<B>) = {b} FOLLOW (<B>) = {c, ⊣ }

SELECT(1)={b, a} SELECT(2)={c} SELECT(3)={a, c, ⊣}

SELECT(4)={b} SELECT(5)={b}

Gramatika jeste LL(1) jer su selekcioni skupovi smena sa istom levom stranom disjunktni.

U vrsti <S>, u skladu sa skupovima SELECT(1) i SELECT(2), akcijama za uparivanje 1. i 2. smene, ispravno su popunjeni ulazi:

(<S>, b) i (<S>, a) i (<S>, c)

U vrsti <A>, u skladu sa skupovima SELECT(3) i SELECT(4), akcijama za uparivanje 3. i 4. smene, ispravno su popunjeni ulazi:

(<A>, a) i (<A>, b) i (<A>, c) i (<A>, ⊣)

U vrsti <B>, u skladu sa skupom SELECT(5), akcijama za uparivanje 5. smene, ispravno su popunjeni ulazi:

(<B>, b) i (a, a) i (c, c) i (∇,⊣)

U tabeli je popunjen i ulaz (<S>, ⊣) akcijama REPLACE (<B> a <A>), RETAIN. Ove akcije možemo da zamenimo akcijom REJECT jer se marker kraja ulaza ne pojavljuje u selekcionim skupovima smena koje imaju <S> na levoj strani.

Zadatak 3.2.9

a) Eliminisati mrtve i nedostižne neterminale iz date gramatike sa startnim simbolom <A>.

b) Iz gramatike dobijene pod a) eliminisati levu rekurziju.

c) Izvršiti levu faktorizaciju u gramatici dobijenoj pod b).

c) Izračunati selekcione skupove smena gramatike dobijene pod c). Da li se radi o LL(1) gramatici?

1. <A> → <A> c <B>
2. <A> → <C> c b
3. <A> → c <D>
4. <A> → <D>
5. <B> → b <B>
6. <B> → d
7. <C> → <C> c <E>
8. <D> → <D> a <B>
9. <D> → <B> b <B>
10. <D> → <B>
11. <E> → a b

Rešenje

a)

Prvo se eliminišu mrtvi neterminali. Određujemo skup živih neterminala Ž. Inicijalno, skup Ž uključuje one neterminale koji se pojavljuju na levim stranama smena na čijim desnim stranama se pojavljuju isključivo terminali:

Ž = {<B>, <E>}

Zatim u skup Ž dodajemo sve one neterminale koji se pojavljuju na levim stranama smena na čijim desnim stranama se pojavljuju samo terminali i oni neterminali koji se već nalaze u skupu Ž. Ovaj postupak se iterativno ponavlja sve dok postoji promena skupa Ž.

Ž = {<B>, <E>, <D>}

Ž = {<B>, <E>, <D>, <A>}

Ž = {<B>, <E>, <D>, <A>}

Po završetku formiranja skupa Ž svi preostali neterminali koji se ne pojavljuju u Ž pripadaju skupu mrtvih neterminala M.

M = {<C>}

Izbacujemo smene 2. i 7. u kojima se pojavljuje neterminal <C>. Gramatika sada glasi:

1. <A> → <A> c <B>
3. <A> → c <D>
4. <A> → <D>
5. <B> → b<B>
6. <B> → d
8. <D> → <D> a <B>
9. <D> → <B> b <B>
10. <D> → <B>
11. <E> → a b

Zatim se eliminišu nedostižni neterminali. U tom cilju formirano skup dostižnih neterminala D koji se inicijalno sastoji od startnog neterminala.

D = {<A>}

Zatim u skup D dodajemo sve neterminale koji se pojavljuju na desnim stranama smena na čijim se levim stranama pojavljuju neterminali iz <D>. Postupak se iterativno ponavlja sve dok postoji promena skupa D.

D = {<A>, <B>, <D>}

D = {<A>, <B>, <D>}

Kada je skup D formiran, preostali neterminali koji se ne pojavljuju u D pripadaju skupu nedostižnih neterminala ND.

ND = {<E>}

Iz gramatike izbacujemo smenu 9. u kojoj se pojavljuje neterminal <E>. Gramatika glasi:

1. <A> → <A> c <B>
3. <A> → c <D>
4. <A> → <D>
5. <B> → b<B>
6. <B> → d
8. <D> → <D> a <B>
9. <D> → <B> b <B>
10. <D> → <B>

b)

Analiza problema

Leva rekurzija se iz gramatike eliminiše sistematičnom primenom sledećeg pravila transformacije smena:

$$\begin{array}{lcl} <X> \to <X>\alpha_1 & & <X> \to \beta_1 <X'> \\ \quad \cdots & & \quad \cdots \\ <X> \to <X>\alpha_m & & <X> \to \beta_n <X'> \\ <X> \to \beta_1 & \to & <X'> \to \alpha_1 <X'> \\ \quad \cdots & & \quad \cdots \\ <X> \to \beta_n & & <X'> \to \alpha_1 <X'> \\ & & <X'> \to \varepsilon \end{array}$$

Rešenje

U datoj gramatici neterminali <A> i <D> su levo rekurzivni. Primenjujemo navedeno pravilo na smene za <A>, pri čemu je m = 1 i n = 2, α_1 = c<B>, β_1 = c<D>, β_2 = <D>.

$$\begin{array}{lcl} 1.\ <A> \to <A>\ c\ <B> & & 1'.\ <A> \to c\ <D>\ <A'> \\ 3.\ <A> \to c\ <D> & \to & 2'.\ <A> \to <D>\ <A'> \\ 4.\ <A> \to <D> & & 3'.\ <A'> \to c\ <B>\ <A'> \\ & & 4'.\ <A'> \to \varepsilon \end{array}$$

Primenjujemo pravilo na smene za <D>, pri čemu je m = 1 i n = 2, α_1 = a<B>, β_1 = <B>b<B>, β_2 = <B>:

$$\begin{array}{lcl} 8.\ <D> \to <D>\ a\ <B> & & 7'.\ <D> \to <B>\ b\ <B>\ <D'> \\ 9.\ <D> \to <B>\ b\ <B> & \to & 8'.\ <D> \to <B>\ <D'> \\ 10.\ <D> \to <B> & & 9'.\ <D'> \to a\ <B>\ <D'> \\ & & 10'.\ <D'> \to \varepsilon \end{array}$$

Bitno je napomenuti da u ovoj gramatici nije bilo indirektnih rekurzija, jer <B> uvek počinje terminalom, a <D> nigde na svojoj desnoj strani nema pozivanje na <A>, pa je zbog toga indirektna rekurzija nemoguća.

Gramatika sada ima oblik:

$$\begin{array}{ll} 1'.\ <A> \to c\ <D>\ <A'> & 6'.\ <B> \to d \\ 2'.\ <A> \to <D>\ <A'> & 7'.\ <D> \to <B>\ b\ <B>\ <D'> \\ 3'.\ <A'> \to c\ <B>\ <A'> & 8'.\ <D> \to <B>\ <D'> \\ 4'.\ <A'> \to \varepsilon & 9'.\ <D'> \to a\ <B>\ <D'> \\ 5'.\ <B> \to b\ <B> & 10'.\ <D'> \to \varepsilon \end{array}$$

c)

Analiza problema

Pravilo leve faktorizacije glasi:

$$\begin{array}{lcl} <S> \to \alpha\ \beta & & <S> \to \alpha\ <S'> \\ <S> \to \alpha\ \gamma & \to & <S'> \to \beta \\ & & <S'> \to \gamma \end{array}$$

Rešenje

Primenom ovog pravila na smene 7′. i 8′., pri čemu je α =

$$\begin{array}{lcl} 7'.\ <D> \to <B>\ b\ <B>\ <D'> & \to & 7''.\ <D> \to <B>\ <D''> \\ 8'.\ <D> \to <B>\ <D'> & & 8''.\ <D''> \to b\ <B>\ <D'> \\ & & 9''.\ <D''> \to <D'> \end{array}$$

Na kraju će gramatika imati sledeći oblik:

$$\begin{array}{ll} 1''.\ <A> \to c\ <D>\ <A'> & 7''.\ <D> \to <B>\ <D''> \\ 2''.\ <A> \to <D>\ <A'> & 8''.\ <D''> \to b\ <B>\ <D'> \\ 3''.\ <A'> \to c\ <B>\ <A'> & 9''.\ <D''> \to <D'> \\ 4''.\ <A'> \to \varepsilon & 10''.\ <D'> \to a\ <B>\ <D'> \\ 5''.\ <B> \to b\ <B> & 11''.\ <D'> \to \varepsilon \\ 6''.\ <B> \to d & \end{array}$$

d)

SELECT (1″) = {c}
SELECT (2″) = FIRST (<D>) = FIRST (<B>) = {b, d}
SELECT (3″) = {c}
SELECT (4″) = FOLLOW (<A′>) = FOLLOW (<A>) = {┤}
SELECT (5″) = {b}
SELECT (6″) = {d}
SELECT (7″) = FIRST (<B>) = {b, d}
SELECT (8″) = {b}
SELECT (9″) = FIRST (<D′>) ∪ FOLLOW (<D″>) = {a} ∪ FOLLOW (<D>)
= {a} ∪ FIRST (<A′>) ∪ FOLLOW (<A>)
= {a} ∪ {c} ∪ {┤}
= {a, c, ┤}

SELECT (10″) = {a}

SELECT (11″) = FOLLOW (<D′>) = FOLLOW (<D″>) = FOLLOW(<D>)

= FIRST (<A′>) ∪ FOLLOW (<A>)

= {c, ⊣ }

Ova gramatika jeste LL (1) tipa jer su selekcioni skupovi disjunktni.

Zadatak 3.2.10

Konstruisati parser na principu rekurzivnog spusta koji odgovara prepoznavaču aritmetičkog izraza pomoću sledećih produkcija.

1. <E> → <T> <E_list>
2. <E_List> → + <T> <E_List>
3. <E_list> → ε
4. <T> → <P> <T_list>
5. <T_List> → * <P> <T_List>
6. <T_list> → ε
7. <P> → (<E>)
8. <P> → I

Analiza problema

Realizaciju parsera u višem programskom jeziku možemo sprovesti tako da umesto potisnog automata koji isključivo radi sa sadržajem steka i ulaza, napišemo program koga čini skup uzajamno rekurzivnih potprograma. U tom slučaju ulogu steka potisnog automata preuzima izvršni stek programa. Ovo rešenje pogodno je u slučaju gramatika sa manjim brojem smena i neterminala.

Rešenje

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. U priloženom pseudoprogramu globalna promenljiva IN predstavlja tekući ulazni znak. U potprogramu PROCP, korišćenjem CASE konstrukcije, prepoznaje se, na primer, smena 7 u slučaju da je tekući ulazni znak (. Ovo je naravno uslovljeno selekcionim skupom 7. smene. Sledeći fragment koda odgovara tom slučaju, odnosno uparivanju desne strane smene <P> → (<E>) sa tekućim ulazom:

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

```
                    end if
```

U ovom fragmentu, sa ulaza se čita sledeći znak pozivom funkcije NEXTCHAR() i smešta u IN (time je uparena i sa ulaza sklonjena otvorena zagrada), zatim se pozivom PROCE ostvaruje uparivanje neterminala <E> sa delom preostalog ulaza i potom se ispituje da li je posle ovoga uparivanja tekući znak ‘)’, ukoliko jeste ovaj znak se sklanja sa ulaza. Time je cela desna strana 7. smene uparena sa ulazom.

Glavni program inicijalizuje promenljivu IN da sadrži prvi ulazni znak. Zatim, pozivom potprograma PROCE, inicira uparivanje ulazne sekvence sa startni neterminalom. Po povratku iz potprograma PROCE u glavnom programu 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.

```
    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();
                    return;
            else    REJECT;
        end if;

    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;                                    end procedure;
```

Zadatak 3.2.11

a) Kojoj gramatici odgovara sledeći rekurzivni prepoznavač napisan na pseudojeziku:

```
glavni program:
    INP = prvi ulazni simbol
    call PROCS;
    if (INP<> '⊣' )
        then   REJECT;
        else   ACCEPT;
    end if;
end;

procedure PROCS:
    case INP of
        'a', 'c', 'd': {call PROCA;
                            if (INP <> 'a')
                        then REJECT
                        else call ADVANCE;
                        call PROCA;
                      end if;
                     }
        'b':          call ADVANCE;
        '⊣':          REJECT;
    end case;
    return;
end procedure;

procedure PROCA:
    case INP of
        'c' :  { call PROCA;
                 if (INP <> 'c')
                     then REJECT
                     else ADVANCE;
                 end if;
               }
        'd' :  call ADVANCE;
        'a', '⊣': /* ništa */;
        'b':  REJECT;
    end case;
    return;
end procedure;

procedure ADVANCE:
    INP = sledeći ulazni simbol;
    return;
end procedure;
```

b) Nalaženjem SELECT skupova odgovoriti na pitanje da li je gramatika iz tačke a) tipa LL(1).

Rešenje

U rekurzivnom prepoznavaču svakom neterminalu u gramatici odgovara jedna procedura. U konkretnom slučaju gramatika ima dva neterminala <S> i <A> pri čemu je <S> startni neterminal s obzirom da se odgovarajuća procedura PROCS poziva iz glavnog programa. Analizom procedure PROCS nalazimo smene koje imaju <S> na levoj strani:

- Programski fragment:

```
case INP of
    'a', 'c', 'd': {call PROCA;
                        if (INP <> 'a')
                    then REJECT
                    else call ADVANCE;
```

```
                    call PROCA;
                  end if;
                }
```

odgovara smeni <S> → <A> a <A>

- Programski fragment:

```
'b':    call ADVANCE;
```

odgovara smeni <S> → b

Ovo su jedine smene koje odgovaraju neterminalu <S>. Analizom PROCA utvrđujemo smene za <A>:

- Programski fragment:

```
case INP of
  'c' :  { call PROCA;
           if (INP <> 'c')
              then REJECT
              else ADVANCE;
           end if;
         }
```

odgovara smeni <A> → <A> c

- Programski fragment:

```
'd' :   call ADVANCE;
```

odgovara smeni <A> → d

- Programski fragment

```
'a', '⊣': /* ništa */;
```

odgovara smeni <A> → ε

Prema tome, gramatika na osnovu koje je napravljen program ima sledeći izgled:

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

b)

Neterminal <A> je poništiv, a neterminal <S> nije, tako da imamo:

FIRST(<A>) = FIRST(<A>) ∪ {c} ∪ {d} = {c, d}

FIRST(<S>) = FIRST(<A>) ∪ {a} ∪ {b} = {a, b, c, d}

FOLLOW(<S>) = {┤}

FOLLOW(<A>) = {a} ∪ FOLLOW(<S>) ∪ {c} = {a, c, ┤}

Sada se mogu odrediti selekcioni skupovi smena:

SELECT(1) = FIRST(<A> a <A>) = FIRST(<A>) ∪ {a} = {a, c, d}

SELECT(2) = {b}

SELECT(3) = FIRST(<A> c) = FIRST(<A>) ∪ {c} = {c, d}

SELECT(4)= {d}

SELECT(5) = FOLLOW(<A>) = {a, c, ┤}

Vidimo da selekcioni skupovi 3., 4, i 5. smene nisu disjunktni (što se moglo očekivati s obzirom da smena 3. poseduje direktnu levu rekurziju), pa gramatika nije LL(1). Samim tim, ni zadati rekurzivni prepoznavač ne radi ispravno, odnosno, može da uđe u beskonačnu rekurziju ako je c tekući ulazni simbol u trenutku izvršavanja PROCA.

Zadatak 3.2.12

Navesti primere gramatika pogodnih za procesiranje od vrha ka dnu, za opis lista entiteta u sledećim varijantama:

- lista od nula ili više elemenata
- lista od jednog ili više elemenata

u kombinaciji sa sledećim varijantama:

- nema posebnog separatora entiteta
- separator je zarez.

Analiza problema

Mnogi programski jezici sadrže konstrukcije koje uključuju liste entiteta, poput promenljivih, parametara, dimenzija i izraza. Ove konstrukcije se pojavljuju toliko često da projektant koji poželi da opiše jezik LL(1) gramatikom obično mora da uključi pogodne smene za generisanje listi.

Rešenje

Pet LL(1) gramatika za liste na Sl. 3.2.15. mogu da posluže kao osnova pri projektovanju smena za generisanje listi.

Gramatika I opisuje liste od jednog ili više simbola *a*. Neterminal <L> generiše samu listu, a <R> se može posmatrati kao onaj koji generiše ostatak liste posle početnog *a*.

Gramatika II opisuje liste od jednog ili više simbola *a*, koji su razdvojeni zarezima.

Gramatika III opisuje liste od nula ili više simbola *a*. Gramatika III' je alternativna gramatika za ovaj jezik. Gramatika III' je jednostavnija ali ponekad nepogodna za svrhe translacije.

Gramatika IV služi za generisanje liste od nula ili više simbola *a* razdvojenih zarezima. Ovu gramatiku možemo da posmatramo kao glavni prototip, pošto gramatika bez punktuacije (poput III) može da se dobije iz IV brisanjem zareza; gramatika koja zabranjuje listu nulte dužine (kao II) se dobija brisanjem prazne smene za <L>; gramatika I se dobija obavljanjem obe izmene.

	Bez punktuacije	Punktuacija zarezom
Prazna lista nije dozvoljena	I. <L> → a <R> <R> → a <R> <R> → ε	II. <L> → a <R> <R> → , a <R> <R> → ε
Prazna lista je dozvoljena	III. <L> → ε <L> → a <R> <R> → a <R> <R> → ε ili III'. <L> → ε <L> → a <L>	IV. <L> → ε <L> → a <R> <R> → , a <R> <R> → ε

Sl. 3.2.15

Diskusija

U svakoj gramatici "ostatak" liste stvoren od <R> je lista od nula ili više elemenata. U gramatici II, na primer, <R> generiše listu od nula ili više dešavanja simbola *a*. U svakoj gramatici smene za <R> su inspirisane gramatikom III' koja daje najjednostavniji način da se generiše lista od nula ili više elemenata. Tako je, u izvesnom smislu, gramatika III' osnovna gramatika listi, a ostale su tek nešto više razmatrane gramatike koje sadržavaju liste od nula ili više elemenata. S te tačke gledišta, gramatika II je opisana kao ona koja generiše *a* kome sledi lista sa nula ili više simbola *a*.

3.3. Programski primeri

Zadatak 3.3.1

Realizovati program koji računa poništive neterminale, FIRST i FOLLOW skupove neterminala za zadatu proizvoljnu bezkontekstnu gramatiku. Gramatika se zadaje u tekstualnoj datoteci. Svaki red odgovara po jednoj smeni. Svako slovo odgovara jednom gramatičkom simbolu (neterminalima velika slova, terminalima mala). Prvo slovo u redu odgovara simbolu leve strane smene, zatim ide razmak pa niz simbola desne strane smene; na primer, smena <X> → <Y><Z>vw se zadaje sa

X Yvw

Iz praktičnih razloga može se uzeti da dužina desnih strana smena ne prelazi dvadeset simbola.

Rešenje

Priloženi program za ulaz na Sl. 3.3.1(a) koji predstavlja ranije korišćenu gramatiku (Zadatak 3.2.7) daje izlaz na Sl. 3.3.1(b), gde je sa ! označen marker kraja ulaza.

```
S aABbCD
S
A Asd
A
B Sac
B eC
B
C Sf
C Cg
C
D aBD
D
```

(a)

```
NULLABLE: S A B C D

FIRST(S)={a}
FIRST(A)={ad}
FIRST(B)={adce}
FIRST(C)={afg}
FIRST(D)={a}

FOLLOW(S)={!dacf}
FOLLOW(A)={adceb}
FOLLOW(B)={ba!dcf}
FOLLOW(C)={a!bgdcf}
FOLLOW(D)={!dacf}
```

(b)

Sl. 3.3.1

U programu koji sledi, leve strane gramatičkih smena pamte se u znakovnom vektoru LHS[], a desne strane u vektoru znakovnih nizova RHS[]. Vektor NULLABLE[], za proizvoljan neterminal X čuva informaciju da li je X poništiv, ukoliko je NULLABLE[‘X’] jednako 1, ili nije poništiv, ukoliko je NULLABLE[‘X’] jednako 0. Za pamćenje FIRST i FOLLOW skupova neterminala koriste se istoimeni vektori znakovnih nizova. Za neterminal X, element vektora FIRST[‘X’] čuva u obliku znakovnog niza skup FIRST(X), slično tome pamte se i FOLLOW skupovi.

Gramatika se čita sa standardnog ulaza procedurom read_grammar(), zatim se određuju i na izlazu ispisuju poništivi neterminali procedurama calc_nullable() i print_nullable(). Prcedurom calc_first() izračunavaju se FIRST skupovi iterativnim metodom skupovnih jednakosti (Zadatak 3.2.5), koji se zatim ispisuju procedurom print_first(). Procedurom calc_follow() izračunavaju

se FOLLOW skupovi (Zadatak 3.2.5) pa zatim ispisuju procedurom print_follow(). Procedure calc_first() i calc_follow() koriste pomoćnu funkciju AddSet(dest, src) koja u odredišni skup dest dodaje sve elemente izvorišnog skupa src, vraćajući kao rezultat 1 ukoliko je došlo do promene odredišnog skupa, a 0 ako je odredišni skup neizmenjen.

```
#include <stdio.h>
#include <string.h>

#define MAX_PROD 100
#define MAX_RHS   20

char LHS[MAX_PROD];
char RHS[MAX_PROD][MAX_RHS];
int NULLABLE[128];
char *FIRST[128][27];
char *FOLLOW[128][27];

int num_prod;

void read_grammar(void){
   char line[80];
   num_prod=0;
   while ( gets(line) != NULL ){
      LHS[num_prod] = line[0];
      strcpy( RHS[num_prod++], ( ( strlen( line ) > 1 ) ? line + 2 : "" ) );
   }
}

void calc_nullable(void){
   int i;
   int change;
   for ( i = 0; i < 128; i++ )
      NULLABLE[i] = 0;
   do{
      change = 0;
      for ( i = 0; i < num_prod; i++ )
         if (!NULLABLE[LHS[i]]){
            int j;
            int null_rhs = 1;
            for ( j = 0; j < strlen( RHS[i] ); j++ )
               if ( !NULLABLE[RHS[i][j]] )
                  null_rhs = 0;
            if (null_rhs){
               NULLABLE[LHS[i]] = 1;
               change = 1;
            }
         }
   } while ( change );
}
```

```
void print_nullable(void){
   int i;
   int printed[128];
   for ( i = 0; i < 128; i++ )
      printed[i] = 0;
   printf("\n\nNULLABLE:");
   for ( i = 0; i < num_prod; i++ )
      if ( NULLABLE[LHS[i]] && !printed[LHS[i]] ){
         printf(" %c", LHS[i]);
         printed[LHS[i]] = 1;
      }
   printf("\n\n");
}

int AddSet( const char *dest, const char *src ){
   int i;
   int change = 0;
   for ( i = 0; i < strlen(src); i++ )
      if ( strchr( dest, src[i] ) == NULL ){
         strncat( dest, src + i, 1 );
         change = 1;
      }
   return change;
}

void calc_first(void){
   int i,j;
   int change;
   for ( i = 'A'; i <= 'Z'; i++ )
      FIRST[i][0] = '\0';
   for ( i = 'a'; i <= 'z'; i++ ){
      FIRST[i][0] = i;
      FIRST[i][1] = '\0';
   }
   do{
      change = 0;
      for ( i = 0; i < num_prod; i++ )
         for ( j = 0; j < strlen( RHS[i] ); j++ ){
            change = change | AddSet( FIRST[LHS[i]], FIRST[RHS[i][j]] );
            if ( !NULLABLE[RHS[i][j]] )
               break;
         }
   } while ( change );
}

void print_first(void){
   int i;
   int printed[128];
```

```
    for ( i = 0; i < 128; i++ )
        printed[i] = 0;
    for ( i = 0; i < num_prod; i++ )
        if ( !printed[LHS[i]] ){
            printf("FIRST(%c)={%s}\n", LHS[i], FIRST[LHS[i]]);
            printed[LHS[i]] = 1;
        }
}
void calc_follow(void){
    int i,j;
    int change;
    for ( i = 'A'; i <= 'Z'; i++ )
        FOLLOW[i][0] = '\0';
    FOLLOW[LHS[0]][0] = '!';
    FOLLOW[LHS[0]][1] = '\0';
    do
    {
        change = 0;
        for ( i = 0; i < num_prod; i++ )
            for ( j = 0; j < strlen( RHS[i] ); j++ )
                if ( isupper( RHS[i][j] ) ){
                    int k;
                    int null_rest = 1;
                    for ( k = j+1; null_rest && k < strlen( RHS[i] ); k++ ){
                        change = change | AddSet( FOLLOW[RHS[i][j]], FIRST[RHS[i][k]] );
                        null_rest = NULLABLE[RHS[i][k]];
                    }
                    if ( null_rest )
                        change = change | AddSet( FOLLOW[RHS[i][j]], FOLLOW[LHS[i]] );
                }
    } while ( change );
}
void print_follow(void){
    int i;
    int printed[128];
    for ( i = 0; i < 128; i++ )
        printed[i] = 0;
    printf("\n");
    for ( i = 0; i < num_prod; i++ )
        if ( !printed[LHS[i]] ){
            printf("FOLLOW(%c)={%s}\n", LHS[i], FOLLOW[LHS[i]]);
            printed[LHS[i]] = 1;
        }
}
main(){
```

```
   read_grammar();
   calc_nullable();
   print_nullable();
   calc_first();
   print_first();
   calc_follow();
   print_follow();
}
```

Zadatak 3.3.2

Realizovati generator parsera na bazi rekurzivnog spusta za zadatu S-gramatiku (bez atributa i akcionih simbola). Gramatika se zadaje u tekstualnoj datoteci. Svaki red odgovara po jednoj smeni. Svako slovo odgovara jednom gramatičkom simbolu (neterminalima velika slova, terminalima mala). Prvo slovo u redu odgovara simbolu leve strane smene, zatim ide razmak pa niz simbola desne strane smene, npr. smena <X> $\rightarrow$ <Y><Z>vw se zadaje sa

X Yvw

Iz praktičnih razloga može se uzeti da dužina desnih strana smena ne prelazi dvadeset simbola. Izlaz programa treba da bude tekstualna datoteka sa programom koji predstavlja parser na bazi rekurzivnog spusta za zadatu gramatiku. Program treba da proverava da li je zadata gramatika zaista S-gramatika.

Rešenje

Razmotrimo prvo rezultate rada traženog generatora parsera. Za ulaz:

```
S aDb
D dSaD
S b
D ba
```

kao izlaz dobija se sledeći program na C-u:

```
#include<stdio.h>
#include<stdlib.h>

int in;

void PROCS(void);

void PROCD(void);

void ACCEPT(void){ printf("Accepted.\n"); }

void REJECT(void){ printf("Rejected.\n"); exit(1); }

main() {
  in = getchar();
  PROCS();
  if ( in == '\n' )
     ACCEPT();
```

```
void PROCD(void){
   switch ( in ) {
      case 'd': goto P01;
      case 'b': goto P03;
      default: REJECT();
```

```
  else
     REJECT();
}

void PROCS(void){
   switch ( in ) {
      case 'a': goto P00;
      case 'b': goto P02;
      default: REJECT();
   }
P00:in = getchar();
   PROCD();
   if ( in != 'b' )
       REJECT();
   else
       in = getchar();
   return;
P02:in = getchar();
   return;
}
   }
P01:in = getchar();
   PROCS();
   if ( in != 'a' )
       REJECT();
   else
       in = getchar();
   PROCD();
   return;
P03:in = getchar();
   if ( in != 'a' )
       REJECT();
   else
       in = getchar();
   return;
}
```

Ovaj program predstavlja prepoznavač za zadatu gramatiku koji na izlazu izdaje poruke ‘Accepted.’ ili ‘Rejected’ za sekvencu sa standardnog ulaza (kraj reda označava kraj ulazne sekvence).

U programu koji sledi, koji realizuje generator parsera, leve strane gramatičkih smena pamte se u znakovnom vektoru LHS[], a desne strane u vektoru znakovnih nizova RHS[]. Gramatika se učitava sa standardnog ulaza procedurom read_grammar(). Procedura is_s_grammar() vrši ispitivanje da li su zadovoljeni uslovi za S-gramatiku (Zadatak 3.2.3). Ukoliko data gramatika jeste S-gramatika, procedura generate_parser() na standarnom izlazu daje kod parsera na principu rekurzivnog spusta (Zadatak 3.2.10) koristeći pri tome nekoliko pomoćnih procedura. Procedura generate_prologue(), generiše potrebne deklaracije, procedure ACCEPT() i REJECT() i glavni program parsera. Procedura generate_procedure_for_nonterminal() poziva se za svaki gramatički neterminal X da generiše odgovarajuću proceduru PROCX. U okviru ove procedure, za svaku smenu koja odgovara neterminalu X poziva se procedura generate_code_for_production() da izgeneriše deo koda koji predstavlja akcije parsera na prepoznavanju te smene. Za S-gramatike selekcioni skupovi smena se trivijalno računaju, tako da je generisanje koda pravolinijsko na bazi šablona (Zadatak 3.2.10).

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAX_PROD 100
#define MAX_RHS   20

char LHS[MAX_PROD];
char RHS[MAX_PROD][MAX_RHS];

int num_prod;
```

```
void read_grammar(void){
   char line[80];
   num_prod=0;
   while ( gets(line) != NULL ){
      LHS[num_prod] = line[0];
      strcpy( RHS[num_prod++], ( ( strlen( line ) > 1 ) ? line + 2 : "" ) );
   }
}

int is_s_grammar(void){
   int i,j;
      for ( i = 0; i < num_prod; i++ )
         if ( RHS[i][0] == '\0' || isupper( RHS[i][0] ) )
            return 0;
      for ( i = 0; i < num_prod; i++ )
         for ( j = i + 1; j < num_prod; j++ )
            if ( LHS[i] == LHS[j] && RHS[i][0] == RHS[j][0] )
               return 0;
   return 1;
}

void generate_prologue(void){
   int i,j;
   int processed[128];
   for ( i = 0; i < 128; i++ )
      processed[i] = 0;
   printf("#include<stdio.h>\n"
          "#include<stdlib.h>\n\n"
          "int in;\n\n");
   for ( i = 0; i < num_prod; i++ )
      if ( !processed[LHS[i]] ){
         printf("void PROC%c(void);\n\n", LHS[i]);
         processed[LHS[i]]=1;
      }
   printf("void ACCEPT(void){ printf(\"Accepted.\\n\"); }\n\n");
   printf("void REJECT(void){ printf(\"Rejected.\\n\"); exit(1); }\n\n");
   printf("main() {\n"
          "   in = getchar();\n"
          "   PROC%c();\n"
          "   if ( in == '\\n' )\n"
          "      ACCEPT();\n"
          "   else\n"
          "      REJECT();\n"
          "}\n\n", LHS[0]);
}

void generate_code_for_production( int i ){
```

```
    int j;
    printf("P%02d:in = getchar();\n", i);
    for ( j = 1; j < strlen( RHS[i] ); j++ )
        if ( isupper( RHS[i][j] ) )
            printf("    PROC%c();\n", RHS[i][j]);
        else
            printf("    if ( in != '%c' )\n"
                   "        REJECT();\n"
                   "    else\n"
                   "        in = getchar();\n", RHS[i][j]);
    printf("    return;\n");
}

void generate_procedure_for_nonterminal( char nt ){
    int i;
    printf("void PROC%c(void){\n"
           "    switch ( in ) {\n", nt);
    for ( i = 0; i < num_prod; i++ )
        if ( LHS[i] == nt )
            printf("        case '%c': goto P%02d;\n", RHS[i][0], i);
    printf("        default: REJECT();\n"
           "    }\n");
    for ( i = 0; i < num_prod; i++ )
        if ( LHS[i] == nt )
            generate_code_for_production( i );
    printf("}\n\n");
}

void generate_parser(void){
    int i;
    int processed[128];
    for ( i = 0; i < 128; i++ )
        processed[i] = 0;
    generate_prologue();
    for ( i = 0; i < num_prod; i++ )
        if ( !processed[LHS[i]] ){
            generate_procedure_for_nonterminal( LHS[i] );
            processed[LHS[i]] = 1;
        }
}

main(){
    read_grammar();
    if ( is_s_grammar() )
        generate_parser();
    else
        printf("\n\nNije S gramatika.\n\n");
```

```
}
```

4. Sintaksna analiza od dna ka vrhu

4.1. Relacioni pristup

4.1.1. Metod potisni–identifikuj

Zadatak 4.1.1

Parser potisni–identifikuj od dna ka vrhu, napravljen na osnovu zadate gramatike, prepoznaje primenu smena u zadatoj ulaznoj sekvenci po sledećem redosledu:

4, 4, 4, 1, 4, 4, 1, 3, 2

a) O kojoj ulaznoj sekvenci je reč?

b) Prikazati rad parsera za ulaznu sekvencu iz tačke a).

c) Kako bi izgledao parser od vrha ka dnu za ovu gramatiku?

1. <S> $\rightarrow$ <S> <S> a

2. <S> $\rightarrow$ <S> b

3. <S> $\rightarrow$ <S> <S> <S> c

4. <S> $\rightarrow$ d

Rešenje

a)

Na osnovu činjenice da parser od dna ka vrhu prepoznaje primenu smena u redosledu koji je obrnut od redosleda smena u krajnje desnom izvođenju posmatrane ulazne sekvence, zaključujemo da se radi o sledećem izvođenju sekvence dddaddacb:

<S> $\Rightarrow_{rm}$ <S>b $\Rightarrow_{rm}$ <S><S><S>cb $\Rightarrow_{rm}$ <S><S><S><S>acb $\Rightarrow_{rm}$ <S><S><S> dacb $\Rightarrow_{rm}$

$$\underset{2}{\Rightarrow_{rm}} \underset{\uparrow 1}{\text{<S><S>ddacb}} \underset{3}{\Rightarrow_{rm}} \underset{\uparrow 4}{\text{<S><S><S>addacb}} \underset{1}{\Rightarrow_{rm}} \underset{\uparrow 4}{\text{<S><S>daddacb}} \underset{4}{\Rightarrow_{rm}}$$

$$\underset{4}{\Rightarrow_{rm}} \underset{\uparrow 4}{\text{<S>ddaddacb}} \Rightarrow_{rm} \text{dddaddacb}$$

b)

Kod parsera potisni–identifikuj svakom gramatičkom simbolu odgovara po jedan simbol steka. Dno steka označavamo posebnim simbolom ∇. U konkretnom slučaju simboli steka su ∇, <S>, a, b, c, d. Sledi prikaz rada parsera za posmatrani ulaz:

Stanje steka :	Ulazni simbol :	Primenjena smena:
1. ∇	dddaddacb⊣	SHIFT
2. ∇ d	ddaddacb⊣	REDUCE(4)
3. ∇ <S>	ddaddacb⊣	SHIFT
4. ∇ <S> d	daddacb⊣	REDUCE(4)
5. ∇ <S> <S>	daddacb⊣	SHIFT
6. ∇ <S> <S> d	addacb⊣	REDUCE(4)
7. ∇ <S> <S> <S>	addacb⊣	SHIFT
8. ∇ <S> <S> <S> a	ddacb⊣	REDUCE(1)
9. ∇ <S> <S>	ddacb⊣	SHIFT
10. ∇ <S> <S> d	dacb⊣	REDUCE(4)
11. ∇ <S> <S> <S>	dacb⊣	SHIFT
12. ∇ <S> <S> <S> d	acb⊣	REDUCE(4)
13. ∇ <S> <S> <S> <S>	acb⊣	SHIFT
14. ∇ <S> <S> <S> <S> a	cb⊣	REDUCE(1)
15. ∇ <S> <S> <S>	cb⊣	SHIFT
16. ∇ <S> <S> <S> c	b⊣	REDUCE(3)
17. ∇ <S>	b⊣	SHIFT
18. ∇ <S> b	⊣	REDUCE(2)
19. ∇ <S>	⊣	ACCEPT

Objašnjenje akcija:

SHIFT ≡ PUSH(tekući ulazni simbol), ADVANCE

REDUCE(1) ≡ POP, POP, POP, PUSH(<S>)

REDUCE(2) ≡ POP, POP, PUSH(<S>)

REDUCE(3) ≡ POP, POP, POP, POP, PUSH(<S>)

REDUCE(4) ≡ POP, PUSH(<S>)

Startno stanje steka je ∇. U svakom trenutku rada parsera sekvenca simbola koja odgovara sadržaju steka nadovezana na preostalu ulaznu sekvencu formira jednu od sentencijalnih formi, nazovimo je F, u krajnje desnom izvođenju posmatrane ulazne sekvence polazeći od startnog neterminala i pod pretpostavkom da je ulazna sekvenca prihvatljiva. Parser funkcioniše tako što na stek redom potiskuje (operacija SHIFT) ulazne simbole dok se na vrhu steka ne kompletira takozvana ručka – desna strana smene koja je poslednja primenjena u krajnje desnom izvođenju sentencijalne forme F. Tada parser zamenjuje na steku desnu stranu primenjene smene njenom levom stranom (operacija svođenja smene *i*, engl. REDUCE(*i*)).

c)

Zbog postojanja direktne leve rekurzije ova gramatika ne spada u klasu LL(k) gramatika, pa se ne može deterministički procesirati metodom od vrha ka dnu.

Zadatak 4.1.2

Projektovati parser potisni–identifikuj u vidu potisnog automata za sledeću gramatiku sa startnim simbolom <S>:

1. <S> → a <S> <A> 1

2. <S> → a <A> 1

3. <S> → a <S> 0

4. <A> → b <A> <S> 0

5. <A> → b <A> <A> 1

6. <A> → a b 0

Rešenje

Zadata gramatika ima specifičan oblik – svaka od smena završava se nekim od terminala 0 ili 1, ovi simboli ne pojavljuju se unutar desnih strana smena i dodatno nijedna desna strana nije sufiks neke druge desne strane smene. Radi se o bezsufiksnoj SI gramatici (vidi Zadatak 4.1.9). Ovo olakšava konstrukciju parsera, bez potrebe da se određuju pomoćne relacije.

Svakom gramatičkom simbolu odgovara tačno jedan simbol steka. Simbol steka je i marker dna steka ∇. Svakom ulaznom simbolu odgovara tačno jedan gramatički terminal. Kontrolnu tabelu potisnog automata (Sl. 4.1.1) proširujemo i kolonom za marker kraja ulaza ⊣.

Vrste kontrolne tabele za simbole koji se pojavljuju isključivo na krajevima smena, popunjavamo IDENTIFY rutinama u kojima se na osnovu nekoliko vršnih simbola steka identifikuje ručka i vrši odgovarajuća REDUCE operacija, ili sekvenca odbija ukoliko stek ne sadrži ručku.

U vrstama kontrolne tabele za ∇ i simbole koji se ne pojavljuju na krajevima smena, sve ulaze osim onoga u koloni za ⊣ popunjavamo akcijom SHIFT. Simbol ⊣ nikada se ne potiskuje na stek. Kolonu ⊣ popunjavamo akcijom ID-3, prema kojoj je ulazna sekvenca prihvatljiva ako i samo ako se na steku u tom trenutku nalazi samo simbol koji odgovara startnom neterminalu.

	a	b	0	1	⊣
<S>	SHIFT	SHIFT	SHIFT	SHIFT	ID – 3
<A>	SHIFT	SHIFT	SHIFT	SHIFT	ID – 3
a	SHIFT	SHIFT	SHIFT	SHIFT	ID – 3
b	SHIFT	SHIFT	SHIFT	SHIFT	ID – 3
0	ID – 1	ID – 1	ID – 1	ID – 1	ID – 1
1	ID – 2	ID – 2	ID – 2	ID – 2	ID – 2
∇	SHIFT	SHIFT	SHIFT	SHIFT	ID – 3

Sl. 4.1.1: Kontrolna tabela parsera potisni–identifikuj

Operacije:

SHIFT: PUSH (tekući ulazni simbol), ADVANCE

```
ID – 1:  if (vrh steka = a<S>0)
                then REDUCE(3)
         else if (vrh steka = b<A><S>0)
                then REDUCE(4)
         else if (vrh steka = ab0)
                then REDUCE(6)
         else REJECT
         end if

ID – 2:  if (vrh steka = a<S><A>1)
                then REDUCE(1)
         else if (vrh steka = a<A>1)
                then REDUCE(2)
         else if (vrh steka = b<A><A>1)
                then REDUCE(5)
         else REJECT
         end if

ID – 3:  if (vrh steka = ∇<S>)
                then ACCEPT
                else REJECT
         end if
```

```
REDUCE(1) = POP, POP, POP, POP, PUSH(<S>)
REDUCE(2) = POP, POP, POP, PUSH(<S>)
REDUCE(3) = POP, POP, POP, PUSH(<S>)
REDUCE(4) = POP, POP, POP, POP, PUSH(<A>)
REDUCE(5) = POP, POP, POP, POP, PUSH(<A>)
REDUCE(6) = POP, POP, POP, PUSH(<A>)
```

Diskusija

Redosled prepoznavanja smena u IDENTIFY rutinama u slučaju bezsufiksnih gramatika nije bitan. U datom rešenju, za svaki simbol kojim se može završiti smena uvedena je posebna IDENTIFY rutina. Alternativno je moguće imati jedinstvenu IDENTIFY rutinu u kojoj bi bile ispitivane sve smene.

Primenom formalne metodologije konstrukcije parsera za datu gramatiku preciznije bi se odredio smeštaj SHIFT i IDENTIFY akcija, odnosno neki od ulaza u dobijenom parseru bili bi popunjeni REJECT akcijama. Na primer, u vrsti a i koloni 0 trebalo bi da stoji REJECT, jer data gramatika ne prihvata ulazne sekvence u kojima se iza simbola a pojavljuje simbol 0, pošto se u smenama ispred nule pojavljuje isključivo neterminal <S>, a sve smene za <S> završavaju

se isključivo simbolima 0 ili 1. Ovo ne dovodi do pogrešnog rada parsera, u smislu odbijanja legalne sekvence (očigledno) ili prihvatanja nelegalne sekvence – detekcija greške biće u okviru IDENTIFY rutine. Ova odložena detekcija greške može da predstavlja problem kod oporavka od greške.

U drugoj varijanti parsiranja od dna ka vrhu, potisni i svedi, u okviru simbola steka kodira se i informacija o kompletiranosti ručke, tako da se REDUCE operacija bira neposredno na osnovu vršnog simbola steka i tekućeg ulaznog simbola. Ovakvi parseri su brži, ali imaju veće kontrolne tabele.

Zadatak 4.1.3

Projektovati parser na bazi gramatike slabog prvenstva (engl. *weak-precedence grammar*) za sledeću gramatiku sa startnim simbolom <S>:

1. <S> $\rightarrow$ <S> <A>

2. <S> $\rightarrow$ <A>

3. <A> $\rightarrow$ 1 <S> 0

4. <A> $\rightarrow$ 1 0

Analiza problema

Postupak konstrukcije kontrolne tabele parsera potisni-identifikuj formalizuje se uvođenjem relacija BELOW i REDUCED_BY na osnovu kojih se formulišu principi potiskivanja i svođenja.

Za proizvoljan simbol steka A i ulazni simbol x, važi da je:

A BELOW x

ako i samo ako važi jedan od sledećih uslova:

a) Postoji gramatička smena oblika <X> $\rightarrow \alpha$ A B β, i važi da je x $\in$ FIRST(B), gde su α i β proizvoljne sekvence od nula ili više gramatičkih simbola, B proizvoljan gramatički simbol, ili

b) A je marker dna steka, a x $\in$ FIRST(<S>), gde je <S> startni neterminal.

Drugim rečima, relacija BELOW pokazuje koji se sve simboli x mogu naći neposredno iznad simbola A na steku.

Princip potiskivanja glasi:

Akcija SHIFT mora da se nalazi u svakom od ulaza kontrolne tabele parsera potisni–identifikuj u vrsti A i koloni x, ako je ispunjeno A BELOW x. Ostali ulazi ne moraju sadržati ovu akciju.

Za proizvoljan simbol A na vrhu steka i proizvoljan ulazni simbol ili marker kraja ulaza x, važi da je

A REDUCED_BY x

ako i samo ako važi jedan od sledećih uslova:

a) Postoji gramatička smena oblika <L> $\rightarrow \alpha$ A pri čemu je x $\in$ FOLLOW(<L>), gde je α proizvoljna sekvenca od nula ili više gramatičkih simbola, <L> proizvoljan neterminal, ili

b) A je startni neterminal a x je marker kraja ulaza ⊣.

Princip svođenja glasi:

Akcija IDENTIFY mora da se nalazi u svakom od ulaza kontrolne tabele parsera potisni–identifikuj u vrsti A i koloni x, ako je ispunjeno A REDUCED_BY x. Ostali ulazi ne moraju sadržati ovu akciju.

Princip potiskivanja određuje koji sve ulazi kontrolne tabele moraju da sadrže akciju SHIFT, dok princip svođenja određuje isto za akcije IDENTIFY. Ostali ulazi mogu po želji biti popunjeni nekom od akcija REJECT, SHIFT, IDENTIFY uz ograničenje da se u koloni za ⊣ ne pojavljuju SHIFT akcije jer se marker kraja nikada ne potiskuje na stek.

Gramatika poseduje konflikt potisni-identifikuj ako postoji takav simbol steka A i ulazni simbol x za koje istovremeno važi:

A BELOW x i A REDUCED_BY x

Iako ovo znači da se za datu gramatiku deterministički parser potisni-identifikuj ne može konstruisati, u praksi je moguće pribeći rešavanju konflikta izborom pogodnije od dve akcije za dati ulaz.

Gramatike koje ne poseduju konflikte tipa potisni-identifikuj nazivaju se SI gramatikama.

Rešenje

Izračunajmo FIRST skupove simbola date gramatike:

FIRST(<S>) = { <S>, <A>, 1} FIRST(1) = {1}

FIRST(<A>) = { <A>, 1} FIRST(0) = {0}

Definicija FIRST skupova je proširena tako da su u njih uvršćeni i neterminali. Ovo nije neophodno za konstrukciju kontrolne tabele, ali je potrebno radi definisanja IDENTIFY rutina kod gramatika koje nisu bezsufiksne, kako će biti objašnjeno u nastavku. Relacija BELOW za datu gramatiku prikazana je na Sl. 4.1.2. Relacija je proširena u odnosu na definiciju kolonama za neterminale koje odgovaraju proširenim FIRST skupovima, što nije neophodno za bezsufiksne gramatike, ali jeste za gramatike slabog prvenstva.

	<S>	<A>	1	0
<S>		1	1	1
<A>				
1	1	1	1	1
0				
∇	1	1	1	

Sl. 4.1.2 Proširena relacija BELOW

Za datu gramatiku FOLLOW skupovi neterminala glase:

FOLLOW(<S>) = {1, 0, ⊣}

FOLLOW(<A>) = FOLLOW(<S>) = {1, 0, ⊣}

Relacija REDUCED_BY predstavljena je na Sl. 4.1.3.

	1	0	⊣
<S>			1
<A>	1	1	1
1			
0	1	1	1
∇			

Sl. 4.1.3 Relacija REDUCED_BY

Na osnovu ovih relacija konstruišemo kontrolnu tabelu automata (Sl. 4.1.4). U cilju konstrukcije IDENTIFY rutina potrebno je obaviti dodatno razmatranje gramatičkih smena.

	0	1	⊣
<S>	SHIFT	SHIFT	IDENTIFY–1
<A>	IDENTIFY–2	IDENTIFY–2	IDENTIFY–2
1	SHIFT	SHIFT	REJECT
0	IDENTIFY–3	IDENTIFY–3	IDENTIFY–3
∇	REJECT	SHIFT	REJECT

Sl. 4.1.4: Kontrolna tabela parsera potisni–identifikuj

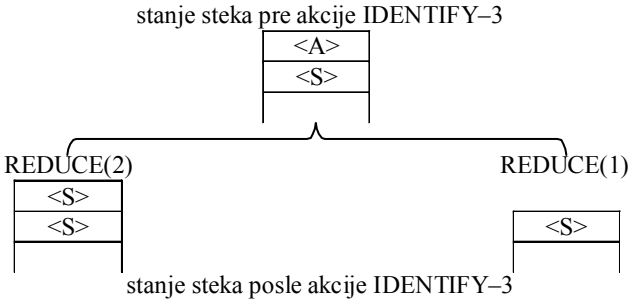
Data gramatika nije bezsufiksna, jer je desna strana druge smene:

<S> → <A>

sufiks desne strane prve smene:

<S> → <S> <A>

Ovo dovodi do sledeće konfliktne situacije: Kad se, u toku rada parsera, na vrhu steka pojavi <A>, a ispod ovog simbola se nalazi <S>, moguće je u okviru IDENTIFY–2 rutine preduzeti ili akciju REDUCE(2) ili akciju REDUCE(1):



Međutim, za prihvatljive ulazne sekvence nikada se ne pojavljuje situacija u kojoj je za opisanu konfiguraciju steka neophodno primeniti REDUCE(2) jer bi se tada na vrhu steka pojavila dva simbola <S>, pa bi trebalo da važi

<S> BELOW <S>

što ovde nije slučaj (vidi Sl. 4.1.2). Ovo znači da se u datoj situaciji uvek primenjuje svođenje prve smene. U suprotnom, došlo bi do konflikta jer bi u nekim situacijama trebalo primeniti REDUCE(1) a u nekim REDUCE(2) zavisno od konteksta (ostatka steka i preostalog dela ulaza), pa se ne bi ovim metodom mogao napraviti parser.

U opštem slučaju, ako gramatika poseduje smene oblika

$<A> \rightarrow \alpha\ B\ \beta$

$<C> \rightarrow \beta$

gde je B proizvoljan gramatički simbol a α i β proizvoljni nizovi gramatičkih simbola, i ako nije ispunjeno

B BELOW <C>

onda nikada u radu parsera ne dolazi do situacije da je desna strana duže smene na vrhu steka, a da je potrebno svesti kraću smenu (ne postoji konflikt svedi–svedi). U tom slučaju, u odgovarajuću identifikacionu rutinu treba ugraditi proveru za dužu smenu pre provere za kraću smenu, da bi se u datoj situaciji uvek vršilo svođenje duže smene.

Gramatike koje nemaju konflikte potisni–identifikuj, nemaju smene sa istom desnom stranom, za sufiksne smene je ispunjen gornji uslov i dodatno, ne važi da $<S> \Rightarrow^{+} <S>$ (da bi se eliminisala dvosmislenost) nazivaju se gramatike slabog prvenstva.

Za datu gramatiku izgled IDENTIFY rutina je sledeći:

```
IDENTIFY–3:   if (vrh steka = 1<S>0)
                    then REDUCE(3)
              else if (vrh steka = 10)
                    then REDUCE(4)
              else REJECT
              end if

IDENTIFY–1:   if (vrh steka = ∇<S>)
                    then ACCEPT
```

```
                else REJECT
        end if
IDENTIFY–2:  if (vrh steka = <S><A>)
                then REDUCE(1)
                else REDUCE(2)  /* ispitivanje steka nije potrebno, jer je  */
        end if                  /* desna strana 2. smene dužine 1           */
```

Zadatak 4.1.4

Dat je jezik $\{ a^n b^n \}$, $n > 0$.

a) Naći gramatiku slabog prvenstva za dati jezik.

b) Za nađenu gramatiku konstruisati parser potisni-identifikuj.

Rešenje

a)

Jedna od gramatika koje opisuju traženi jezik je sledeća:

1. <S>→ a<S>b
2. <S>→ <S1><S2>
3. <S1>→ a
4. <S2>→ b

Smene 2, 3 i 4. opisuju najkraću sekvencu ab. Duže sekvence se dobijaju uzastopnim primenama prve smene. Da bismo bili sigurni da je u pitanju gramatika slabog prvenstva, moramo proveriti zadovoljenost sledećih uslova:

← Nema konflikata potisni–identifikuj.

↑ Ne postoje dve smene sa istom desnom stranom.

→ Ako imamo neke smene <A>→α B β i <C>→ β ne sme da važi B BELOW <C>.

↓ Ne sme da važi <S> $\overset{+}{\Rightarrow}$ <S>, gde je <S> startni neterminal.

Nije teško utvrditi da su uslovi ↑ i ↓ ispunjeni. Za proveru ispunjenosti uslova ← i → nalazimo, formalno, relacije BELOW i REDUCED_BY po sledećoj proceduri:

1. Određivanje relacija BEGINS_DIRECTLY_WITH i njenog refleksivno-tranzitivnog zatvaranja (Sl. 4.1.5).

	<S>	<S1>	<S2>	a	b
<S>	←	1		1	
<S1>		←		1	

<S2>			←		1
a				←	
b					←

Sl. 4.1.5: Relacije BEGINS_DIRECTLY_WITH (samo 1) i BEGINS_DIRECTLY_WITH* (1 i ←)

2. Određivanje relacije IS_DIRECT_END_OF, njenog tranzitivnog i refleksivno-tranzitivnog zatvaranja (Sl. 4.1.6).

	<S>	<S1>	<S2>	a	b
<S>	←				
<S1>		←			
<S2>	1		←		
a		1		←	
b	1		1		←

Sl. 4.1.6 Relacije IS_DIRECT_END_OF i IS_DIRECT_END_OF $^+$ (samo 1) i IS_DIRECT_END_OF * (1 i ←)

3. Određivanje relacije IS_FOLLOWED_DIRECTLY_BY (Sl. 4.1.7).

	<S>	<S1>	<S2>	a	b
<S>					1
<S1>			1		
<S2>					
a	1				
b					

Sl. 4.1.7: Relacija IS_FOLLOWED_DIRECTLY_BY

4. Računanje relacije $\leq\cdot$ (dela relacije BELOW bez vrste ∇) na sledeći način (Sl. 4.1.8):
$\leq\cdot$ = IS_FOLLOWED_DIRECTLY_BY $\cdot$ BEGINS_DIRECTLY_WITH*

5. Puna relacija BELOW (Sl. 4.1.8) se dobija kada se relaciji $\leq\cdot$ doda vrsta ∇, u koju se prepiše vrsta <S> relacije BEGINS_DIRECTLY_WITH*, odnosno relaciji $\leq\cdot$ dodaju se svi parovi (∇, X) za sve gramatičke simbole X za koje važi <S> BEGINS-DIRECTLY_WITH* X.

	<S>	<S1>	<S2>	a	b

<S>					1
<S1>			1		1
<S2>					
a	1	1		1	
b					
∇	1	1		1	

Sl. 4.1.8: Relacije ≤· (bez vrste ∇) i BELOW

6. Računanje relacije ·> (dela relacije REDUCED_BY bez kolone ⊣) na sledeći način (Sl. 4.1.9):
   ·> = IS_DIRECT_END_OF $^{+}$ · ≤·

7. Puna relacija REDUCED_BY (Sl. 4.1.9) dobija se proširivanjem relacije ·> (samo kolone za terminale) kolonom za marker kraja ulaza, u koju se prepiše kolona <S> relacije IS_DIRECT_END_OF *, odnosno relaciji ·> dodaju se svi parovi (X, ⊣) za sve gramatičke simbole X za koje važi X IS_DIRECT_END_OF * <S>.

	<S>	<S1>	<S2>	a	b	⊣
<S>						1
<S1>						
<S2>					1	1
a			1		1	
b					1	1

Sl. 4.1.9: Relacije ·> (sve kolone osim ⊣) i REDUCED_BY (samo kolone a, b, i ⊣)

Zadovoljenost uslova ←, odnosno nepostojanje konflikta potisni–identifikuj, potvrđujemo tako što ustanovljavamo da ne postoji nijedan par gramatičkih simbola Q i X za koje bi istovremeno važilo:

Q BELOW X ∧ Q REDUCED_BY X

S obzirom da je desna strana 4. smene sufiks desne strane prve smene, proverom ustanovljavamo da ne važi

<S> BELOW <S2>

što znači da je ispunjen i uslov →, čime smo konačno potvrdili da se radi o gramatici slabog prvenstva.

b)

Na osnovu dobijenih relacija popunjavamo kontrolnu tabelu (Sl. 4.1.10) prema pravilima iz prethodnog zadatka.

	a	b	⊣
<S>	REJECT	SHIFT	IDENTIFY–0
<S1>	REJECT	SHIFT	REJECT
<S2>	REJECT	IDENTIFY–1	IDENTIFY–1
a	SHIFT	IDENTIFY–2	REJECT
b	REJECT	IDENTIFY–3	IDENTIFY–3
∇	SHIFT	REJECT	REJECT

Sl. 4.1.10: Kontrolna tabela parsera potisni–identifikuj

```
IDENTIFY–0:  if (vrh steka = ∇<S>)
                 then ACCEPT
                 else REJECT
             end if

IDENTIFY–1:  if (vrh steka = <S1><S2>)
                 then REDUCE(2)
                 else REJECT
             end if

IDENTIFY–2:  REDUCE(3)

IDENTIFY–3:  if (vrh steka = <S1>b)
                 then REDUCE(4)
             else if (vrh steka = <S>b)
                 then REDUCE(1)
             else REJECT
             end if
```

Zadatak 4.1.5

Data je sledeća gramatika sa startnim simbolom <S>:

1. <S> → a <A>
2. <S> → b <B>
3. <A> → <C> <A> 1
4. <A> → <C> 1
5. <B> → <D> <B> <E> 1
6. <B> → <D> <E> 1
7. <C> → 0
8. <D> → 0
9. <E> → 1

Konstruisati prepoznavač na osnovu jednostavne mešovite strategije prvenstva.

Analiza problema

Gramatika pripada klasi gramatika jednostavne mešovite strategije prvenstva ako i samo ako:

← Nema konflikata potisni-identifikuj.

↑ Za bilo koje dve smene sa istom desnom stranom:

<A> → α

<B> → α

ni za jedan simbol steka X nije istovremeno ispunjeno X BELOW <A> i X BELOW <B>.

→ Ako imamo smene <A>→αBβ i <C>→ β ne sme da važi B BELOW <C>.

↓ Ne sme da važi <S> $\overset{+}{\Rightarrow}$ <S>, gde je <S> startni neterminal.

U odnosu na gramatike slabog prvenstva, razlika je u uslovu ②. Klasa gramatika jednostavne mešovite strategije prvenstva je nadskup klase gramatika slabog prvenstva, pošto je dozvoljena i pojava smena sa istom desnom stranom. Pri radu parsera, pri prepoznavanju smena sa istom desnom stranom u okviru IDENTIFY rutine, potrebno je ispitati i stek simbol neposredno ispod desne strane smene. U takvoj situaciji, uslov ② obezbeđuje jednoznačno određivanje smene.

Rešenje

Za datu gramatiku, FIRST i FOLLOW skupovi neterminala su dati sa:

FIRST(<S>) = {<S>, a, b}	FOLLOW(<S>) = {⊣}
FIRST(<A>) = {<A>, <C>, 0}	FOLLOW(<A>) = {1, ⊣}
FIRST(<B>) = {<B>, <D>, 0}	FOLLOW(<B>) = {1, ⊣}
FIRST(<C>) = {<C>, 0}	FOLLOW(<C>) = {1,0}
FIRST(<D>) = {<D>, 0}	FOLLOW(<D>) = {1,0}
FIRST(<E>) = {<E>, 1}	FOLLOW(<E>) = {1}

Relacija BELOW prikazana je na Sl. 4.1.11, relacija REDUCED_BY na Sl. 4.1.12, a kontrolna tabela parsera, popunjena po pravilima koja važe i za gramatike slabog prvenstva, prikazana je na Sl. 4.1.13. Prazni ulazi tabele predstavljaju akciju REJECT.

	<S>	<A>	<B>	<C>	<D>	<E>	a	b	0	1
∇	1						1	1		
<S>										
<A>										1
<B>						1				1
<C>		1		1					1	1
<D>			1		1	1			1	1
<E>										1
a		1		1					1	
b			1		1				1	
0										
1										

Sl. 4.1.11: Relacija BELOW

	a	b	0	1	⊣
<S>					1

<A>					1
<B>					1
<C>					
<D>					
<E>					
a					
b					
0			1	1	
1				1	1

Sl. 4.1.12: Relacija REDUCED-BY

	a	b	0	1	⊣
∇	SHIFT	SHIFT			
<S>					IDENTIFY-3
<A>				SHIFT	IDENTIFY-4
<B>				SHIFT	IDENTIFY-5
<C>			SHIFT	SHIFT	
<D>			SHIFT	SHIFT	
<E>				SHIFT	
a			SHIFT		
b			SHIFT		
0			IDENTIFY-1	IDENTIFY-1	
1				IDENTIFY-2	IDENTIFY-2

Sl. 4.1.13: Kontrolna tabela

Zadata gramatika nije bezsufiksna. Desna strana 9. smene predstavlja sufiks desnih strana 3, 4, 5 i 6. smene. Utvrđujemo da ne važi:

<A> BELOW <E> <C> BELOW <E> <E> BELOW <E>

pa je moguće u okviru rutine IDENTIFY-2 prvo vršiti ispitivanja steka za smene 3, 4, 5 i 6. (međusobni redosled ispitivanja nije bitan) pa tek potom vršiti prepoznavanje smene 9.

Drugi, novi problem su 7. i 8. smena koje imaju iste desne strane. Da bismo znali kada 0 na vrhu steka treba svesti na <C>, a kada na <D>, treba koristiti relaciju BELOW. Važi da je:

<C> BELOW <C> b BELOW <D>

a BELOW <C> <D> BELOW <D>

odnosno, ne postoji simbol X tako da istovremeno važi X BELOW <C> i X BELOW <D>. Prema tome, gramatika može da se parsira jednostavnom mešovitom strategijom, što znači da se, u okviru IDENTIFY–1 rutine, ispitivanjem steka ispod 0, to jest desne strane 7. i 8. smene, može odrediti da li se vrši redukcija 7. smene (kada je ispod nule <C> ili a) ili redukcija 8. smene (kada je ispod nule <D> ili b).

```
IDENTIFY-1:
        if (vrh steka = <C>0)
                then REDUCE(7)
        else if (vrh steka = a0)
                then REDUCE(7)
        else if (vrh steka = <D>0)
                then REDUCE(8)
        else if (vrh steka = b0)
                then REDUCE(8)
        else REJECT
        end if

IDENTIFY-2:
        if (vrh steka = <C><A>1)
                then REDUCE(3)
        else if (vrh steka = <C>1)
                then REDUCE(4)
        else if (vrh steka = <D><B><E>1)
                then REDUCE(5)
        else if (vrh steka = <D><E>1)
                then REDUCE(6)
        else REDUCE(9)
        end if

IDENTIFY-3:
        if (vrh steka = ∇<S>)
                then ACCEPT
                else REJECT
        end if

IDENTIFY-4:
        if (vrh steka = a<A>)
                then REDUCE(1)
                else REJECT
        end if

IDENTIFY-5:
        if (vrh steka = b<B>)
                then REDUCE(2)
                else REJECT
        end if
```

Zadatak 4.1.6

Kod nekih gramatika koje nemaju konflikt potisni-identifikuj ali nisu ni gramatike jednostavne mešovite strategije prvenstva, identifikacione rutine se mogu projektovati tako što koriste sledeći ulaz u izboru smene ručke. Za sledeću gramatiku projektovati parser potisni-identifikuj koristeći ovu mogućnost.

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

4. <B> → a

Analiza problema

U prethodnom zadatku definisana su četiri uslova koje gramatika treba da zadovolji da bi pripadala klasi gramatika jednostavne mešovite strategije prvenstva.

Uslovi 2. i 3. treba da utvrde da li postoji konflikt tipa svedi-svedi. Uslov 2. je karakterističan za gramatiku jednostavne mešovite strategije prvenstva, a 3. za sufiksnu gramatiku. Da bismo znali da li je data gramatika jednostavne mešovite strategije prvenstva, potrebno je da je ispitamo u odnosu na sva navedena četiri uslova, to znači da moramo odrediti skupove FIRST i FOLLOW, relacije BELOW i REDUCED-BY i kontrolnu tabelu.

Rešenje

Skupovi FIRST su:

FIRST (<S>) = {<S>, <A>, <B>, a}

FIRST (<A>) = {<A>, a}

FIRST (<B>) = {<B>, a}

FIRST (a) = {a}

FIRST (c) = {c}

FIRST (d) = {d}

Skupovi FOLLOW su:

FOLLOW (<S>) = {⊣ }

FOLLOW (<A>) = {c}

FOLLOW (<B>) = {d}

Polazeći od skupova FIRST može se lako izračunati relacija BELOW na osnovu definicije (Sl. 4.1.14).

	a	c	d	<S>	<A>	<B>
a						
c						
d						
<S>						
<A>		1				
<B>			1			
∇	1			1	1	1

Sl. 4.1.14: Proširena relacija BELOW

Iz ove relacije se odmah vidi da drugi uslov nije ispunjen jer imamo da je

$$\nabla \text{ BELOW } <A> \wedge \nabla \text{ BELOW } <B>$$

što znači da postoji konflikt svedi–svedi.

Lako je takođe utvrditi da su 3. i 4. uslov ispunjeni jer nema smena oblika datog u trećem uslovu i nema sekvencijalnog izvođenja koje dovodi do cikličnog zatvaranja. Ostaje da se još proveri da li postoji konflikt tipa potisni-identifikuj. Da bismo to utvrdili, potrebno je konstruisati kontrolnu tabelu, a njeno definisanje zahteva formiranje proširenih relacija BELOW i REDUCED-BY. Proširena relacija BELOW već je izračunata.

Proširena relacija REDUCED-BY se dobija uvođenjem kolone za marker kraja ulazne sekvence i primenu relacije:

$$X \text{ IS-DIRECT-END-OF}^{*} <S>$$

za parove (X, ⊣), gde je <S> startni simbol, na osnovu relacije REDUCED-BY datu sa:

$$\text{IS-DIRECTED-END-OF}^{+} \leq$$

gde ≤ označava osnovnu relaciju BELOW, koja se dobija od proširene kada joj se ukloni vrsta za marker dna steka ∇. Relacija IS-DIRECT-END-OF za zadatu gramatiku prikazana je na Sl. 4.1.5. U ovom slučaju relacija IS-DIRECT-END-OF^{+} poklapa se sa relacijom IS-DIRECT-END-OF.

	a	c	d	<S>	<A>	<B>
a					1	1
c				1		
d				1		
<S>						
<A>						
<B>						

Sl. 4.1.15: Relacija IS-DIRECT-END-OF

Relacija REDUCED_BY prikazana je na Sl. 4.1.6, a njena proširena varijanta na Sl. 4.1.7.

	a	c	d	<S>	<A>	<B>
a		1	1			
c						
d						
<S>						

<A>						
<B>						

Sl. 4.1.16: Relacija REDUCED-BY

	a	c	d	<S>	<A>	<B>	⊣
a		1	1				
c							1
d							1
<S>							1
<A>							
<B>							

Sl. 4.1.17: Proširena relacija REDUCED_BY

Na osnovu proširenih relacija BELOW i REDUCED-BY dobija se kontrolna tabela prikazana na Sl. 4.1.18, pri čemu su u vrsti a razdvojene IDENTIFY akcije za prepoznavanje 3., odnosno 4. smene na osnovu tekućeg ulaznog simbola. Smena 3. se svodi kada tekući ulazni simbol pripada skupu FOLLOW(<A>), jer je <A> na levoj strani 3. smene. Na sličan način, smena 4. se svodi kada tekući ulazni simbol pripada skupu FOLLOW(<B>), s obzirom da je <B> na levoj strani 4. smene.

	a	c	d	⊣
S	REJECT	REJECT	REJECT	IDENTIFY–1
A	REJECT	SHIFT	REJECT	REJECT
B	REJECT	REJECT	SHIFT	REJECT
a	REJECT	IDENTIFY–2	IDENTIFY–3	REJECT
c	REJECT	REJECT	REJECT	IDENTIFY–4
d	REJECT	REJECT	REJECT	IDENTIFY–5
∇	SHIFT	REJECT	REJECT	REJECT

Sl. 4.1.18: Kontrolna tabela parsera potisni–identifikuj

SHIFT : PUSH (tekući_ulazni_simbol), ADVANCE

```
IDENTIFY1:   if (vrh steka = ∇<S>)
                then ACCEPT
                else REJECT
             end if

IDENTIFY4:   if (vrh steka = <A>c)
                then REDUCE(1)
                else REJECT
```

```
IDENTIFY2:  if (vrh steka = a)
                then REDUCE(3)
                else REJECT
            end if
IDENTIFY3:  if (vrh steka = a)
                then REDUCE(4)
                else REJECT
            end if
            end if
IDENTIFY5:  if (vrh steka = <B>d)
                then REDUCE(2)
                else REJECT
            end if
```

Posmatrajući dobijenu kontrolnu tabelu dolazimo do interesantnih zaključaka. Ne samo da ne postoji konflikt potisni-identifikuj, već ne postoji ni konflikt svedi-svedi koji sugeriše relacija BELOW, što znači da je dobijena ispravna kontrolna tabela za datu gramatiku. To je ostvareno posmatranjem sledećeg simbola pre izvođenja operacije svođenja.

Diskusija

Ovaj pojedinačan slučaj gramatike koja nema konflikt potisni-identifikuj ali nije tipa jednostavne mešovite strategije prvenstva (uslov 2. nije ispunjen) moguće je podići na nivo opšteg principa. Dakle, iako relacija BELOW sugeriše konflikt svedi-svedi, ne treba odmah prestati sa traženjem rešenja za kontrolnu tabelu koja važi za gramatike jednostavne mešovite strategije prvenstva. Za smeštaj IDENTIFY akcija moguće je koristiti tekući ulazni simbol pri odabiranju smene ručke. Tek tada se može otkriti da li se konflikt svedi-svedi otkriven relacijom BELOW proširio i na kontrolnu tabelu. Da bismo to otkrili ranije te izbegli nepotrebno izračunavanje kontrolne tabele, potrebno je uraditi sledeće:

Ako se posle izračunavanja relacije BELOW utvrdi neispunjavanje uslova 2, onda za smene tipa

$$<A> \rightarrow \alpha$$

$$<B> \rightarrow \alpha$$

treba izračunati FOLLOW (<A>) i FOLLOW (<B>). Ako je ispunjen uslov

$$\text{FOLLOW}(<A>) \cap \text{FOLLOW}(<B>) \neq \varnothing$$

znači da konflikt i dalje postoji i da treba odustati od popunjavanja tabele. Ako navedeni uslov nije ispunjen, to znači da ne postoji ulazni simbol x takav da je

$$x \in \text{FOLLOW}(<A>) \wedge x \in \text{FOLLOW}(<B>)$$

čime se neutrališe konflikt svedi-svedi otkriven relacijom BELOW. U tom slučaju, pravila popunjavanja kontrolne tabele iz prethodnog zadatka modifikujemo na sledeći način:

U vrsti tabele gde se smeštaju akcije za identifikaciju navedenih smena, posebna IDENTIFY akcija koja svodi smenu $<A> \rightarrow \alpha$ smešta se u kolone tabele koje odgovaraju ulaznim simbolima iz skupa FOLLOW(<A>), a druga, različita IDENTIFY akcija, za svođenje smene $<B> \rightarrow \alpha$, u kolone koje odgovaraju ulaznim simbolima iz skupa FOLLOW(<B>).

Zadatak 4.1.7

Data gramatika opisuje izraze u kojima se upotrebljavaju binarni operatori OP1, OP2 i unarni operator OP3 pri čemu su svi operatori levo asocijativni, i OP1 i OP2 imaju isti prioritet dok OP3 ima viši prioritet.

1. <E> → <E> OP1 <E>
2. <E> → <E> OP2 <E>
3. <E> → OP3 <E>
4. <E> → c

a) Odrediti BELOW i REDUCED_BY za datu gramatiku i preko njih pokazati da gramatika poseduje konflikte potisni–identifikuj.

b) Transformisati gramatiku radi eliminacije konflikata i napraviti parser potisni-identifikuj za transformisanu gramatiku.

Rešenje

a)

Relacije BELOW i REDUCED_BY za zadatu gramatiku prikazane su na Sl. 4.1.19 i Sl. 4.1.20, respektivno.

	<E>	OP1	OP2	OP3	c
∇	1			1	1
<E>		1	1		
OP1	1			1	1
OP2	1			1	1
OP3	1			1	1
c					

Sl. 4.1.19: Proširena relacija BELOW

	OP1	OP2	OP3	c	⊣
<E>	1	1			1
OP1					
OP2					
OP3					
c	1	1			1

Sl. 4.1.20: Proširena relacija REDUCED_BY

Iz ovih tabela se vidi da postoje dva konflikta tipa potisni-identifikuj s obzirom da važi:

<E> BELOW OP1 i <E> REDUCED_BY OP1

<E> BELOW OP2 i <E> REDUCED_BY OP2

b)

Ovi konflikti posledica su dvosmislenosti gramatike, na primer, sentenca c OP1 c OP1 c poseduje dva različita stabla izvođenja. Dvosmislenost eliminišemo transformacijom date gramatike uvođenjem novog neterminala <T>:

1. <E> $\rightarrow$ <E> OP1 <T>
2. <E> $\rightarrow$ <E> OP2 <T>
3. <E> $\rightarrow$ <T>
4. <T> $\rightarrow$ OP3 <T>
5. <T> $\rightarrow$ c

Neterminal <T> opisuje izraze u kojima se pojavljuju samo operator OP3 i konstanta c. Novodobijena gramatika opisuje isti jezik kao i zadata, odnosno, u notaciji regularnih izraza:

$OP3^{*}\ c\ ((OP1|OP2)\ OP3^{*}\ c)^{+}$

Radi se o listi jednog ili više članova razdvojenih operatorima OP1 ili OP2, pri čemu je član konstanta c na koju je nula ili više puta primenjen operator OP3.

Za ovu gramatiku imamo da su:

FIRST (<E>) = {<E>, <T>, OP3, c}

FIRST (<T>) = FIRST (E) = {<T>, OP3, c}

FOLLOW (<E>) = {OP1, OP2, ⊣}

FOLLOW (<T>) = FOLLOW (<E>) = {OP1, OP2, ⊣}

Proširene relacije BELOW i REDUCED_BY prikazane su na Sl. 4.1.21(a) i (b), respektivno. Uočavamo da nema konflikata tipa potisni-identifikuj. Kontrolna tabela parsera prikazana je na Sl. 4.1.22.

	<E>	<T>	OP1	OP2	OP3	c
<E>			1	1		
<T>						
OP1		1			1	1
OP2		1			1	1
OP3		1			1	1
c						
∇	1	1			1	1

(a) Proširena relacija BELOW

	OP1	OP2	OP3	c	⊣
<E>					1
<T>	1	1			1
OP1					
OP2					
OP3					
c	1	1			1

(b) Proširena relacija REDUCED_BY

Sl. 4.1.21

	OP1	OP2	OP3	c	⊣
<E>	SHIFT	SHIFT			IDENTIFY–1
<T>	IDENTIFY–2	IDENTIFY–2			IDENTIFY–2
OP1			SHIFT	SHIFT	
OP2			SHIFT	SHIFT	
OP3			SHIFT	SHIFT	
c	IDENTIFY–3	IDENTIFY–3			IDENTIFY–3
∇			SHIFT	SHIFT	

Sl. 4.1.22: Kontrolna tabela

U transformisanoj gramatici, desna strana 3. smene je sufiks desnih strana 1., 2. i 4. smene. Utvrđujemo da ne važi:

OP1 BELOW <E> OP2 BELOW <E> OP3 BELOW <E>

tako da, uz ispunjene ostale definicione uslove, gramatika spada u klasu gramatika slabog prvenstva. IDENTIFY i REDUCE akcije imaju sledeći izgled:

```
IDENTIFY–1: if vrh steka = ∇ <E>
                then ACCEPT
                else REJECT
            end if

IDENTIFY–2: if vrh steka = <E> OP1 <T>
                then REDUCE(1)
            else if vrh steka = <E> OP2 <T>
                then REDUCE(2)
            else if vrh steka = OP3 <T>
                then REDUCE(4)
            else REDUCE(3)
            end if

IDENTIFY–3: REDUCE(5)
```

```
REDUCE(1):
POP, POP, POP, PUSH(<E>)

REDUCE(2):
POP, POP, POP, PUSH(<E>)

REDUCE(3):
POP, PUSH(<E>)

REDUCE(4):
POP, POP, PUSH(<T>)

REDUCE(5):
POP, PUSH(<T>)
```

Zadatak 4.1.8

Sledeća gramatika odgovara jeziku kod koga se i pozivi funkcija i obraćanja elementima nizova pišu običnim zagradama.

1. <stmt> → <expr> := <expr>
2. <expr> → <fun_call>
3. <expr> → <arr_ref>
4. <expr> → id
5. <fun_call> → id (<parm_list>)
6. <parm_list> → <parm_list> , <expr>
7. <parm_list> → <expr>
8. <arr_ref> → id (<expr_list>)
9. <expr_list> → <expr_list> , <expr>
10. <expr_list> → <expr>

a) Pokazati da je gramatika dvosmislena.

b) Dvosmislenost se može ukloniti uvođenjem posebnog terminala *procid* za imena funkcija. Za ovaj slučaj detaljno objasniti rad leksičkog analizatora.

c) Za transformisanu gramatiku u skladu sa tačkom b) prikazati rad parsera potisni-identifikuj za ulaz:

id := id (id, id)

Nije potrebno konstruisati parser.

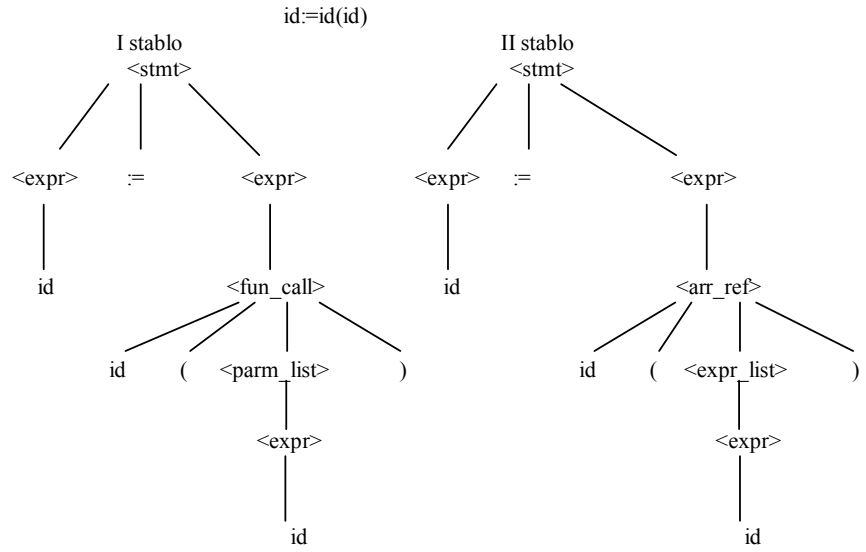
Rešenje

a)

Gramatika je dvosmislena jer ima dva različita stabla izvođenja za isti niz ulaznih simbola:

id := id (id)

kao što se vidi sa slike Sl. 4.1.23. Levo stablo odgovara semantici poziva procedure u izrazu sa desne strane iskaza dodele, a desno stablo odgovara obraćanju elementu niza.



Sl. 4.1.23

b)

U ovom slučaju u 5. smeni terminal id zamenjen je terminalom procid čime se eliminiše dvosmislenost. Gramatika sada ima sledeći izgled:

1. <stmt> → <expr> := <expr>
2. <expr> → <fun_call>
3. <expr> → <arr_ref>
4. <expr> → id
5. <fun_call> → procid (<parm_list>)
6. <parm_list> → <parm_list> , <expr>
7. <parm_list> → <expr>
8. <arr_ref> → id (<expr_list>)
9. <expr_list> → <expr_list> , <expr>
10. <expr_list> → <expr>

Uloga leksičkog analizatora je da u nizu znakova koji predstavlja izvorni program identifikuje pojedine leksičke jedinice (lekseme) i parseru, za svaku leksemu, dostavi odgovarajući interni leksički kod (engl. *token*). Ovi kodovi odgovaraju pojedinim terminalima.

U slučaju identifikatora promenljivih, odnosno procedura, leksički analizator po izdvajanju lekseme konsultuje tabelu simbola. S obzirom da deklaracije promenljivih, odnosno procedura prethode njihovoj upotrebi, leksički analizator na osnovu informacije iz tabele simbola prosleđuje odgovarajući leksički kod parseru.

c)

U nastavku je data obrada ulazne sekvence od strane parsera. REDUCE akcije selektuju se u okviru IDENTIFY rutine.

stek	ulazni niz	akcije parsera
∇	id:=id(id,id) ⊣	SHIFT
∇id	:=id(id,id) ⊣	REDUCE(4)
∇<expr>	:=id(id,id) ⊣	SHIFT
∇<expr>:=	id(id,id) ⊣	SHIFT
∇<expr>:=id	(id,id) ⊣	SHIFT
∇<expr>:=id(	id,id) ⊣	SHIFT
∇<expr>:=id(id	,id) ⊣	REDUCE(4)
∇<expr>:=id(<expr>	,id) ⊣	REDUCE(7)
∇<expr>:=id(<parm_exp_list>	,id) ⊣	SHIFT
∇<expr>:=id(<parm_exp_list>,	id) ⊣	SHIFT
∇<expr>:=id(<parm_exp_list>,id	) ⊣	REDUCE(4)
∇<expr>:=id(<parm_exp_list>,<expr>	) ⊣	REDUCE(6)
∇<expr>:=id(<parm_exp_list>	) ⊣	SHIFT
∇<expr>:=id(<parm_exp_list>)	⊣	REDUCE(8)
∇<expr>:=<arr_ref>	⊣	REDUCE(3)
∇<expr>:=<expr>	⊣	REDUCE(1)
∇<stmt>	⊣	ACCEPT

Zadatak 4.1.9

Pretpostaviti da je bezkontekstna gramatika takva da se svaka smena završava različitim terminalnim simbolom i da se ti terminalni simboli ne javljaju na drugim mestima u gramatici. Pokazati da se takva gramatika uvek može parsirati metodom potisni-identifikuj.

Rešenje

Dokazaćemo da posmatrana gramatika spada u klasu bezsufiksnih SI gramatika. Iz postavke zadatka sledi da se ne može naći smena čija desna strana predstavlja sufiks desne strane neke druge smene (s obzirom da se svaka smena završava unikatnim terminalom).

Da bismo dokazali da gramatika nema konflikata potisni-identifikuj, pođimo od obrnute pretpostavke: postoje gramatički simboli X i Y takvi da istovremeno važi:

X BELOW Y i X REDUCED_BY Y

Pošto važi X REDUCED_BY Y sledi da se X mora nalaziti na kraju desne strane neke smene. Iz uslova zadatka sledi da je X terminalni simbol i da se ne pojavljuje na drugim mestima u gramatici. Iz ovoga neposredno sledi da ne postoji simbol Y tako da važi X BELOW Y. Dakle, polazna pretpostavka nije tačna, odnosno gramatika ne poseduje konflikte potisni-identifikuj.

Takođe, ne postoji konflikt svedi–svedi jer se svaka smena završava različitim terminalnim simbolom. Radi se dakle o bezsufiksnoj SI gramatici za kakvu je uvek moguće konstruisati potisni-identifikuj parser.

Zadatak 4.1.10

Pronaći jednu gramatiku sa startnim simbolom <S> koja ima sledeću kontrolnu tabelu po metodu parsiranja potisni-identifikuj. Sračunati relacije BELOW i REDUCED_BY za nađenu gramatiku.

	0	1	⊣
<S>	REJECT	SHIFT	IDENTIFY
<A>	SHIFT	IDENTIFY	IDENTIFY
0	IDENTIFY	IDENTIFY	IDENTIFY
1	SHIFT	SHIFT	REJECT
∇	SHIFT	REJECT	REJECT

Sl. 4.1.24: Kontrolna tabela parsera potisni–identifikuj

Analiza problema

Na osnovu date kontrolne tabele, pod pretpostavkom da je popunjena na standardan način, moguće je odrediti skup gramatičkih simbola, relaciju REDUCED_BY i delimično relaciju

BELOW (bez kolona koje odgovaraju neterminalima). Na osnovu toga moguće je odrediti neke zavisnosti među gramatičkim simbolima i kombinatornim metodom odrediti gramatiku koja zadovoljava nađene zavisnosti. Zadatak u opštem slučaju nema jedinstveno rešenje, dakle moguće je odrediti veći broj gramatika koje zadovoljavaju uslove zadatka.

Rešenje

Skup gramatičkih simbola je {<S>, <A>, 0, 1}. Pozicije akcija SHIFT u kontrolnoj tabeli (Sl. 4.1.24) određuju relaciju BELOW, bez kolona za neterminalne simbole, kao na Sl. 4.1.25(a). Pozicije akcija IDENTIFY određuju relaciju REDUCED_BY, kao što je prikazano na Sl. 4.1.25(b).

	0	1
<S>		1
<A>	1	
0		
1	1	1
∇	1	

(a) Parcijalna relacija BELOW

	0	1	⊣
<S>			1
<A>		1	1
0	1	1	1
1			

(b) Relacija REDUCED_BY

Sl. 4.1.25

Na osnovu relacija može se zaključiti sledeće:

1. postoje pojavljivanja simbola <S>, <A> i 1 na desnim stranama smena na pozicijama koje nisu krajnje desne, pri tome <S> se pojavljuje neposredno pre 1 (ili neterminala u čijem se FIRST skupu nalazi 1), <A> neposredno pre 0 (ili neterminala u čijem se FIRST skupu nalazi 0), a 1 neposredno pre 0 ili 1 ili odgovarajućih neterminala; sve ovo se zaključuje na osnovu relacije BELOW.
2. FIRST(<S>) sadrži od terminala samo 0; ovo sledi iz poslednje vrste relacije BELOW.
3. na osnovu relacije REDUCED_BY, desne strane smena završavaju se samo simbolima <A> i 0. Prema definiciji relacije REDUCED_BY, vrste <A> i 0 popunjene su u skladu sa FOLLOW skupovima neterminala koji se nalaze na levim stranama smena koje se završavaju sa <A> i 0, respektivno. FOLLOW skup jednog od neterminala sadrži 1 i ⊣, a FOLLOW skup drugog 0, 1 i ⊣. S obzirom da smo već zaključili da se <A> pojavljuje neposredno ispred 0, sledi:

   FOLLOW(<S>) = {1, ⊣}

   FOLLOW(<A>) = {0, 1, ⊣}

Pošto za svaki neterminal mora da postoji bar po jedna smena, probajmo da nađemo gramatiku sa dve smene koja zadovoljava uslove 2. i 3. Jedna od smena mora da se završi simbolom <A>, druga simbolom 0 (uslov 3.). Prema uslovu 2., sledi da nulom mora da počne desna strana smene, a to mora biti smena za <A>, inače bi gramatika imala nedostižnu smenu, ili bi postojala

pojava 0 koja nije striktno na kraju desne strane smene. Dalje sledi da smena za <S> mora da počne sa <A> da bi bio zadovoljen uslov 3. Smene dakle imaju oblik:

1. <S> → <A><A>
2. <A> → 0

Ostaje da preciziramo smenu koja zadovoljava uslov 1. U ovoj smeni neophodna je jedna pojava simbola <S> i dve pojave simbola 1 prema uslovu 1, iza kojih sledi <A>. Gramatika ima konačni izgled:

1. <S> → <S>11<A>
2. <S> → <A><A>
3. <A> → 0

Za ovu gramatiku imamo:

FIRST (<S>) = {<S>, <A>, 0}

FIRST (<A>) = {<A>, 0}

FOLLOW (<S>) = {1, ⊣}

FOLLOW (<A>) = { t | t ∈ V_t ∧ t ∈ FIRST(<A>) ∪ FOLLOW(<S>) }

= {0, 1, ⊣}

Na osnovu ovoga nalazimo relacije BELOW i REDUCED_BY, prikazane na Sl. 4.1.26.

	<S>	<A>	0	1
<S>				1
<A>		1	1	
0				
1		1	1	1
∇	1	1	1	

(a) Proširena relacija BELOW

	0	1	⊣
<S>			1
<A>		1	1
0	1	1	1
1			

(b) Proširena relacija REDUCED_BY

Sl. 4.1.26

Vidimo da se nađene relacije poklapaju sa onima određenim na osnovu tabele, prema tome rešenje je ispravno uz konstataciju da gramatika pripada klasi bezsufiksnih SI gramatika.

Diskusija

Čitaocu se preporučuje da nađe drugačiju gramatiku koja će zadovoljiti uslove zadatka.

4.1.2. Metod potiskivanja i svođenja

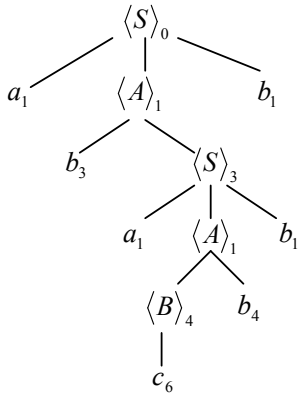
Zadatak 4.1.11

Data je gramatika sa startnim simbolom <S>. Nacrtati izgled steka kada parser potisni-svedi (engl. *SHIFT–REDUCE*) procesira niz znakova *abacbbb*.

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

Rešenje

Preko stabla izvođenja (Sl. 4.1.27) može se pokazati da gramatika opisuje niz *abacbbb*.



Sl. 4.1.27: Stablo izvođenja za niz *abacbbb*

U stablu izvođenja svaki simbol je dodatno obeležen indeksom primenjene smene. Na ovaj način jednoznačno opisujemo pojedina dešavanja (pojave) simbola u gramatičkim smenama. Parser prepoznaje sekvencu abacbbb na sledeći način:

Stek	Ulazni niz	Operacija mašine
1. ∇	abacbbb⊣	SHIFT
2. ∇a_1	bacbbb⊣	SHIFT
3. ∇a_1b_3	acbbb⊣	SHIFT
4. ∇$a_1b_3\,a_1$	cbbb⊣	SHIFT
5. ∇ $a_1b_3a_1c_6$	bbb⊣	REDUCE(6)
6. ∇$a_1b_3a_1$<B>$_4$	bbb⊣	SHIFT
7. ∇$a_1b_3a_1$<B>$_4b_4$	bb⊣	REDUCE(4)
8. ∇$a_1b_3a_1$<A>$_1$	bb⊣	SHIFT

9. $\nabla a_1 b_3 a_1 <A>_1 b_1$	$b \dashv$	REDUCE(1)
10. $\nabla a_1 b_3 <S>_3$	$b \dashv$	REDUCE(3)
11. $\nabla a_1 <A>_1$	$b \dashv$	SHIFT
12. $\nabla a_1 <A>_1 b_1$	$\dashv$	REDUCE(1)
13. $\nabla <S>_0$	$\dashv$	ACCEPT

Diskusija

Kao što se može videti, rad parsera sličan je radu potisni-identifikujparsera s tom razlikom što simbolima steka odgovaraju pojedina gramatička dešavanja, čime je omogućeno da se odluka o sledećoj akciji parsera (SHIFT, REDUCE, ACCEPT ili REJECT) donese isključivo na osnovu vršnog simbola steka i (eventualno) tekućeg ulaznog simbola. Ovime je ukinuta potreba za IDENTIFY akcijama.

Akcije SHIFT i REDUCE($<X> \rightarrow A_1A_2...A_n$) imaju sličnu semantiku kao kod metoda potisni-identifikuj:

SHIFT: PUSHT(*tekući ulazni simbol*), ADVANCE

REDUCE($<X> \rightarrow A_1A_2...A_n$): n puta akcija POP, PUSHT(<X>)

Jedina razlika je pojava akcije PUSHT umesto PUSH, jer se na stek ne stavljaju neposredno gramatički simboli nego njihova dešavanja. Akcija PUSHT(X) konsultuje posebnu, takozvanu potisnu tabelu, da na osnovu vršnog simbola steka i simbola X odredi koje gramatičko dešavanje simbola X treba potisnuti na stek.

Napomena: takozvano startno dešavanje $<S>_0$ i odgovaraći simbol steka, koji se javlja u poslednjem koraku rada parsera, ne pojavljuje se u gramatičkim smenama. Ovo dešavanje startnog neterminala i odgovarajući simbol steka posebno se dodaju da označe konfiguraciju steka kada je startni neterminal neposredno iznad dna steka.

Ukoliko se na desnoj strani neke gramatičke smene isti simbol pojavljuje više puta koristi se dvostruki indeks, redni broj smene i redni broj posmatranog dešavanja u okviru smene.

U opštem slučaju, simboli steka odgovaraju skupovima gramatičkih dešavanja a ne samo pojedinačnim dešavanjima.

Na primer, ako bismo gramatici dodali smenu:

7. $< B > \rightarrow b\ c$

tada za datu ulaznu sekvencu ab..., kada treba potisnuti *b* na stek, ne zna se da li je to b_3 ili b_7 (iz 3. ili 7. smene). U tom slučaju postojao bi simbol steka b_x koji bi odgovarao skupu gramatičkih dešavanja $\{b_3, b_7\}$.

Zadatak 4.1.12

a) Za sledeću gramatiku sa startnim simbolom <S> izračunati skupove OFIRST za svako dešavanje gramatičkog simbola na desnoj strani smene.

b) Izračunati relaciju OBELOW.

c) Projektovati LR(0) prepoznavač.

1. <S> → *a*
2. <S> → (<S> <R>
3. <R> →, <S> <R>
4. <R> →)

Analiza problema

Skupovi OFIRST i relacija OBELOW odgovaraju FIRST skupovima i relaciji BELOW kod metoda potisni-identifikuj, s tom razlikom što su definisani za gramatička dešavanja.

Za proizvoljna gramatička dešavanja X_i i Y_j simbola X i Y respektivno, važi da je

$$X_i \in \text{OFIRST}(Y_j)$$

ako i samo ako važi jedan od sledećih uslova:

a) X_i je identično Y_j ili

b) X_i odgovara početnom simbolu neke sentencijalne forme izvedene iz Y bez upotrebe praznih smena, odnosno

$$Y \Rightarrow^* <L> \beta \Rightarrow X \alpha \beta$$

pri čemu je X_i krajnje levo dešavanje u smeni $<L> \rightarrow X \alpha$.

Za proizvoljno gramatičko dešavanje (ili marker dna steka) A i proizvoljno gramatičko dešavanje Y_j važi da je:

$$A \text{ OBELOW } Y_j$$

ako i samo ako važi jedan od sledećih uslova:

a) A je gramatičko dešavanje, njega u smeni neposredno sledi dešavanje Z_k i $Y_j \in \text{OFIRST}(Z_k)$.

b) A je marker dna steka i $Y_j \in \text{OFIRST}(<S>_0)$ gde $<S>_0$ predstavlja startno dešavanje.

Radi računanja skupova OFIRST i relacije OBELOW uvode se neke pomoćne relacije. Skupovi OFIRST za gramatička dešavanja računaju se korišćenjem relacije O_BEGINS_DIRECTLY_WITH koja odgovara relaciji BEGINS_DIRECTLY_WITH, ali se definiše za gramatička dešavanja.

Za proizvoljna gramatička dešavanja X_i i Y_j simbola X i Y respektivno, važi da je:

$$X_i \text{ O\_BEGINS\_DIRECTLY\_WITH } Y_j$$

ako i samo ako postoji smena $X \rightarrow Y \beta$ i Y_j je krajnje levo dešavanje desne strane ove smene.

Za proizvoljna gramatička dešavanja X_i i Y_j važi da je $X_i \in \text{OFIRST}(Y_j)$ ako i samo ako Y_j O_BEGINS_DIRECTLY_WITH* X_i.

Sledi definicija relacije O_IS_FOLLOWED_DIRECTLY_BY koja se koristi u računanju relacije OBELOW.

Za proizvoljna gramatička dešavanja X_i i Y_j simbola X i Y respektivno, važi da je:

X_i O_IS_FOLLOWED_DIRECTLY_BY Y_j

ako i samo ako postoji smena $<Z> \rightarrow \alpha\ X\ Y\ \beta$, na čijoj desnoj strani Y_j neposredno sledi X_i.

Relacija OBELOW se sada računa tako što se relacionom proizvodu:

O_IS_FOLLOWED_DIRECTLY_BY · O_BEGINS_DIRECTLY_WITH*

dodaju svi parovi (∇, Z_k) za sva gramatička dešavanja Z_k za koja važi $<S>_0$ O_BEGINS_DIRECTLY_WITH* Z_k.

Rešenje

a)

Relacije O_BEGINS_DIRECTLY_WITH i O_BEGINS_DIRECTLY_WITH* za datu gramatiku prikazane su na Sl. 4.1.28.

Na osnovu ovih relacija računamo OFIRST skupove:

OFIRST $(a_1) = \{a_1\}$

OFIRST $((_2) = \{(_2\}$

OFIRST $(<S>_2) = \{<S>_2, a_1, (_2\}$

OFIRST $(<R>_2) = \{,_3, <R>_2,)_4\}$

OFIRST $(,_3) = \{,_3\}$

OFIRST $(<S>_3) = \{a_1, (_2, <S>_3\}$

OFIRST $(<R>3) = \{,_3,)_4, <R>_3\}$

OFIRST $( )_4) = \{)_4\}$

OFIRST $(<S>_0) = \{a_1, (_2, <S>_0\}$

	$<S>_0$	a_1	$(_2$	$<S>_2$	$<R>_2$	$,_3$	$<S>_3$	$<R>_3$	$)_4$
$<S>_0$	*	1	1						
a_1		*							
$(_2$			*						
$<S>_2$		1	1	*					
$<R>_2$					*	1			1
$,_3$						*			
$<S>_3$		1	1				*		
$<R>_3$						1		*	1
$)_4$									*

Sl. 4.1.28: Relacije O_BEGINS_DIRECTLY_WITH (samo 1) i
O_BEGINS_DIRECTLY_WITH* (1 i *)

b)

Relacije O_IS_FOLLOWED_DIRECTLY_BY i OBELOW prikazane su na Sl. 4.1.29 i Sl. 4.1.30, respektivno.

	$<S>_0$	a_1	$(_2$	$<S>_2$	$<R>_2$	$,_3$	$<S>_3$	$<R>_3$	$)_4$
$<S>_0$									
a_1									
$(_2$				1					
$<S>_2$					1				
$<R>_2$									
$,_3$							1		
$<S>_3$								1	
$<R>_3$									
$)_4$									

Sl. 4.1.29: Relacija O_IS_FOLLOWED_DIRECTLY_BY

	$<S>_0$	a_1	$(_2$	$<S>_2$	$<R>_2$	$,_3$	$<S>_3$	$<R>_3$	$)_4$
$<S>_0$									
a_1									
$(_2$		1	1	1					
$<S>_2$					1	1			1
$<R>_2$									
$,_3$		1	1				1		
$<S>_3$						1		1	1
$<R>_3$									
$)_4$									
∇	1	1	1						

Sl. 4.1.30: Relacija OBELOW

c)

Analiza problema

Na osnovu relacije OBELOW konstruišu se potisna i kontrolna tabela parsera potisni-svedi. Prvi korak u konstrukciji potisne tabele je konstrukcija karakterističnog konačnog automata – detektora ručki. Detektor ručki konstruiše se na sledeći način:

- Stanja automata odgovaraju pojedinim gramatičkim dešavanjima i markeru dna steka (njemu odgovara startno stanje automata).
- Ulaznu azbuku čine gramatički simboli.
- Pravilo popunjavanja tabele prelaza: iz stanja A postoji prelaz u stanje B pod ulazom Y, ako i samo ako stanje B odgovara nekom dešavanju Yj gramatičkog simbola Y i ispunjeno je A OBELOW B.

Ukoliko sva stanja koja odgovaraju krajnje desnim dešavanjima smena i dešavanju $<S>_0$ proglasimo za stanja prihvatanja, a ostala stanja za stanja odbijanja, automat će prihvatati prefikse koji se završavaju u ručki svih sentencijalnih formi iz krajnje desnog izvođenja proizvoljne gramatičke sentence.

U opštem slučaju dobija se nedeterministički konačni automat. Određivanjem ekvivalentnog determinističkog automata dobija se ujedno i potisna tabela parsera. Stanja automata odgovaraju simbolima steka parsera (zbog načina konstrukcije, pojedinim simbolima steka odgovara skup gramatičkih dešavanja). Prazni ulazi tabele automata popunjavaju se akcijom REJECT.

Kontrolna tabela parsera klase LR(0) ima vrste označene simbolima steka i kolone označene ulaznim simbolima (gramatičkim terminalima) i markerom kraja ulaza. Tabela se popunjava po vrstama primenom sledećih pravila (uzmimo da razmatramo simbol steka X kojem odgovara skup S gramatičkih dešavanja):

a) Ako S sadrži jedino startno dešavanje, sve kolone se popunjavaju akcijom REJECT, osim kolone ⊣ koja se popunjava akcijom ACCEPT.

b) Ako S sarži startno dešavanje i jedno ili više dešavanja koja nisu krajnje desna u smenama, sve kolone se popunjavaju akcijom SHIFT, osim kolone ⊣ koja se popunjava akcijom ACCEPT.

c) Ako S sadrži samo jedno dešavanje koje predstavlja krajnje desno dešavanje u i-toj smeni, tada se cela vrsta popunjava akcijom REDUCE(i).

d) Ako je X marker dna steka, ili S sadrži samo dešavanja koja nisu krajnje desna u smeni, sve kolone se popunjavaju akcijom SHIFT, osim kolone ⊣ koja se popunjava akcijom REJECT.

Ako neka vrsta ne potpada pod prethodna pravila, znači da gramatika nije u klasi LR(0) i da se za nju ne može konstruisati LR(0) parser. Pri konstrukciji kontrolne tabele mogu se dogoditi dve vrste konflikata:

1. Konflikt tipa potisni–svedi nastaje ukoliko pravila propisuju i akciju SHIFT i akciju REDUCE(i) za isti ulaz tabele.

2. Konflikt tipa svedi–svedi nastaje ukoliko pravila propisuju REDUCE akciju, ali za dve različite smene, za isti ulaz tabele.

Rešenje

Na osnovu relacije OBELOW pravimo tabelu prelaza detektora ručki (Sl. 4.1.31).

	a	(	,	)	<S>	<R>
∇	a_1	$(_2$			$<S>_0$	
$<S>_0$						
a_1						
$(_2$	a_1	$(_2$			$<S>_2$	
$<S>_2$			$,_3$	$)_4$		$<R>_2$
$<R>_2$						
$,_3$	a_1	$(_2$			$<S>_3$	
$<S>_3$			$,_3$	$)_4$		$<R>_3$
$<R>_3$						
$)_4$						

Sl. 4.1.31: Potisna tabela LR(0) parsera

S obzirom da je dobijeni automat deterministički, njegova tabela prelaza je istovremeno i potisna tabela parsera. Kotrolna tabela parsera prikazana je na Sl. 4.1.32.

	a	(	,	)	⊣
∇	SHIFT	SHIFT	SHIFT	SHIFT	REJECT
$<S>_0$	REJECT	REJECT	REJECT	REJECT	ACCEPT
a_1	REDUCE (1)	REDUCE (1)	REDUCE (1)	REDUCE (1)	REDUCE (1)
$(_2$	SHIFT	SHIFT	SHIFT	SHIFT	REJECT
$<S>_2$	SHIFT	SHIFT	SHIFT	SHIFT	REJECT
$<R>_2$	REDUCE (2)	REDUCE (2)	REDUCE (2)	REDUCE (2)	REDUCE (2)
$,_3$	SHIFT	SHIFT	SHIFT	SHIFT	REJECT
$<S>_3$	SHIFT	SHIFT	SHIFT	SHIFT	REJECT
$<R>_3$	REDUCE (3)	REDUCE (3)	REDUCE (3)	REDUCE (3)	REDUCE (3)
$)_4$	REDUCE (4)	REDUCE (4)	REDUCE (4)	REDUCE (4)	REDUCE (4)

Sl. 4.1.32: Kontrolna tabela LR(0) parsera

Startni stek: ∇

REDUCE(1): POP, PUSHT(<S>)

REDUCE(2): POP, POP, POP, PUSHT(<S>)

REDUCE(3): POP, POP, POP, PUSHT(<R>)

REDUCE(4): POP, PUSHT(<R>)

Diskusija

Razmotrimo primer koji ilustruje svojstvo detekcije ručki od strane automata sa Sl. 4.1.31; krajnje desno izvođenje sentence (a , a) iz startnog neterminala u skladu sa zadatom gramatikom je:

$$\underset{\uparrow}{\langle S\rangle} \Rightarrow_{rm} \underset{\uparrow}{(\ \langle S\rangle\ \langle R\rangle} \Rightarrow_{rm} (\ \langle S\rangle\ ,\ \langle S\rangle\ \underset{\uparrow}{\langle R\rangle} \Rightarrow_{rm} (\ \langle S\rangle\ ,\ \underset{\uparrow}{\langle S\rangle}\ ) \Rightarrow_{rm} (\ \langle S\rangle\ ,\ \underset{\uparrow}{a}\ ) \Rightarrow_{rm} (\ a\ ,\ a\ )$$

2 3 4 1 1

Posmatrajmo jednu od sentencijalnih formi koje se pojavljuju u ovom izvođenju, na primer sentencijalnu formu (<S> , a). Simbol a predstavlja ručku, to jest, pojavu desne strane 1. smene koja je poslednja primenjena u izvođenju posmatrane sentencijalne forme iz prethodne.

Prefiks sentencijale forme koji se završava ručkom je, prema tome,

(<S> , a

Ukoliko ovaj prefiks dovedemo na ulaz detektora ručki, automat će završiti rad u stanju a_1 koje je stanje prihvatanja. Sve druge prefikse sentencijalne forme (<S> , a):

(

(<S>

(<S>,

kao i kompletnu formu, automat odbija.

Zadatak 4.1.13

Projektovati SLR(1) parser za zadatu gramatiku sa startnim simbolom <S>.

1. <S> → a <A>
2. <S> → b <B>
3. <A> → 1 <A> 0
4. <A> → ε
5. <B> → 1 <B> 0 0
6. <B> → ε

Analiza problema

Potisna tabela SLR(1) parsera konstruiše se na isti način kao i kod LR(0) parsera, na osnovu determinističkog detektora ručki izvedenog iz relacije OBELOW. Pri konstrukciji relacije OBELOW zanemaruje se postojanje praznih smena u gramatici.

Pri konstrukciji kontrolne tabele SLR(1) tipa uzimaju se u obzir i FOLLOW skupovi gramatičkih neterminala. Vrste kontrolne tabele odgovaraju pojedinim simbolima steka određenim prilikom konstrukcije potisne tabele. Kolone tabele odgovaraju pojedinim ulaznim simbolima i markeru kraja ulazne sekvence. Ulaz kontrolne tabele u vrsti obeleženoj stek simbolom Q, kome odgovara skup dešavanja S, i koloni obeleženoj ulaznim simbolom z popunjava se primenom jednog od sledećih pravila:

a) Ako S sadrži startno dešavanje, a z je marker kraja, akcija je ACCEPT.

b) Ako z nije marker kraja, a ulaz potisne tabele u vrsti Q i koloni z nije REJECT, akcija je SHIFT.

c) Ako S sadrži krajnje desno dešavanje neke smene p sa neterminalom <X> na levoj strani i z $\in$ FOLLOW(<X>), akcija je REDUCE(p).

d) (Ukoliko gramatika sarži prazne smene) Ako je q prazna smena sa neterminalom <X> na levoj strani, važi da je z $\in$ FOLLOW(<X>) i ulaz potisne tabele u vrsti Q i koloni <X> nije REJECT, akcija je REDUCE(q).

e) Ukoliko neki od ulaza tabele ne potpada pod pravila od a) do d), popunjava se akcijom REJECT

Ukoliko pravila od a) do d) za neki od ulaza postavljaju konfliktne zahteve, gramatika ne pripada klasi SLR(1). Iz potisne tabele moguće je ukloniti vrste koje sadrže isključivo REJECT akcije, jer se takve vrste nikada ne konsultuju u akcijama iz kontrolne tabele.

Rešenje

Prvo računamo OFIRST skupove za gramatička dešavanja simbola kao da nema praznih smena:

OFIRST ($<S>_0$) = {$<S>_0$, a_1 , b_2}

OFIRST (a_1) = {a_1}

OFIRST ($<A>_1$) = {$<A>_1$, 1_3}

OFIRST (b_2) = {b_2}

OFIRST ($<B>_2$) = {$<B>_2$, 1_5 }

OFIRST (1_3) = {1_3}

OFIRST ($<A>_3$) = {$<A>_3$, 1_3}

OFIRST (0_3) = {0_3}

OFIRST (1_5) = {1_5}

OFIRST ($<B>_5$) = {$<B>_5$, 1_5}

OFIRST $(0_{51}) = \{0_{51}\}$

OFIRST $(0_{52}) = \{0_{52}\}$

Napomena: U 4. i 6. smeni nema gramatičkih dešavanja. U 5. smeni prvo dešavanje simbola 0 označili smo sa 0_{51}, a drugo sa 0_{52}.

Relacija OBELOW prikazana je na Sl. 4.1.33.

	$<S>_0$	a_1	$<A>_1$	b_2	$<B>_2$	1_3	$<A>_3$	0_3	1_5	$<B>_5$	0_{51}	0_{52}
∇	1	1		1								
$<S>_0$												
a_1			1			1						
$<A>_1$												
b_2					1				1			
$<B>_2$												
1_3						1	1					
$<A>_3$								1				
0_3												
1_5									1	1		
$<B>_5$											1	
0_{51}												1
0_{52}												

Sl. 4.1.33: Relacija OBELOW

Iz tabele prelaza detektora ručki sa Sl. 4.1.34 (leva strana tabele do dvostruke linije) se vidi da je u pitanju deterministički automat koji ujedno predstavlja i potisnu tabelu (prazni ulazi odgovaraju akciji REJECT).

Određujemo FOLLOW skupove neterminala:

FOLLOW (< S >) = {⊣}

FOLLOW (< A >) = {⊣, 0}

FOLLOW (< B >) = {⊣, 0}.

Kontrolna tabela prikazana je na Sl. 4.1.34 (desno od dvostruke linije).

	<S>	<A>	<B>	a	b	1	0	a	b	1	0	⊣
∇	$<S>_0$			a_1	b_2			SHIFT	SHIFT			
$<S>_0$												ACCEPT
a_1		$<A>_1$				1_3				SHIFT	REDUCE(4)	REDUCE(4)

$<A>_1$											REDUCE(1)
b_2			$<B>_2$			1_5			SHIFT	REDUCE(6)	REDUCE(6)
$<B>_2$											REDUCE(2)
1_3		$<A>_3$				1_3			SHIFT	REDUCE(4)	REDUCE(4)
$<A>_3$							0_3			SHIFT	
0_3										REDUCE(3)	REDUCE(3)
1_5			$<B>_5$			1_5			SHIFT	REDUCE(6)	REDUCE(6)
$<B>_5$							0_{51}			SHIFT	
0_{51}							0_{52}			SHIFT	
0_{52}										REDUCE(5)	REDUCE(5)

Sl. 4.1.34: Potisna i kontrolna tabela parsera

Startni stek: ∇

REDUCE(1): POP, POP, PUSHT(<S>)
REDUCE(2): POP, POP, PUSHT(<S>)
REDUCE(3): POP, POP, POP, PUSHT(<A>)
REDUCE(4): PUSHT(<A>)
REDUCE(5): POP, POP, POP, POP, PUSHT(<B>)
REDUCE(6): PUSHT(<B>)

Diskusija

Razmotrimo rad parsera za legalnu ulaznu sekvencu b:

	stek	ulaz	operacija
1.	∇	b⊣	SHIFT
2.	∇b_2	⊣	REDUCE (6)
3.	$\nabla b_2 < B >_2$	⊣	REDUCE (2)
4.	$\nabla < S >_0$	⊣	ACCEPT

Rad parsera za slučaj neispravnog ulaznog niza b0:

	stek	ulaz	operacija
1.	∇	b0⊣	SHIFT
2.	∇b_2	0⊣	REDUCE (6)
3.	$\nabla b_2 < B >_2$	0⊣	REJECT

Primetimo da je REJECT mogao doći već u koraku 2. Kako je FOLLOW (<B>) = {⊣, 0} otuda je u tim kolonama (za ⊣ i 0) data operacija REDUCE(6). Poboljšanje algoritma popunjavanja kontrolne tabele je da se posmatra FOLLOW skup za pojedine gramatičke pojave:

FOLLOW $(<B>_5) = \{0\}$

FOLLOW $(<B>_2) = \{\dashv\}$.

Ćelije u kontrolnoj tabeli $(b_2, 0)$ i $(1_5, \dashv)$ bile bi u tom slučaju popunjene akcijom REJECT što bi omogućilo bržu detekciju neispravne ulazne sekvence, a to je od značaja za evenualni

oporavak parsera od greške. Razmatranje koje smo sproveli je u osnovi konstrukcije parsera za šire klase LR gramatika – LALR(i) i LR(i), i≥1.

Zadatak 4.1.14

a) Za gramatiku

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

sa startnim simbolom <S> konstruisati SLR(1) prepoznavač.

b) Prikazati rad prepoznavača iz tačke a) za ulaznu sekvencu abca.

Rešenje

a)

Računamo OFIRST skupove gramatičkih dešavanja:

OFIRST($<S>_0$) = {$<S>_0$, $<A>_1$, $<S>_2$}

OFIRST($<A>_1$) = {$<A>_1$, $<S>_2$}

OFIRST(a_1) = {a_1}

OFIRST($<S>_2$) = {$<S>_2$, $<A>_1$}

OFIRST(b_2) = {b_2}

OFIRST($<B>_2$) = {$<B>_2$, $<A>_4$, c_5, $<S>_2$, $<A>_1$}

OFIRST($<A>_4$) = {$<A>_4$, $<A>_1$, $<S>_2$}

OFIRST(b_4) = {b_4}

OFIRST(c_5) = {c_5}

Relacija OBELOW prikazana je na Sl. 4.1.35, a odgovarajući nedeterministički detektor ručki na Sl. 4.1.36.

	$<S>_0$	$<A>_1$	a_1	$<S>_2$	b_2	$<B>_2$	$<A>_4$	b_4	c_5
$<S>_0$									
$<A>_1$			1						
a_1									

$<S>_2$					1				
b_2		1		1		1	1		1
$<B>_2$									
$<A>_4$								1	
b_4									
c_5									
∇	1	1		1					

Sl. 4.1.35: Relacija OBELOW

	<S>	<A>	<B>	a	b	c
$<S>_0$						
$<A>_1$				a_1		
a_1						
$<S>_2$					b_2	
b_2	$<S>_2$	$<A>_1$, $<A>_4$	$<B>_2$			c_5
$<B>_2$						
$<A>_4$					b_4	
b_4						
c_5						
∇	$<S>_0$, $<S>_2$	$<A>_1$				

Sl. 4.1.36: Nedeterministički detektor ručki

Deterministički detektor ručki, dobijen procedurom konstrukcije ekvivalentnog determinističkog automata na osnovu zadatog nedeterminističkog automata, prikazan je na Sl. 4.1.37. To je ujedno i potisna tabela parsera.

	<S>	<A>	<B>	a	b	c
∇	$\{<S>_0,<S>_2\}$	$<A>_1$				
$<S>_x = \{<S>_0,<S>_2\}$					b_2	
$<A>_1$				a_1		
a_1						
$<S>_2$					b_2	
b_2	$<S>_2$	$\{<A>_1,<A>_4\}$	$<B>_2$			c_5

$<B>_2$						
$<A>_x = \{<A>_1, <A>_4\}$				a_1	b_4	
b_4						
c_5						

Sl. 4.1.37: Deterministički detektor ručki

Odredimo sada FOLLOW skupove:

FOLLOW(<S>) = {b, ⊣}

FOLLOW(<A>) = {a, b}

FOLLOW(<B>) = FOLOW(<A>) = {a, b}

Ovi skupovi određuju smeštaj REDUCE akcija u kontrolnoj tabeli parsera, prikazanoj na Sl. 4.1.38. Za pravila popunjavanja kontrolne tabele SLR(1) parsera videti Zadatak 4.1.13.

	a	b	c	⊣
∇	REDUCE(3)	REDUCE(3)		
$<S>_x$		SHIFT		ACCEPT
$<A>_1$	SHIFT			
a_1		REDUCE(1)		REDUCE(1)
$<S>_2$		SHIFT		
b_2	REDUCE(3)	REDUCE(3)	SHIFT	
$<B>_2$	REDUCE(2)	REDUCE(2)		
$<A>_x$	SHIFT	SHIFT		
b_4	REDUCE(4)	REDUCE(4)		
c_5	REDUCE(5)	REDUCE(5)		

Sl. 4.1.38: Kontrolna tabela SLR(1) parsera

Startni stek: ∇

REDUCE(1): POP, POP, PUSHT(<S>)

REDUCE(2): POP, POP, POP, PUSHT(<A>)

REDUCE(3): PUSHT(<A>)

REDUCE(4): POP, POP, PUSHT(<B>)

REDUCE(5): POP, PUSHT(<B>)

b)

Rad parsera na prepoznavanju ulazne sekvence abca:

stek	ulaz	akcija
∇	abca⊣	REDUCE(3)

$\nabla<A>_1$	abca⊣	SHIFT
$\nabla<A>_1 a_1$	bca⊣	REDUCE(1)
$\nabla <S>_x$	bca⊣	SHIFT
$\nabla <S>_x b_2$	ca⊣	SHIFT
$\nabla <S>_x b_2 c_5$	a⊣	REDUCE(5)
$\nabla <S>_x b_2 <B>_2$	a⊣	REDUCE(2)
$\nabla<A>_1$	a⊣	SHIFT
$\nabla<A>_1 a_1$	⊣	REDUCE(1)
$\nabla <S>_x$	⊣	ACCEPT

Zadatak 4.1.15

Napisati gramatiku koja je

a) LL(1) ali nije LR(0)

b) LR(0) ali nije LL(1)

c) LL(1) ali nije SLR(1).

Rešenje

a) Rešenje predstavlja bilo koja LL(1) gramatika sa praznom smenom, s obzirom da LR(0) gramatike ne poseduju prazne smene, na primer:

1. <S>→ a
2. <S>→ ε

b) Rešenje predstavlja bilo koja LR(0) gramatika sa levom rekurzijom, na primer:

1. <S>→ <S> a SELECT(1) = {a}
2. <S>→ a SELECT(2) = {a}

c) Svaka LL(1) gramatika jeste i LR(1) gramatika, ali ne nužno i SLR(1) gramatika. Dok se kod LR(1) gramatika pri svođenju ručke na neki neterminal <A> gleda tačan kontekst u kome se neterminal <A> javlja, kod SLR(1) gramatika stvar se uprošćava korišćenjem skupa FOLLOW(<A>). Sledi primer LL(1) gramatike u kome se <A> javlja dva puta u različitom kontekstu – prvo dešavanje je praćeno terminalom a, a drugo terminalom b.

1. <S>→ <A> a <A> b SELECT(1) = FIRST(<A>) ∪ {a} = ∅ ∪ {a} = {a}
2. <S>→ b SELECT(2) = {b}
3. <A>→ ε SELECT(3) = FOLLOW(<A>) = {a, b}

Važi da je OFIRST($<S>_0$) = {$<S>_0$, $<A>_1$, b_2}, odnosno:

(1) ∇ OBELOW $<S>_0$

(2) ∇ OBELOW $<A>_1$

(3) ∇ OBELOW b_2

Iz (3) sledi da se ulaz kontrolne tabele u vrsti ∇ i koloni b popunjava akcijom SHIFT. S druge strane, s obzirom na (2) i činjenicu da je FOLLOW(<A>) = {a, b}, sledi da ulaze kontrolne tabele u vrsti ∇ i kolonama a i b treba popuniti akcijom REDUCE(3). Prema tome, u kontrolnoj tabeli javlja se "potisni–svedi" konflikt u vrsti ∇ i koloni b.

Diskusija

Za gramatiku iz tačke c) konflikt se neće javiti kod LR(1) parsera gde se za smeštaj REDUCE(3) akcija u vrsti ∇ posmatra tačan kontekst, a to je prvo dešavanje simbola <A> u prvoj smeni, tako da se samo kolona a popunjava sa REDUCE(3).

4.1.3. Optimizacija parserskih tabela

Zadatak 4.1.16

Data je sledeća gramatika:

1. <E> $\rightarrow$ <E> + <T>
2. <E> $\rightarrow$ <T>
3. <T> $\rightarrow$ <T> * <P>
4. <T> $\rightarrow$ <P>
5. <P> $\rightarrow$ (<E>)
6. <P> $\rightarrow$ a

Kontrolna tabela za datu gramatiku, konstruisana metodom potisni-identifikuj, prikazana je na Sl. 4.1.39.

	+	*	a	(	)	+
<E>	SHIFT	REJECT	REJECT	REJECT	SHIFT	ID–1
<T>	ID–2	SHIFT	REJECT	REJECT	ID–2	ID–2
<P>	ID–3	ID–3	REJECT	REJECT	ID–3	ID–3
+	REJECT	REJECT	SHIFT	SHIFT	REJECT	REJECT
*	REJECT	REJECT	SHIFT	SHIFT	REJECT	REJECT
a	ID–4	ID–4	REJECT	REJECT	ID–4	ID–4

(	REJECT	REJECT	SHIFT	SHIFT	REJECT	REJECT
)	ID–5	ID–5	REJECT	REJECT	ID–5	ID–5
∇	REJECT	REJECT	SHIFT	SHIFT	REJECT	REJECT

Sl. 4.1.39: Kontrolna tabela parsera potisni–identifikuj

Startni stek: ∇

SHIFT: PUSH(tekući ulaz), ADVANCE

ID–1: if (vrh steka = ∇<E>)
then ACCEPT
else REJECT
end if

ID–2: if (vrh steka = <E>+<T>)
then REDUCE(1)
else REDUCE(2)
end if

ID–3: if (vrh steka = <T>*<P>)
then REDUCE(3)
else REDUCE(4)
end if

ID–4: REDUCE(6)

ID–5: if (vrh steka = **(<E>)**)
then REDUCE(5)
else REJECT
end if

Smanjiti datu kontrolnu tabelu primenom funkcija prvenstva.

Analiza problema

Veličina kontrolne tabele za mašinu potisni-identifikuj može se, u nekim slučajevima, smanjiti primenom funkcija prvenstva. Neka je f(x) ceo broj koji je pridružen simbolu steka x, a g(y) ceo broj koji je pridružen ulaznom simbolu y. Uočimo da su terminalnim simbolima pridružene po dve vrednosti (pošto su oni istovremeno i ulazni simboli i simboli steka). Pretpostavimo da za simbol steka x i ulazni simbol y važi f(x) < g(y) kadgod su oni povezani relacijom x BELOW y. Pretpostavimo, takođe, da kad god postoji relacija x REDUCED_BY y, važi f(x) > g(y). Tada kontrolna tabela parsera može da bude zamenjena tabelama funkcija prvenstva f i g.

Rešenje

Relacije BELOW i REDUCED_BY za zadatu gramatiku prikazane su na Sl. 4.1.40 i Sl. 4.1.41, respektivno.

	+	*	a	(	)	<E>	<T>	<P>
+			1	1			1	1
*			1	1				1
a								
(			1	1		1	1	1
)								
<E>	1				1			

<T>		1						
<P>								
∇			1	1				

Sl. 4.1.40: Relacija BELOW

	+	*	a	(	)	+
+						
*						
a	1	1			1	1
(						
)	1	1			1	1
<E>						
<T>	1					
<P>	1	1			1	1

Sl. 4.1.41: Relacija REDUCED_BY

Na osnovu analize problema, ako važi

x BELOW y

tada je f(x)<g(y), odnosno ako je

x REDUCED_BY y

onda je f(x)>g(y). Odavde sledi da za slučaj REJECT važi

f(x)=g(y).

Međutim, ovakav pristup može dovesti do toga da se ne može naći funkcija prvenstva. Imajući na umu da sadržaj steka spojen s preostalim delom ulazne sekvence predstavlja sentencijalnu formu u krajnjem desnom izvođenju, nikakva kombinacija operacija SHIFT i REDUCE ne može prevesti ulaznu sentencu u startni neterminal ako ona ne pripada jeziku definisanom datom gramatikom. Polazeći od ovog stava, nema nikakve prepreke da se u ulazne kontrolne tabele umesto REJECT stavi SHIFT; parser će, istina, pri obradi neke sekvence, potrošiti više vremena, ali će raditi ispravno jer se neće moći izvršiti odgovarajuće svođenje koje bi dovelo do startnog neterminala. Na osnovu ovoga, kontrolna tabela dobija sledeći oblik:

	+	*	a	(	)	+
<E>	SHIFT	SHIFT	SHIFT	SHIFT	SHIFT	ID–1
<T>	ID–2	SHIFT	SHIFT	SHIFT	ID–2	ID–2

<P>	ID–3	ID–3	SHIFT	SHIFT	ID–3	ID–3
+	SHIFT	SHIFT	SHIFT	SHIFT	SHIFT	ID–6
*	SHIFT	SHIFT	SHIFT	SHIFT	SHIFT	ID–6
a	ID–4	ID–4	SHIFT	SHIFT	ID–4	ID–4
(	SHIFT	SHIFT	SHIFT	SHIFT	SHIFT	ID–6
)	ID–5	ID–5	SHIFT	SHIFT	ID–5	ID–5
∇	SHIFT	SHIFT	SHIFT	SHIFT	SHIFT	ID–6

Sl. 4.1.42: Modifikovana kontrolna tabela

U koloni za simbol + ulazi koji sadrže REJECT su zamenjeni sa ID–6 iz istih razloga zbog kojih je REJECT zamenjeno sa SHIFT. Izgled ove akcije je:

ID–6: REJECT

Princip određivanja funkcija prvenstva je sada jednostavan:

- ako ulaz u kontrolnoj tabeli definisan parom (x,y) sadrži SHIFT, odabrati f(x) i g(y) tako da važi f(x) < g(y).
- ako ulaz u kontrolnoj tabeli definisan parom (x,y) sadrži IDENTIFY, odabrati f(x) i g(y) tako da važi f(x) > g(y).

Posmatrajmo prvo kolonu za y = +; u njoj su samo identifikacione rutine, te g(+) moramo da izaberemo tako da za svako x važi

f(x) > g(+)

Neka je, stoga, g(+) = 0. Na isti način, posmatrajmo kolone koje su popunjene samo sa SHIFT. Zaključujemo da za svako x mora da važi f(x) < g(a) i f(x) < g((. Stoga stavljamo da je g(a) = g(() = b.

Analizirajmo sada redove za simbole steka: <E>, +, *, (i ∇. Vidi se da treba da bude f(x) > 0 i f(x) < g(y) za svako y. Stoga, stavljamo:

f(<E>) = f(+) = f(*) = f(() = f(∇) = 1

Na osnovu ulaza za x = <T> i y = * vidimo da treba da važi f(<T>) < g(*); s druge strane, treba da je f(<T>) > g(+) i f(<T>) > g()). Imajući to u vidu, stavljamo:

g(+) = g()) = 2, f(<T>) = 3, g(*) = 4

Preostaje da se odrede vrednosti funkcije prvenstva za simbole steka a,) i <P>. Za njih mora da važi f(x) > g(+), f(x) > g(*), f(x) < g(a) i f(x) > g()), pa je razumljiv izbor:

f(<P>) = f(a) = f()) = 5

Tabela funkcija prvenstva izgleda sada ovako:

stek simboli

x	<E>	<T>	<P>	+	*	a	(	)	∇
f(x)	1	3	5	1	1	5	1	5	1

ulazni simboli

y	+	*	a	(	)	+
g(y)	2	4	6	6	2	0

Parser funkcioniše sada prema sledećem algoritmu:

```
repeat
        Naći f(x) za tekući simbol steka i g(y) za tekući ulazni simbol;
        if  (f(x) < g(y))
                then    SHIFT y
                else    case x of
                                <E>: ID–1
                                <T>: ID–2
                                <P>: ID–3
                                   a: ID–4
                                   ): ID–5
                                +, (,∇: ID–6
                        end case
        end if
until false
```

Izlaz algoritma je ACCEPT ili REJECT u odgovarajućim IDENTIFY rutinama.

Diskusija

Ostavlja se čitaocu da na osnovu kontrolne tabele, relacija BELOW i REDUCED_BY sačini program koji bi automatski izračunavao funkcije prvenstva i generisao upravljački program.

4.2. Konfiguracioni pristup

Zadatak 4.2.1

Za datu gramatiku konstruisati

a) LR(0)

b) SLR(1)

parser konfiguracionim metodom.

1. <S> → a

2. <S> → (<S> <R>

3. <R> → , <S> <R>

4. <R> →)

Analiza problema

LR(0) konfiguracija predstavlja proizvoljnu gramatičku smenu dopunjenu simbolom • na početku desne strane smene ili na njenom kraju ili između bilo koja dva simbola na desnoj strani. Na primer, 2. smeni date gramatike odgovaraju sledeće konfiguracije:

<S> → • (<S> <R> <S> → (• <S> <R>

<S> → (<S> • <R> <S> → (<S> <R> •

Konfiguracije sa tačkom na početku nazivaju se konfiguracije zatvaranja, ostale konfiguracije nazivaju se bazičnim. Bazičnim konfiguracijama jednoznačno odgovaraju dešavanja gramatičkih simbola (Zadatak 4.1.11). Na primer, konfiguraciji <S> → (• <S> <R> odgovara dešavanje $(_2$, konfiguraciji <S> → (<S> • <R> dešavanje $<S>_2$, a konfiguraciji <S> → (<S> <R> • odgovara dešavanje $<R>_2$.

Praznoj smeni <X> → ε, ukoliko se pojavljuje u gramatici, odgovara jedna konfiguracija, u oznaci <X> → ε • ili <X> → •.

Iz potrebe da se uniformišu pravila konstrukcije karakterističnog automata i parserskih tabela, uvek se uvodi poseban startni neterminal i gramatika proširuje smenom rednog broja nula, na čijoj se desnoj strani nalaze startni neterminal originalne gramatike i marker kraja ulazne sekvence:

0. <S'> → <S> ⊣

Modifikovana gramatika opisuje sekvence koje pripadaju jeziku originalne gramatike, produžene markerom kraja.

Stanjima karakterističnog LR(0) automata (detektora ručki) odgovaraju skupovi LR(0) konfiguracija, a ulazi automata određeni su simbolima proširene gramatike. Algoritam konstrukcije karakterističnog LR(0) automata glasi:

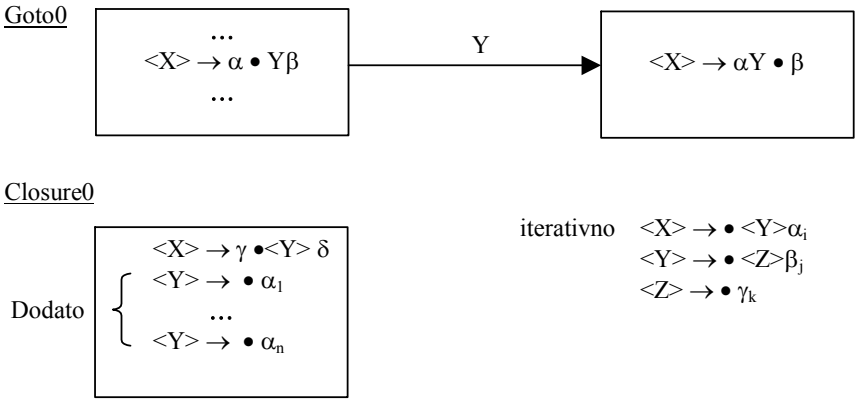
```
Izračunati skup konfiguracija startnog stanja S0 := Closure0 {<S'> → • <S> ⊣ }.
Označiti vrstu tabele prelaza sa ∇. Ovoj vrsti odgovara skup S0.
while (postoji označena, nepopunjena vrsta V tabele prelaza)
        Neka je S skup konfiguracija koji odgovara vrsti V.
        for ( ∀X iz skupa gramatičkih simbola)
            S1 := Goto0(S, X);
            if ( S1 se ne poklapa sa skupovima stanja već prisutnih u tabeli)
                then Dodati tabeli prelaza vrstu V1 kojoj odgovara skup S1.
                Napomena: Oznaka vrste jednaka je gramatičkom dešavanju koje
                odgovara bazičnoj konfiguraciji iz skupa S1. Ako u S1 ima više
                bazičnih konfiguracija, sve one odgovaraju različitim gramatičkim
                dešavanjima istog simbola, pa se u označavanju vrste upotrebljava
```

```
                    taj simbol sa alfanumeričkim indeksom koji služi da se ovo stanje u
                    tabeli jednoznačno identifikuje.
                end if;
                popuniti ulaz u vrsti V i koloni X sa V1;
            end for;
    end while;
```

U gornjoj proceduri koriste se pomoćne funkcije Closure0 i Goto0. Operacija Closure0 zadati skup bazičnih konfiguracija dopunjava (zatvara) odgovarajućim konfiguracijama zatvaranja, dok Goto0 izračunava novo stanje u koje se prelazi iz datog stanja pod datim ulaznim simbolom. Algoritmi ovih operacija glase:

```
function Closure0(ulaz: skup bazičnih konfiguracija S) rezultat: skup konfiguracija;
    Inicijalno S' := S;
    for (∀K ∈ S')
        if (u konfiguraciji K tačka se nalazi ispred neterminala <X>)
            then  for (∀ smenu P)
                        if (P je oblika <X>→ α)
                            Napomena: α je proizvoljan niz znakova)
                            then S' := S' ∪ { <X> → • α};
                        end if;
                    end for;
        end if
    end for;
    return S';
end function Closure0;
function GoTo0(ulaz: skup konfiguracija S; simbol X) rezultat: skup konfiguracija;
    Inicijalno S' := ∅;
    for (∀K ∈ S)
        if  (K je oblika <Y> → α • X β)
                Napomena: α i β proizvoljne sekvence
                then S' := S' ∪ {<Y> → α X • β}
        end if;
    end for;
    return Closure0(S');
end function GoTo0;
```

Operacije Closure0 i Goto0 grafički su predstavljene na Sl. 4.2.1. Za svaku konfiguraciju oblika $<X> \rightarrow \alpha \bullet Y\ \beta$ u zadatom stanju, operacija Goto0 dodaje konfiguraciju oblika $<X> \rightarrow \alpha\ Y \bullet \beta$ u novo stanje za koje se prelazi po zadatom simbolu Y. Za svaku konfiguraciju oblika $<X> \rightarrow \gamma \bullet <Y>\ \delta$ u zadatom stanju, operacija Closure0 u isto stanje dodaje po jednu konfiguraciju oblika $<Y> \rightarrow \bullet\ \alpha_i$ za svaku smenu sa neterminalom <Y> na levoj strani smene. Postupak dodavanja konfiguracija se iterativno ponavlja, dakle ako je dodata konfiguracija oblika $<Y> \rightarrow \bullet <Z>\ \beta_j$, tada će u narednoj iteraciji biti dodate i sve konfiguracije oblika $<Z> \rightarrow \bullet\ \gamma_k$, i tako dalje, sve dok se ne obrade sve konfiguracije postojeće i dodate konfiguracije iz zadatog stanja.



Sl. 4.2.1: Operacije Goto0 i Closure0

Dobijeni automat je deterministički, tako da njegova tabela prelaza istovremeno predstavlja potisnu tabelu LR(0) parsera.

Vrste kontrolne tabele LR(0) parsera odgovaraju pojedinim stanjima karakterističnog automata, a postoji samo jedna kolona jer se tekući ulaz uopšte ne razmatra pri izboru sledeće akcije LR(0) parsera – ovo je posledica potiskivanja markera kraja ulaza na stek. Pravila popunjavanja LR(0) kontrolne tabele na osnovu karakterističnog automata su sledeća (razmatra se ulaz u vrsti V, kojoj odgovara skup konfiguracija S):

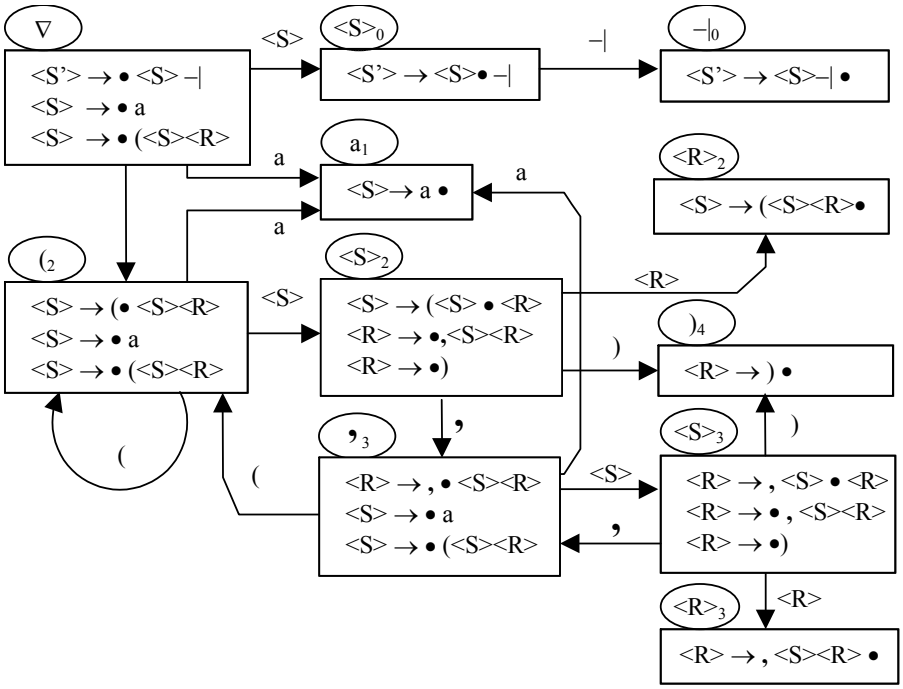
1. Ako S sadrži konfiguraciju oblika <X> → α • x β, odnosno tačka se nalazi ispred terminalnog simbola, ulaz treba popuniti akcijom SHIFT.
2. Ako S sadrži konfiguraciju oblika <Y> → γ •, odnosno tačka se nalazi na kraju desne strane, ulaz treba popuniti akcijom REDUCE(<Y> → γ). Izuzetno, ako se radi o nultoj smeni, akcija je ACCEPT.
3. Vrste koje nisu pokrivene ni 1. ni 2. pravilom treba da sadrže akciju REJECT.

Ukoliko gornja pravila na jednoznačan način određuju akciju za svaki ulaz kontrolne tabele, gramatika pripada klasi LR(0). U suprotnom, može doći do konflikata tipa potisni–svedi i svedi–svedi, što znači da se gramatika ne može deterministički parsirati LR(0) parserom.

Rešenje

a)

Graf prelaza karakterističnog LR(0) automata prikazan je na Sl. 4.2.2. U stanjima su navedeni odgovarajući skupovi konfiguracija, a simboličke oznake pojedinih stanja su zaokružene. Na slici Sl. 4.2.3 prikazane su tabela prelaza ovog automata koja istovremeno predstavlja potisnu tabelu parsera, kao i kontrolna tabela parsera. Gramatika jeste u klasi LR(0).



Sl. 4.2.2: Karakteristični LR(0) automat

Diskusija

Ovako konstruisan parser, za razliku od parsera dobijenog na osnovu relacije OBELOW, potiskuje marker kraja ulaza na stek; osim ovoga, dobijena tabela treba da je identična tabeli za istu gramatiku konstruisanoj po relacionom modelu. Potiskivanje markera kraja na stek omogućava kompletiranje ručke nulte smene <S'> → <S> ⊣ i uniformnu formulaciju pravila konstrukcije parserskih tabela.

Moguće je eliminisati simbol steka ⊣$_0$ i odgovarajuću vrstu u potisnoj i kontrolnoj tabeli, a u vrstu <S>$_0$ kontrolne tabele staviti akciju ACCEPT, pod uslovom da stanje <S>$_0$ karakterističnog automata sadrži samo jednu konfiguraciju. Ovo nije ispunjeno za sledeću gramatiku, na primer:

0. <S'> → <S>⊣
1. <S> → <S>a
2. <S> → b

kod koje stanje <S>$_0$ sadrži skup konfiguracija {<S'> → <S>•⊣, <S> → <S>•a}, pa bi opisano uprošćenje kontrolne tabele eliminisalo mogućnost prepoznavanja prve smene.

	<S>	<R>	(	,	)	a	⊣	
∇	<S>$_0$		(2			a1		SHIFT
<S>$_0$							⊣$_0$	SHIFT

⊣$_0$								ACCEPT
a$_1$								REDUCE(1)
($_2$	<S>$_2$		($_2$			a$_1$		SHIFT
<S>$_2$		<R>$_2$		,$_3$	)$_4$			SHIFT
<R>$_2$								REDUCE(2)
,$_3$	<S>$_3$		($_2$			a$_1$		SHIFT
<S>$_3$		<R>$_3$		,$_3$	)$_4$			SHIFT
<R>$_3$								REDUCE(3)
)$_4$								REDUCE(4)

potisna tabela LR (0) kontrolna tabela

Sl. 4.2.3: LR(0) parser

b)

Analiza problema

Potisna tabela SLR(1) parsera identična je potisnoj tabeli LR(0) parsera. Vrste kontrolne tabele SLR(1) parsera odgovaraju pojedinim stanjima karakterističnog automata, a kolone pojedinim ulaznim simbolima gramatike proširene 0. smenom. Pravila popunjavanja SLR(1) kontrolne tabele na osnovu karakterističnog LR(0) automata i FOLLOW skupova gramatičkih neterminala su sledeća (razmatra se ulaz u vrsti V, kojoj odgovara skup konfiguracija S i koloni x):

1. Ako S sadrži konfiguraciju oblika $<X> \rightarrow \alpha \bullet x\, \beta$, odnosno tačka se nalazi ispred terminalnog simbola x, ulaz treba popuniti akcijom SHIFT.
2. Ako S sadrži konfiguraciju oblika $<Y> \rightarrow \gamma \bullet$, odnosno tačka se nalazi na kraju desne strane, a važi da je $x \in$ FOLLOW(<Y>), tada ulaz treba popuniti akcijom REDUCE($<Y> \rightarrow \gamma$). Izuzetno, ako se radi o nultoj smeni, akcija je ACCEPT.
3. Vrste koje nisu pokrivene ni 1. ni 2. pravilom treba da sadrže akciju REJECT.

Ukoliko gornja pravila na jednoznačan način određuju akciju za svaki ulaz kontrolne tabele, gramatika pripada klasi SLR(1). U suprotnom, može doći do konflikata tipa potisni–svedi i svedi–svedi, što znači da se gramatika ne može deterministički parsirati SLR(1) parserom.

Rešenje

Odredimo FOLLOW skupove neterminala proširene gramatike:

FOLLOW (<S'>) = { ⊣ }

FOLLOW (<S>) = { ,) ⊣ }

FOLLOW (<R>) = { ,) ⊣ }

Na osnovu ovoga i automata sa Sl. 4.2.2 određujemo kontrolnu tabelu (Sl. 4.2.4). Gramatika jeste klase SLR(1).

	(	,	)	a	⊣
∇	SHIFT			SHIFT	
<S>$_0$					SHIFT
⊣$_0$					ACCEPT
a$_1$		REDUCE(1)	REDUCE(1)		REDUCE(1)
($_2$	SHIFT			SHIFT	
<S>$_2$		SHIFT	SHIFT		
<R>$_2$		REDUCE(2)	REDUCE(2)		REDUCE(2)
,$_3$	SHIFT			SHIFT	
<S>$_3$		SHIFT	SHIFT		
<R>$_3$		REDUCE(3)	REDUCE(3)		REDUCE(3)
)$_4$		REDUCE(4)	REDUCE(4)		REDUCE(4)

Sl. 4.2.4: Kontrolna tabela SLR(1) parsera

Zadatak 4.2.2

Za datu gramatiku konstruisati

a) LR(1)

b) LALR(1)

parser konfiguracionim metodom.

1. <S> → <A> <S>
2. <S> → ε
3. <A> → <B> b <B> c
4. <A> → <C> c <B>
5. <B> → a <D>
6. <C> → a <D>
7. <D> → ε

Analiza problema

LR(1) konfiguracije definišu se kao uređeni par u kome prva komponenta predstavlja LR(0) konfiguraciju, a druga, takozvana predikciona (engl. *lookahead*) komponenta, terminalni simbol gramatike koja je proširena nultom smenom kao što je opisano u prethodnom zadatku. Predikciona komponenta x konfiguracije <A> → α • β, x označava jedan od mogućih tekućih

ulaznih simbola u trenutku rada parsera kada se na vrhu steka kompletira ručka smene $<A> \rightarrow \alpha\beta$.

Navedimo za primer četiri LR(1) konfiguracije za datu gramatiku sa istom prvom komponentom :

$<S> \rightarrow \bullet <A> <S>$, a $<S> \rightarrow \bullet <A> <S>$, b

$<S> \rightarrow \bullet <A> <S>$, c $<S> \rightarrow \bullet <A> <S>$, ⊣

Skup koji sadrži navedene četiri konfiguracije može se kraće obeležiti sa:

$<S> \rightarrow \bullet <A> <S>$, {a, b, c, ⊣}

pri čemu {a, b, c, ⊣} nazivamo predikcioniom skupom.

Stanjima karakterističnog LR(1) automata (detektora ručki) odgovaraju skupovi LR(1) konfiguracija, a ulazi automata određeni su simbolima proširene gramatike. Algoritam konstrukcije karakterističnog LR(1) automata glasi:

```
Izračunati skup konfiguracija startnog stanja S0 := Closure1 {<S'> → • <S> ⊣ , ∅}.
Označiti vrstu tabele prelaza sa ∇. Ovoj vrsti odgovara skup S0.
while (postoji označena, nepopunjena vrsta V tabele prelaza)
        Neka je S skup konfiguracija koji odgovara vrsti V.
        for (∀X iz skupa gramatičkih simbola)
            S1 := Goto1(S, X);
            if ( S1 se ne poklapa sa skupovima stanja već prisutnih u tabeli)
                then Dodati tabeli prelaza vrstu V1 kojoj odgovara skup S1.
                Napomena: Vrste se simbolički označavaju prema prvim
                komponentama konfiguracija, po istim pravilima kao kod
                LR(0) parsera. Dodatno, ukoliko su dva stanja identična kada
                se posmatraju samo prve komponente konfiguracija, moguće
                je upotrebiti istu oznaku i donji indeks indeks, a razlikovati po
                gornjem indeksu, na primer, ax' i ax".
            end if;
            popuniti ulaz u vrsti V i koloni X sa V1;
        end for;
end while;
```

U gornjoj proceduri koriste se pomoćne funkcije Closure1 i Goto1. Operacija Closure0 zadati skup bazičnih konfiguracija dopunjava (zatvara) odgovarajućim konfiguracijama zatvaranja, dok Goto1 izračunava novo stanje u koje se prelazi iz datog stanja pod datim ulaznim simbolom. Algoritmi ovih operacija glase:

```
function Closure1( ulaz: skup bazičnih konfiguracija S) rezultat: skup konfiguracija;
    Inicijalno S' := S;
    for (∀K ∈ S')
        if (K je oblika <Y> → α • <X> β, t )
        then    for ( ∀smenu P)
                    if (P je oblika <X> → γ)
                        Napomena: α, β i γ su proizvoljne sekvence simbola)
```

```
                                then  if (β je poništiva sekvenca)
                                          then Dodati u S' sve konfiguracije oblika
                                              <X> → • γ, u gde u ∈ FIRST(β) ∪ {t}
                                          else Dodati u S' sve konfiguracije oblika
                                              <X> → • γ, u gde u ∈ FIRST(β).
                                      end if;
                              end if;
                          end for;
                  end if;
          end for;
          return S';
    end function Closure1;
```

```
    function GoTo1( ulaz: skup konfiguracija S; simbol X) rezultat: skup konfiguracija;
        Inicijalno S' := ∅;
        for (∀K ∈ S)
                if  (K je oblika <Y> → α • X β, t)
                        Napomena:  α i β proizvoljne sekvence
                         t je proizvoljan terminal
                        then S' := S' ∪ {<Y> → α X • β, t}
                end if;
        end for;
        return Closure1(S');
    end function GoTo1;
```

Dobijeni automat je deterministički, tako da njegova tabela prelaza istovremeno predstavlja potisnu tabelu LR(1) parsera.

Vrste kontrolne tabele LR(1) parsera odgovaraju pojedinim stanjima karakterističnog automata, a kolone pojedinim ulaznim simbolima gramatike proširene 0. smenom. Pravila popunjavanja LR(1) kontrolne tabele na osnovu karakterističnog LR(1) automata su sledeća (razmatra se ulaz u vrsti V, kojoj odgovara skup konfiguracija S i koloni x):

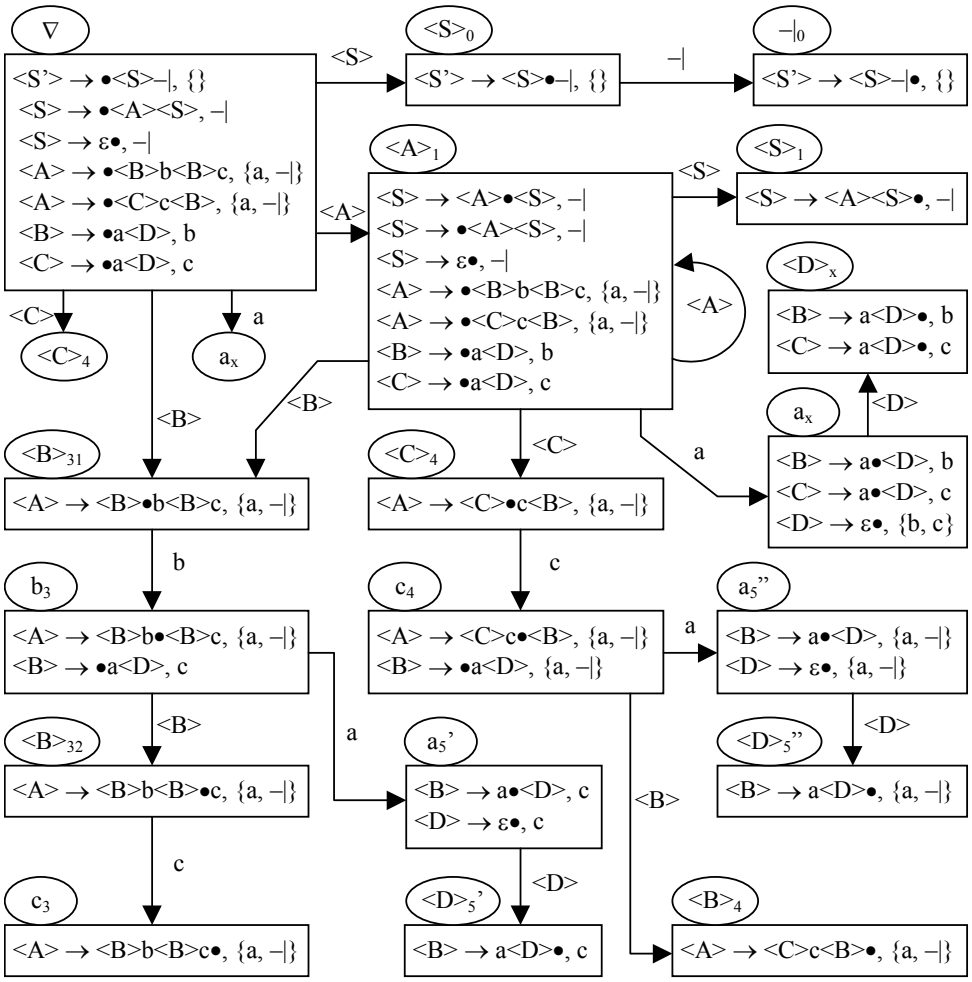
1. Ako S sadrži konfiguraciju oblika <X> → α • x β, t odnosno tačka se nalazi ispred terminalnog simbola x, ulaz treba popuniti akcijom SHIFT. Izuzetno, ako se radi o nultoj smeni i konfiguraciji <S'> → <S> • ⊣, akcija je ACCEPT.
2. Ako S sadrži konfiguraciju oblika <Y> → γ •, x odnosno tačka se nalazi na kraju desne strane, a predikciona komponenta je x, tada ulaz treba popuniti akcijom REDUCE(<Y> → γ).
3. Vrste koje nisu pokrivene ni 1. ni 2. pravilom treba da sadrže akciju REJECT.

Ukoliko gornja pravila na jednoznačan način određuju akciju za svaki ulaz kontrolne tabele, gramatika pripada klasi LR(1). U suprotnom, može doći do konflikata tipa potisni–svedi i svedi–svedi, što znači da se gramatika ne može deterministički parsirati LR(1) parserom.

Rešenje

a)

Na Sl. 4.2.5 prikazan je karakteristični LR(1) automat za zadatu gramatiku proširenu nultom smenom. Potisna i kontrolna tabela LR(1) parsera prikazane su na Sl. 4.2.6. Prazni ulazi u tabelama predstavljaju akciju REJECT. Gramatika pripada klasi LR(1).



Sl. 4.2.5: Karakteristični LR(1) automat

	<S>	<A>	<B>	<C>	<D>	a	b	c	a	b	c	–|
∇	$<S>_0$	$<A>_1$	$<B>_{31}$	$<C>_4$		a_x			SHIFT			RED(2)
$<S>_0$												ACC
$<A>_1$	$<S>_1$	$<A>_1$	$<B>_{31}$	$<C>_4$		a_x			SHIFT			RED(2)
$<B>_{31}$							b_3			SHIFT		
$<C>_4$								c_4			SHIFT	
a_x					$<D>_x$					RED(7)	RED(7)	
$<S>_1$												RED(1)
b_3			$<B>_{32}$			a_5'			SHIFT			
c_4			$<B>_4$			a_5"			SHIFT			
$<D>_x$										RED(5)	RED(6)	
$<B>_{33}$								c_3			SHIFT	
a_5'					$<D>_5$'						RED(7)	
a_5"					$<D>_5$"				RED(7)			RED(7)
$<B>_4$									RED(4)			RED(4)
c_3									RED(3)			RED(3)
$<D>_5$'											RED(5)	
$<D>_5$"									RED(5)			RED(5)

potisna tabela kontrolna tabela

Sl. 4.2.6: LR(1) parser

b)

Analiza problema

Karakteristični LR(0) automat za neku gramatiku može se dobiti na osnovu karakterističnog LR(1) automata za istu gramatiku na taj način što se:

1. uklone predikcione komponente LR(1) konfiguracija iz svih stanja LR(1) automata i
2. stanja koja sadrže identične skupove LR(0) konfiguracija posle primene tačke 1. navedu samo jedanput u tabeli ili dijagramu prelaza.

Svakom stanju S karakterističnog LR(0) automata odgovara jedno ili više stanja karakterističnog LR(1) automata od kojih se, uklanjanjem predikcionih komponenti LR(1) konfiguracija, dobija stanje S. Stanje S u tom slučaju naziva se jezgrom odgovarajućih stanja LR(1) automata.

Karakteristični LALR(1) automat za zadatu gramatiku određuje se prema sledećim pravilima:

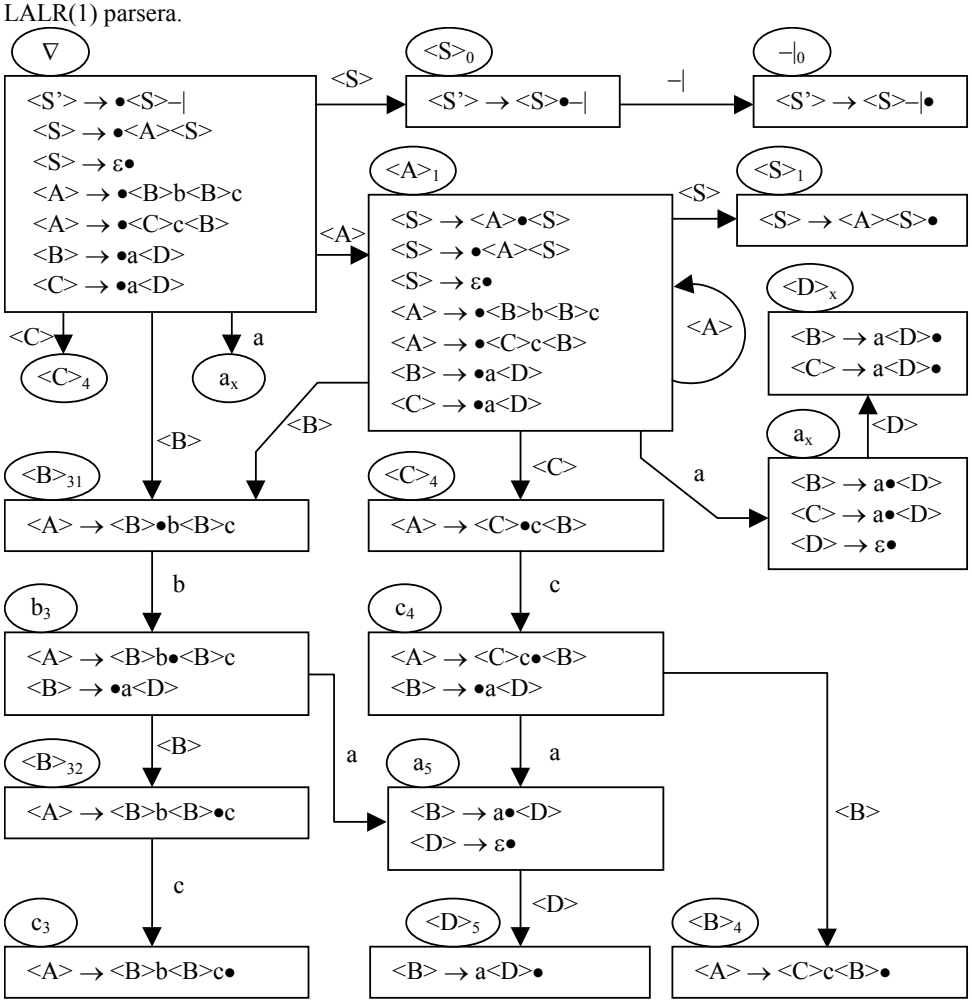
- Dijagram prelaza LALR(1) automata isti je dijagramu prelaza LR(0) automata za zadatu gramatiku, što znači da su stanja ova dva automata u 1-1 korespondenciji. S obzirom na ovu činjenicu, stanja LALR(1) automata takođe predstavljaju jezgra odgovarajućih stanja LR(1) automata.

- Svakom stanju S LALR(1) automata odgovara skup LR(1) konfiguracija, koji predstavlja uniju svih LR(1) konfiguracija iz svih stanja LR(1) automata koja imaju jezgro S.

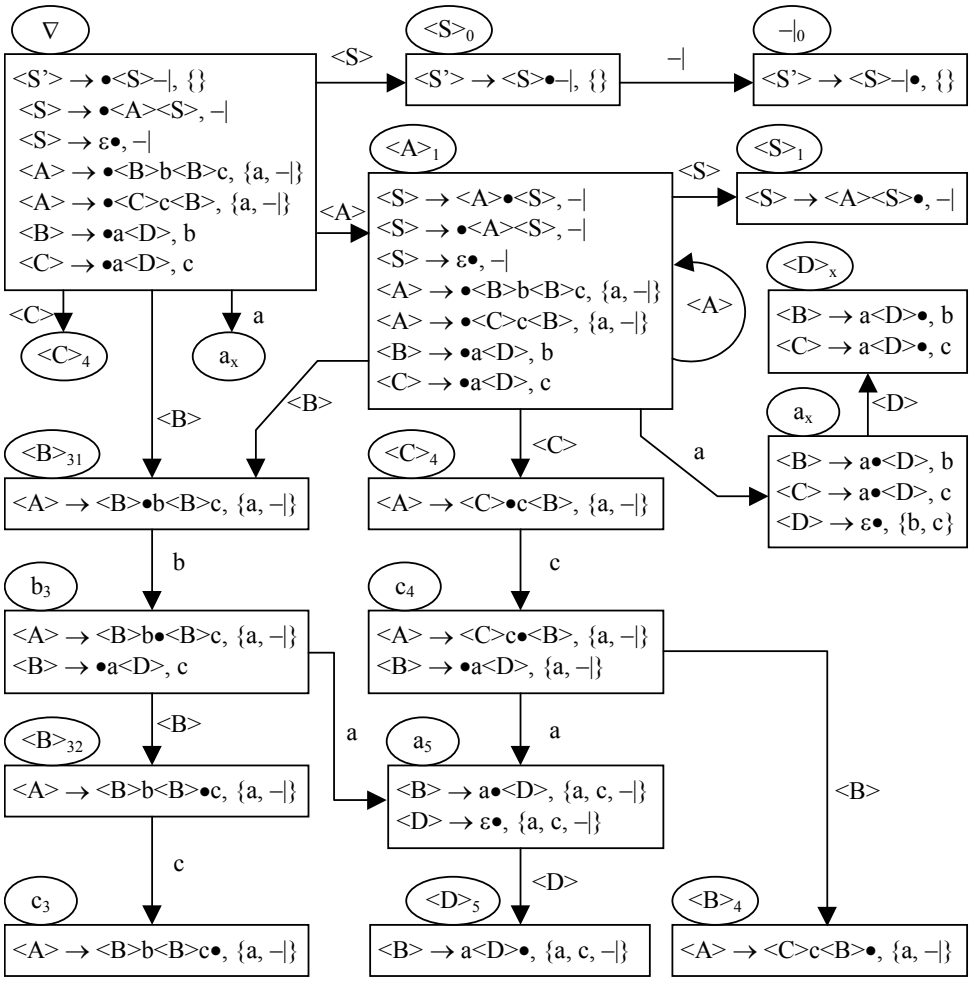
Karakteristični LALR(1) automat je deterministički, pa istovremeno predstavlja potisnu tabelu LALR(1) parsera. Kontrolna tabela određuje se na osnovu ovog automata istim pravilima kao i u slučaju LR(1) parsera.

Rešenje

Na Sl. 4.2.7 prikazan je karakteristični LR(0) automat za proširenu gramatiku. Vidimo da je stanje a_5 ovog automata jezgro stanja a_5' i a_5" LR(1) automata sa Sl. 4.2.5, a stanje $<D>_5$ jezgro stanja $<D>_5$' i $<D>_5$". Na Sl. 4.2.8 prikazan je karakteristični LALR(1) automat za datu gramatiku proširenu nultom smenom. Na Sl. 4.2.9 prikazane su potisna i kontrolna tabela LALR(1) parsera.



Sl. 4.2.7: Karakteristični LR(0) automat



Sl. 4.2.8: Karakteristični LALR(1) automat

Diskusija

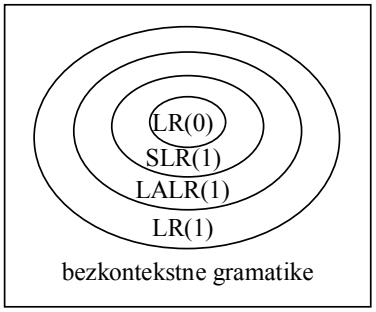
Odnos pojedinih klasa LR gramatika prikazan je na Sl. 4.2.10.

LALR(1) parseri u praksi predstavljaju najčešće korišćeni kompromis između generalnosti i veličine parsera. Veličina parsera ista je kao kod SLR(1) parsera, mnogo manja nego kod LR(1) parsera, ali je smeštaj REDUCE akcija u kontrolnoj tabeli preciznije određen nego kod SLR(1) parsera što rezultuje u većem skupu gramatika za koje je moguće konstruisati parser. Ovaj skup je ipak manji nego kod LR(1) metoda. Poznati generator parsera *yacc* zasnovan je na LALR(1) metodu parsiranja. Problem veličine LR(1) parsera rešava se jednim originalnim metodom autora (videti Zadatak 4.3.2).

	<S>	<A>	<B>	<C>	<D>	a	b	c	a	b	c	–|
∇	$<S>_0$	$<A>_1$	$<B>_{31}$	$<C>_4$		a_x			SHIFT			RED(2)
$<S>_0$												ACC
$<A>_1$	$<S>_1$	$<A>_1$	$<B>_{31}$	$<C>_4$		a_x			SHIFT			RED(2)
$<B>_{31}$							b_3			SHIFT		
$<C>_4$								c_4			SHIFT	
a_x					$<D>_x$					RED(7)	RED(7)	
$<S>_1$												RED(1)
b_3			$<B>_{32}$			a_5'			SHIFT			
c_4			$<B>_4$			a_5''			SHIFT			
$<D>_x$										RED(5)	RED(6)	
$<B>_{33}$								c_3			SHIFT	
a_5					$<D>_5$'				RED(7)		RED(7)	RED(7)
$<B>_4$									RED(4)			RED(4)
c_3									RED(3)			RED(3)
$<D>_5$									RED(5)		RED(5)	RED(5)

potisna tabela kontrolna tabela

Sl. 4.2.9: LALR(1) parser



Sl. 4.2.10: Klase LR gramatika

Čitaocu se ostavlja da proveri da li gramatika:

1. <S> → (<A>)
2. <S> → [<A>]
3. <S> → (<B>]
4. <S> → [<B>)
5. <A> → c
6. <B> → c

pripada klasama LR(0), SLR(1), LALR(1) ili LR(1).

Zadatak 4.2.3

Za gramatiku iz prethodnog zadatka odrediti karakteristični LALR(1) automat na bazi LR(0) automata, bez konstruisanja LR(1) automata.

Analiza problema

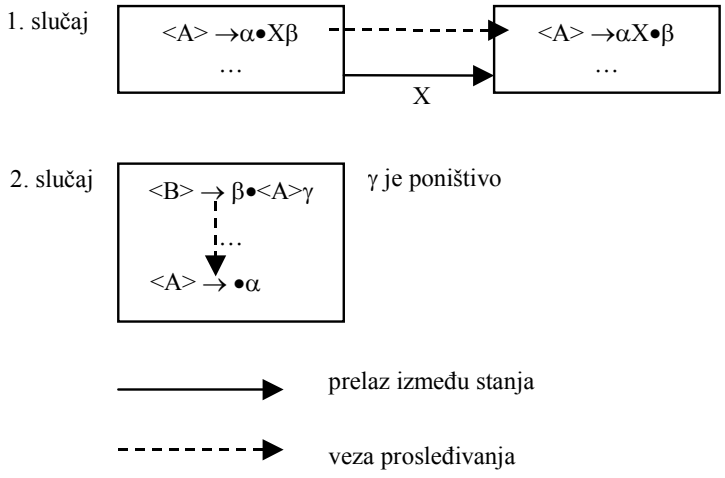
Za dobijanje LALR(1) automata potrebno je odrediti predikcione skupove LR(0) konfiguracija u pojedinim stanjima LR(0) automata.

U cilju računanja predikcionih skupova prvo se odrede takozvani skupovi spontanih predikcionih simbola (skr. SLA) za sve konfiguracije zatvaranja u svakom od stanja LR(0) detektora ručki. SLA skup za proizvoljnu LR(0) konfiguraciju zatvaranja $<A> \rightarrow \bullet \alpha$ u stanju s računa se tako što se nađu sve konfiguracije oblika $<B> \rightarrow \beta \bullet <A> \gamma$ u stanju s i u skup SLA($<A> \rightarrow \bullet\alpha$) uključe svi terminali $t \in FIRST(\gamma)$.

Između LR(0) konfiguracija u stanjima LR(0) automata definišu se *veze prosleđivanja* prema sledećim pravilima:

1. Za svaku konfiguraciju oblika $<A> \rightarrow \alpha\bullet X\beta$ u posmatranom stanju S, gde su α i β proizvoljne sekvence od nula ili više gramatičkih simbola, a X proizvoljan gramatički simbol, uvodi se veza prosleđivanja ka konfiguraciji oblika $<A> \rightarrow \alpha X\bullet\beta$ u stanju u koje se prelazi iz S po X.
2. Za svaku konfiguraciju oblika $<B> \rightarrow \beta\bullet <A>\gamma$ u posmatranom stanju S, gde je β proizvoljna sekvenca od nula ili više gramatičkih simbola, γ je poništiva sekvenca od nula ili više gramatičkih simbola, a <B> je proizvoljan neterminal, uvode se veze prosleđivanja ka svim konfiguracijama oblika $<A> \rightarrow \bullet\alpha$ u posmatranom stanju S.

Pravila određivanja veza prosleđivanja ilustrovana su na Sl. 4.2.11.



Sl. 4.2.11: Pravila prosleđivanja predikcionih simbola

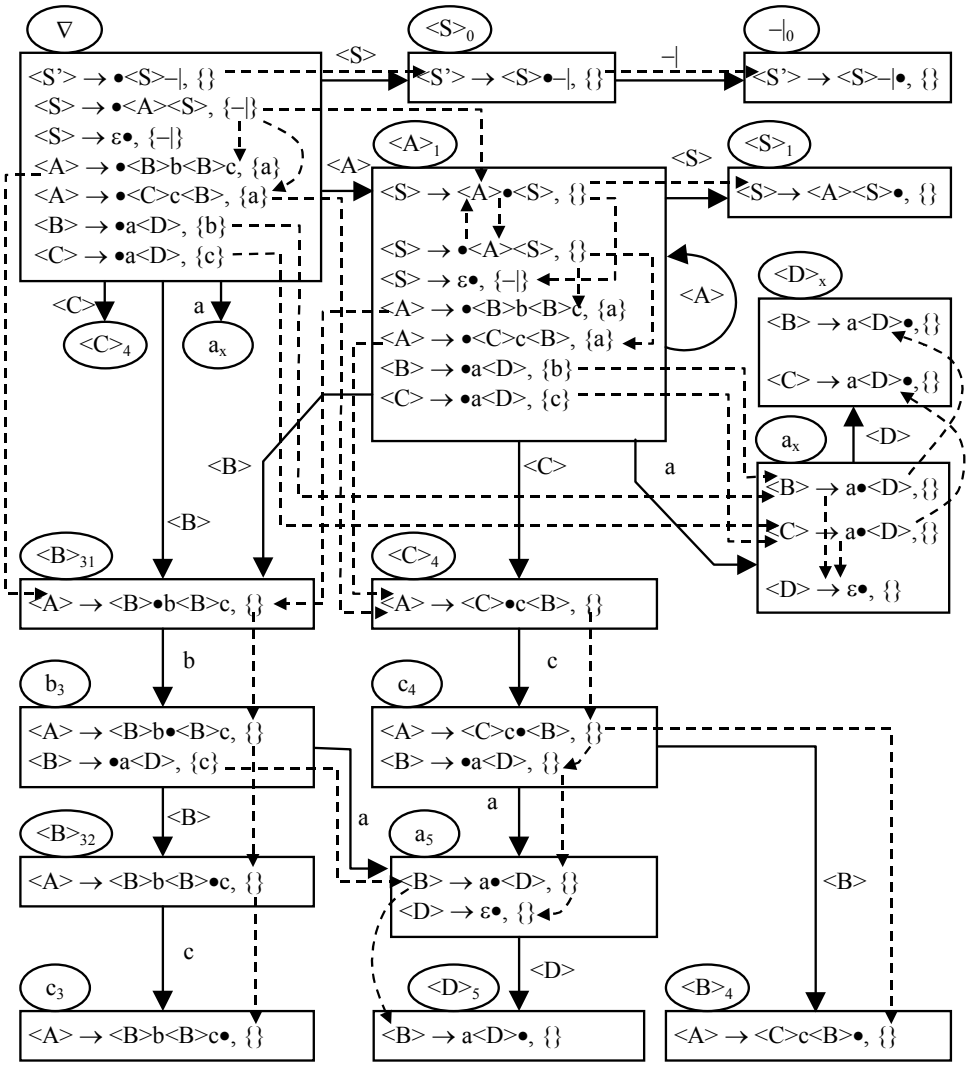
Predikcioni (skr. LA) skupovi računaju se sledećim algoritmom:

```
ulaz:   LR(0) automat sa vezama prosleđivanja između konfiguracija, SLA skupovi
        svih konfiguracija zatvaranja u svim stanjima;
izlaz:  LA skupovi svih konfiguracija u svim stanjima.
stek je inicijalno prazan;
for (∀ stanje S)
    for (∀ konfigiraciju K ∈ S)
        if (K je oblika <A> → •α)
            then  LA(K) := SLA(K);
                  for (t ∈ SLA(K))
                      push( S, K, t);
                  end for;
            else LA(K) := ∅;
        end if
    end for;
end for;
while (stek nije prazan)
    pop( S, K, t);
    for (∀ konfiguraciju K1 ka kojoj ide veza prosleđivanja od K)
        if ( t ∉ LA(K1))
            then  dodati t u LA(K1);
                  S1 := stanje automata u kome se nalazi K1;
                  push(S1, K1, t);
        end if;
    end for;
end while;
```

Inicijalno su LA skupovi konfiguracija zatvaranja u svakom od stanja jednaki SLA skupovima odgovarajućih konfiguracija, dok su LA skupovi ostalih konfiguracija prazni. Algoritam koristi stek na koji se stavljaju trojke (stanje, konfiguracija, predikcioni simbol). Inicijalno na stek idu sve trojke koje odgovaraju konfiguracijama zatvaranja iz svih stanja i simbolima iz njihovih SLA skupova. Trojke sa steka se redom razmatraju, pri čemu se ažuriraju LA skupovi konfiguracija koje su povezane vezama prosleđivanja sa razmatranom konfiguracijom. Na stek idu novo dodati elementi ažuriranih LA skupova. Postupak se završava kada se stek isprazni.

Rešenje

Sl. 4.2.12 prikazuje početno stanje pri računanju predikcionih simbola iterativnim algoritmom za gramatiku iz prethodnog zadatka. LR(0) konfiguracijama u svakom stanju LR(0) detektora ručki sa Sl. 4.2.7 dodati su predikcioni skupovi koji su inicijalizovani na skupove spontanih predikcija za konfiguracije zatvaranja, dok su predikcioni skupovi bazičnih konfiguracija inicijalno prazni. Isprekidane linije predstavljaju veze prosleđivanja. Po završetku rada algoritma, dobija se LALR(1) detektor ručki prikazan na Sl. 4.2.8.



Sl. 4.2.12: Ilustracija iterativnog određivanja predikcionih simbola

Zadatak 4.2.4

Naći gramatiku sa sledećim svojstvima:

a) da je SLR(3) ali ne i SLR(2).

b) da je LALR(2) ali ne i LALR(1).

c) da je LR(1).

Analiza problema

LR tehnike parsiranja mogu se generalizovati da koriste predikcije dužine k ulaznih simbola. Definišimo najpre generalizaciju FIRST i FOLLOW skupova da, umesto pojedinih simbola, uključe nizove simbola dužine k:

Sa $FIRST_k(\alpha)$, gde je α proizvoljna sekvenca gramatičkih simbola, označavamo skup prefiksa **u** svih nizova terminala β koje je moguće izvesti iz α, pri čemu dužina **u** iznosi k ako je β duže od k simbola, inače se **u** poklapa sa β.

$$FIRST_k(\alpha) = \{u \mid \alpha \stackrel{*}{\Rightarrow} \beta = uv \wedge u, v \in V_T^* \wedge |u| = \min(k, |\beta|)\}$$

Sa $FOLLOW_k(<X>)$, gde je <X> proizvoljan gramatički neterminal, označavamo skup prefiksa **u** svih nizova terminala β koji mogu slediti <X> u nekoj sentencijalnoj formi izvedenoj iz <S>⊣, pri čemu dužina **u** iznosi k ako je β duže od k simbola, inače se **u** poklapa sa β i <S> predstavlja startni neterminal.

$$FOLLOW_k(<X>) = \{u \mid <S>\dashv \stackrel{*}{\Rightarrow} \alpha<X>\beta \ \wedge \beta = uv \wedge u, v \in V_T^* \wedge |u| = \min(k, |\beta|)\}$$

Na osnovu ovoga generalizuju se pojedine LR tehnike (navedene su izmene u odnosu na slučaj k=1):

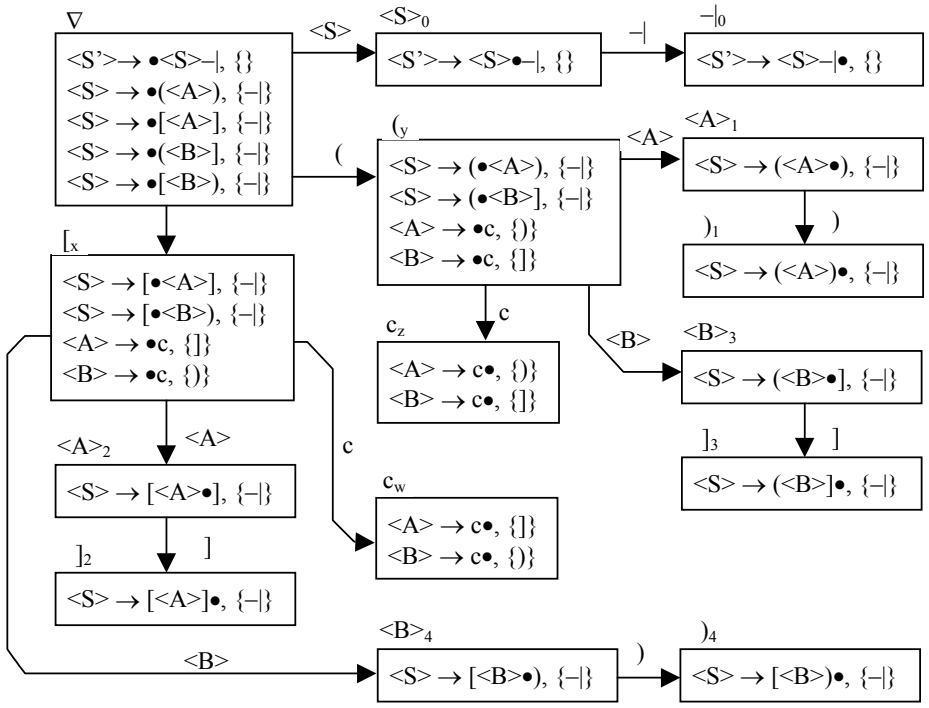
- LR(k): Pri konstrukciji karakterističnog automata, u operaciji zatvaranja Closure-k skupu konfiguracija koji sadrži $<A> \rightarrow \alpha\bullet<B>\beta$, x dodati sve konfiguracije oblika $<B> \rightarrow \bullet\gamma$, y gde je $y \in FIRST_k(\beta x)$. Akcija svođenja smene $<A> \rightarrow \alpha$ za predikciju x na ulazu parsera preduzima se za vrh steka koji odgovara stanju karakterističnog automata sa konfiguracijom $<A> \rightarrow \alpha\bullet$, x. Akcija potiskivanja za predikciju x na ulazu preduzima se za vrh steka koji odgovara stanju sa konfiguracijom $<A> \rightarrow \alpha\bullet a\beta$, y, gde a predstavlja terminalni simbol i $x \in FIRST_k(a\beta y)$.
- SLR(k): Akcija svođenja smene $<A> \rightarrow \alpha$ za predikciju x na ulazu parsera preduzima se za vrh steka koji odgovara stanju karakterističnog automata sa konfiguracijom $<A> \rightarrow \alpha\bullet$ i pri tome je $x \in FOLLOW_k(<A>)$. Akcija potiskivanja za predikciju x na ulazu preduzima se za vrh steka koji odgovara stanju sa konfiguracijom $<A> \rightarrow \alpha\bullet a\beta$, gde a predstavlja terminalni simbol i $x \in FIRST_k(a\beta u)$ i $\mathbf{u} \in FOLLOW_k(<A>)$.
- LALR(k): Spojiti stanja karakterističnog LR(k) automata sa istim jezgrom. Akcije parsera defnišu se tada na isti način kao i u LR(k) slučaju za ovako dobijeni automat.

Rešenje

Polazna tačka za konstrukciju tražene gramatike biće sledeća LR(1) gramatika iz Fischer-a [6] koja nije LALR(1):

1. $<S> \rightarrow (<A>)$
2. $<S> \rightarrow [<A>]$
3. $<S> \rightarrow (<B>]$
4. $<S> \rightarrow [<B>)$
5. $<A> \rightarrow c$
6. $<B> \rightarrow c$

Na Sl. 4.2.13 prikazan je karakterističan LR(1) automat za ovu gramatiku (dodata je smena $<S'> \rightarrow <S>\dashv$), bez konflikata.



Sl. 4.2.13: Karakteristični LR(1) automat

Karakteristični LALR(1) automat poseduje ista stanja i prelaze kao i LR(1) automat, sa jednim izuzetkom: stanjima c_z i c_w, koja imaju isto jezgro, odgovara jedinstveno stanje sa sledećim konfiguracijama:

<A> → c•, {),]}
<B> → c•, {],)}

koje izaziva REDUCE–REDUCE konflikt u kontrolnoj tabeli LALR(1) parsera.

Da bismo od posmatrane gramatike dobili gramatiku koja nije LALR(1) a jeste LALR(2), odnosno da bismo izbegli konflikt, potrebno je prateće kontekste za <A>, odnosno za <B> u smenama 1.-4. učiniti jedinstvenim za slučaj predikcija dužine 2 dodavanjem novih terminala čime dobijamo sledeću gramatiku:

1. <S> → (<A>)]
2. <S> → [<A>]]
3. <S> → (<B>])
4. <S> → [<B>))
5. <A> → c
6. <B> → c

Nije teško utvrditi da karakteristični LALR(1) automat za ovu gramatiku poseduje isto konfliktno stanje kao i automat za polaznu gramatiku. U LALR(2) automatu, odgovarajuće stanje ima sledeći izgled (predikcije su dužine 2 simbola):

<A> → c•, {)],]]}
<B> → c•, {]),))}

Konflikt je, dakle, izbegnut. Ostala stanja nisu kritična (konflikti se ne pojavljuju čak ni kada se uklone predikcije), tako da gramatika jeste LALR(2). Gramatika jeste i SLR(2) s obzirom da predikcije iz "kritičnog" stanja LALR(2) automata ujedno predstavljaju $FOLLOW_2$ skupove neterminala <A>, odnosno <B>. Da bismo zadovoljili uslov da gramatika ne bude SLR(2), a da bude SLR(3) načinićemo konačnu modifikaciju dodavanjem nove pojave neterminala <A> u pogodnom kontekstu, bez promene konfiguracija "kritičnog" stanja, da se ne naruše uslovi LALR(2) i LR(1) parsiranja:

1. <S> → <A>)))
2. <S> → (<A>)]
3. <S> → [<A>]]
4. <S> → (<B>])
5. <S> → [<B>))
6. <A> → c
7. <B> → c

Karakteristični LR(1) automat za ovu gramatiku prikazan je na Sl. 4.2.14. Gramatika jeste LR(1), ali nije LALR(1), jer se spajanjem stanja c_z i c_w dobija konfliktno stanje. Konstrukcijom LALR(2) automata (Sl. 4.2.15) utvrđujemo da je gramatika LALR(2). Ukoliko iz LALR(2) automata iz stanja uklonimo predikcione komponente, dobija se karakteristični LR(0) automat. Za SLR parsiranje kritično je stanje c sa konfiguracijama:

<A> → c•

<B> → c•

$FOLLOW_2$ skupovi neterminala <A> i <B> su sledeći:

$FOLLOW_2$(<A>) = {)),)],]]}

$FOLLOW_2$(<B>) = {]),))}

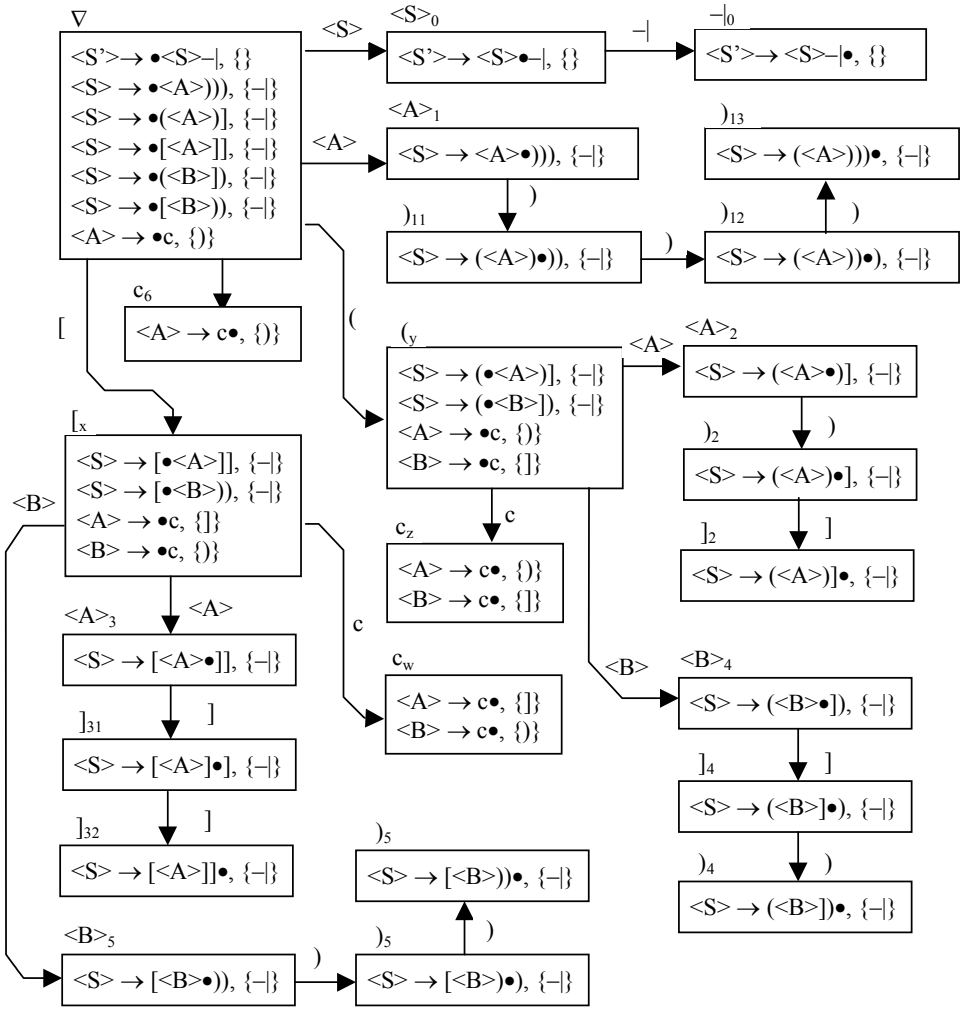
S obzirom da oba skupa sadrže)), pojavljuje se REDUCE-REDUCE konflikt u kritičnom stanju, pa zaključujemo da gramatika nije SLR(2).

$FOLLOW_3$ skupovi neterminala <A> i <B> su sledeći:

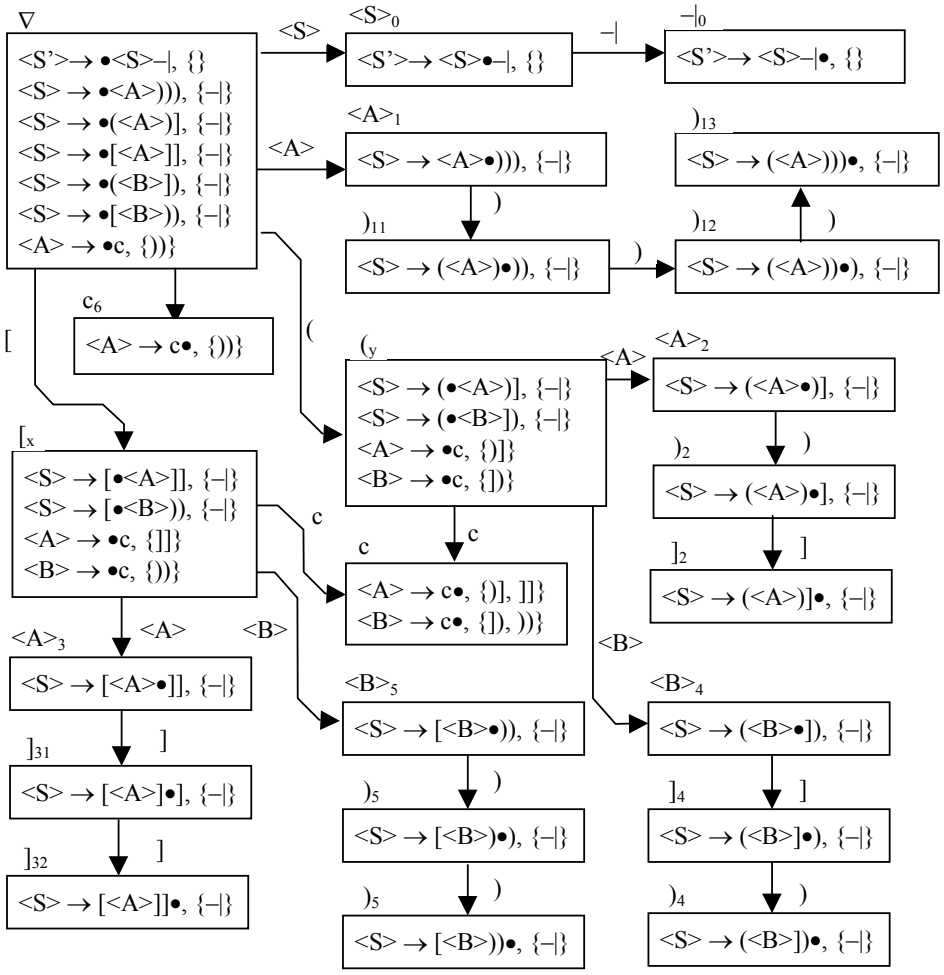
$FOLLOW_3$(<A>) = {))),)]–|,]]–|}

$FOLLOW_3$(<B>) = {])–|,))–|}

S obzirom da su skupovi disjunktni, u kritičnom stanju ne dolazi do konflikta, pa zaključujemo da je u pitanju SLR(3) gramatika čime su zadovoljeni uslovi zadatka.



Sl. 4.2.14: Karakteristični LR(1) automat



Sl. 4.2.15: Karakteristični LALR(2) automat

Zadatak 4.2.5

Razmotrimo sledeću gramatiku koja ima $O(n^2)$ smena:

$<S> \rightarrow <X_i>z_i \quad 1 \le i \le n$

$<X_i> \rightarrow y_j<X_i> \quad 1 \le i, j \le n, \quad i \ne j$

$<X_i> \rightarrow y_j \quad 1 \le i, j \le n, \quad i \ne j$

a) Pokazati da karakteristični LR(0) automat za ovu gramatiku ima $O(2^n)$ stanja.

b) Ispitati da li je data gramatika SLR(1).

Analiza problema

Za opis konstrukcije karakterističnog LR(0) automata i definiciju SLR(1) gramatika pogledati Zadatak 4.2.1.

Rešenje

a)

Označimo sa $P_{a1,a2,...,am}$, $1 \le m \le n$, $1 \le a_i \le n$, $a_i \ne a_j$ za $i \ne j$, proizvoljan podskup skupa neterminala $\{<X_i> \mid 1 \le i \le n\}$ koji NE sadrži neterminale $<X_{a1}>, <X_{a2}>, ..., <X_{an}>$:

$$P_{a1,a2,...,an} = \{<X_i> \mid 1 \le i \le n\} - \{<X_{a1}>, <X_{a2}>, ..., <X_{an}>\}$$

Pokažimo da se iz startnog stanja karakterističnog LR(0) automata za datu gramatiku, za proizvoljnu ulaznu sekvencu:

$y_{a1}y_{a2}...y_{am}$ $\quad 1 \le m \le n,\ 1 \le a_i \le n,\ a_i \ne a_j$ za $i \ne j$

dolazi u stanje, nazovimo ga $S_{a1,a2,...,am}$ koje sadrži sledeće konfiguracije:

$<X_i> \to y_{am}\bullet<X_i>$ $\quad <X_i> \in S_{a1,a2,...,am}$

$<X_i> \to y_{am}\bullet$ $\quad <X_i> \in S_{a1,a2,...,am}$

$<X_i> \to \bullet y_k<X_i>$ $\quad <X_i> \in S_{a1,a2,...,am}$ $\quad 1 \le k \le n,$ $\quad k \ne i$

$<X_i> \to \bullet y_k$ $\quad <X_i> \in S_{a1,a2,...,am}$ $\quad 1 \le k \le n,$ $\quad k \ne i$

Dokaz se može izvesti indukcijom po dužini m ulazne sekvence. Startno stanje karakterističnog LR(0) automata za datu gramatiku sadrži sledeći skup konfiguracija:

$<S'> \to \bullet<S>\dashv$

$<S> \to \bullet<X_i>z_i$ $\quad 1 \le i \le n$

$<X_i> \to \bullet y_j<X_i>$ $\quad 1 \le i, j \le n,$ $\quad i \ne j$

$<X_i> \to \bullet y_j$ $\quad 1 \le i, j \le n,$ $\quad i \ne j$

Prelaz iz startnog stanja po y_{a1} je u stanje sa sledećim skupom konfiguracija:

$<X_i> \to y_{a1}\bullet<X_i>$ $\quad 1 \le i \le n,$ $\quad i \ne a_1$

$<X_i> \to y_{a1}\bullet$ $\quad 1 \le i \le n,$ $\quad i \ne a_1$

$<X_i> \to \bullet y_k<X_i>$ $\quad 1 \le i, k \le n,\ i \ne a_1,\ k \ne i$

$<X_i> \to \bullet y_k$ $\quad 1 \le i, k \le n,\ i \ne a_1,\ k \ne i$

Radi se o stanju koje po uvedenoj konvenciji označavamo sa S_{a1}, čime je okončan dokaz za slučaj m=1. Prelaz iz stanja S_{a1} po y_{a2} je u stanje sa sledećim skupom konfiguracija:

$<X_i> \to y_{a2}\bullet<X_i>$ $\quad 1 \le i \le n,$ $\quad i \ne a_1,$ $\quad i \ne a_2$

$<X_i> \to y_{a2}\bullet$ $\quad 1 \le i \le n,$ $\quad i \ne a_1,$ $\quad i \ne a_2$

$<X_i> \to \bullet y_k<X_i>$ $\quad 1 \le i, k \le n,\ i \ne a_1,\ i \ne a_2,\ k \ne i$

$<X_i> \rightarrow \bullet y_k \qquad 1 \le i, k \le n, i \ne a_1, i \ne a_2, k \ne i$

Vidimo da se radi o stanju koje po uvedenoj konvenciji označavamo sa S_{a1a2}, čime je tvrđenje dokazano za m=2. Čitaocu se ostavlja da sam sprovede indukcioni korak od dužine ulazne sekvence k na dužinu k+1.

Koliko ukupno ima različitih stanja $S_{a1,a2,...,am}$ karakterističnog automata? S obzirom da svako stanje jednoznačno identifikuju bazične konfiguracije:

$<X_i> \rightarrow y_{am}\bullet<X_i> \qquad <X_i> \in S_{a1,a2,...,am}$

$<X_i> \rightarrow y_{am}\bullet \qquad <X_i> \in S_{a1,a2,...,am}$

svako stanje jednoznačno odgovara jednom podskupu $P_{a1,a2,...,am}$. S obzirom da skup svih neterminala $<X_i>$ ima ukupno n elemenata, broj svih podskupova ovoga skupa je 2^n.

Ostala stanja karakterističnog automata, u koja se dolazi ili iz startnog stanja ili iz nekog od stanja $S_{a1,a2,...,am}$ sadrže sledeće skupove konfiguracija:

$\{<S'> \rightarrow <S>\bullet\dashv\}, \ \{<S'> \rightarrow <S>\dashv\bullet\}$; ukupno 2 stanja

$\{<S> \rightarrow <X_i>\bullet z_i \mid 1 \le i \le n\}, \{<S> \rightarrow <X_i>z_i\bullet \mid 1 \le i \le n\}$; ukupno 2n stanja

$\{<X_i> \rightarrow y_j<X_i>\bullet\}, 1 \le i, j \le n, i \ne j$; ukupno n(n–1) stanja

$\{<X_i> \rightarrow y_j\bullet\}, 1 \le i, j \le n, i \ne j$; ukupno n(n–1) stanja

Prema tome, broj stanja $S_{a1,a2,...,am}$ određuje red veličine ukupnog broja stanja automata koji iznosi 2^n.

b)

Za utvrđivanje da li je gramatika SLR(1) nema potrebe razmatrati stanja koja sadrže samo jednu konfiguraciju, jer se kod njih ne može pojaviti konflikt. Konflikt se potencijalno može javiti jedino u stanju koje sadrži sve ili neke od konfiguracija oblika:

$<X_i> \rightarrow y_j\bullet<X_i> \qquad 1 \le i, j \le n, \qquad i \ne j$

$<X_i> \rightarrow y_j\bullet \qquad 1 \le i, j \le n, \qquad i \ne j$

Prva konfiguracija sugeriše akciju SHIFT, a druga akciju REDUCE u odgovarajućoj vrsti kontrolne tabele. Za svako y_j, $1 \le j \le n$, ulaz u odgovarajućoj koloni kontrolne tabele ne može sadržati REDUCE, jer $y_j \notin$ FOLLOW($<X>_i$), za bilo koje i. Za svako z_j, $1 \le j \le n$, ulaz u odgovarajućoj koloni kontrolne tabele ne može sadržati SHIFT, s obzirom da se z_j pojavljuju isključivo kao krajnje desni simboli u smenama. Time je pokazano da se SHIFT-REDUCE konflikt ne može desiti ni za jedan ulazni simbol, pa gramatika jeste SLR(1).

4.3. Hibridni pristup

Zadatak 4.3.1

Data je sledeća gramatika:

1. <S> → a b <A>
2. <S> → <A> <C>
3. <A> → a <C> b
4. <A> → ε
5. <C> → c <A>
6. <C> → d

d) Izračunati proširene leve uglove za datu gramatiku.

e) Konstruisati karakteristični ELC automat – detektor levih uglova.

f) Konstruisati potisnu i kontrolnu tabelu hibridnog ELC/LALR(1) parsera.

g) Prikazati rad parsera za ulaznu sekvencu adbc.

Analiza problema

ELC parsiranje (*Extended Left-Corner Parsing* ili *Enhanced Left-Corner Parsing*) zasniva se na pojmu *proširenog levog ugla* smene. Formalna definicija ovog pojma data je u nastavku rešenja, a neformalno se pojam proširenog levog ugla smene definiše na sledeći način:

Desna strana proizvoljne gramatičke smene može se podeliti na *prošireni levi ugao* i *prateći deo*. Po definiciji, da bi parser koji radi na principu 'od dna ka vrhu' jednoznačno prepoznao upotrebu smene, neophodan i dovoljan uslov je da konzumira kompletni prošireni levi ugao te smene.

ELC parser procesira proširene leve uglove smena u maniru 'od dna ka vrhu' tj. potiskujući simbole na stek dok se na vrhu steka ne kompletira levi ugao. Tada ELC parser prepoznaje smenu, skida sa steka njen levi ugao a na stek stavlja simbole koji odgovaraju pratećem delu te smene tako da se prateći deo procesira u maniru 'od vrha ka dnu' uparivanjem sadržaja steka sa tekućim ulazom.

Za procesiranje proširenih levih uglova smena ELC parser koristi metod potiskivanja i svođenja (*shift-reduce*). Kao što je poznato, postoji više klasa tzv. *shift-reduce* parsera koji odgovaraju različitim klasama LR gramatika. Kao posledica toga, može se definisati više klasa ELC parsera, u zavisnosti od toga koji LR metod se koristi za procesiranje proširenih levih uglova. Korišćeni LR metod pri tome definiše klasu LR gramatika za koje se može konstruisati odgovarajući ELC parser. Tako se, na primer ELC/LALR(1) parser može konstruisati za one i samo one gramatike koje pripadaju LALR(1) klasi gramatika (bez prethodnih transformacija gramatike).

Konstrukcija ELC parsera sastoji se iz tri koraka:

1. Određivanje proširenih levih uglova za zadatu gramatiku

2. Konstrukcija ELC automata - detektora levih uglova; analogno LALR(1) parsiranju, uloga ovoga automata je da tokom parsiranja detektuje trenutke kada je na vrhu steka kompletiran levi ugao neke od smena. Tada parser treba da izvrši REDUCELC akciju.
3. Konstrukcija parserskih tabela na osnovu ELC automata.

Za sprovođenje prvog i drugog koraka potrebno je konstruisati LALR(1) automat za zadatu gramatiku.

a)

Određivanje proširenih levih uglova

Nekoliko pomoćnih pojmova uvodi se radi formalne definicije proširenih levih uglova:

Dešavanje gramatičkog simbola X u smeni <Y> $\rightarrow \alpha$ X β je *kandidat za prateći deo smene* ako i samo ako je konfiguracija <Y> $\rightarrow \alpha$ X $\bullet$ β jedina bazična LR(0) konfiguracija u svim stanjima LALR(1) automata u kojima se ova konfiguracija pojavljuje.

Svaka smena, izuzev praznih, poseduje *granično dešavanje* gramatičkog simbola. To je početno dešavanje najdužeg sufiksa desne strane smene koji se sastoji isključivo od kandidat-dešavanja.

Po definiciji, granično dešavanje razgraničava prošireni levi ugao smene od njenog pratećeg dela. Granično dešavanje pripada pratećem delu smene ako i samo ako je reč o:

- dešavanju terminala ili
- dešavanju neterminala <X> pri čemu sve smene za <X> imaju prazne leve uglove.

Prethodna definicija je rekurzivna; zato se gramatičke smene razmatraju takvim redosledom da se smene sa neterminalom <X> na levoj strani razmatraju pre smena sa <Y> na levoj strani ako važi da je <X> $\in$ FIRST(<Y>). Pri tome levo rekurzivni neterminali ne prave probleme jer se sva njihova dešavanja uvek nalaze u levim uglovima smena.

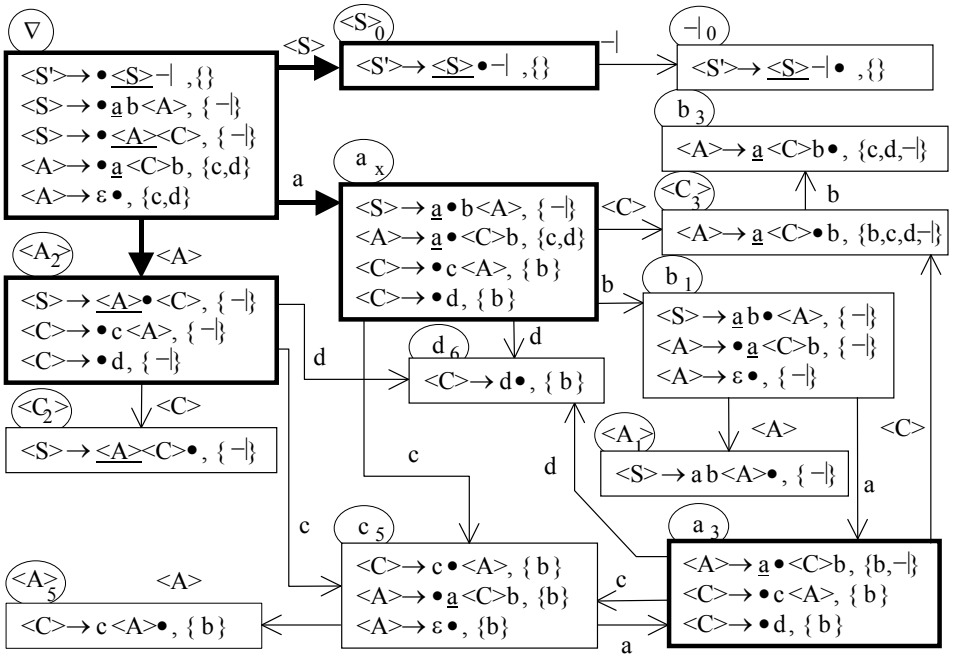
Po definiciji se usvaja da levi ugao nulte smene obuhvata krajnje levi neterminal njene desne strane, bez ozbira što se primenom prethodno opisane procedure može za ovu smenu dobiti i prazan levi ugao. Na ovaj način obezbeđuje se da ELC parser uvek počinje parsiranje u režimu 'od dna ka vrhu' tako da se kontrolna tabela konstruiše na unforman način.

Rešenje

Na Sl. 4.3.1 prikazan je karakteristični LALR(1) automat za datu gramatiku prethodno proširenu smenom:

0. <S'> $\rightarrow$ <S> $\dashv$

Pregledom stanja $<S>_0$ i $\dashv_0$ zaključujemo da su sva dešavanja 0. smene kandidat–dešavanja. Pregledom stanja a_x, b_1 i $<A>_1$ zaključujemo da dešavanje simbola a u prvoj smeni nije kandidat–dešavanje, a ostala dešavanja jesu. U drugoj smeni, sva dešavanja jesu kandidat–dešavanja što zaključujemo pregledom stanja $<A>_2$ i $<C>_2$. U trećoj smeni, dešavanje simbola a nije kandidat–dešavanje, a ostala dešavanja jesu, na osnovu stanja a_x, $<C>_3$ i b_3. Četvrta smena je prazna, tako da ne poseduje kandidat–dešavanja, dok su sva dešavanja 5. i 6. smene kandidat–dešavanja, na osnovu stanja c_5, $<A>_5$ i d_6.



Sl. 4.3.1: Karakteristični LALR(1) automat

Granična dešavanja pojedinih smena prikazana su podebljano:

0. <S'> → **<S> ⊣** 4. <A> → ε
1. <S> → a **b** <A> 5. <C> → **c** <A>
2. <S> → **<A>** <C> 6. <C> → **d**
3. <A> → a **<C>** b

Granična dešavanja 1., 5. i 6. smene pripadaju pratećem delu, jer se radi o terminalima. Granično dešavanje 0. smene pripada po definiciji levom uglu te smene. Granično dešavanje 2. smene pripada levom uglu, jer treća smena, koja ima neterminal <A> na levoj strani, očigledno nema prazan levi ugao. Granično dešavanje 3. smene pripada pratećem delu smene, jer sve smene sa neterminalom <C> na levoj strani imaju prazne leve uglove.

Prošireni levi uglovi prikazani su podvučeno:

0. <S'> → <u><S></u> ⊣ 4. <A> → ε

1. <S> → <u>a</u> b <A> 5. <C> → c <A>

2. <S> → <u><A></u> <C> 6. <C> → d

3. <A> → <u>a</u> <C> b

b)

Konstrukcija detektora levih uglova

Za LALR(1) gramatike, ELC automat dobija se sledećom modifikacijom LALR(1) automata:

- Iz grafa prelaza LALR(1) automata ukloniti sva stanja koja sadrže bazične konfiguracije sa tačkom unutar pratećeg dela odgovarajuće smene (u to se ne ubrajuju konfiguracije oblika <X> → <u>α</u> • β gde β predstavlja prateći deo smene <X> → αβ). Preostala stanja LALR(1) automata nazivaćemo *ugaonim stanjima*. Ova stanja jednoznačno odgovaraju ugaonim simbolima steka ELC parsera.
- U modifikovani graf dodati po jedan čvor za svaki gramatički simbol koji se pojavljuje u nekom od pratećih delova smena (bez ozbira da li se isti simbol pojavljuje i u levim uglovima). Na ovaj način dobijaju se *predikciona* stanja koja jednoznačno odgovaraju predikcionim simbolima steka ELC parsera.
- Dodati čvorovi povezuju se sa ostatkom grafa na sledeći način:

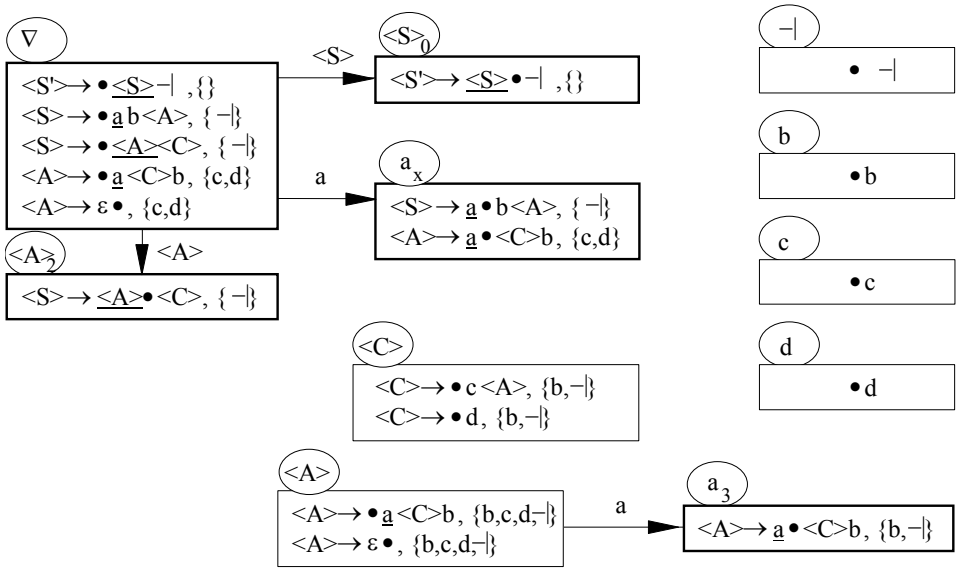
  (1) Ako čvor odgovara terminalnom simbolu u pratećem delu smene, ili neterminalu koji se u pratećim delovima smena javlja isključivo kao granični neterminal, nijedna veza nije potrebna. Neformalno, veze su potrebne u režimu rada parsera 'od dna ka vrhu', a ovakvi gramatički simboli se obrađuju strogo u režimu 'od vrha ka dnu'.

  (2) (Za čvorove neterminala koji ne spadaju u pravilo 1): Za dati neterminal <X>, izabrati proizvoljnu smenu <Y> → <u>α</u>β<X>γ koja sadrži pojavu <X> striktno u pratećem delu (tj. β nije prazan niz). U originalnom LALR(1) detektoru pronaći stanje S sa konfiguracijom <Y> → <u>α</u>β • <X>γ. Posmatrani čvor u modifikovanom grafu posedovaće sve prelaze u ugaona stanja koje poseduje i stanje S u originalnom LALR(1) automatu. Prelazi iz stanja S koji ne predstavljaju prelaze u ugaona stanja ne uzimaju se u obzir. Dokazuje se da efekat ovog pravila ne zavisi od smene koju izaberemo.
- Svako od dodatih stanja opisano je unijom skupova konfiguracija svih stanja S koja se definišu u prethodnoj tački pri čemu su iz tih stanja uklonjene sve bazične konfiguracije.
- Posle primene prethodnih koraka iz svih stanja se uklanjaju konfiguracije K za koje važi: Konfiguracija K je dodata primenom operacije zatvaranja na neku konfiguraciju oblika <Y>→<u>α</u>β•<X>γ pri čemu je α levi ugao smene <Y>→<u>α</u>β<X>γ.

Može se pokazati da je dobijeni automat za slučaj LALR(1) gramatika deterministički.

Rešenje

Na Sl. 4.3.1 ugaona stanja LALR(1) automata prikazana su punom linijom, ostala koja se uklanjaju pri konstrukciji ELC automata prikazana su isprekidanom linijom. ELC automat prikazan je na Sl. 4.3.2. Punom linijom prikazana su ugaona stanja, a isprekidanom dodatna predikciona stanja. Prelaz iz stanja <A> u a_3, dodat je posmatranjem bilo stanja c_5 bilo stanja b_1 LALR(1) detektora.



Sl. 4.3.2: ELC/LALR(1) automat

c)

Konstrukcija potisne i kontrolne tabele

Skup ulaznih simbola parsera odgovara skupu svih terminalnih simbola proširene gramatike. Ulazni simboli označavaju kolone kontrolne tabele.

Simboli steka označavaju vrste kako potisne, tako i kontrolne tabele parsera i jednoznačno odgovaraju stanjima ELC automata.

Ulazi potisne tabele mogu da sadrže:

- REJECT što označava da ulazna sekvenca nije u jeziku specificiranom proširenom gramatikom;
- oznaku simbola steka koji treba potisnuti na vrh steka;
- POP što označava skidanje vršnog simbola sa steka.

Ulazi kontrolne tabele mogu da sadrže sledeće akcije:

- REJECT sa istim značenjem kao ranije;

- ACCEPT što označava kraj parsiranja legalnog ulaznog niza;
- SHIFT označava sledeće akcije:
  (1) PUSHT(*tekući ulazni simbol*) - ažuriranje steka u skladu sa ulazom potisne tabele za tekući simbol steka i tekući ulazni simbol.
  (2) ADVANCE - konzumiranje tekućeg ulaznog simbola i prelazak na sledeći ulazni simbol.
- REDUCELC(*i*), gde *i* predstavlja redni broj smene, izaziva sledeće akcije:
  (1) sa vrha steka se uklanja (POP) onoliko stek simbola koliko se gramatičkih simbola nalazi u proširenom levom uglu *i*-te smene. Drugim rečima, prošireni levi ugao *i*-te smene se uklanja sa steka;
  (2) akcija PUSHT(*neterminal leve strane i-te smene*) ažurira vrh steka u skladu sa ulazom potisne tabele u vrsti označenoj vršnim simbolom steka (posle primene POP akcija) i koloni označenoj neterminalom leve strane *i*-te smene.
  (3) predikcioni stek simboli koji neposredno predstavljaju gramatičke simbole pratećeg dela *i*-te smene (dakle stek simboli iz drugog podskupa skupa svih stek simbola) se potiskuju na stek, pri čemu se krajnje desni simbol smene potiskuje na stek prvi. Pri ovome se NE konsultuje potisna tabela, tako da će ova akcija u narednim primerima biti označena sa PUSH.

Potisna i kontrolna tabela parsera konstruišu se na bazi ELC automata za zadatu gramatiku. Svakom simbolu steka ELC parsera jednoznačno odgovara stanje ELC automata. Potisna tabela dobija se iz tabele prelaza ELC automata uz sledeće modifikacije:

- Vrsta tabele prelaza za stanje koje odgovara markeru kraja ulazne sekvence ⊣ eliminiše se pošto se nikad ne potiskuje na stek (ACCEPT akcija sledi odmah po prepoznavanju levog ugla nulte smene)
- Za svako predikciono stanje S koje odgovara gramatičkom simbolu X, u vrstu S i kolonu X stavlja se POP. Ova akcija se preduzima po uparivanju simbola X sa ulaznim podnizom (režim od vrha ka dnu).
- Svi prelazi u stanje greške kod ELC automata interpretiraju se kao REJECT akcije u potisnoj tabeli.

Kontrolna tabela popunjava se prema sledećim pravilima:

1. Ulaz kontrolne tabele popunjava se SHIFT akcijom ako i samo ako ulaz potisne tabele za isti red i kolonu ne sadrži REJECT.
2. Neka X predstavlja proizvoljni simbol steka kome odgovara stanje ELC automata sa konfiguracijom oblika $<A> \rightarrow \underline{\alpha} \bullet \beta$, u koja pripada i-toj smeni (zapaziti da se tačka nalazi između levog ugla α i pratećeg dela β). U ovom slučaju, svaki ulaz tabele u vrsti X i svakoj od kolona $y \in FIRST(\beta u)$ popunjava se sa REDUCELC(i) za $i \neq 0$, odnosno sa ACCEPT za $i = 0$.
3. Svi ulazi kontrolne tabele koji nisu obuhvaćeni pravilima 1. i 2. sadrže REJECT.

U slučaju da, posle primene gornjih pravila, svaki ulaz kontrolne tabele sadrži jedinstvenu akciju, procedura konstrukcije je uspešno okončana. U suprotnom, gramatika ne pripada klasi LALR(1).

Rešenje

Na Sl. 4.3.3 prikazane su potisna i kontrolna tabela ELC parsera za zadatu gramatiku. Prazni ulazi potisne tabele sadrže REJECT akciju. REDUCELC akcije (skraćeno obeležene sa RLC) imaju sledeći izgled:

REDUCELC(1): POP, PUSHT(<S>), PUSH(<A>), PUSH(b)

REDUCELC(2): POP, PUSHT(<S>), PUSH(<C>)

REDUCELC(3): POP, PUSHT(<A>), PUSH(b), PUSH(<C>)

REDUCELC(4): PUSHT(<A>)

REDUCELC(5): PUSHT(<C>), PUSH(<A>), PUSH(c)

REDUCELC(6): PUSHT(<C>), PUSH(d)

Parser započinje rad u režimu 'od dna ka vrhu'. Prema tome startna konfiguracija steka sastoji se od praznog steka, tj. na vrhu steka se nalazi simbol dna steka ∇.

	<S>	<A>	<C>	a	b	c	d	a	b	c	d	⊣
∇	$<S>_0$	$<A>_2$		a_x				SHIFT		RLC(4)	RLC(4)	
$<S>_0$												ACC
$<A>_2$										RLC(2)	RLC(2)	
a_x									RLC(1)	RLC(3)	RLC(3)	
a_3										RLC(3)	RLC(3)	
<A>		POP		a_3				SHIFT	RLC(4)	RLC(4)	RLC(4)	RLC(4)
<C>			POP							RLC(5)	RLC(6)	
b					POP				SHIFT			
c						POP				SHIFT		
d							POP				SHIFT	

potisna tabela kontrolna tabela

Sl. 4.3.3: ELC/LALR(1) parser

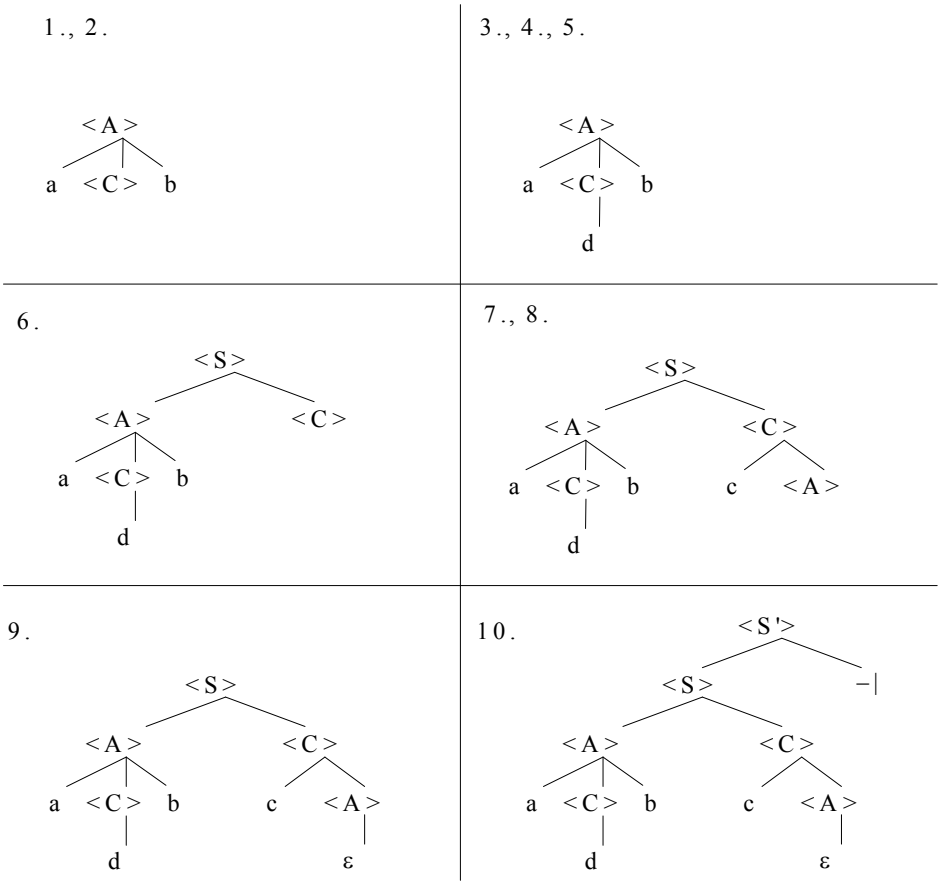
d)

Razmotrimo kako parser sa Sl. 4.3.3 procesira ulazni niz abac:

	sadržaj steka	preostali ulaz	akcija parsera
1.	∇	adbc⊣	SHIFT
2.	∇ a_x	dbc⊣	REDUCELC(3)
3.	∇$<A>_2$ b <C>	dbc⊣	REDUCELC(6)
4.	∇$<A>_2$ b d	dbc⊣	SHIFT
5.	∇$<A>_2$ b	bc⊣	SHIFT

6.	$\nabla\langle A\rangle_2$	c⊣	REDUCELC(2)
7.	$\nabla\langle S\rangle_0\langle C\rangle$	c⊣	REDUCELC(5)
8.	$\nabla\langle S\rangle_0\langle A\rangle$ c	c⊣	SHIFT
9.	$\nabla\langle S\rangle_0\langle A\rangle$	⊣	REDUCELC(4)
10.	$\nabla\langle S\rangle_0$	⊣	ACCEPT

Akcije parsera u prethodnom primeru odgovaraju građenju stabla izvođenja na način prikazan na Sl. 4.3.4. Prikazan je izgled otkrivenog dela stabla izvođenja u svakom od koraka parsiranja iz prethodnog primera.



Sl. 4.3.4: Ilustracija rada ELC/LALR(1) parsera

Diskusija

Na osnovu prethodnog primera vidi se da sekvenca primena smena koju prepoznaje ELC parser ne odgovara ni krajnje levom izvođenju ulaznog niza (kao što je to slučaj kod LL parsera) niti

obrnutom krajnje desnom izvođenju (kao što je slučaj kod LR parsera). Za ELC parser važi da se prateći delovi smena prepoznaju na način koji odgovara njihovom krajnje levom izvođenju, dok se prošireni levi uglovi prepoznaju na način koji odgovara obrnutom krajnje desnom izvođenju.

U slučaju LALR(1) gramatika koje su takođe LL(1), prošireni levi uglovi svih smena su prazni (sa izuzetkom nulte smene gde se startni neterminal eksplicitno ubacuje u ugao). U ovom slučaju, potisna tabela ELC parsera poseduje isključivo POP akcije u nepraznim ulazima, sa izuzetkom ulaza (∇, <S>) koji je popunjen sa $<S>_0$. Potisnu tabelu je moguće u potpunosti eliminisati, zamenom PUSHT akcija u REDUCELC operacijama odgovarajućim akcijama iz potisne tabele, dok je kontrolna tabela istog izgleda kao kod LL(1) parsera, osim što ima jednu vrstu više (radi se o simbolu steka $<S>_0$) s obzirom na odluku da parser uvek započinje rad u režimu od dna ka vrhu.

Postoje LL(1) gramatike koje nisu LALR(1), na primer, gramatika sa Sl. 4.3.5(a), izaziva svedi–svedi konflikt u stanju LALR(1) automata prikazanom na Sl. 4.3.5(b), tako da nije moguće napraviti kako LALR(1) parser, tako ni ELC/LALR(1) parser.

1. <S> → a<A>
2. <S> → b<B>
3. <A> → <C>a
4. <A> → <D>b
5. <B> → <C>b
6. <B> → <D>a
7. <C> → <E>
8. <D> → <E>
9. <E> → ε

(a)

<C>→ <E>• , {a, b}
<D>→ <E>• , {a, b}

(b)

Sl. 4.3.5: Primer LL(1) gramatike koja nije LALR(1)

Zadatak 4.3.2

Za sledeću gramatiku konstruisati hibridni ELC/LR(1) parser.

0. <S'> → <S> ¬
1. <S> → <C> a <C> a
2. <S> → <D> b <C>
3. <S> → <S> <C> b
4. <S> → <S> <D> a
5. <C> → c
6. <D> → c
7. <D> → ε

Analiza problema

Konstrukcija ELC/LR(1) parsera sastoji se iz tri koraka:

1. Određivanje proširenih levih uglova za zadatu gramatiku.
2. Konstrukcija karakterističnog ELC/LR(1) automata - detektora levih uglova.
3. Konstrukcija parserskih tabela na osnovu karakterističnog automata.

Određivanje proširenih levih uglova

Procedura određivanja proširenih levih uglova opisana je u prethodnom zadatku.

Konstrukcija karakterističnog automata

Procedura konstrukcije karakterističnog ELC/LR(1) automata glasi:

1. Za startni neterminal <S'> i svaki neterminal koji se bar jednom pojavljuje u pratećim delovima smena, formirati po jedno stanje, takozvano *predikciono stanje*. Predikciono stanje neterminala <X> je skup svih LR(1) konfiguracija oblika $<X> \rightarrow \bullet\alpha, u$ gde je $<X> \rightarrow \alpha$ gramatička smena, a $u \in$ FOLLOW(<X>).
2. Na svako od stanja iz tačke 1 primeniti operaciju ELCClosure1.
3. Formirati skup Set koji se sastoji od stanja dobijenih u tački 2. Dok skup Set ne postane prazan raditi sledeće:
   3.1. Udaljiti proizvoljno stanje S iz skupa Set.
   3.2. Za svaki gramatički simbol X, primeniti operaciju ELCGoTo1(S,X). Ako rezultat primene procedure nije prazan skup konfiguracija, rezultatom je definisano novo stanje ELC/LR(1) detektora - naslednik stanja S po simbolu X. Ako novo stanje nije jednako nekom od postojećih, ubaciti novo stanje u skup Set.
4. Za svaki terminalni simbol koji se bar jednom pojavljuje u pratećim delovima smena, formira se po jedno stanje. Ova stanja ne sadrže LR(1) konfiguracije.

Procedure ELCGoTo1 i ELCClosure1 predstavljaju modifikovane verzije procedura GoTo1 i Closure1 koje se koriste u konstrukciji karakterističnog LR(1) automata (Zadatak 4.2.2). Modifikacija (označena podebljano) se ogleda u tome što u proceduri konstrukcije ne učestvuju sve LR(1) konfiguracije, već samo one kod kojih se tačka nalazi unutar levog ugla (zaključno sa konfiguracijama kod kojih se tačka nalazi između levog ugla i pratećeg dela smene).

Operacija Closure1 nad skupom LR(1) konfiguracija S:

1. Dok postoji promena u skupu S raditi sledeće:
   1.1. Izabrati iz skupa S konfiguraciju oblika $<X> \rightarrow \alpha \bullet <Y> \beta, u$ **pri čemu neterminal <Y> pripada levom uglu smene** $<X> \rightarrow \alpha <Y> \bullet \beta$.
   1.2. Dodati u skup S sve konfiguracije oblika $<Y> \rightarrow \bullet \gamma, v$ gde je $v \in$ FIRST(βu)

Operacija GoTo1 nad skupom LR(1) konfiguracija S i gramatičkim simbolom X konstruiše skup LR(1) konfiguracija S':

1. Neka je S' prazan skup.
2. Za svaku od konfiguracija oblika $<X> \rightarrow \alpha \bullet Y \beta, u$ iz skupa S **pri čemu se simbol Y nalazi unutar levog ugla smene** $<X> \rightarrow \alpha Y \beta$,
   2.1. Dodati u skup S' konfiguraciju oblika $<X> \rightarrow \alpha Y \bullet \beta, u$.
3. Primeniti operaciju Closure1 na skup S'.

Za razliku od ELC/LALR(1) parsera, gde se karakteristični ELC automat formira na osnovu kompletnog LALR(1) automata, u ovom pristupu izbegnuta je konstrukcija LR(1) automata jer bi takav pristup u opštem slučaju bio neefikasan zbog veličine LR(1) automata.

Konstrukcija potisne i kontrolne tabele

Potisna i kontrolna tabela parsera konstruišu se na bazi ELC automata za zadatu gramatiku. Svakom simbolu steka ELC parsera jednoznačno odgovara stanje ELC automata. Potisna tabela dobija se iz tabele prelaza ELC automata uz sledeće modifikacije:

- Vrsta tabele prelaza za stanje koje odgovara markeru kraja ulazne sekvence ⊣ elimiše se pošto se nikad ne potiskuje na stek (ACCEPT akcija sledi odmah po prepoznavanju levog ugla nulte smene)
- Za svako predikciono stanje X koje odgovara gramatičkom simbolu X, u vrstu X i kolonu X stavlja se POP. Ova akcija se preduzima po uparivanju simbola X sa ulaznim podnizom (režim od vrha ka dnu).
- Svi prelazi u stanje greške kod ELC automata interpretiraju se kao REJECT akcije u potisnoj tabeli.

Kontrolna tabela popunjava se prema sledećim pravilima:

1. Ulaz kontrolne tabele popunjava se SHIFT akcijom ako i samo ako ulaz potisne tabele za isti red i kolonu ne sadrži REJECT.
2. Neka X reprezentuje proizvoljni simbol steka kome odgovara stanje ELC automata sa konfiguracijom oblika $\langle A\rangle \rightarrow \underline{\alpha} \bullet \beta$, u koja pripada i-toj smeni (zapaziti da se tačka nalazi između levog ugla α i pratećeg dela β). U ovom slučaju, svaki ulaz tabele u vrsti X i svakoj od kolona $y \in FIRST(\beta u)$ popunjava se sa REDUCELC(i) za $i \neq 0$, odnosno sa ACCEPT za $i = 0$.
3. Svi ulazi kontrolne tabele koji nisu obuhvaćeni pravilima 1. i 2. sadrže REJECT.

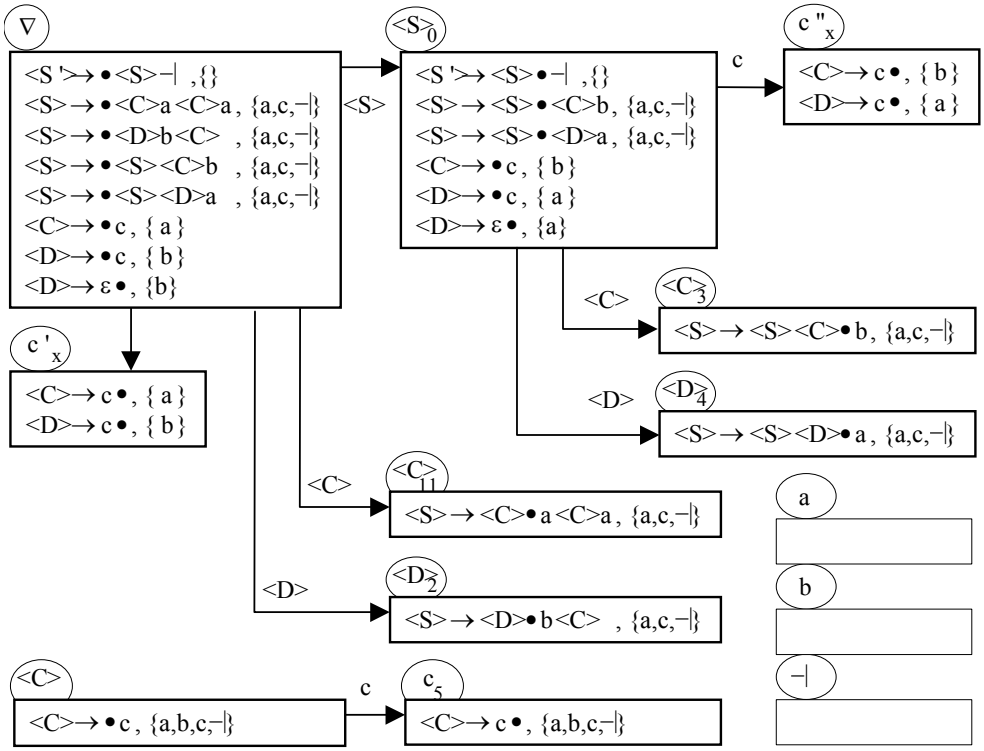
U slučaju da, posle primene gornjih pravila, svaki ulaz kontrolne tabele sadrži jedinstvenu akciju, procedura konstrukcije je uspešno okončana. U suprotnom, gramatika ne pripada klasi LR(1).

Rešenje

Prošireni levi uglovi smena zadate gramatike prikazani su podvučeno:

0. $\langle S'\rangle \rightarrow \underline{\langle S\rangle} \dashv$
1. $\langle S\rangle \rightarrow \underline{\langle C\rangle}$ a $\langle C\rangle$ a
2. $\langle S\rangle \rightarrow \underline{\langle D\rangle}$ b $\langle C\rangle$
3. $\langle S\rangle \rightarrow \underline{\langle S\rangle\ \langle C\rangle}$ b
4. $\langle S\rangle \rightarrow \underline{\langle S\rangle\ \langle D\rangle}$ a
5. $\langle C\rangle \rightarrow \underline{c}$
6. $\langle D\rangle \rightarrow \underline{c}$
7. $\langle D\rangle \rightarrow \varepsilon$

Karakteristični ELC/LR(1) automat za ovu gramatiku prikazan je na Sl. 4.3.6. Potisna i kontrolna tabela parsera prikazane su na Sl. 4.3.7.



Sl. 4.3.6: Karakteristični ELC/LR(1) automat

	<S>	<C>	<D>	a	b	c
∇	$<S>_0$	$<C>_{11}$	$<D>_2$			c'_x
$<S>_0$		$<C>_3$	$<D>_4$			c''_x
$<C>_{11}$						
$<D>_2$						
$<C>_3$						
$<D>_4$						
c'_x						
c''_x						
c_5						
<C>		POP				c_5
a				POP		
b					POP	

(a) POTISNA TABELA

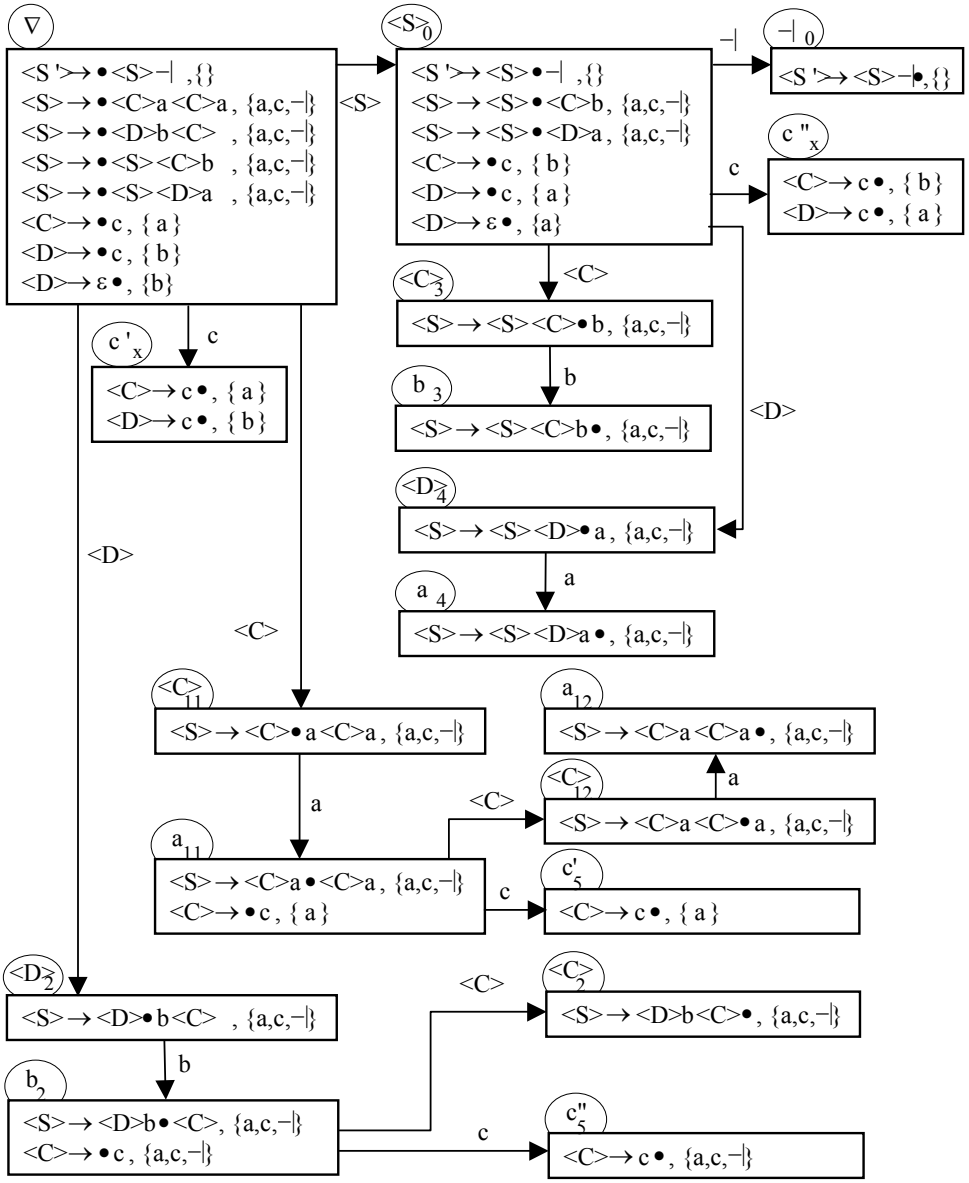
	a	b	c	⊣
∇		RLC(7)	SHIFT	
$<S>_0$	RLC(7)		SHIFT	ACC
$<C>_{11}$	RLC(1)			
$<D>_2$		RLC(2)		
$<C>_3$		RLC(3)		
$<D>_4$	RLC(4)			
c'_x	RLC(5)	RLC(6)		
c''_x	RLC(6)	RLC(5)		
c_5	RLC(5)	RLC(5)	RLC(5)	RLC(5)
<C>			SHIFT	
a	SHIFT			
b		SHIFT		

(b) KONTROLNA TABELA

Sl. 4.3.7: ELC/LR(1) parser

Diskusija

U svrhu poređenja sa LR(1) tehnikom, na Sl. 4.3.8 je prikazan LR(1) automat za datu gramatiku.



Sl. 4.3.8: Karakteristični LR(1) automat

Ovaj automat ima ukupno 18 stanja. Međusobnim spajanjem stanja c'_5 i c''_5 koja imaju isto jezgro, kao i stanja c'_x i c''_x dobio bi se karakteristični LALR(1) automat koji ima 16 stanja. Pri tome bi nastao konflikt u novom stanju c_x iz čega se vidi da gramatika nije LALR(1). Prema tome, ELC automat sa Sl. 4.3.6 koji ima 13 stanja, manji je kako od LR(1), tako i od LALR(1) automata. To pokazuje da je ELC/LR(1) tehnika superiornija od tehnike redukcije LR(1) automata po principu kompatibilnosti stanja, opisane u Fischer-u [6] koja spaja isključivo stanja sa istim jezgrom i u najboljem slučaju LR(1) automat može da svede na veličinu odgovarajućeg LALR(1) automata. Detaljnija diskusija na temu poređenja veličina parsera kod ovih tehnika može se naći u [4].

4. Prilozi

4.1. Lex, generator leksičkih analizatora

4.1.1. Uvod

Mnoge primene računara u sebi sadrže neku vrstu obrade teksta. Tipični primeri su leksička analiza kod kompilatora, razne statističke analize teksta (na primer program *wc* – "word count" na UNIX operativnim sistemima), ili pronalaženje i obradu delova teksta koji odgovaraju zadatom šablonu (na primer, alat *grep* za izdvajanje linija u tekstualnim datotekama na osnovu zadatog regularnog izraza, alat *awk* za obradu teksta).

U svim ovim slučajevima postavlja se problem grupisanja znakova sa ulaznog toka u pojedine leksičke jedinice.

Postupak za pisanje programa koji obavlja ovaj zadatak je potpuno definisan, i predstavlja jedan rutinski, nekreativan posao. Zbog toga su napravljeni različiti generatori koda koji ga automatizuju. Program *lex* je jedan od njih.

U nastavku teksta opisan je način upotrebe programa *lex*. Na početku se na nekoliko jednostavnih primera, bez zalaženja u detalje sintakse i semantike, ilustruje korišćenje *lex*-a. Zatim sledi formalni opis postupka pisanja leksičkih analizatora upotrebom ovog programa i na kraju detaljni opis svih njegovih mogućnosti koje su uglavnom taksativno nabrojane sa ponekim primerom. U daljem tekstu podrazumeva se da čitalac ima izvesna znanja o radu sa operativnim sistemom UNIX i programskim jezikom C.

4.1.2. Kako se upotrebljava lex

Primer: Iz ulaznog teksta izostaviti sva pojavljivanja teksta "programski prevodioci".

Princip rešavanja ovog problema sa programom *lex* je sledeći:

i) Pripremi se datoteka proizvoljnog imena (na primer izostavi.l) u kojoj se specificira leksička struktura ulaznog teksta. U našem slučaju ona se sastoji iz samo dva reda i ima sledeći izgled:

```
%%
```

“programski prevodioci” ;

U ulaznom tekstu traži se pojavljivanje sekvence *programski prevodioci* (bez navodnika koji služe samo za grupisanje reči s obzirom da sekvenca sadrži razmak). Pronađena sekvenca, s obzirom da nije navedena nikakva akcija (što je označeno korišćenjem znaka ;) biće ignorisana i neće biti prepisana na standardni izlaz, za razliku od preostalog dela ulaznog teksta koji se prepisuje, što je podrazumevano ponašanje.

ii) Datoteka izostavi.l se obradi komandom

lex izostavi.l

Rezultat prethodne obrade je datoteka lex.yy.c u tekućem katalogu.

iii) Sledećom komandom dobijamo gotov leksički analizator u formi filtra:

cc -o izostavi lex.yy.c -ll

Njen rezultat je izvršni program izostavi u tekućem katalogu.

Da bismo proverili rad našeg analizatora, pripremimo datoteku proba sa sledećim tekstom:

programski prevodioci i savremena ekonomija
su moji
omiljeni predmeti.

Komandom

izostavi <proba

na standardni izlaz će biti ispisan sledeći tekst:

i savremena ekonomija
su moji
omiljeni predmeti.

Iz ovog primera može se izvući nekoliko zaključaka. Prvo, razvoj ovakvih programa korišćenjem *lex*-a neuporedivo je jednostavniji (manja veličina koda i kraće vreme razvoja) u odnosu na rešavanje istog problema u klasičnom programskom jeziku. Efikasnost programa dobijena na ovaj način je sasvim zadovoljavajuća i ”ručnim kodiranjem” je nemoguće dobiti veća poboljšanja.

Drugo, pošto je poslednji korak u pisanju leksičkog analizatora bio aktiviranje prevodioca za C, može se zaključiti da je *lex* formirao C program na osnovu ulazne specifikacije (datoteka lex.yy.c). U daljem tekstu videćemo značaj ove činjenice.

Treće, iz načina upotrebe dobijenog leksičkog analizatora vidi se da se ulazni tekst čita sa standardnog ulaza i ispisuje na standardni izlaz. Kasnije će biti objašnjeno kako se ovo ponašanje može promeniti.

Primer: U ulaznom tekstu zameniti pojavljivanja reči “prevodioci” sa “kompilatori”.

lex program za ovu namenu ima sledeći izgled:

```
%%
prevodioci printf( “kompilatori”);
```

lz ovog primera vidimo da je u *lex* program moguće ubaciti sopstveni programski kod, u konkretnom slučaju poziv bibliotečke funkcije *printf*, pisan u jeziku C. Taj kod će biti ubačen na odgovarajuće mesto u izvornom C programu koji formira *lex* i aktiviran prilikom uparivanja ulaza sa zadatom sekvencom *prevodioci*.

Izvršni program dobijamo gore opisanim komandama i pokrećemo komandom:

```
zameni <proba
```

Kao rezultat obrade datoteke proba, na standardnom izlazu dobija se sledeći tekst

```
programski kompilatori i savremena ekonomija
su moji
omiljeni predmeti.
```

Primer: Izbrojati sva pojavljivanja slova ‘a’ u ulaznom tekstu i na njegovom kraju dati odgovarajući izveštaj.

Sledeća ulazna specifikacija formiraće leksički analizator sa traženom funkcijom:

```
        int x;
%%
a       {
        x++;
        putchar('a');
        }
%%
main() {
  yylex();
  printf(“\n%d slova a\n", x);
  }
```

Celobrojna promenljiva x uvećava se za 1 na svaku pojavu slova a u ulaznom tekstu (a se pri tome ispisuje na standardni izlaz, da bi ulazni tekst ostao neizmenjen).

U prethodna dva primera nismo specificirali našu main funkciju. Zbog toga je prilikom povezivanja programa korišćena ona koja se nalazi u biblioteci libl.a (to je prekidač -ll u pozivu C prevodioca). Ona je oblika

```
main() {
  yylex();
  }
```

i koristi se ukoliko ne dostavimo povezivaču našu verziju. Ovu, a i druge potrebne funkcije i deklaracije potrebnih globalnih promenljivih možemo da navedemo iza drugog znaka %%, kao u primeru. Naravno, ništa nas ne sprečava da ove funkcije smestimo u posebne datoteke, prevedemo ih nezavisno i povezemo sa predmetnom datotekom lex.yy.o. Drugo što može da se primeti je forma leksičkog analizatora koju daje *lex*. Vidimo da on ima oblik C funkcije sa imenom yylex().

Na osnovu ovog pregleda programa *lex* moguće je shvatiti osnovne principe njegovog korišćenja. U daljem tekstu sledi formalni opis strukture *lex* specifikacije.

4.1.3. Forma specifikacije u lex-u

Lex program u opštem slučaju ima sledeći oblik:

regularne definicije
%%
pravila
%%
dopunski programski kod na jeziku C

Nije potrebno da postoji svaki od ovih delova, ali bar jedan separator sekcije %% mora da se pojavi. Pre opisa sadržaja svake od sekcija, objasnićemo ukratko strukturu generisanog analizatora koji je u obliku izvornog programa.

Leksički analizator koji je proizveo program *lex* sastoji se iz jedne centralne funkcije i nekoliko pomoćnih koje ova poziva. Centralna funkcija je yylex(), i vraća rezultat tipa **int**. Funkcija yylex() realizuje konačni automat koji čita tekst sa standardnog ulaza dok ne upari deo teksta (jednu leksemu) nekim od korisnički definisanih pravila. Tada se izvršava korisnički definisana akcija (sve akcije nalaze se u okviru **switch** strukture gde se na bazi rednog broja pravila vrši izbor akcije). Ako se u okviru akcije upotrebi iskaz **return** sa celobrojnom povratnom vrednošću, funkcija završava rad i predaje kontrolu pozivaocu (koji će je ponovo pozvati za nastavak obrade ulaznog teksta, odnosno za prepoznavanje sledeće lekseme). Ako akcija ne sadrži iskaz **return**, posle završetka korisničke akcije u okviru funkcije yylex() prelazi se na prepoznavanje sledeće lekseme, bez povratka kontrole pozivaocu. Kada se celokupan ulazni tekst pročita, funkcija yylex() završava sa radom i vraća vrednost EOF.

4.1.3.1. Pravila

Pravila se sastoje iz dva dela. Prvi deo ide od početka linije i završava se prvim razmakom ili tab-znakom. U terminologiji *lex*-a ovaj deo se zove uzorak (pattern) ili regularni izraz. Regularnim izrazom opisuje se niz znakova koji treba prepoznati u ulaznom tekstu.

Drugi deo je iza razmaka je akcija. Kada je početak ulaznog teksta opisan nekim regularnim izrazom, izvršava se akcija pridružena tom izrazu. Ako ulazni tekst nije opisan ni jednim regularnim izrazom, tekst se samo ispisuje na standardni izlaz. Ukoliko više regularnih izraza opisuje ulazni tekst, primenjuje se pravilo koje može da upari više znakova sa ulaza. Ako više pravila uparuje isti broj znakova, ono koje je prvo navedeno ima prioritet.

Iskazi na C-u u okviru akcije standardno se pišu u okviru jednog reda. Akcija može da se protegne i na više redova, ukoliko se uokviri vitičastim zagradama.

U okviru akcije moguće je pristupiti tekstu koji odgovara uparenom regularnom izrazu. Taj tekst se nalazi u promenljivoj char *yytext, i završava se znakom null. Njegova dužina, ne računajući završni znak, može se pročitati u celobrojnoj promenljivoj yyleng.

Ako više pravila ima istu akciju, postoji mogućnost da se ona napiše samo jednom. Niz pravila oblika

izraz1 akcija
izraz2 akcija

može da se napiše u obliku

```
izraz1          |
izraz2          akcija
```

Vertikalna crta znači "upotrebi akciju iz sledećeg pravila".

4.1.3.2. Regularni izrazi

U dosadašnjim primerima upotrebljavani su regularni izrazi najjednostavnijeg tipa: niz znakova koji može da se upari jedino sa samim sobom. *Lex* omogućava pisanje regularnih izraza kojima odgovara više različitih nizova znakova. Ovakvi regularni izrazi pišu se uz pomoć simbola koji imaju specijalno značenje. Sledeći znaci su rezervisani za tu namenu:

```
" \ ( ) < > { } [ ] % * + ? - ^ / $ . |
```

Da bi se neki od od tih znakova upotrebio kao običan (da mu se ukine specijalno značenje) može da se uokviri navodnicima ili da se napiše iza znaka \. Ovaj znak čak i pod navodnicima zadržava specijalno značenje. Tako se regularni izraz koji prepoznaje par znakova * piše kao

```
"\\*"
```

Navodnik se u kontekstu običnog znaka piše kao \".

Regularni izraz u *lex*-u je, u opštem slučaju, oblika

[<*početni uslo*v,...>] [^] uzorak1 [/uzorak2] [$]

gde uglaste zagrade označavaju opcione elemente. Značenje pojedinih polja u navedenom opštem obliku izraza je sledeće:

<početni uslov,...>	uparivanje je samo u slučaju navedenih početnih uslova (objašnjeno kasnije)
^	uparuje se sa početkom linije
uzorak1	regularni izraz kojim se opisuje niz za uparivanje (objašnjeno u nastavku)
/uzorak2	regularni izraz kojim se opisuje kontekst uparivanja
$	uparuje se sa krajem linije

Regularni izrazi *uzorak1* i *uzorak2* formiraju se po sledećim pravilima:

i) Osnovni regularni izrazi su:

regularni izraz	značenje
abc	tekst abc
\a	znak a, čak i kada je specijalan
.	bilo koji znak osim '\n'
[l234567890]	klasa znakova; bilo koji znak unutar uglastih zagrada, u primeru bilo koja cifra
[0-9]	klasa znakova; specificiranje opsega; u primeru sve cifre od 0 do 9

regularni izraz	značenje
[a-zA-Z]	klasa znakova; kombinovanje prethodne dve tehnike; u primeru sva mala i velika slova
[^a-z]	klasa znakova; ^ upotrebljen iza [označava bilo koji znak osim specificiranih; u primeru bilo koji znak osim malog slova
(izraz)	isto značenje kao i bez zagrada; zagrade služe za grupisanje prilikom primene operatora
(*ime_definicije*}	specificiranje regularne definicije (objašnjeno kasnije)

ii) Ako su r i s regularni izrazi, tada se primenom sledećih operatora mogu dobiti novi regularni izrazi:

izraz	značenje
r+	jedno ili više pojavljivanja r
*r**	nula ili više pojavljivanja r
r?	nula ili jedno pojavljivanje r
r{*n*}	n pojavljivanja r
r{*n*,*m*}	od n do m pojavljivanja r
rs	sekvenca sastavljena od r i s
r|*s*	r ili s

iii) Svi regularni izrazi dobijaju se konačnom primenom pravila *i* i *ii*.

Lex prepoznaje i standardne sekvence iz C jezika za specificiranje grafičkih znakova: \n, \t, \b, \r, \f i \ooo (oktalna reprezentacija). Unutar klasa (uglastih zagrada) jedino znaci \, -, ^ i] zadržavaju specijalno značenje.

Kao ilustracija mogućnosti koje pruža *lex* može da posluži regularni izraz kojim se opisuje komentar u C jeziku, koji počinje nizom "/*" a završava se nizom "*/". Unutar komentara smeju se pojaviti svi znaci, pa i znaci '*' i '/', osim što nije dopuštena pojava znaka '/' neposredno iza '*', s obzirom da ta kombinacija znakova označava kraj komentara.

```
"/*"\/*([^*/] |[^ *]\/|\*[^/])*\**"*/"
```

U gornjem izrazu, unutrašnji deo komentara opisan je na sledeći način: na početku je nula ili više pojava znaka '/', zatim sledi sekvenca od nula ili više elemenata sledećeg oblika:

- bilo koji znak osim znakova '/' i '*', ili
- bilo koji znak osim '*' iza koga sledi znak '/', ili
- znak '*' iza koga sledi bilo koji znak osim '/'

Iza ove sekvence može se pojaviti nula ili više znakova '*' čime je opis unutrašnjeg dela komentara kompletiran.

4.1.3.3. Početni uslovi

Početni uslovi omogućavaju dinamičku promenu skupa aktivnih pravila leksičkog analizatora. Početni uslovi se deklarišu u delu sa definicijama koristeći direktivu %s za inkluzivne, odnosno %x za ekskluzivne početne uslove. Na primer:

%s *ime1 ime2*

*%*x *ime3 ime4*

uvodi dva inkluzivna početna uslova *ime1* i *ime2* i dva ekskluzivna početna uslova *ime3* i *ime4*. Pravilo je uslovljeno ako na njegovom početku stoji niz imena početnih uslova uokvirenih znakovima < i >.

U inicijalnom režimu rada aktivna su jedino pravila koja nisu uslovljena. Početni uslov *id* aktivira se u okviru akcije iskazom BEGIN(*id*). Tada leksički analizator prelazi u režim rada u kome su aktivna jedino pravila uslovljena sa *id*, u slučaju da se radi o ekskluzivnom uslovu, odnosno pravila uslovljena sa *id* i sva pravila bez početnog uslova, u slučaju da se radi o inkluzivnom uslovu. Sa BEGIN 0 leksički analizator vraća se u podrazumevani režim rada.

Sledeći primer ilustruje upotrebu početnog uslova pri obradi komentara u jeziku C. Generisani leksički analizator izostavlja sva pojavljivanja reči *jezici* izvan komentara, a u okviru komentara izostavlja sve osim te reči. Početak komentara /* prebacuje leksički analizator u režim početnog uslova KOM, kada se reč *jezici* uparuje prvim pravilom. U tom trenutku aktivno je i poslednje pravilo, ali se izbor pravila vrši po redosledu njihovog pojavljivanja. Leksički analizator vraća se u inicijalni režim kada naiđe kraj komentara */.

```
%s KOM
%%
<KOM>jezici     printf( "\"jezici\" u komentaru\n" );
<KOM>"*/"       BEGIN 0;
<KOM>.|\n       ;
"/*"            BEGIN (KOM);
jezici          ;
%%
```

4.1.3.4. Uparivanje u kontekstu

Konstrukcija /*uzorak2*, gde *uzorak2* označava proizvoljan regularni izraz, koja se može upotrebiti na kraju regularnog izraza *uzorak* definiše prateći kontekst za prepoznavanje izraza *uzorak.* U tom slučaju, iza teksta koji se uparuje izrazom *uzorak*, mora se pojaviti tekst koji se može upariti izrazom *uzorak2*. Tekst uparen izrazom *uzorak2* ne predstavlja deo prepoznatog teksta, već samo uslov da tekst uparen izrazom *uzorak* bude prepoznat. U sledećem primeru prvo pravilo prepoznaje niz cifara kao neoznačeni ceo broj jedino ako iza njih neposredno reč int. U okviru pravila funkcija atoi najpre određuje binarnu vrednost broja na osnovu niza cifara prosleđenog *lex* promenljivom yytext, a zatim se funkcijom printf na izlazu ispisuje broj ponovo u obliku niza znakova. Drugo pravilo uparuje reč int i na izlaz ispisuje reč float.

```
%%
[0-9]+/int      printf("ceo broj %d ", atoi( yytext ) );
int             printf("float");
```

4.1.3.5. Regularne definicije

Kao što je moglo da se zapazi, regularni izrazi mogu da budu veoma složeni. Da se olakšala njihova upotreba, upotrebljavaju se regularne definicije. One su oblika

ime regularni_izraz

i zasnivaju se na makro-procesiranju. Kad god se od mesta takve definicije naiđe na

{ime}

ono se zamenjuje tekstom regularni izraz. Ovo važi i u samoj sekciji za regularne definicije, pa se na ovaj način mogu inkrementalno graditi složeni izrazi.

4.1.3.6. Iiskazi na C-u u lex-u

Osim u pravilima, različiti iskazi na C-u se mogu pojaviti i na drugim mestima. Prvo, u delu sa regularnim definicijama, ispred prvog %%, mogu da se jave deklaracije promenljivih, definicije tipova i slično, ukoliko se napišu u okviru jednog reda. Takav red mora da počinje jednim razmakom ili tab-znakom. U generisanom C kodu ovaj red se pojavljuje izvan i pre bilo koje funkcije. Znači, to je mesto pogodno za deklaracije globalnih promenljivih.

Problem nastaje kada je potrebno u taj deo ubaciti potrebne pretprocesorske direktive. Te direktive se pišu od prve kolone, pa je nemoguće da se koristi mehanizam sa uvlačenjem reda. Tada se koristi sledeći metod:

```
%{
... tekst ...
%}
```

pri čemu su simboli % na početku linije. Tekst napisan između ovih graničnika se direktno prepisuje u izlazni C program, takođe na početak datoteke.

Ako se deklaracija, uvučena od početka linije, javi iza prvog znaka %%, a pre prvog pravila, u rezultujućem kodu ona postaje lokalna deklaracija funkcije yylex().

Na kraju, ostaje područje iza drugog %%. Ovaj deo se direktno prepisuje u izlaznu datoteku, bez ikakvih ograničenja.

4.1.3.7. Napredne tehnike

Lex u izlaznom kodu definiše nekoliko makroa (direktivama #define) koji se mogu koristiti u okviru akcija. Ovi makroi sažeti su u sledećem spisku:

- input() makro leksički analizator koristi da pročita novi znak sa ulaza. Ovaj znak čita se iz datoteke na koju pokazuje tok yyin.
- output(c) ispisuje jedan znak u okviru podrazumevane akcije. Datoteka u koju se vrši ispisivanje određena je sa yyout.
- yyin i yyout su globalne promenljive tipa FILE* inicijalizovane na stdin i stdout, respektivno. Njihovim postavljanjem na druge vrednosti, na primer u okviru naše funkcije main, ovo može lako da se promeni. Na taj način moguće je promeniti ulazni i izlazni tok u rezultujućem leksičkom analizatoru.

- unput(c) vraća znak u ulazni tok, tako da se može koristiti u sledećem uparivanju.
- REJECT vraća upareni tekst nazad i omogućava da ga neko drugo pravilo prepozna. Time je omogućeno procesiranje preklapajućih nizova znakova.
- yymore() utiče da se novi upareni niz znakova akumulira na prethodni, tako da su posle toga oba u promenljivoj yytext.
- yyless(n) skraćuje upareni tekst za n znakova koje vraća u ulazni tok.

Značenje ovih makro definicija može se jednostavno promeniti pretprocesorskim direktivama #undef i #define u delu sa regularnim definicijama unutar ograničavača %{ i %}. Takođe može da bude korisno definisati funkciju yywrap(). Ona se poziva kada se pri čitanju ulaznog toka naiđe na kraj datoteke. Jedna verzija ove funkcije postoji u pomenutoj biblioteci libl.a.

4.1.4. Primer

Sledeći, malo kompletniji primer predstavlja leksički analizator za jezik C. On nije potpun jedino u smislu da je u cilju uštede prostora izbačeno prepoznavanje svih ključnih reči, operatora i sl. Pošto je zamišljen kao komponenta kompilatora, napravljen je tako da sa nekim terminalnim simbolima vraća i njihove atribute. Atributi se dostavljaju parseru kroz globalnu promenljivu yylval koja je u datoteci y.tab.h definisana kao unija. U istoj datoteci definisana su i simbolička imena za celobrojne kodove terminalnih simbola koji se u okviru akcija vraćaju iskazima return. Ovu datoteku kreira generator parsera. Kod nekih terminalnih simbola predstavljenih jednim ulaznim znakom, na primer '(', parseru se vraća ASCII kod znaka kao oznaka terminala.

```
%{

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "y.tab.h"
long atol();
void *emalloc();
unsigned nlines = 1;

char c;
extern void *emalloc();
%}

slovo                   [_A-Za-z]
cifra                   [0-9]
nenulta_cifra           [1--9]
oktalna_cifra           [0-7]
heksadecimalna_cifra    [0-9A-Fa-f]
identifikator           {slovo}({slovo}|{cifra})*
celobrojni_sufiks       (([lL]?[uU]?)|([uU]?[lL]?))
```

```
decimalna_konstanta                 {nenulta_cifra}{cifra}*{celobrojni_sufiks}
oktalna_konstanta                   0{oktalna_cifra}*{celobrojni_sufiks}
heksadecimalna_konstanta            0[xX]{heksadecimalna_cifra}*{celobrojni_sufiks}
znak1                               [ ^\\\n\‘]
znak2                               \\[ntvbrfa\\\?\‘\"]
znak3                               \\{oktalna_cifra}{1,3}
znak4                               \\x{heksadecimalna_cifra}+
znak                                ({znak1}|{znak2}|{znak3}|{znak4})
znakovna_konstanta                  \‘{znak}\'
niska                               ([^\“\n]|(\\\“))*
konstanta_sa_pokretnim_zarezoml1 {cifra}+\.{cifra}*([eE]{cifra}+)?[fFIL]?
konstanta_sa_pokretnim_zarezom2  {cifra}*\.{cifra}+([eE]{cifra}+)?[fFIL]?
konstanta_sa_pokretnim_zarezom3  {cifra}+\.?{cifra}*[eE]{cifra}+[fFIL]?
konstanta_sa_pokretnim_zarezom4  {cifra}*\.?{cifra}+[eE]{cifra}+ [RIL]?

%%
[\t ]                   |
\n                      nlines++ ;
auto                    return( AUTO_ );
break                   return( BREAK );
case                    return( CASE );
/* itd. */

\(                      return( ‘(’ );
\)                      return( ‘)’ );
\{                      return( ‘{’ );
\}                      return( ‘}’ );
\[                      return( ‘[’ );
/* itd. */

{identifikator}     {
                        yylval.name = emalloc( yyleng+ 1 );
                        strcpy( yylval.name, yytext );
                        return( IDENTIFIKATOR );
                }
{znakovna_konstanta}        return( ZNAKOVNA_KONSTANTA );
{decimalna_konstanta}       |
{oktalna konstanta}         |
{heksadecimalna_konstanta}  {
                        yylval.value = atol( yytext );
                        return( CELOBROJNA_KONSTANTA );
                        }
{konstanta_sa_pokretnim_zarezom1}           |
{konstanta_sa_pokretnim_zarezom2}           |
{konstanta_sa_pokretnim_zarezom3}           |
{konstanta_sa_pokretnim_zarezom4}           {
                        yylval.name = emalloc( yyleng+1 );
```

```
                    strcpy( yylval.name, yytext );
                    return( KONSTANTA_SA_POKRETNIM_ZAREZOM );
                    }
"/*"                {
        loop1 :
                    while ((c = input()) != '*')
                            if (c == '\n') (
                                    nlines++;
                                    orglines++;
}
                            else if ( c == EOF )
                                    fatal( "neocekivan kraj datoteke" );
                    switch (input()) {
                            case '/': break;
                            case '*': unput('*');
                            default: goto loop1;
                            }
                    }
{niska}             {
                            yylval.name = emalloc( yyleng+1);
                            strcpy( yylval.name, yytext );
                            return( NISKA );
                    }
%%
void *emalloc( size )
size_t size;
{
        void *p;
        if ((p = emalloc( size )))
                return p;
        fprintf( stderr, "Out of memory\n" );
        exit( 1 );
}
```

4.2. Gramatičke transformacije

4.2.1. Uvod

Mnoge procedure projektovanja u ovoj knjizi zahtevaju da data bezkontekstna gramatika ima određene osobine (da, recimo, bude LL(1) ili POMERI-IDENTIFIKUJ). Često gramatika opisana iz uputstva za jezik ili nekako drugačije ne poseduje traženu osobinu. U ovom dodatku, predstavljamo izvestan broj metoda transformacije koji se mogu koristiti da se data gramatika preinači, da bi se ostvarila nova gramatika koja generiše isti jezik, ali ima neke od željenih osobina. Sekcije od 4.2.2 do 4.2.5 sadrže transformacije pogodne za ostvarivanje parsirajućih gramatika "od vrha ka dnu", a sekcije 4.2.6 do 4.2.8 sadrže transformacije za parsiranje od dna ka vrhu. U celom dodatku pretpostavljamo da su iz gramatike uklonjene sve smene koje sadrže suvišne simbole.

Nije garantovano da će ove transformacije da preinače neku proizvoljnu gramatiku u gramatiku sa parsiranjem od vrha ka dnu, ili od dna ka vrhu, jer neki bezkontekstni jezici nemaju gramatiku koja se može parsirati potisnom mašinom od vrha ka dnu ili od dna ka vrhu. Ipak, po našem iskustvu za većinu realnih programskih jezika projektovanje sintaksnih procesora se može zasnovati na jednom od sledećih tipova gramatika: *q*-gramatika, LL(1), slabog prvenstva, prostog-mešovitog prvenstva, SLR(1) ili LALR(1). Transformacije predstavljene u ovom dodatku obično su adekvatne za pretvaranje gramatike programskog jezika u bilo koji od zahtevanih tipova.

4.2.2. Leva faktorizacija

Pretpostavimo da gramatika sadrži dve smene

<S> → IF <B> THEN <S1>

<S> → IF <B> THEN <S1> ELSE <S>

Gramatika ne može da bude LL(1) jer obe smene imaju terminal IF u svojim selekcionim skupovima. Kad god dve smene sa istim levim stranama imaju desne strane koje počinju istim simbolom ili simbolima, i ovi zajednički simboli generišu bar jedan neprazan niz terminala, onda selekcioni skupovi nisu disjunktni, i gramatika nije LL(1).

U projektovanju LL(1) gramatike važno je da jezičke konstrukcije sa istim počecima budu generisane iz zajedničkih smena. Za gore navedeni primer zajednički početak mogao bi se postići zamenom dve date smene sledećim smenama:

<S> → IF <B> THEN <S1> <something>

<something> → ELSE <S>

<something> → ε

gde je <something> novi neterminal koji se ne javlja nigde drugde u gramatici. Za ove tri smene se kaže da su dobijene od originalne dve "levom faktorizacijom", pošto je zajednički levi deo desnih strana smena "faktorizovan" u jednu smenu.

Princip leve faktorizacije se može izraziti i u simboličkim terminima:

U gramatici koja sadrži *n* smena

$$<A> \rightarrow \alpha\ \beta_1$$
$$\dots$$
$$<A> \rightarrow \alpha\ \beta_n$$

gde je <A> neterminal, a α i β_i za $1 \le i \le n$ su sekvence neterminala i terminala, ove smene mogu da se zamene sledećim smenama:

$$<A> \rightarrow \alpha\ <new>$$
$$<new> \rightarrow \beta_1$$
$$\dots$$
$$<new> \rightarrow \beta_n$$

gde je <new> neterminalni simbol koji nije iz originalne gramatike. Gramatika dobijena ovom zamenom generiše isti jezik kao i originalna i kaže se da je dobijena levom faktorizacijom.

Leva faktorizacija može da bude primenjena i u translacionim gramatikama. Na primer, dve smene

<S> → IF <B> {JUMPF} THEN <S1> {LABEL}

<S> → IF <B> {JUMPF} THEN <S1> {JUMP} {LABEL} ELSE <S> {LABEL}

se mogu faktorizovati u

<S> → IF <B> {JUMPF} THEN <S1> <else clause>

<else clause> → {LABEL}

<else clause> → {JUMP} {LABEL} ELSE <S> {LABEL}

4.2.3. Ugaona zamena

Razmotrimo dato gramatičko dešavanje neterminala na desnoj strani smene. Svaki put kad se smena koristi u izvođenju, dešavanje mora svuda da se zameni desnom stranom neke smene tog neterminala.

Moguće je tako izmeniti datu gramatiku da se posmatrano dešavanje simbola zamenom odstrani. Ta izmena se zove "supstitucija" i simbolički je opisana ovako:

Ako gramatika sadrži *n* smena

$$<A> \rightarrow \alpha_1$$
$$\dots$$
$$<A> \rightarrow \alpha_n$$

sa levim stranama smena <A> (i ni jednim drugim), i ako gramatika ima datu smenu oblika

$$<B> \rightarrow \beta<A>\gamma$$

gde je <B> neterminal, a β i γ su nizovi neterminala i terminala, onda data smena može da se zameni sa *n* smena:

$$<B> \rightarrow \beta\ \alpha_1\ \gamma$$

$$\ldots$$

$$<B> \rightarrow \beta\ \alpha_n\ \gamma$$

Gramatika dobijena ovim zamenama generiše isti jezik kao i originalna i kaže se da je dobijena zamenom dešavanja neterminala <A> u datoj smeni.

Za bilo koju gramatičku smenu, prvi levi simbol (ako ga ima) na desnoj strani ćemo zvati ugao smene. Prazna smena nema ugao. Ako je zamena načinjena na uglu date smene tu zamenu ćemo zvati ugaona zamena. Ugaona zamena je osnovna tehnika u stvaranju i poboljšanju LL(1) gramatika.

Da bismo ilustrovali kako se ugaona zamena može koristiti u stvaranju LL(1) gramatika, razmotrimo sledeću gramatiku sa početnim simbolom <A>:

1. $<A> \rightarrow a$
2. $<A> \rightarrow <B>c$
3. $<B> \rightarrow a<A>$
4. $<B> \rightarrow b<B>$

Ova gramatika nije LL(1), jer selekcioni skupovi smena 1 i 2 sadrže *a*. Zamenom ugla smene 2 (odn. <B> u desnoj strani smene 2), dobijamo gramatiku

1. $<A> \rightarrow a$

2a. $<A> \rightarrow a<A>c$

2b. $<A> \rightarrow b<B>c$

3. $<B> \rightarrow a<A>$
4. $<B> \rightarrow b<B>$

gde je smena 2 sada zamenjena dvema smenama, po jednom za svaku smenu sa levom stranom <B>.

Nova gramatika nije LL(1) jer smene 1 i 2a imaju *a* u svojim selekcionim skupovima. Za očekivati je da jedna od zamenjenih smena generiše sekvencu koja počinje sa *a* pošto i originalna smena generiše takvu sekvencu. Važna osobina nove gramatike je da obe konfliktne smene počinju sa *a* i da se mogu levo faktorizovati. Posle izvođenja leve faktorizacije gramatika postaje LL(1).

Ugaona zamena je korisna i u pretvaranju LL(1) gramatika u *q*–gramatike. Na primer, razmotrimo sledeću LL(1) gramatiku sa početnim simbolom <S>:

1. $<S> \rightarrow c<S>$
2. $<S> \rightarrow <A>c$
3. $<A> \rightarrow a<S>$

4. <A> → *b*

Ova gramatika nije *q*–gramatika jer je ugao smene 2 neterminal. Zamena ovog ugla daje gramatiku

1. <S> → *c*<S>
2a. <S> → *a*<S>*c*
2b. <S> → *bc*

Pošto zamena uklanja iz gramatike dešavanja neterminala <A> u smeni 2, smene 3 i 4 postaju nedostižne i uklanjaju se. Nova gramatika je *q*–gramatika.

Ugaona zamena čuva LL(1) osobinu i može se uvek koristiti za pretvaranje LL(1) gramatike u *q*–gramatiku. Osnovni princip je sledeći:

Ako je LL(1) gramatika modifikovana ugaonom zamenom, dobijena gramatika je isto LL(1). Ako LL(1) gramatiku višestruko modifikujemo ugaonom zamenom sve dok svi uglovi ne postanu terminali, onda je rezultujuća gramatika q-gramatika.

Prednost modifikovanja LL(1) gramatike supstitucijom uglova je ta što potisne mašine za novu gramatiku mogu parsirati u manje koraka. Mana je što rezultujuća gramatika obično ima više smena i odgovarajuća mašina je veća.

4.2.4. Singlton supstitucija

Kada želimo da dobijemo LL(1) gramatiku, kontraproduktivno je vršiti supstituciju onih pojava koje nisu uglovi. Ovakve supstitucije zamenjuju jednu smenu sa nekoliko smena koje imaju isti ugao i zajednički ugao uzrokuje konflikte u selekcionom skupu. Izuzetak je slučaj kada postoji samo jedna smena sa datom levom stranom. Smena koja jedina ima datu levu stranu zove se *singlton*.

Važna osobina singlton smena je da, ako se njena leva strana pojavljuje na desnoj strani smene *p*, supstitucija te pojave jednostavno rezultuje zamenom smene *p* jednom novom smenom. Tako se pojave leve strane smene mogu višestruko supstituisati bez povećanja broja smena.

Razmotirimo na primer sledeću gramatiku sa startnim simbolom <S>:

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

U ovom primeru, smena 4 je jedina smena sa levom stranom <A>. Supstitucija <A> u smenu 2 rezultuje zamenom smene 2 jednom smenom

<S> → ba<S>b

Daljom supstitucijom pojava <A> u smeni 1, dobija se gramatika

1. <S> → aa<S>ba<S>b
2. <S> → ba<S>b
3. <S> → c

gde je smena 4 izbačena jer je nedostižna. Nova gramatika je jednostavnija jer ima manje smena i manje neterminala; kompleksnija je jer se više simbola javlja na desnim stranama. Obe gramatike su q-gramatike.

Koristimo izraz *singlton supstitucija* za supstituciju pojava neterminala koji su leve strane singlton smena.

Važna osobina singlton supstitucije je da svaka nova smena ima isti selekcioni skup kao i stara smena, pa tako singlton supstitucija čuva LL(1) osobinu. Kao u primeru, singlton supstitucija može se ponavljati dok se ne uklone sve pojave date leve strane na desnim stranama smena. Singlton smena se može odbaciti, osim ako je njena leva strana startni simbol. Za odbačenu smenu kaže se da je eliminisana singlton supstitucijom. Rezultujuća gramatika je LL(1) ako je to i originalna gramatika, a odgovarajuća mašina ima manje stek simbola.

Postoji jedan nebitan izuzetak: kada je desna strana singlton smene sadrži levu stranu kao u smeni

<L> → a<L>b

U takvom slučaju, simbol sa leve strana je mrtav i smena nestaje eliminacijom suvišnih simbola.

4.2.5. Eliminacija leve rekurzije

Neterminal je (indirektno) levo rekurzivan ako se sentenca koja počinje tim neterminalom može izvesti iz istog tog neterminala primenom jedne ili više smena. Smena je (indirektno) levo rekurzivna ako se može koristiti kao prvi korak u takvom izvođenju. Simbolički, neterminal <A> je levo rekurzivan ako i samo ako postoji niz β takav da

$$< A > \overset{+}{\Rightarrow} < A > \beta$$

a smena

<A> → α

je levo rekurzivna ako i samo ako postoji β takvo da

$$\alpha \overset{*}{\Rightarrow} < A > \beta$$

Smena je *direktno levo rekurzivna* (self left recursive) ako su joj ugao i leva strana jednaki. Takva smena mora biti i levo rekurzivna zato što je desna strana sentenca koja počinje levom stranom i izvođenje se ostvaruje u nula koraka.Direktna leva rekurzija je ilustrovana smenom 1 na Sl. 4.2.1(a) koja ima <S> kao ugao i levu stranu. Druga smena u ovoj gramatici nije levo rekurzivna.

1. <S> → <S>a
2. <S> →b

(a)

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

(b)

Sl. 4.2.1: Gramatike sa levom rekurzijom

Primer indirektne leve rekurzije ilustrovan je gramatikom na Sl. 4.2.1(b). Ovde su neterminali <A> i <B> levo rekurzivni što dokazujemo izvođenjima

<A> ⇒ <B>b ⇒ <A>cb

<B> ⇒ <A>c ⇒ <B>bc

Izvođenja počinju smenama 2 i 3, respektivno, koje su prema tome levo rekurzivne smene. Leva rekurzivnost pokazuje se relacijama

$$< B > b \overset{*}{\Rightarrow} < A > cb \qquad \text{i} \qquad < A > c \overset{*}{\Rightarrow} < B > bc$$

Smene 1 i 4 nisu levo rekurzivne.

Gramatika sa levo rekurzivnim neterminalom ne može biti LL(1) gramatika. Centralna ideja dokaza ove tvrdnje je: selekcioni skup bilo koje levo rekurzivne smene sa levom stranom <X> mora sadržavati i FIRST(<X>), i tako dolazi u konflikt sa selekcionim skupovima svih ostalih smena koje imaju levu stranu <X>. U slučaju na Sl. 4.2.1(b) konflikt selekcionih skupova smene 1 i levo rekurzivne smene 2 pokazuje se

SELECT(1) = FIRST(a<B>) = {a}

SELECT(2) = FIRST(<B>b) = {a,d}

Levo rekurzivna gramatika se uvek može preurediti u gramatiku bez leve rekurzije. Opšte tehnike za ovo su prilično složene, ali postoji specijalni slučaj kada je pruređivaje gramatike pravolinijsko. To je slučaj kada je svaka levo rekurzivna smena direktno levo rekurzivna. Ovaj slučaj je primenljiv za većinu praktičnih primena u programskim jezicima.

Osnovna ideja u ovom slučaju je da rekurzivne neterminale posmatramo kao generatore nekog niza posle koga dolazi lista od nula ili više članova. Na Sl. 4.2.1(a) na primer posmatramo <S> kao generator niza *b* posle koga sledi lista od nula ili više *a*-ova.Prirodni metod od vrha ka dnu (top-down) za izražavanje ovog načina posmatranja je

<A> → b<list>

<list>→ a<list>

<list>→ ε

Prethodna ideja generalizuje se sledećom transformacijom za uklanjanje direktne leve rekurzije:

Pretpostavimo da neterminal <A> ima m direktno rekurzivnih smena

<A> → <A>α_i za $1 \le i \le m$

i *n* smena

<A> → β_j za $1 \le j \le n$

koje nisu direktno levo rekurzivne. Zamenimo te smene smenama

<A> → β_j <list A> za $1 \le j \le n$

<list A> → α_i<list A> za $1 \le i \le m$

<list A> → ε

gde je <list A> novi neterminal.

Nove smene se interpretiraju: <A> generiše jedan od β_j -ova koga sledi lista od nula ili više α_i-ova.

Sa izuzetkom određenih dvosmislenih gramatika: Ako je data gramatika takva da su sve levo rekurzivne smene direktno levo rekurzivne, onda se ekvivalentna gramatika bez leve rekurzije može dobiti primenom predhodnih transformacija na sve levo rekurzivne neterminale.

Kada se koristi zajedno sa supstitucijom uglova, ovo transformaciono pravilo može eliminisati levo rekurzivne smene čak i onda kada neke smene nisu direktno levo rekurzivne. Uzmimo gramatiku sa Sl. 4.2.1(b) kao primer, leva rekurzija može se eliminisati prvo supstitucijom ugla smene 2 i zatim primenom transformacije na neterminal <A>.

1. <A> → a<B>	1. <A> → a<B><list>
2.4. <A> → db	2.4. <A> → db<list>
2.3. <A> → <A>cb	2.3. <list> → cb<list>
3. <B> → <A>c	x. <list> → ε
4. <B> → d	3. <B> → <A>c
	4. <B> → d
(a)	(b)

Sl. 4.2.2: Gramatike za ilustraciju postupka eliminacije leve rekurzije

Supstitucija ugla smene 2 rezultuje gramatikom na Sl. 4.2.2(a) gde je smena 2.4 rezultat supstitucije smene 4 u ugao smene 2, a smena 2.3 je rezultat supstitucije smene 3. Eliminacija direktne leve rekurzije za neterminal <A> rezultuje gramatikom na Sl. 4.2.2(b). Ova gramatika nije levo rekurzivna.

Eliminacija indirektne leve rekurzije odvija se po sledećoj proceduri:

1. Usvojiti neki redosled neterminala:
   $<A_1>, <A_2>, <A_3>, ..., <A_n>$
2. Transformisati gramatiku primenom sledećeg algoritma:

```
for (i := 1 to n)
    for (j:= 1 to i–1)
        Zameniti svaku smenu oblika
                <Ai> → <Aj>γ
        smenama
                <Ai> → δ1γ
                <Ai> → δ2γ
                    …
                <Ai> → δkγ
        gde su
                <Ai> → δ1γ
                <Ai> → δ2γ
                    …
                <Ai> → δkγ
        tekuće smene za <Aj>; Napomena: ovim se indirektna leva rekurzija pretvara
```

u desnu.
end for;
Ukloniti sve direktne leve rekurzije (ako ih ima) među smenama za $<A_i>$ ranije izloženom procedurom;
end for;

Primena ovog algoritma je moguća ako nema praznih smena $<A> \overset{*}{\Rightarrow} <A>$ i ciklusa u gramatici.

Primer: Neka je data gramatika

1. $<S> \rightarrow <A>a$
2. $<S> \rightarrow b$
3. $<A> \rightarrow <A>c$
4. $<A> \rightarrow <S>d$
5. $<A> \rightarrow \varepsilon$

Gramatika je indirektno levo rekurzivna jer je:

$$<S> \overset{1}{\Rightarrow} <A>a \overset{4}{\Rightarrow} <S>da$$

Uredimo neterminale na sledeći način:

$$<S>, <A> \Rightarrow (<A_1> = <S>, <A_2> = <S>)$$

Promenljiva u spoljnoj petlji (i) ide od 1 do 2. Ispratimo ovu transformaciju sledećim koracima:

1. i = 1, j = 1
   $<A_i> = <A_1> = <S>$; $<A_j> = <A_1> = <S>$
   $<S> \rightarrow <S>\gamma$ ne postoji

   Zamena smena:

2. i = 2, j = 1
   $<A_i> = <A_2> = <A>$; $<A_j> = <A_1> = <S>$
   potrebno je zameniti $<A> \rightarrow <S>d$

   Gramatika sada postaje:

3. $<A_j> = <A_1> = <S> \rightarrow <A>a$ (δ_1)
   $<A_j> = <A_1> = <S> \rightarrow <A>a$ (δ_1)
   $|<A> \rightarrow <A>ad$ dodajemo u gramatiku
   $<A> \rightarrow bd$ dodajemo u gramatiku

   Eliminisanje direktne rekurzije:

4. $<A> \rightarrow <A>ad$ ($ad = \alpha_1$)
   $<A> \rightarrow <A>c$ ($c = \alpha_2$)
   $<A> \rightarrow bd$
   $<A> \rightarrow \varepsilon$

   Gramatika bez levih rekurzija:

3. $<A> \rightarrow bd<list>$
   $<A> \rightarrow <list>$
   $<list> \rightarrow ad<list>$
   $<list> \rightarrow c<list>$
   $<list> \rightarrow \varepsilon$

4. $<S> \rightarrow <A>a$
   $<S> \rightarrow b$
   $<A> \rightarrow bd<list>$
   $<A> \rightarrow <list>$
   $<list> \rightarrow ad<list>$
   $<list> \rightarrow c<list>$
   $<list> \rightarrow \varepsilon$

Algoritam je primenjen na gramatiku koja ima praznih smena. U odabranom uređenju neterminala to nije predstavljalo problem. Međutim, ako se neterminali urede kao <A>, <S>, eliminacija indirektne leve rekurzije neće biti moguća.

4.2.6. Eliminacija praznih smena

Ako je jezik L generisan gramatikom $G = (V_N, V_T, P, <S>)$ i svaka smena u P je oblika $<A> \to \alpha$, gde je $\alpha \in V^*$, $V = V_T \cup V_N$, tada se L može generisati gramatikom u kojoj je svaka smena oblika $<B> \to \beta$, gde je $\beta \in (V - \{<S>\})^+$, ili $<S> \to \varepsilon$, odnosno startni neterminal <S> se ne pojavljuje na desnoj strani bilo koje smene i smene nisu prazne, osim što startni neterminal može posedovati praznu smenu.

Prvi korak u nalaženju ovakve gramatike je da se pronađe gramatika kod koje se <S> ne pojavljuje na desnoj strani bilo koje smene.

Neka je $<S_1>$ gramatički simbol koji ne pripada ni V_T ni V_N. Neka je $G_1 = (V_N \cup \{<S_1>\}, V_T, P_1, <S_1>)$. P_1 se sastoji od svih smena u P, uz smene oblika $<S_1> \to \alpha$, gde je $<S> \to \alpha$ smena u P. Treba imati na umu da se $<S_1>$ ne pojavljuje na desnoj strani bilo koje smene. Tvrdimo da je $L(G) = L(G_1)$.

Pretpostavimo da je $<S> \overset{*}{\underset{G}{\Rightarrow}} w$. Neka je prva korišćena smena u ovom izvođenju $<S> \to \alpha$. Tada se može staviti $<S> \underset{G}{\Rightarrow} \alpha \overset{*}{\underset{G}{\Rightarrow}} w$. Prema definiciji za P_1, $<S_1> \to \alpha$ je u P_1, tako da je $<S_1> \underset{G_1}{\Rightarrow} \alpha$. Takođe, pošto P_1 sadrži sve smene od P, $\alpha \overset{*}{\underset{G_1}{\Rightarrow}} w$. Stoga je $<S_1> \overset{*}{\underset{G_1}{\Rightarrow}} w$. Dakle, zaključujemo da $L(G) \subseteq L(G_1)$.

Neka je polazna gramatika G već u obliku u kojem se startni simbol ne pojavljuje na desnoj strani bilo koje smene. Za bilo koji neterminal <A> gramatike G može se utvrditi da li je $<A> \overset{*}{\underset{G}{\Rightarrow}} \varepsilon$. Ako je tako, tada postoji izvođenje čije stablo nema putanju koja je duža od broja neterminala u gramatici G.

Neka su $<A_1>, <A_2>,..., <A_k>$ neterminali iz kojih se može izvesti prazna sekvenca a $<B_1>, <B_2>,..., <B_m>$ oni iz kojih to nije moguće. Konstruišemo novi skup smena P_1 prema sledećim pravilima:

1. Ako je $<S> \overset{*}{\underset{G}{\Rightarrow}} \varepsilon$, tada je $<S> \to \varepsilon$ u P_1.
2. Nijedna druga smena oblika $<A> \to \varepsilon$ ne pojavljuje se u P_1.
3. Ako se u P nalazi smena oblika:

   $$<A> \to C_1C_2\ldots C_r, \qquad r \geq 1$$

   tada se u P_1 dodaju sve smene oblika $<A> \to \alpha_1\alpha_2\ldots\alpha_r$, pri čemu je $\alpha_i = C_i$ za $C_i \in V_T \cup \{<B_1>, <B_2>,..., <B_m>\}$, odnosno $\alpha_i = C_i$ ili $\alpha_i = \varepsilon$ za $C_i \in \{<A_1>, <A_2>,..., <A_k>\}$, $1 \leq i \leq r$, uz ograničenje u poslednjoj varijanti da ne mogu svi simboli α_i biti jednaki ε.

Jasno je da za $G_1 = (V_N, V_T, P_1, <S>)$ važi $L(G_1) \subseteq L(G)$. Moramo sada pokazati indukcijom prema broju koraka u izvođenju da ako je $<A> \overset{*}{\underset{G}{\Rightarrow}} w$, $w \neq \varepsilon$, tada je $<A> \overset{*}{\underset{G_1}{\Rightarrow}} w$, za $<A> \in V_N$. U prvom koraku je to očigledno, pa pretpostavimo da je tačno za k koraka. Pretpostavimo da je $<A> \overset{*}{\underset{G}{\Rightarrow}} w$ dobijeno u k+1. koraku izvođenja i pretpostavimo da je $<A> \to C_1C_2\ldots C_r$ prva korišćena smena. Možemo staviti $w = w_1w_2\ldots w_r$, gde za $1 \leq i \leq r$ važi $C_i \overset{*}{\underset{G_1}{\Rightarrow}} w_i$. Dalje, postoji smena u P_1 oblika $<A> \to \alpha_1\alpha_2\ldots\alpha_r$ gde je $\alpha_i = C_i$ ako je $w_i \neq \varepsilon$ i $\alpha_i = \varepsilon$ ako je $w_i = \varepsilon$. Otuda je

$<A> \overset{*}{\underset{G_1}{\Rightarrow}} w$. Dakle, gramatike G i G_1 su ekvivalentne.

4.2.7. Dobijanje poljske translacione gramatike

Pretpostavimo da želimo da realizujemo prevođenje specificirano smenom

<statement> → IF <Boolean expression> {JUMPF} THEN <statement> {LABEL}

metodom od dna ka vrhu, koja zahteva da gramatika bude u obliku poljske translacione gramatike. Ovaj problem rešavamo zamenom ove smene dvema smenama tako što {JUMPF} dolazi na krajnje desnu poziciju jedne smene, a {LABEL} druge smene. Nove smene se dobijaju uvođenjem novog neterminala <if clause> koji generiše niz IF <Boolean expression> {JUMPF} i zamenom pojave datog niza u postojećoj smeni novim neterminalom. Nove smene su:

<statement> → <if clause> THEN <statement> {LABEL}

<if clause> → IF <Boolean expression> {JUMPF}

Uopšteno, ako imamo smene oblika

$$<L> \rightarrow \alpha\{A\}\beta$$

gde su α i β nizovi gramatičkih simbola, može se uvesti novi neterminal <new>, a postojeća smena se zameniti smenama

$$<L> \rightarrow <new>\beta$$

$$<new> \rightarrow \alpha\{A\}$$

Rezultat je da je akcioni simbol {A} smešten na krajnju desnu poziciju.

Ponavljanjem ove tehnike smenu po smenu, uvek je moguće preurediti gramatiku u poljsku translacionu gramatiku. Ako data gramatika nema akcione simbola na krajnje levim pozicijama, preuređivanje se može izvesti bez dodavanja praznih smena.

Ukoliko pojava praznih smena nije problem za izabrani metod parsiranja, na raspolaganju je sledeća transformacija u kojoj se smena oblika

$$<L> \rightarrow \alpha\{A\}\beta$$

zamenjuje sledećim smenama:

$$<L> \rightarrow \alpha<new>\beta$$

$$<new> \rightarrow \{A\}$$

gde <new> predstavlja novi neterminal, koja takođe smešta akcioni simbol na krajnje desnu stranu u smeni.

4.2.8. Pretvaranje gramatike u "pomeri–identifikuj" konzistentnu

Sada dajemo metodu pomoću koje je moguće svaku bezkontekstnu gramatiku bez ε-smena preurediti u potisni–identifikuj konzistentnu gramatiku. Posmatrajmo sledeću gramatiku sa startnim simbolom <S>:

1. <S> → c<A>a
2. <A> → c<B>
3. <A> → d<B>a
4. <B> → d<S>e<B>f
5. <B>→ g

Postoji konflikt tipa pomeri–identifikuj za simbol steka <B> i ulazni simbol a zbog toga što kada je <B> na vrhu steka, ne znamo da li je <B> iz smene 3 kada bi trebalo da izvršimo SHIFT, ili je iz smene 2 kada bi trebalo da izvršimo IDENTIFY (a potom i REDUCE(2)). Naš plan je da uvedemo novi neterminal <B'> pomoću singlton smene

<B'> → <B>

i da zamenimo sve pojave <B> sa <B'> u smenama na desnoj strani, osim kada se nalazi na krajnje desnoj poziciji. Nove smene su:

1. <S> → c <A> a
2. <A> → c <B>
3. <A> → d <B'> a
4. <B> → d <S> e <B'> f
5. <B> → g
6. <B'> → <B>

U novoj gramatici <B> se pojavljuje samo na krajnje desnoj poziciji smena i zbog toga ne može biti ispod (BELOW) nekog terminala. Slično tome <B'> se pojavljuje samo na pozicijama koje nisu krajnje desne i zbog toga ne može biti REDUCED-BY od strane nekog terminala. Nova gramatika je potisni–identifikuj konzistentna.

Tehnika ilustrovana u prethodnom primeru je opšta i može se koristiti da pojedinačno otkloni sve konflikte tipa potisni–identifikuj.

> Opšta procedura je:
>
> Za svaki gramatički simbol A (neterminal ili terminal) za koji postoji SHIFT-IDENTIFY konflikt, tj. postoji terminal Z takav da
>
> A REDUCED-BY Z i A BELOW Z
>
> uvesti novi neterminal <A'> pomoću singlton smene
>
> <A'> → <A>
>
> i zameniti <A> sa <A'> u svi slučajevima kada je A sa desne strane, a nije u krajnje desnoj poziciji.

Ako nam je data ulazna gramatika za koju se želi projektovati parser potisni–identifikuj, preporučujemo da se primene metode iz odeljaka od 4.2.6 do 4.2.8 i to sledećim redom:

1. Ukloniti prazne smene, ako postoje, koristeći metod opisan u 4.2.6.

2. Ubaciti akcione simbole i to tako da ni jedan nije u krajnje levoj poziciji.
3. Preurediti translacionu gramatiku kao poljsku translacionu gramatiku koristeći metod opisan u 4.2.7
4. Preurediti gramatiku u "pomeri–identifikuj" konzistentnu gramatiku koristeći metod opisan u 4.2.8.

Primetimo da preuređivanje gramatike u "pomeri–identifikuj" konzistentnu ne narušava osobine poljske translacije pošto metode iz koraka 4 ne utiču na krajnje desne pozicije smena.

Koristeći ove metode svaka bezkontekstna gramatika se može preurediti u potisni–identifikuj konzistentnu gramatiku. Pa ipak ne postoje garancije da će rezultujuća gramatika zadovoljiti bilo koji od preostalih uslova koje zahteva parsiranje potisni–identifikuj (na primer, da će biti bezsufiksna, slabog prvenstva ili prostog mešovitog prvenstva).

LITERATURA

[1] A. V. Aho, R. Sethi, J. D. Ullman, *Compilers/Principles, Techniques and Tools*, Addison-Wesley 1986.

[2] A. V. Aho, J. D. Ullman, *The Theory of Parsing, Translation, and Compiling*, Prentice-Hall 1972.

[3] D. Bojić, D. Velašević, Determinističko parsiranje hibridnim metodom proširenih levih uglova, *Zbornik radova konferencije YUINFO*, Brezovica, april 1995, pp. 405-410

[4] D. Bojić, D. Velašević, LR(1) parsiranje hibridnim metodom, *Zbornik radova konferencije YUINFO*, Brezovica, april 1996, pp. 80

[5] F. DeRemer, T. Pennello, Efficient computation of LALR(1) lookahead sets, *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 4, October 1982, pp. 615-649

[6] C. N. Fischer, R. J. LeBlanc, *Crafting A Compiler*, The Benjamin/Cummings Publishing Company 1988.

[7] P. M. Lewis, D. J. Rosenkrantz, R. E. Stearns, *Compiler Design Theory*, Addison-Wesley 1976.