PREVAJALNIK JEZIKA PL/I ZA RAČUNALNIK CYBER 72 – 24

B. Vilfan, V. Batagelj, I. Lajovic, A. Jerman–Blažič
Institut Jožef Stefan, Ljubljana, Jugoslavija

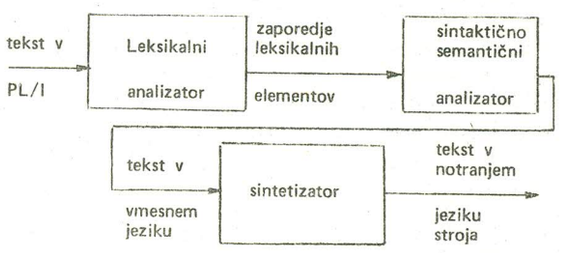
Opisani so glavni principi prevajalnika za PL/I, ki je trenutno v fazi izdelave. Poudarjene so predvsem tiste lastnosti, ki izražajo sodobno programsko prakso: modularnost in splošnost ter prilagodljivost celotne konstrukcije.

A PL/I COMPILER FOR THE CDC CYBER 72–24. We describe the general design principles of a PL/I compiler currently being written. We emphasize particularly those features that represent modern programming philosophy: modularity, generality, and the adaptability of the design.

V začetku leta 1973 smo na Institutu Jožef Stefan začeli z delom na prevajalniku jezika PL/I za računalnik CDC Cyber 72–24. Za specifikacijo jezika smo vzeli nek standardni dokument ([1]) in naš namen je realizirati čim večji del jezika, kot je tam opisan.

Eden od poglavitnih namenov celotnega projekta je pridobivanje izkušenj pri pisanju podobnih programov. Zaradi tega smo se takoj opredelili, da se ne bomo usmerili v neko togo konstrukcijo, čeprav bi taka morda bila boljša glede hitrosti, potrebnega spomina itd., temveč bomo izbrali elastično konstrukcijo, ki dovoljuje enostavno zamenjavo delov, kar olajšuje eksperimentiranje. Na osnovi tega kar je bilo do sedaj narejenega, mislimo, da smo ta cilj uresničili.

Danes velika večina prevajalnikov temelji na opisu jezika v obliki kontekstno svobodne gramatike oz. nekega podrazreda kontekstno svobodnih gramatik. Bolj precizno, kontekstno svobodna gramatika je nekako prvi približek opisu jezika, za katerega je prevajalnik narejen. Dejansko se v podrobnostih jezik precej oddaljuje od kontekstno svobodnega modela. Gramatike vgrajene v prve prevajalnike ki so sloneli na tej ideji (npr. prvi prevajalniki za ALGOL 60) so za končne (terminalne) simbole imele posamezne simbole luknjača oz. teleprinterja. Iz teh simbolov so bila sestavljena števila, imena in ključne besede jezika in nato šele bolj zapletene (višje) gramatične enote. Vendar so kaj hitro ugotovili, da se najnižje gramatične enote (števila, imena) dajo opisati bolj enostavno kot preko kontekstno svobodne gramatike in da uporaba bolj enostavnega opisa tudi ustrezno zmanjša velikost prevajalnika. Zato se danes skoraj izključno uporablja groba razdelitev prevajalnika na sestavne dele kot je to razvidno na sliki 1. Leksikalni analizator je končni avtomat, sintaktično semantični analizator je skladovni avtomat, sintetizator deluje pa kot asembler.



Slika 1. Blok shema prevajalnika

Leksikalni analizator (LA) razpoznava najnižje enote jezika kot so imena in številčne ter znakovne konstante – imenujemo jih leksikalni elementi. Opisani so preko neke regularne gramatike in potemtakem LA deluje v bistvu kot končni avtomat. Pri vsakem prehodu iz enega stanja v drugo, ki je povzročen od vhodnega simbola, se sproži neka „semantična" rutina. Končni učinek vseh semantičnih rutin, ki so delovale med obdelavo nekega leksikalnega elementa, je notranja ponazoritev slednjega.

Naš analizator ima 36 stanj (potrebno je omeniti, da število stanj ni popolna mera zamotanosti analizatorja, ker se število stanj lahko zmanjša na ta način, da se uvedejo razne pomožne spemenljivke, ki označujejo različne situacije) in 27 semantičnih rutin. 63 vhodnih simbolov je grupiranih v 14 skupin, ki dejansko rabijo za vhode avtomata. Vsi znaki znotraj ene skupine imajo isti učinek na analizator, znaki iz različnih skupin pa imajo vsaj enkrat različen učinek.

Notranja ponazoritev leksikalnih elementov je prikazana na sliki 2. LA uporablja slučajnostno kodiranje (angleško, hash coding) katerega bistvo je v tem, da je razdalja dveh različnih leksikalnih elementov v adresnem prostoru enakomerno porazdeljena slučajna spremenljivka neodvisna od leksikografske razdalje ☆ istih elementov ([2]). Tak način kodiranja omogoča razmeroma hitro iskanje nekega zaporedja v seznamu.

LA vsebuje dve tabeli, KAZALO in TEKST, ki imata sledeči funkciji: Naslovi tabele KAZALO rabijo kot slučajnostni kodi leksikalnih elementov, naslovi tabele TEKST pa rabijo kot kodi leksikalnih elementov v ostalih blokih prevajalnika ☆☆. Vsebine (zasedenih) naslovov tabele KAZALO so naslovi v tabeli TEKST, slednji pa vsebujejo podatke o leksikalnih elementih (npr. dolžino, binarno predstavo, itd.). Ko LA sprejme nek element, se na osnovi binarne predstave enoumno izračuna naslov tabele KAZALO. Funkcija, ki realizira to preslikavo (iz binarne predstave elementa v naslov tabele KAZALO), je take narave, da dobimo slučajnostno kodiranje, kot je to bilo opisano zgoraj. Konkretno, če vzamemo neko množico parov elementov katerih binarne predstave se razlikujejo za, vzemimo, c, potem tej množici ustrezajo pari naslovov, katerih razlika je porazdeljena enakomerno po možnih relativnih naslovih tabele KAZALO. V primeru, da je tako dobljeni naslov tabele KAZALO nezaseden, postavimo v njega prvi prosti naslov tabele TEKST, v slednjega pa spravimo podatke o sprejetemu leksikalnemu elementu (njegovo binarno predstavo, dolžino in druge karakteristike). Če je naslov v tabeli KAZALO zaseden, je to lahko rezultat dveh vrst okoliščin: (1) sprejeti leksikalni element je enak nekemu, ki je bil že sprejet. V tem primeru se o tem prepričamo s primerjanjem binarne predstave sprejetega elementa z binarno predstavo elementa, ki se hrani v ustreznem naslovu tabele TEKST. Tedaj že imamo notranjo predstavo sprejetega elementa. (2) Lahko pa se zgodi, da ima nek drugi element (slučajno) isti naslov v tabeli KAZALO. V tem primeru povečujemo naslov tabele KAZALO toliko časa dokler ne najdemo bodisi isti element ali pa prosti naslov.

KAZALO

1	2	3	4	5	6	7	8	9	10	11	...
		9							1		

TEKST

1	2	3	4	5	6	7	8	9	10	11	12	13	.....
x	x	x	4	R	E	A	L	x	x	x	8	F	U	C	T	I	O	N

Najenostavnejši primer delovanja LA

Na vhodu je bil sprejet zadnji znak zaporedja FUNCTION; f(FUNCTION) = 3; KAZALO(3) je nezasedeno; prvi prosti naslov tabele TEKST je 9; v KAZALO(3) postavimo 9; v naslove TEKST(9) in naprej pa podatke o sprejetem elementu (x označuje podatke, ki nas tu ne zanimajo, f: funkcija za izračun slučajnostnih kodov).

Slika 2. Kodiranje elementov v leksikalnem analizatorju

Zgornji opis delovanja LA je nujno malo površen – bolj podroben opis vseh programov prevajalnika bo tiskan kot poročilo skupine, ki dela na nalogi.

Sintaktično – semantični analizator (SSA) ima vgrajeno konteksno–svobodno gramatiko, katere končni simboli so leksikalni elementi oz. podrazredi leksikalnih elementov. Za vmesni jezik smo izbrali nek nabor makroukazov zbirnega jezika COMPASS tako, da kot končni blok prevajalnika (sintetizator) lahko rabi kar COMPASS assembler.

Da bi dosegli kar najbolj enostavno konstrukcijo SSA in čim večjo zamenljivost delov, smo omejili vrsto gramatike, ki jo rabimo. Uporabljamo LL(1) gramatiko, ki spada med deterministične kontekstno–svobodne gramatike ([3,4]). Če LL(1) gramatiko zapišemo v Greibach normalni formi (vse desne strani produkcij se začenjajo s terminalnim simbolom) ugotovimo, da se vse desne strani produkcij, ki imajo isto levo stran, začenjajo z drugačnim terminalnim simbolom. To omogoča analizo od leve proti desni in izbiro produkcije na osnovi enega simbola v skladu in enega simbola na vhodu. Gramatika, ki jo rabimo v prevajalniku je v nekoliko ojačani Greibach obliki, kajti dovoljujemo še uporabo praznih produkcij (tj. produkcij s prazno desno stranjo) oz. če nek neterminalni simbol nima prazne produkcije, lahko dodamo največ eno produkcijo, ki ima ta simbol kot levo stran, desna stran se pa ne začenja s terminalnim simbolom.

V SSA igra osrednjo vlogo razpoznavalni algoritem, ki bere vhod in izbira produkcije. V grobem je definiran takole:
(Pred začetkom delovanja je v skladu aksiom gramatike, na vhodu pa prvi leksikalni element)
1. Če je v skladu terminalni element, skok na 2.
Uporabi produkcijo, ki ustreza paru <element na vhodu, simbol v skladu> in pokliči ustrezno semantično rutino. Če je bila uporabljena produkcija, katere desna stran se ne začenja s terminalnim simbolom, program nadaljuje od 1, drugače od 3.

2. Primerjaj simbol sklada s terminalnim elementom na vhodu.
V primeru enakosti, izvrši skok na semantično rutino, ki je povezana s simbolom v skladu (po izvršitvi semantične rutine, program nadaljuje od 3). V primeru neenakosti, je nastopila napaka.

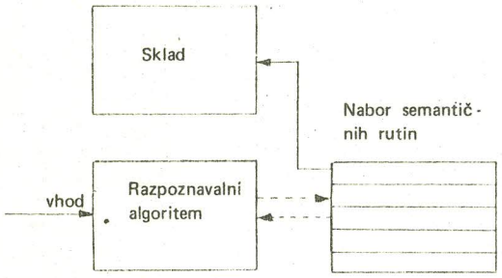
☆ Leksikografsko ureditev neke množice končnih zaporedij znakov nad neko abecedo dobimo, če množico uredimo po istem principu, kot so urejene besede v slovarjih. Pri tem izhajamo iz neke podane ureditve znakov v abecedi. Npr., če znake abecede predstavljamo binarno in je torej ureditev abecede definirana preko binarnih vrednosti znakov, potem prvo definiramo leksikografsko vrednost nekega končnega zaporedja kot binarni ulomek, ki ga dobimo, če postavimo binarno vejico na levi strani zaporedja. Potem definiramo leksikografsko razdaljo dveh zaporedij kot absolutno vrednost razlike njihovih leksikografskih vrednosti.

☆☆ Slučajnostne kode potrebujemo samo v LA, kajti rabijo samo za to, da kar se da hitro ugotovimo, ali je leksikalni element, ki je trenutno na vhodu, že bil obdelan, ali se je pa pojavil prvič in potrebuje novi kod.

3. Preberi nov element vhoda (pokliči LA). Skok na 1.

Omenili smo že, da SSA želimo realizirati tako, da lahko čim lažje zamenjujemo dele. To se nanaša predvsem na gramatiko in semantične rutine. Izbrali smo konstrukcijo,, ki je prikazana na sliki 3. Vsebuje sklad, razpoznavalni algoritem s potrebnimi tabelami ter množico semantičnih rutin. Pri tej konstrukciji je edino razpoznavalni algoritem definiran enkrat za vselej, gramatika in semantične rutine se pa lahko zelo enostavno menjajo.

Sklad je definiran kot COMMON blok poljubne velikosti. Razpoznavalni algoritem je napisan v COMPASS–u, gramatika se pa priključi z večkratno uporabo posebnega makroukaza. Semantične rutine, napisane bodisi v COMPASS–u, ali v FORTRAN–u so lahko dodane kot neodvisni programi.



Prekinjene črte označujejo skoke v programu, neprekinjene črte pa tok podatkov.

Slika 3. Blok shema sintaktično semantičnega analizatorja

Trenutno smo pri delu v fazi preizkušanja gramatike jezika in izdelave semantičnih rutin.

Primer delovanja sintaktično–semantičnega analizatorja

Prevod enostavnega prireditvenega stavka bo vseboval naslednje ukaze v zbranem jeziku

CL	ACC	postavi akumulator na 0
AD	x	prištej vsebino celice x vsebini akumulatorja
SB	x	odštej vsebino celice x od vsebine akumulatorja
ST	x	prenesi vsebino akumulatorja v celico x

Stavki, ki jih bomo prevajali naj bodo zgrajeni glede na sledeča pravila:

		semantične rutine
P1 < stavek >	→ IME = < izraz >	(IME, S1), (= ; S2)
P2	→ NAPAKA	(N, NAPAKA)
P3 < izraz >	→ IME < rep >	(IME, S3)
P4	→ NAPAKA	(N, NAPAKA)
P5 < rep >	→ + < izraz >	(+, S4)
P6	→ − < izraz >	(−, S5)
P7	→ ϵ	(ϵ, S6)

Vsakemu terminalu (posebni znak ali imena, napisana z velikimi črkami) ustreza semantična rutina, ki opravi nalogo, predpisano z razpoznanjem določenega terminala.

S1 spravi ime v celico IM, v celico OP postavi + in generira ukaz CL ACC

S2 nič

S3 če celica OP vsebuje +, generiraj ukaz AD x; sicer generiraj ukaz SB x (x je ime trenutno razpoznanega terminala IME)

S4 v celico OP spravi +

S5 v celico OP spravi –

S6 generiraj ukaz ST (IM), to je, poišči vsebino celice IM ter ga dodaj desno k znakoma ST.

Oglejmo si korakoma potek prevajanja ukaza:

U = X + Y – Z

ki ga računalnik bere od leve proti desni.

korak	številka produkcije	vsebina sklada po koraku	semantična rutina	vsebina OP	vsebina IM	generirani tekst
0		<stavek>				
1	P1	= < izraz >	S1	+	U	CL ACC
2		< izraz >	S2	+	U	
3	P3	< rep >	S3	+	U	AD X
4	P4	< izraz >	S4	+	U	
5	P5	< rep >	S3	+	U	AD Y
6	P6	< izraz >	S5	–	U	
7	P3	< rep >	S3	–	U	SB Z
8	P7		S6	–	U	ST U

Gornjo gramatiko smo zapisali v obliki, ki jo SSA razume in uporabi za prevajanje vhodnega niza – v našem primeru gornjega enostavnega aritmetičnega prireditvenega stavka

```
          IDENT  PRIM
TRI       XTEXT  MAKRO
TRI       XTEXT  EQUSIM
TRI       XTEXT  RAZ
          LIST   -R
          GLAVA  STAVEK
          P      STAVEK,((S1+1,(IM,0),(S2+1,PZ,ENACAJ),(,NT,IZRAZ)),
,                        (N+1,(0,0)))
          P      IZRAZ,((S3+1,(IM,0),(,NT,REP)),
,                        (N+1,(0,0)))
          P      REP,((S4+1,(PZ,PLUS),(,NT,IZRAZ)),
,                        (S5+1,(PZ,MINUS),(,NT,IZRAZ)),
,                        (S6+1,(0,0)))
          KONEC
          END
```

Semantične rutine, ki pripadajo posameznim desnim stranem produkcij so:

```
      SUBROUTINE S1
      INTEGER TEKST
      COMMON /KAZALO/KAZALO(512),TEKST(1024),X/KOD/KOD
      COMMON /PRIM/IM,OP/SIMB/SIMB(4),MASKA,PRVD(20),IPR
      DATA SIMB/10L CL ACC    ,4L AD ,4L SB ,4L ST /
      MASKA=COMPL(MASK(24))
      PRINT 1
    1 FORMAT(#1  PRIMER DELOVANJA LEKSICNEGA ANALIZATORJA#//)
      IPR=1
      IM=KOD.AND.MASKA
      OP=1R+
      PRVD(IPR)=SIMB(1)
      RETURN
      END

      SUBROUTINE S2
      RETURN
      END

      SUBROUTINE S3
      COMMON /PRIM/IM,OP/SIMB/SIMB(4),MASKA,PRVD(20),IPR
      COMMON /KAZALO/KAZALO(512),TEKST(1024),X/KOD/KOD
      IPR=IPR+1
      I=3
      IF(OP.EQ.1R+) I=2
      PRVD(IPR)=SIMB(I).OR.KOD.AND.MASKA
      RETURN
      END

      SUBROUTINE S4
      COMMON /PRIM/IM,OP
      OP=1R+
      RETURN
      END

      SUBROUTINE S5
      COMMON /PRIM/IM,OP
      OP=1R-
      RETURN
      END
```

```
      SUBROUTINE S6
      INTEGER TEKST
      COMMON /KAZALO/KAZALO(512),TEKST(1024),X/KOD/KOD
      COMMON /PRIM/IM,OP/SIMB/SIMB(4),MASKA,PRVD(20),IPR
      IPR=IPR+1
      PRVD(IPR)=SIMB(4).OR.IM
      DO 2 I=1,IPR
    2 PRINT 3,PRVD(I)
    3 FORMAT(1H0,A10)
      RETURN
      END

      SUBROUTINE N
      PRINT 1
    1 FORMAT(1H ,T30,'NEPRAVILEN STAVEK')
      STOP
      END
```

Analiza zgornjega stavka je dala naslednji rezultat:

```
      PRIMER DELOVANJA LEKSICNEGA ANALIZATORJA


     CL ACC
     AD X
     AD Y
     SR Z
     ST U
```

LITERATURA

([1]) IBM Oprating System 1360, PL/I: Language Specifications. IBM Systems Reference Library, Form (28–657)–2, File No. S360–29.

([2]) John Donovan, Systems Programming (Chapter III). McGrow Hill 1971.

([3]) P.M. Lewis II and D.J. Rosenkrantz, An ALGOL Compiler Designed Using Automata Theory. Polytechnic Institute of Brooklyn Symposium on Computers and Automata, April 1971.

([4]) P.M. Lewis II and R.E. Stearns, Syntax--Directed Transduction. Journal of the ACM, Vol. 15, No. 3, July 1968, pp. 465–488.