GENERIRANJE LALR TABEL

PETER KOKOL
VILJEM ŽUMER

UDK: 681.3.06:808.61

VISOKA TEHNIŠKA ŠOLA MARIBOR, VTO ELEKTROTEHNA MARIBOR

V članku na kratko opisujemo postopek za generacijo LALR(1) tabel, dodajamo osnovne algoritme in procedure. Generiranje LALR(1) tabel smo preverili za različne jezike s tem, da je potrebno kot vhodni podatek vnesti gramatiko ustreznega jezika v BNF meta jeziku. Na koncu dodajamo krajši primer.

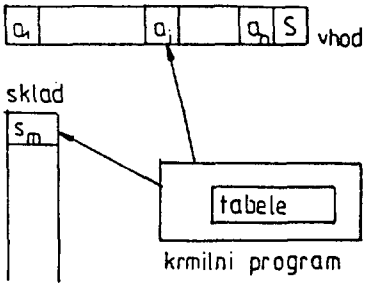
LALR TABLES GENERATOR. Algorithms and procedures for LALR(1) tables generator are given. To produce the LALR(1) tables for some language we must input the grammar of this language in BNF meta language. The described generator was tested for different languages. A short example is given at the end of this paper.

1. UVOD

Eden prvih korakov pri implementaciji jezika je sintaksna analiza oz. razpoznavanje jezika. V zadnjem času se vse več uporabljajo LR razpoznavniki, ki imajo več prednosti pred ostalimi. LR razpoznavnik je zelo preprost program in splošen za vse jezike, pač pa zahteva pri svojem delu obsežne tabele. Te LR tabele vsebujejo lastnosti posameznega jezika in jih je potrebno generirati za vsak jezik. Glede na večino današnjih programskih jezikov lahko obsežne LR tabele skrčimo na LALR(1) tabele, ki zavzamejo mnogo manj prostora. Generiranje teh tabel je zamudno opravilo, ki se ga da mehanizirati. Osnovna naloga razpoznavnika je učinkovito odkrivanje in javljanje sintaksnih napak. Zato morajo LALR(1) tabele vsebovati podatke za odpravljanje napak.

2. LR razpoznavniki

V splošnem je razpoznavnik sestavljen iz dveh delov: iz tabel in programa, ki ga te tabele krmilijo. Za različne gramatike se spreminjajo le tabele. Zgradba razpoznavnika je prikazana na sliki 1.



Slika 1: Zgradba razpoznavnika

Razpoznavnik ima sklad, vhod in tabelo. Vhod se bere iz leve proti desni, simbol za simbolom. Sklad vsebuje niz stanj $s_0\ s_1\ s_2\ s_3 \cdots s_m$, kjer je s_m na vrhu sklada in vsebuje informacije o skladu.

Tabela je sestavljena iz dveh delov, iz funkcij ACTION in GOTO. Funkcija ACTION (s_m, a_i) ima lahko naslednje vrednosti:

1. pomakni s
2. reduciraj $A \rightarrow \gamma$
3. sprejmi
4. napaka

Funkcija GOTO (s_m, a_i) generira novo stanje s in predstavlja tabelo prehajanja stanj determinističnega končnega avtomata, katerega vhodna abeceda so simboli gramatike.

Konfiguracija LR razpoznavnika je dvojica:

$(s_0\ s_1\ s_2 \cdots s_m, a_i\ a_{i+1} \cdots a_n\ \$)$

Prva komponenta je vsebina sklada, druga komponenta pa še ne prebranidel vhodnega teksta. Naslednja akcija razpoznavnika je odvisna od trenutnega vhodnega simbola a_i in stanja na vrhu sklada s_m. Konfiguracija po tej akciji je glede na štiri vrednosti funkcije ACTION naslednja:

1. če je funkcija ACTION (s_m, a_i) = pomakni s, razpoznavnik izvede pomik in dobimo naslednjo konfiguracijo

   $(s_0\ s_1\ s_2 \cdots s_m\ \ s, a_{i+1} \cdots a_n\ \$)$

2. Če je funkcija ACTION (s_m, a_i) = reduciraj $A \rightarrow \gamma$, razpoznavnik reducira niz v neterminal A in dobimo naslednjo konfiguracijo:

$(s_0\ s_1\ s_2 \dots s_{m-r}\ s,\ a_i\ a_{i+1} \dots a_n\ \$)$

Stanje s dobimo iz funkcije GOTO (s_{m-r}, A), kjer je r dolžina niza γ, to je desne strani produkcije.

3. Če je funkcija ACTION (s_m, a_i) = sprejmi, je razpoznavanje končano.

4. Če je funkcijo ACTION (s_m, a_i) = napaka, je razpoznavnik odkril napako in se pokličejo ustrezni podprogrami.

2.1. Postopek za generiranje LALR tabel

Prva naloga pri gradnji tabel je, da skonstruiramo množico LR(1) postavk za gramatiko G, ki opisuje programski jezik. Najprej G razširimo s produkcijo $S \rightarrow S'$, kjer je S startni simbol gramatike G. S to dodatno produkcijo dodamo informacijo, kdaj naj se razpoznavnik ustavi. Generira se stanje "sprejmi" v ACTION delu tabele.

LR(1) postavka je urejen par oblike $(A \rightarrow \alpha . \gamma, a)$, kjer je prva komponenta jedro, druga pa množica terminalov s končnim simbolom $.

Prvi del postavke-jedro nam pokaže, kolikšen del produkcije je bil že razpoznan pri razpoznavanju. Iz zgornje produkcije dobimo naslednje postavke:

1. $(A \rightarrow . \alpha\gamma, a)$
2. $(A \rightarrow \alpha . \gamma, a)$
3. $(A \rightarrow \alpha\gamma ., a)$

Prva postavka pove, da na vhodu pričakujemo niz, ki je izpeljiv iz $\alpha\gamma$, druga pa, da pričakujemo niz izpeljiv iz γ. Tretje postavka se nanaša na drugo komponento in pove, da je potrebna redukcija $A \rightarrow \alpha\gamma$, če je na vhodu simbol a. Formalno pravimo, da je postavka $(A \rightarrow \alpha . \gamma, a)$ veljavna za prefiks β, če obstaja izpeljava $S \rightarrow^* \delta A w \rightarrow \delta\alpha\gamma w$, kjer velja

1. $\beta = \delta\alpha$ in
2. a je prvi simbol w ali w je ε in a je $.

Algoritem za konstrukcijo LR (1) postavk

Vhod: Razširjena gramatika G

Izhod: Množica LR (1) postavk

Metoda: Procedure CLOSURE, GOTO in izračunaj - postavke

```
procedure CLOSURE(P)
begin
  repeat
    for  za vsako postavko (A--->b.Bg,a) v P,vsako produkcijo B--->d in
         vsak terminal b v FIRST(ga) tako da B--->.d se ni v P
    do   dodaj (B--->.d,b) k P;
    endfor;
  until nobena postavka ne more vec biti dodana k P;
  return P
end


procedure GOTO(P,X);
begin
  naj bo J mnozica postavk (A--->aX.g,a) tako da je (A--->a.Xg,a) v P;
  return CLOSURE(J)
end


program IZRACUNAJ_POSTAVKE;
  begin
    C:=$CLOSURE({S'--->.S,$})$
    repeat
      for za vsako mnozico postavk P v C in vsak simbol gramatike X,
          tako da GOTO(P,X) ni prazna mnozica in da se ni v C
      do  dodaj GOTO(P,X) k C
      endfor
    until nobene mnozice postavk ne moremo vec dodati k C
end.
```

Naslednji korak pri konstrukciji tabel je pretvorba množic LR (1) postavk v množice LALR (1) postavk. To dosežemo tako, da združimo vse množice postavk, ki imajo enaka jedra, se pravi enake prve komponente.

Algoritem za konstrukcijo LALR (1) postavk

Vhod: Razširjena gramatika G
Izhod: Množica LALR (1) postavk
Metoda:

1. Konstruiraj $C = P_0, P_1 \dots P_n$, množice LR (1) postavk

2. Za vsako jedro v LR (1) množici postavk najdi vse množice z enakim jedrom in jih združi v unijo.

Zadnji korak v konstrukciji tabele je pretvorba množice LALR (1) postavk v LALR (1) tabelo. Če v tabeli ni večkratno definiranih vstopov (konfliktov) pravimo, da je tabela LALR (1) tabela.

Poznamo dve vrsti konfliktov:

1. Reduciraj - reduciraj konflikt
2. Pomakni - reduciraj konflikt

Algoritem za konstrukcijo LALR (1) tabele

Vhod: Razširjena gramatika G

Izhod: LALR (1) tabela v primeru, da ni konfliktov

Metoda:

1. Skonstruiraj množico LALR (1) postavk C za G, kjer je $C = P_0, P_1 \ldots P_n$

2. Stanje i za razpoznavnika se skonstruira iz P_i. Funkcija ACTION je določena takole:
   a) Če je postavka $(A \rightarrow \alpha . a \beta, b)$ v P_i, GOTO $(P_i, a) = P_j$ postavi ACTION (i, a) na "pomakni j"
   b) Če je postavka $(A \rightarrow \alpha ., a)$ v P_i, potem postavi ACTION (i, a) na "reduciraj" $A \rightarrow \alpha$.
   c) Če je postavka $(S' \rightarrow S., \$)$ v P_i, potem postavi ACTION (i, $) na "sprejmi"

3. GOTO funkcija za stanje P_i se določi na naslednji način: Če je GOTO $(P_i, A) = P_j$ potem je GOTO (i, A)=j, (A je neterminal.)

4. Vsi vstopi, ki niso definirani z 2 ali 3 so postavljeni na "napako"

5. Začetno stanje je stanje skonstruirano iz postavk $(S' \rightarrow S ., \$)$

3. Opis programa

Program za generiranje LALR (1) tabel zahteva kot vhodne podatke gramatiko ustreznega jezika v BNF meta jeziku. Ta ima naslednja pravila:

GRAMATIKA: : = BLOK $

BLOK : : = PRODUKCIJA {';'PRODUKCIJA} *

PRODUKCIJA : : = LEVI DEL ':' : = 'DESNI DEL

LEVI DEL : : = NETERMINAL

DESNI DEL : : IZRAZ {'/'IZRAZ}

IZRAZ : : = {NETERMINAL / TERMINAL}

NETERMINAL : : = ČRKA {ČRKA / ŠTEVILKA} *

TERMINAL : : = '.' {ZNAK} * '.'

DELIMITER : : = '='/'/'/';'/'$'/'.'

Program najprej pretvori vhodne podatke v vmesni tekst, iz tega vmesnega teksta izračuna postavke, nato konstruira tabelo, jo linearizira in izpiše. Glavni program z imenom GENERIRAJ je:

```
program GENERIRAJ(VHODNI_PODATKI)
(* PROG so produkcije ki določajo gramatiko jezika *)
(* F10 je kretnica za izpis postavk                *)
(* MATR je vmesna matrika                          *)
(* IZRACUNAJ_POSTAVKE_MODIF  procedura             *)
(* KONSTRUIRAJ_TABELO        procedura             *)
(* LINEARIZIRAJ              procedura             *)
(* RAZPOZNAJ                 procedura             *)

begin
  preberi VHODNE_PODATKE;RAZPOZNAJ;
  if ni napak then IZRACUNAJ_POSTAVKE_MODIF;
    if F10 do izpisi mnozico postavk end;
    KONSTRUIRAJ-TABELO;
    LINEARIZIRAJ;
    izpisi linearizirano tabelo;
    if odkriti konflikti do izpisi konflikte end
  else
    javi slovnicne napake
  endif
end.
```

Procedura RAZPOZNAJ opravi leksikalno in sintaksno analizo vhodnih podatkov in jih preslika v vmesni tekst z namenom, da se poenostavi nadaljnje generiranje in da je program kar se da modularen.

```
procedure RAZPOZNAJ;
begin
  opravi leksikalno analizo;
  opravi sintaksno analizo;
  if ni napak then pretvori v MATR endif
end
```

Za konstrukcijo postavk smo nekoliko spremenili algoritma in razdelka 2, ker smo tako prihranili precej pomnilniškega prostora.

Glavne razlike so:

1. Shranimo le glave množic postavk. Glava je tisti del množice postavk, ki jih skonstruiramo pri GOTO funkciji preden izračunamo CLOSURE (množica J).

2. Ne skonstruiramo celotne množice LR (1) postavk, ampak zahtevane unije računamo kar sproti. Zaradi tega postane postopek iterativen, torej se poveča čas obdelave.

Prvo komponento postavke predstavimo v računalnik z dvema celima številoma (N, I), kjer je N številka produkcije in I mesto pike v postavki.

Drugo komponento predstavimo z logičnim poljem z n biti, kjer je n število atomov v jeziku.

Izračun LALR postavk je prikazan v proceduri.

```
procedure IZRACUNAJ_POSTAVKE_MODIF;
(* N,L sta stevca                                    *)
(* I(N) je mnozica postavk                           *)
(* I je mnozica vseh mnozic postavk                  *)
(* P1,P delovni pomnilnik                            *)
(* GOTO,CLOSURE sta proceduri definirani prej *)
begin N:=0;L:=0;I(N):=(S'--)S,$)
(* izracun prve iteracije *)
  while niso obdelane vse I(N) do P:=CLOSURE(N);X:=prvi simbol;
    while X se ni obsel vseh simbolov do
      I(L):=GOTO(P,X);L:=L+1;
      if I(K)=I(L-1) do
        I(K):=I(K) unija I(L-1);          (* 0<=K<=L *)
        L:=L-1                            (* brisi I(L-1) iz I *)
      end;
      X:=naslednji simbol;
    end;
    N:=N+1;
  end;
(* izracun naslednjih iteracij *)
  repeat N:=0;
    while niso obdelani vsi I(N) iz I do P:=CLOSURE(I(N));X:=prvi simbol;
      while X se ni obsel vseh simbolov do
        P1:=GOTO(P,X)
        if I(K)=P1 do I(K):=I(K) unija P1 end     (* 0<=K<=L *)
      end
    end
  until ni sprememb v I
end
```

Preslikavo iz LALR postavk v LALR tabelo opravimo kot je opisano v razdelku 2. 1. in je prikazana v proceduri KONSTRUIRAJ - TABELO

```
procedure KONSTRUIRAJ-TABELO
(* TAB je LALR tabela organizirana kot matrika m x n,
   m je stevilo simbolov *)
   n je stevilo stanj *)
begin N:=0
  while niso obdelane se vsi I(N) iz I do P:=CLOSURE(I(N));
    while niso obdelane vse postavke v P do
      if (S'--)S,,$) v P do  TAB(N,$):=(0,"A")
      else (A--)α.,a) v P do TAB(N,a):=(A,"R")
      else (A--)α.bβ,a v P do
        P1:=GOTO(I(N),b);
        najdi v I takšen I(K) da je I(K)=P1;
        if b je terminal do TAB(n,b)=(K,"S")
        else b je neterminal do TAB(n,b):=(K,"G") end
      end
    end
  end
end
```

Kot izhod iz procedure KONSTRUIRAJ - TABELO dobimo matriko m x n, kjer je m število atomov jezika, za katerega konstruiramo tabele, n pa število stanj končnega avtomata za razpoznavnik.

Zadnji korak pri konstrukciji LALR tabele je linearizacija matrike. Linearizacija pomeni, da tabelo, ki je organizirana kot matrika preslikamo v vektor, v katerega ne vpisujemo praznih členov matrike.

```
(* I,K,L,N,J ... so stevci *)
(* TAB1 je linearizirana LALR tabela *)
(* TAB2 je tabela indeksov *)
begin
procedure LINEARIZIRAJ;
  K:=1;TAB2(1):=1;                    (* inicializacija *)
  for N:=1 to n do
    begin L:=0
      for M:=1 to m do
        begin
          if TAB(N,M) ni 0 do TAB1(K):=(K,n,v,M) end;
          L:=L+1;K:=K+1;
        end;
      TAB2(N+1):=TAB2(N)+L
    end
end
```

4. ZAKLJUČEK

Naveden generator LALR tabel je zelo koristen pripomoček in omogoča razpoznavanje kakršnega koli jezika, ki se ga da izvajati v kontekstno svobodni obliki. Torej ga lahko uporabimo pri implementaciji večine današnjih programirnih jezikov, kakor tudi za komandne (ukazne) jezike. Generator smo preizkusili za proceduralni jezik, za aplikativni jezik in za komandni jezik simulacijskega svežnja. Da smo lahko pri tem odpravljali napake, je bilo potrebno generirati poleg osnovne LALR(1) tabele še tabelo napak. V bodoče nameravamo spremeniti generator tabel tako, da bo sprejel kot vhodne podatke tudi gramatike jezikov v razširjeni BNF obliki.

5. PRIMER

```
K O N S T R U K C I J A   L A L R   T A B E L E

VHODNI TEKST

   1   EXP:=EXP ADDOP TERM / TERM ;
   2   TERM:=TERM MULTOP FACTOR / FACTOR ;
   3   FACTOR:=(. EXP .)./ .IDENT. ;
   4   ADDOP:=.+./ .-. ;
   5   MULTOP:=./. / .*. $

0. NAPAK ODKRITIH

          SIMBOLI

KODA     IME          TIP

   1     EXP          NETERMINAL
   2     ADDOP        NETERMINAL
   3     TERM         NETERMINAL
   4     MULTOP       NETERMINAL
   5     FACTOR       NETERMINAL
   6     (            TERMINAL
   7     )            TERMINAL
   8     IDENT.       TERMINAL
   9     +            TERMINAL
  10     -            TERMINAL
  11     /            TERMINAL
  12     *            TERMINAL
  13     $            KONCNI SIMBOL

          PRODUKCIJE

  1       EXP    --)  EXP     ADDOP   TERM
  2       EXP    --)  TERM
  3       TERM   --)  TERM    MULTOP  FACTOR
  4       TERM   --)  FACTOR
  5       FACTOR --)  (       EXP     )
  6       FACTOR --)  IDENT
  7       ADDOP  --)  +
  8       ADDOP  --)  -
  9       MULTOP --)  /
 10       MULTOP --)  *
```

LITERATURA

1. W. A. Barrett, J. D. Couch, Compiler Construction: Theory and Practice, SRA, 1978.
2. A. V. A Ho, J. D. Ullman, Principles of Compiler Design, Addison-Wesley, 1977.
3. V. Žumer, idr., Raziskava mikroprogramirane arhitekture za implementacijo visokega programskega jezika, VTŠ Maribor 1982.

LINEARIZIRANA LALR TABELA

N	STANJE	TIP	J
1	2	G	1
2	3	G	3
3	4	G	5
4	5	S	6
5	6	S	8
6	7	G	2
7	8	S	9
8	9	S	10
9	0	A	13
10	10	G	4
11	3	R	7
12	3	R	9
13	3	R	10
14	11	S	11
15	12	S	12
16	3	R	13
17	5	R	7
18	5	R	9
19	5	R	10
20	5	R	11
21	5	R	12
22	5	R	13
23	13	G	1
24	3	G	3
25	4	G	5
26	5	S	6
27	6	S	8
28	7	R	7
29	7	R	9
30	7	R	10
31	7	R	11
32	7	R	12
33	7	R	13
34	14	G	3
35	4	G	5
36	5	G	6
37	6	S	8
38	8	R	6
39	8	R	8
40	9	R	6
41	9	R	8
42	15	G	5
43	5	S	6
44	6	S	8
45	10	R	6
46	10	R	8
47	11	R	6
48	11	R	8
49	7	G	2
50	16	S	7
51	8	S	9
52	9	S	10
53	10	G	4
54	2	R	7
55	2	R	9
56	2	R	10
57	11	S	11
58	12	S	12
59	2	R	13
60	4	R	7
61	4	R	9
62	4	R	10
63	4	R	11
64	4	R	12
65	4	R	13
66	6	R	7
67	6	R	9
68	6	R	10
69	6	R	11
70	6	R	12
71	6	R	13

TABELA REDUKCIJ

PRODUKCIJA	DOLZINA	LEVA STRAN
1	3	1
2	1	1
3	3	3
4	1	3
5	3	5
6	1	5
7	1	2
8	1	2
9	1	4
10	1	4

TABELA INDEKSOV

I	INDEX
1	1
2	6
3	10
4	17
5	23
6	28
7	34
8	38
9	40
10	42
11	45
12	47
13	49
14	53
15	60
16	66
17	72