Limbaje Formale si Translatoare
Laborator 4

Februarie 2021

1 Scopul Lucrarii

1. Intelegerea modului de functionare al automatului cu stiva generat de Yacc
2. Identificarea si rezolvarea conflictelor
3. Analiza si intelgerea mai multor exemple de cod Yacc
4. Solutionarea unui interpretor pentru un calculator

2 Notiuni Teoretice

2.1 Mediul Yacc - Continuare

Cand utilizatorul introduce o specificatie pentru Yacc, iesirea este un fisier de programe C, numit in general y.tab.c. Functia produsa de Yacc se numeste yyparse() si returneaza un intreg. Ea utilizeaza repetat yylex(), analizorul lexical furnizat de catre utilizator, pentru a obtine tokeni din intrare. cand detecteaza o eroare ce nu poate fi acoperita, yyparse() returneaza valoarea 1.

Pentru a obtine un program functional, utilizatorul trebuie sa-i furnizeze analizorului o anumita cantitate de "mediu". De exemplu, trebuie definit un program main, care eventual apeleaza yyparse(), si o rutina yyerror() care scrie un mesaj cand e detectata o eroare de sintaxa. Aceste doua rutine trebuie furnizate intr-o forma sau alta de catre utilizator.

Pentru a usura efortul de utilizare a Yacc-ului, acestuia i s-a atasat o biblioteca cu versiuni implicite de main si yyerror(). Aceste programe implicite sunt foarte simple:

```
#include <stdio.h>

main()
{
    return(yyparse());
}
yyerror(char *s);
{
    fprintf (stderr, "%s\n",s);
}
```

Argumentul lui yyerror() e un sir de caractere continand un mesaj de eroare (de obicei "syntax error"). In general, la detectarea unei erori de sintaxa programul afiseaza impreuna cu mesajul de eroare si numarul liniei din fisierul de intrare in care a aparut eroarea. Variabila intreaga externa yychar contine numarul token-ului look-ahead din momentul detectarii erorii si poate ajuta la debugging.

Variabila intreaga externa yydebug e setata in mod normal la valoarea zero. Daca ea are o alta valoare, analizorul produce si o descriere detaliata a actiunilor sale, inclusiv informatii despre simbolul de intrare citit, si despre actiunile efectuate.

2.2 Modul de lucru al analizorului sintactic

Analizorul generat de Yacc pe baza specificatiilor date este relativ simplu si consta dintr-un **automat stiva**. Spre deosebire de automatele folosite la analiza lexicala care sunt automate finite de stare, yacc foloseste automat stiva.

Stiva este necesara pentru a putea impinge si extrage de pe stiva stari. Starea curenta este intotdeauna cea din varful stivei. Acest lucru permite parser-ului sa poata trata si limbaje independente de context, nu doar limbaje bazate pe gramatici regulate. Analizorul e capabil sa citeasca si sa memoreze simbolul de intrare urmator, numit token look-ahead.

2.2.1 Automat stiva

Automatul are 4 actiuni : *shift*, *reduce*, *accept* si *error*. Initial masina e in starea 0 (zero), stiva contine starea 0 si nici un token "look-ahead" nu a fost citit.

- Pe baza starii curente, se decide daca trebuie citit inainte ("look-ahead") un token pentru a hotari actiunea de efectuat (shift sau reduce). Daca e nevoie de un nou token, e apelat yylex() pentru a-l obtine.
- Folosind starea curenta si eventual token-ul "look-ahead", analizorul decide actiunea sa urmatoare si o efectueaza. Aceasta poate determina punerea pe stiva a unor stari (push), preluarea de pe stiva a unor stari (pop), consumarea tokenului urmator sau doar examinarea inainte a tokenului urmator.
- actiunea de *shift* a unui token consuma acest token de la intrare. Pentru a exemplifica shift asupra unui *token*, sa presupunem ca intr-o stare avem actiunea

  ```
  IF shift, and go to state 34
  ```

  Atunci, daca token-ul urmator e IF, starea 34 devine stare curenta si e pusa pe varful stivei. Token-ul IF este consumat (sters) din intrare.
- actiunea *Reduce* e realizata cand partea dreapta a unei reguli gramaticale a fost identificata. Se inlocuieste partea dreapta cu partea stanga.
  - pentru a decide actiunea de reduce, e posibil sa fie necesar sa se consulte token "look-ahead" (dar nu e obligatoriu)
  - actiunea de reduce e asociata unei reguli gramaticale individuale
  - se scot de pe stiva atatea stari cate simboluri sunt in partea dreapta a regulii
  - exemplu: fie regula

    $$A : X\ Y\ X;$$

1. Pentru a aplica *reduce*, se elimina de pe stiva primele 3 stari. Aceste stari au fost puse pe stiva cand s-a recunoscut X, Y si Z.
2. Dupa ce s-au eliminat (pop) aceste stari, varful stivei contine starea in care analizorul era cand a inceput sa proceseze regula (pt care s-a facut reduce acum).
3. Folosind aceasta stare si simbolul nonterminal A din stanga regulii, se realizeaza ceea ce ar fi *shift* a lui A, adica se impinge pe stiva o noua stare. In automatul construit de yacc, aceast pas e marcat de *goto*. Pasul de goto nu consuma token de la intrare. De exemplu, starea din varful stivei dupa eliminarea starilor asociate lui X, Y, Z (consideram ca mai sus, ca exista regula A: X Y Z;) contine o descriere ca:

   ```
   A go to state 20
   ```

   Starea 20 e impinsa pe stiva si devine stare curenta. Analizorul se comporta apoi ca si cand ar fi vazut partea stanga in acel moment.

- actiunea de *accept* indica faptul ca intreaga intrare a fost vazuta si ca aceasta se potriveste cu specificatia. Aceasta actiune apare cand token-ul deja citit din intrare este end-marker-ul, si indica faptul ca analizorul a terminat procesarea cu succes.

- actiunea *error* reprezinta un punct din care nu se mai poate continua analiza conform specificatiei. Tokenii deja consumati din intrare impreuna cu token-ul urmator din intrare nu pot fi urmati de nimic ce ar constitui o intrare corecta.

  Analizorul raporteaza o eroare si incearca sa refaca situatia si sa reia analiza.

Actiunea de *reduce* e importanta si in tratarea actiunilor si valorilor furnizate de utilizator. Cand o regula este redusa, **codul furnizat cu regula e executat inainte de modificarea stivei**. In paralel cu stiva ce pastreaza starile, exista o alta stiva ce pastreaza valorile returnate de analizorul lexical si de actiuni. Cand are loc un shift, variabila externa *yylval* e incarcata (push) pe stiva de valori. Dupa revenirea din codul utilizator, e efectuata actiune de *reduce*. Cand e realizata actiunea goto, variabila externa **yyval** e incarcata (push) pe stiva valorilor. Pseudo-variabilele $1, $2, ... refera de fapt stiva de valori.

Intuitia e urmatoarea: "o actiune de tip shift presupune inaintarea in lista de tokeni, iar o actiune reduce marcheaza gasirea unei potriviri cu partea dreapta a unei productii".

Fiecare stare a parser-ului va coincide cu una sau mai multe situatii (situatie = regula gramaticala aflata in stadiu diferit de procesare a partii drepte). Cu punct se identifica pozitia pana la care o regula a fost procesata. De exemplu, avand regula A $\rightarrow$ a b c, unde a si b au fost procesate, dar nu si c, vom scrie:

A $\rightarrow$ a b . c

Fiecare stare a parser-ului va avea asociate astfel de reguli gramaticale cu punct. Primul pas pentru obtinerea automatului generat pentru parser este utilizarea optiunii yacc -v, ce produce fisierul *y.output* continand informatii despre stari.

2.3 Exemplul 1

Sa consideram specificatia de mai jos care, la compilare cu -v, produce fisierul *y.output* din Listing 1.

```
1   %token        DING  DONG  DELL
2
3   %%
4
5   rhyme    :    sound  place;
6   sound    :    DING  DONG;
7   place    :    DELL;
```

Figura 1: Specificatia fisierului Yacc

```
State 0
    0 $accept: . rhyme $end
    DING  shift, and go to state 1
    rhyme  go to state 2
    sound  go to state 3

State 1
    2 sound: DING . DONG
    DONG  shift, and go to state 4

State 2
    0 $accept: rhyme . $end
    $end  shift, and go to state 5

State 3
    1 rhyme: sound . place
    DELL  shift, and go to state 6
    place  go to state 7

State 4
    2 sound: DING DONG .
    $default  reduce using rule 2 (sound)

State 5
    0 $accept: rhyme $end .
    $default  accept

State 6
    3 place: DELL .
    $default  reduce using rule 3 (place)

State 7
    1 rhyme: sound place .
    $default  reduce using rule 1 (rhyme)
```

Listing 1: Fisierul y.output pentru gramatica DING DONG DELL

Stiva	Intrare	Actiune realizata	Stare curenta (urmariti deplasarea .)
#0	DING DONG DELL $end	shift	$accept: . rhyme $end
#01	DONG DELL $end	shift	sound : DING . DONG
#014	DELL $end	reduce cu regula 2 (sound)	sound: DING DONG .
#03	DELL $end	shift	rhyme: sound . place
#036	$end	reduce cu regula 3 (place)	place: DELL .
#037	$end	reduce cu regula 1 (rhyme)	rhyme : sound place .
#02	$end	shift	$accept: rhyme . *end*
#025		accept	$accept: rhyme *end.*
#02			

Tabela 1: Modul de functionare a automatului pe DING DONG DELL

Sa presupunem ca intrarea este: DING DONG DELL. Operatiile pe stiva decurg conform urmatorilor pasi, cu mentiunea ca punctul e utilizat pentru a indica ce a fost vazut si ce trebuie sa urmeze, in fiecare regula (vezi tabelul 1):

1. Initial, starea curenta e 0. Analizorul trebuie sa analizeze intrarea pentru a alege una dintre actiunile disponibile in starea 0, deci primul token, DING, e citit, devenind token look-ahead. In starea 0, actiunea pentru DING e shift, and go to state 1, deci starea 1 este pusa pe stiva si token-ul look-ahead e sters. Starea 1 devine stare curenta.

2. In starea 1, urmatorul token, DONG e citit, devenind token look-ahead. Actiunea pentru token-ul DONG este shift, and go to state 4, deci starea 4 e pusa pe stiva si token-ul look-ahead e sters. Stiva contine acum 0, 1 si 4.

3. In starea 4, fara a fi necesara consultarea look-ahead, analizorul reduce prin regula 2: sound : DING DONG. Aceasta regula are doua simboluri in partea dreapta, deci doua stari, 4 si 1, sunt luate de pe stiva, lasand in varful stivei starea 0.

4. In starea 0, consultand descrierea si cautand un go to pentru sound, se obtine: sound go to state 3. Astfel starea 3 e pusa pe stiva, devenind stare curenta.

5. In starea 3 e citit urmatorul token, DELL. Actiunea e shift, and go to state 6, deci starea 6 e pusa pe stiva, care contine acum 0, 3 si 6 si token-ul look-ahead e sters.

6. In starea 6, singura actiune e reducerea prin regula 3 (place). Aceasta are un simbol in partea dreapta, deci se ia o stare de pe stiva, 6, ramanand starea 3 in varful stivei.

7. In starea 3, actiunea go to pentru place e starea 7. Acum, stiva contine 0, 3 si 7.

8. In starea 7 singura actiune posibila e reducerea prin regula 1 (rhyme). Exista doua simboluri in dreapta, deci sunt preluate de pe stiva doua stari, lasand starea 0 din nou pe varful stivei.

9. In starea 0, exista un go to pentru rhyme, prin care analizorul intra in starea 2.

10. In starea 2 urmatorul token, end-marker, e citit (indicat prin '$end' in fisierul y.output). Actiunea in starea 2 cand se recunoaste end-marker-ul e shift spre starea 5. Acum, stiva contine starile 0, 2 si 5.

11. In starea 5, actiunea implicita este acccept, se sterge tokenul look-ahead, iar analiza este terminata cu succes.

2.4 Exemplul 2 - Conflicte

Se considera gramatica al carei limbaj e 0^{2n}.
S $\rightarrow$ 0 S 0
S $\rightarrow \epsilon$

Regulile corespunzatoare yacc sunt:

```
 9  S: '0' S '0'
10    | /* epsilon */
11    ;
```

Figura 2: Descrierea Yacc pentru gramatica 0^{2n}

Odata invocat yacc, se obtine:
yacc -d -v first.y
first.y: warning: 1 shift/reduce conflict [-Wconflicts-sr]

Prima linie din fisierul y.output descrie locatia si cauza conflictului, in cazul nostru:
State 1 conflicts: 1 shift/reduce

Fisierul generat listeaza regulile gramaticale, plus o extra regula generata de yacc. Se listeaza terminalii si non-terminalii, dupa care se listeaza starile parser-ului alaturi de expresiile punct asociate lor.

Pentru starea 1, in cadrul careia exista un conflict, prima linie contine descrierea starii prin notatie punct. Mai departe, conform productiei S $\rightarrow$ '0' S '0', daca urmeaza un '0' la intrare, se poate realiza o actiune shift, iar automatul ar tranzitiona in starea 1. In acelasi timp, conform productiei S $\rightarrow \epsilon$, daca urmeaza un '0' la intrare, se poate realiza o reducere. Aceasta situatie marcheaza un **conflict**. Ultima linie corespunzatoare starii 1, "S go to state 3" se refera la o tranzitie spre starea 3, in urma actiunii de reducere.

Fisierul y.output arata astfel:

```
    State 1 conflicts: 1 shift/reduce

Grammar
    0 $accept: S $end
    1 S: '0' S '0'
    2  | %empty
...
State 1

    1 S: '0' . S '0'
    '0'  shift, and go to state 1
    '0'  [reduce using rule 2 (S)]
    S  go to state 3
...
```

Listing 2: Fisierul y.output pentru gramatica 0^{2n}

2.4.1 Conflicte Shift/Reduce

Apare cand intr-o stare se poate realiza atat actiune de shift cat si de reduce. Pentru exemplul prezentat anterior, starea 1 e o stare in care s-a inceput recunoasterea unui S cu regula 1, s-a consumat un 0 si se asteapta recunoasterea unui S (cel de dupa .) urmat de un 0. Se poate observa ca Yacc considera doua posibilitati pentru aceeasi intrare '0':

1. Caracterul '0' care urmeaza in input indica ca se va recunoaste un nou S format conform primei reguli ($S \rightarrow 0S0$); acest nou S e cel mentionat dupa punctul descrierii starii 1 (S $\rightarrow$ '0' . S '0') In acest caz, parser-ul realizeaza shift pentru tokenul de intrare si incepe recunoasterea unui S de forma 0S0.

2. Caracterul '0' in input e vazut ca ultimul '0' din S $\rightarrow$ '0' . S '0', pentru cazul in care S din partea dreapta a productiei s-ar descompune prin productia S $\rightarrow \epsilon$. In acest caz, parser-ul ar realizeaza o actiune reduce.

2.4.2 Conflicte Reduce/Reduce

Acest tip de conflict poate aparea atunci cand cel putin doua productii gramaticale pot fi folosite pentru o actiune de reducere intr-o anumita stare a automatului. De exemplu, considerand gramatica de mai jos:

S $\rightarrow$ T
S $\rightarrow \epsilon$
T $\rightarrow$ 0 0 T
T $\rightarrow \epsilon$

Descrierea acestei gramatici in Yacc ar face ca la compilare sa apara un conflict de tip reduce/reduce deoarece la intalnirea in intrare a unui token EOF, acel token ar putea fi derivat fie din S, fie din T, rezultand astfel doua posibilitati de actiuni reduce.

2.5 Eliminarea Conflictelor

Analizand automatul generat, putem sa gasim motivul pentru conflictele ce pot aparea, si se pot face eventuale modificari la gramatica sau la fisierul yacc pentru eliminarea conflictului si pastrarea limbajului. Ca si tehnici, amintim urmatoarele:

1. **Restructurarea Gramaticii:** Se poate schimba gramatica pentru a elimina sursa conflictului. In exemplul din Figura 2, motivul conflictului shift/reduce este acela ca in starea S $\rightarrow$ '0' . S '0' nu se poate determina daca o intrare '0' vine de la recunoasterea lui S (shift), sau de la finalul productiei curente, in cazul in care s-ar recunoaste ϵ ca S (reduce).

   Astfel, s-ar putea restructura gramatica in felul urmator, pentru a elimina conflictele:

   S $\rightarrow$ '0' '0' S
   S $\rightarrow \epsilon$

2. **Informatii Aditionale catre Yacc:** Se pot oferi informatii aditionale Yacc-ului pentru ca acesta sa decida care dintre actiunile in conflict ar trebui aleasa in situatii ambigue.

   In general, ambiguitatea apare atunci cand nu exista destule informatii despre **precedenta** si **asociativitatea** unor operatori folositi in expresii. Aceste tipuri de conflicte shift/reduce se

(a) *shift* action (b) *reduce* action

Figura 3: Variante de parsare ce arata ambiguitatea operatorilor

pot elimina folosind declaratii ca **%left**, **%right** sau **%prec**.

De exemplu, sa consideram gramatica de mai jos:

E → E '+' E
S → id

Atunci cand parser-ul intalneste '+', va exista un conflict intre posibilitatea ca '+'-ul sa fie recunoscut ca parte din E → E . '+' E (shift), conform Figurii 3 a, dar si posibilitatea ca el sa fie recunoscut dintr-o ramura recursiva a lui E, precum in Figura 3 b (reduce).

Acest conflict se poate rezolva si rescriind gramatica, dar se poate solutiona mai usor indicand asociativitatea dorita pentru operatorul '+'.

3. **Default**: Yacc-ul va alege o actiune implicita
   - Intr-un conflict shift-reduce se alege shift.
   - Intr-un conflict reduce-reduce se alege regula gramaticala care apare prima (in secventa de intrare).

E important de retinut ca nu toate conflictele pot fi eliminate. Pentru unele gramatici, pastrarea echivalentei in urma eliminarii conflictelor nu este posibila.

3 Desfasurarea Laboratorului

3.1 Functionare yacc

Analizati modul de functionare al analizorului sintactic folosind y.output pentru fisierele first si pentru exemplul DING-DONG.

3.2 Expresii Matematice

Analizati fisierele dc1.l si dc1.y. analizati rezultatele pe diverse expresii: 2+3*4 si 2*3+4. Se obtine rezultatul corect? Propuneti solutii de corectare

3.3 Database Configuration

Analizati utilizarea union din exemplul db_config din laboratorul din saptamana 4.

3.4 Calculator

Completati codul din calc1 in asa fel incat sa fie un interpretor pentru un limbaj care permite utilizarea variabilelor identificate de o singura litera si instructiuni de atribuire si de printare a unei expresii; o expresie poate fi o variabila, un numar, sau expresii formate folosind adunare/scadere. Orice instructiune se termina cu ';', si se iese din program cu instructiunea exit.

```
a = 1 + 100;
print a;
B = a - 10;
print B;
print a+ B;
exit;
print 2;
```

Pasi:

(a) Mai intai testati codul asa cum e, fara actiuni in codul yacc. Testati cu cod in limbajul descris corect, respectiv cu erori.

(b) Analizati functiile C incluse.

(c) Inlocuiti "your code here" cu actiunile C potrivite.