Cimpoiasu Catalina Ioana 454E 2006-2007
Popa Silviu 455E 2006-2007

Tema SO

1.Compilatoare

Compilatorul este un program de transfer de la un limbaj de nivel înalt, care respectă legile unei gramatici relativ complexe, la o reprezentare în cod maşină, executabil pe o maşină concretă. Rostul compilatorului este de a uşura munca unui programator de calculatoare, care va folosi un limbaj apropiat de limbajul natural, structurat pe legile generale ale logicii. Fiecare limbaj folosit este o variantă de mecanism de conservare şi de transmisie de informaţie, având la bază un set de reguli construite conform legilor logicii. Setul de reguli proprii unui limbaj alcătuiesc gramatica acelui limbaj.

Semnul este o modificare a unei stări staţionare. Semiotica se ocupă cu studiul semnelor.Semantica este o parte a semioticii care studiază relaţia între semn şi realitatea semnificată. Sintaxa este acea parte a semioticii care se ocupă de relaţiile stabilite între semne.Gramatica este un set de reguli ataşate sintaxei unui limbaj.

Legile gramaticale se împart în două categorii:

1. Legi lexicale: set de reguli care stabilesc tipurile şi structura elementelor lexicale permise în reprezentarea limbajului.
2. Legi sintactice: set de reguli care stabilesc compunerea elementelor lexicale în reprezentarea unui limbaj.

Unitatea lexicală este *token*-ul. *Token*-urile sunt formaţiuni de simboluri dispuse în şiruri delimitate precis, de obicei prin caracterul spaţiu. Token-urile sunt adevărate cărămizi cu care se construieşte limbajul (se construiesc aserţiuni). Ele pot fi:

- cuvinte rezervate;
- nume de variabile;
- operatori aritmetici sau logici.

Analiza lexicală

Se ocupă cu examinarea textului sursă din punct de vedere al diferitelor componente (*tokens*) :

- recunoaştere;
- clasificare.

Partea compilatorului care se ocupă cu analiza lexicala se numeşte *scaner* .

Analiza sintactica

Examinează construirea aserţiunilor din elemente lexicale .Partea compilatorului care se ocupa cu analiza sintactica se numeşte *parser* .

Generatorul de cod

Generatorul de cod realizează trecerea din examinarea în limbaj de nivel înalt în cod maşină. El efectuează legătura cu realitatea .Generatorul de cod poate face trecerea spre un limbaj de asamblare sau spre un cod universal. Existenţa celor trei zone ale compilatorului nu presupune trei treceri , fiind uneori posibilă o singură trecere .
Gramatica este un set de legi de compoziţie în care fiecare dintre ele defineşte sintaxa unei construcţii lexicale, astfel încât totalitatea regulilor structurează întreg limbajul.

De exemplu, în legea exprimată în format BNF <read>::=READ(<id_list>)

::= este predicatul, acţiunea asupra părţii din stânga, de a se defini prin explicitatea exprimată în partea dreaptă.
Expresia din stânga este cea care se defineşte.

Expresia din dreapta, definiţia expresiei din stânga, este alcătuită din simboluri neterminale, aşezate între < >, şi simboluri terminale *tokens*, dispuse în afara < >.

În exemplul dat

<read>::=READ(<id_list>)

există următoarele simboluri terminale şi neterminale:

read , id_list = simboluri neterminale;

READ, (,) = simboluri terminale (*tokens*);

Simbolurile neterminale sunt elemente care intră în organizarea legilor gramaticale, legi care stabilesc legaturile între aceste simboluri. Conform unei legi specifice, scanerul află relaţia cu simbolurile mai simple şi urmăreşte următorul simbol neterminal. În cazul nostru, legea care este examinată ulterior va explicita unul dintre simbolurile neterminale întâlnite:

<id_list>::= id | <id_list>,id

în care

| = simbol care separă alternative ;

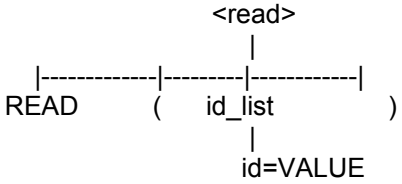
, = simbol care separă identificatorii într-un şir de identificatori.

În acest caz avem un exemplu de definiţie recursivă, (care se defineşte prin ea însăşi).

Identificatorii sunt identificaţi de scaner (care se ocupă cu analiza lexicală).

Rezultatul analizei sintactice este arborele sintactic.

Ex: READ (VALUE)



Analiza lexicală

Modulul compilatorului care face această operaţie se numeşte *scanner .*

Scopul ei este descoperirea şi clasificarea unităţilor lexicale -tokenuri.Ele pot fi:

- cuvinte rezervate ;
- operatori ;
- identificatori ;
- numere întregi , în virgulă mobilă ;
- şiruri de simboluri

Ele provin din limbajul de programare pe care se adaptează compilatorul.

Organizarea lor corespunde gramaticii limbajului.

Am scanare care iau identificatorii ca atare şi scanere care iau identificatorii ca definite din simboluri.

<ident> ::=<letter>|<ident><letter>|<ident><digit>

<letter> ::=A|B|C|D|....|Z

<digit> ::=0|1|2|3|....|9

În acest caz se cunosc ca elemente lexicale simboluri separate litere, cifre ,etc.

Se continuă cu şiruri de simboluri care construiesc identificatorii.Programe ale scanerului aleg identificatorii şi-i clasifică.

Scanerul ia textul , îl scanează şi face un tabel.Rezultatul scanării : şirul de tokens , cărora li se dau coduri.

In timpul scanarii, in afara codarii token-urilor, unde este cazul se pastreaza si alte informatii ale token-urilor.

Specificatoare de token.

Rezultatul: Tabela asemanatoare cu tabela de simboluri din asambloare. Specificatoare de token sint codificari pentru diferite prezentari de date: numere in virgula mobila, siruri de simboluri etc.

Deci functia de baza a scanerului este: descoperirea si codarea token-urilor, alcatuirea tabelei de token-uri cu specificatoare, examinarea textului sursa si controlul listingului.

Observatii: -scanerul trebuie sa decida asupra token-ului in functie de context.
READ cuvint cheie
READ valoare a unei variabile sir
-excluderea comentariilor, daca e cazul
-considerarea organizarii liniei sursa (exista sau nu numarator de linie)
-continuarea asertiunii pe mai multe linii.

Analiza sintactica

Scop: construirea arborelui structurii gramaticale a programului.
Analiza lexicala a identificat elementele de limbaj .
Analiza sintactica releva ordinea in care se efectueaza operatiile.
Metode de descompunere gramaticala:
-metoda ascendenta(de jos in sus)
-metoda descendenta(de sus in jos)
Se porneste de la scopul general care este radacina descompunerii gramaticale si se continua cu nivelurile tot mai de jos conform cu sintaxa asertiunilor.

Generarea codului obiect

Subprograme pentru fiecare lege si pentru fiecare alternativa in lege.
In urma examinarii textului, se decide care subprogram corespunde textului
=> programe semantice
atasarea unei actiuni (cu sens-lexic-sintaxa-semantica) formei de pina acum;
genereaza chiar ele cod
Mai complicat
genereaza o reprezentare intermediara care va fi folosita ulterior la generare de cod.
=>Metode de optimizare a codului
Optimizare independenta de procesor in transferul spre reprezentarea intermediara.
Optimizare dependenta de procesor in transferul intre reprezentarea intermediara si cod masina.

Variante de constructie a compilatoarelor

1. cu o singura trecere

2. cu mai multe treceri

Desi sint 3 sectii, exista o varianta in care la o singura trecere se realizeaza tot; alte variante construiesc fisiere. Pentru o singura trecere, viteza compilatorului e mai mare, dar performantele ar putea fi neoptimizate. Compilatoarele cu o singura trecere sint eficace atunci cind nu se cere o performanta deosebita si cind e nevoie ca de fiecare data sa se faca toata compilarea;

Compilatoarele cu mai multe treceri sint utile atunci cind utilizatorii folosesc ceva din sursa initiala in comun; calitatea compilatorului si a rezultatelor obtinute e superioara dar viteza e mai mica; se folosesc mai eficient resursele masinii; se cistiga atunci cind sursa e folosita o singura data, iar utilizatorii folosesc rezultate specifice fiecaruia.

1. Sa se scrie un program intr-un limbaj oarecare prin care sa se calculeze diferenta intre media geometrica si media aritmetica a 2 numere intregi:
   -sa se descrie setul de legi BNF ale programului;
   -sa se descrie rezultatul actiunii scanerului si al parserului;
   -sa se scrie programul obiect quartet.

```
1. PROGRAM MEDIA;
2. VAR
3. MG, MA, D, A, B : REAL;
4. BEGIN
5. MA:=0;
6. MG:=0;
7. D:=0;
8. READ(A);
9. READ(B);
10. MA:=(A+B) DIV 2;
11. MG:=SQRT(A*B);
12. D:=MA-MG;
13. WRITE (D);
14. END.
```

1.<prog>::=PROGRAM<prog_name> VAR <dec_list> BEGIN <stmt_list>end.
2.<prog_name>::=id
3.<dec_list>::=<dec>|<dec_list>; <dec>
4. <dec>::=<id_list>:<type>
5.<type>::=REAL
6.<id_list>::=id|<id_list>;id
7.<stmt_list>::=<stmt>|<stmt_list>; <stmt>
8.<stmt>::=<assign>|<read>|<write>
9.<assign>::=id::=<exp>
10.<exp>::=<term>|<<exp>+<term>|<exp>-<term>
11.<term>::=<factor>|<term * factor>|<term SQRT factor>|term DIV factor>
12.<factor>::=id|int|(<exp>)
13.<read>::=READ(< id_list>)
14.<write>::=WRITE(<id_list>)
15.<body>::=<stmt>|BEGIN <stmt_list>END

Tabel pentru token-uri; Actiunea scanerului

UNITATEA LEXICALA	COD
PROGRAM	1
VAR	2

BEGIN	3
END.	4
REAL	5
READ	6
WRITE	7
;	8
:	9
,	10
:=	11
+	12
-	13
*	14
DIV	15
SQRT	16
(	17
)	18
ID	19
INT	20

Tabelul specificatorilor de program

linie	token	specificator
1	1 19 8	 ->medie
2	2	
3	19 10 19 10 19 10 19 10 19 9 5	->MG ->MA ->D ->A ->B
4	3	
5	19 11 20 8	->MA
6	19 11 20 8	->MG
7	19	->D
	11	
	20	
	8	

Parserul ascendent

Linia 10: ma:=(a+b) div 2;
id1:=(id2+id3) div int;
<.N1.>
id1:=(<N1>+id3) div int;
<.N2.>
id1:=(<N1>+<N2>) div int;
<.N3.>
id1:=<N3> div int;
<.N4.>
id1:=<N3> div <N4>
<.N5.>
id1:=<N5>

<.N6.>
<N6>:=<N5>
_____ <.N7.>

Programul in quarteti

	Operand0	Operand1	Operand2	Rezultat
1.	:=	#0		MA
2.	:=	#0		MG
3.	:=	#0		D
4.	CALL PARAM	XREAD A		
5.	CALL PARAM	XREAD B		
6.	+ Div :=	A i1 i2	B #2	i1 i2 MA
7.	* Sqrt	A i1	B	i1 i2
8.	:= - :=	i2 MA I1	 MG	MG i1 D
9.	CALL PARAM	XWRITE D		