Γλώσσες Προγραμματισμού-Μεταγλωττιστές

Εργασία 2η

Φαίδρα Θεοχαρίδη ics22069

Πανεπιστήμιο Μακεδονίας 2023

ΠΕΡΙΕΧΟΜΕΝΑ

Παράδειγμα 1

Ξεκινώντας από το 1ο παράδειγμα, έχουμε μια μόνο εντολή“(print (3 4 +))“ και μια εντολή για την αρχή του προγράμματος “start simple1” και μια για το τέλος “end”. Άρα πρέπει να ποσθέσουμε τον αρχικό κανόνα της γαρμματικής S και επίσης έναν (ή περισσότερους) κανόνα που θα μπορεί να υλοποιεί την εντολή που βρίσκετε στο σώμα της συνάρτησης simple. Έτσι προκύπτει:

PROGRAM ::= “start” “name”* BODY “end”
BODY ::= “(” BODY BODY “)” | “print” | “number”* “number” ‘+’ | ε

Όπου “name” μια μεταβλητή η οποία παραδεχόμαστε ότι μπορεί να ξεκινάει μόνο με χαρακτήρα αλλά να περιέχει και νόυμερο και τον ειδικό χαρακτήρα “_”, δηλαδή αντιστοιχίζεται στη κανονική έκφραση {LETTER}({LETTER}| {DIGITS}| _)
Όπου, LETTER=[a-zA-Z] και DIGITS=[0-9]
*Όπου “number” αντιστοιχίζεται στη κανονική έκφραση {INT}
Όπου INT= ([1-9] | 0)+

Κάποια κομμάτια κώδικα που προστέθηκαν
Flex:

```
DIGITS [0-9]
LETTER [a-zA-Z]
NAME {LETTER}({LETTER}|{DIGITS}|"_")*
%%
"start"   {return T_START;}
"end"     {return T_END;}
"print"   {return T_PRINT;}

"(" {return '(';}
")" {return ')';}
"+" {return ‘+';}

{NAME}    {yylval.lexical = strdup(yytext); return T_NAME;}
{NUMBER}  {yylval.lexical = strdup(yytext); return T_NUMBER;}
%%
```

Bison:

%token '('
%token ‘)'

%token T_start "start"
%token T_end "end"
%token T_print "print"
%token T_NAME "name"
%token T_NUMBER “number”

%left ‘+'

Παράδειγμα 2

Στο 2ο παράδειγμα έχουμε τις εντολές “(print (3 4 + 7 *))” και “(print (3.0 4.0 + 7.0 *))” στο BODY το οποίο σημαίνει ότι η γραμματική πρέπει να μετασχηματιστεί ως εξής:

PROGRAM ::= “start” “name” BODY “end”
BODY ::= “(” BODY BODY “)” BODY | “print” | “number”* “number” ‘*’| “number” “number” ‘+’| ε

*Όπου “number” αντιστοιχίζεται στη κανονική έκφραση {INT} | {FLOAT}
Όπου INT= ([1-9] | 0)+ και FLOAT= [0-9]+\.[0-9]+

Θα πρέπει να ορίσουμε τις σημασιολογικές ρουτίνες έτσι ώστε να μην επιτρέπεται η πρόσθεση με ένα ή παραπάνω float ορίσματα.

Κάποια κομμάτια κώδικα που προστέθηκαν
Flex:

"*" {return ‘*';}

Bison:

%left '*'

Παράδειγμα 3

Για το 3ο παράδειγμα έχουμε την εντολή (x 1) της οποίας η μορφή δεν καλύπτεται από την ήδη υπάρχουσα γραμματική και επιπλέον την (y (10 x +)). Μετασχηματίζουμε ως εξής:

PROGRAM ::= “start” “name” BODY “end”
BODY ::= “(” BODY BODY “)” BODY | “print” | MATH | ε
MATH ::= MATH MATH ‘+’ | MATH MATH ‘*’ | “name” | “number” | ε

Παράδειγμα 4

Δεν υπάρχει κάποια εντολή η οποία δεν καλύπτεται η παραπάνω γραμματική.

Παράδειγμα 5

Προσαρμόζουμε την γραμματική ώστε να υποστηρίζει type conversion
(x (3 (int 4.0) +))

PROGRAM ::= “start” “name” BODY “end”
BODY ::= “(” BODY BODY “)” BODY | “print” | MATH | CONV | ε
MATH ::= MATH MATH ‘+’ | MATH MATH ‘*’ | “name” | “number” | ε
CONV ::= “int” BODY

Κάποια κομμάτια κώδικα που προστέθηκαν
Flex:

"int" {yylval.tokentype = type_integer; return T_type;}

Bison:

%token <tokentype> T_type

Παράδειγμα 6

Δεν υπάρχει κάποια εντολή η οποία δεν καλύπτεται η παραπάνω γραμματική.

Παράδειγμα 7

Προσαρμόζουμε την γραμματική ώστε να υποστηρίζει type conversion (int και float) (x (float 3 4 +))

PROGRAM ::= “start” “name” BODY “end”
BODY ::= “(” BODY BODY “)” BODY | “print” | MATH | CONV | ε
MATH ::= MATH MATH ‘+’ | MATH MATH ‘*’ | “name” | “number” | ε
CONV ::= “int” BODY | “float” BODY

Κάποια κομμάτια κώδικα που προστέθηκαν
Flex:

"float" {yylval.tokentype = type_real; return T_type;}

Παράδειγμα 8

Προσαρμόζουμε την γραμματική ώστε να υποστηρίζει την δημιουργία πινάκων (y [forall i in 3..4]) και την εκτύπωση πινάκων (print y[0])

PROGRAM ::= “start” “name” BODY “end”
BODY ::= “(” BODY BODY “)” BODY | PRINT | MATH | CONV | ARRAY| ε
PRINT ::= “print” BODY
MATH ::= MATH MATH ‘+’ | MATH MATH ‘*’ | “name” | “number” | ε
CONV ::= “int” BODY | “float” BODY
ARRAY ::=‘[’ “forall” “name” “in” “number” ‘.’’.’ ”number” ‘]’ | “name” ‘[’ ” number” ‘]’

Κάποια κομμάτια κώδικα που προστέθηκαν
Flex:

"forall" {return T_FORALL;}
"in"   {return T_IN;}

"[" {return '[';}
"]" {return ']';}
"." {return ‘.';}

Bison:
%token '['
%token ']'

%token '.'

%token T_forall "forall"
%token T_in "in"

Παράδειγμα 9

Προσαρμόζουμε την γραμματική ώστε να υποστηρίζει την εκτέλεση πράξεων με πίνακες (x (100 arr[0] +))

PROGRAM ::= “start” “name” BODY “end”
BODY ::= “(” BODY BODY “)” BODY | PRINT | MATH | CONV | ARRAY| ε
PRINT ::= “print” BODY
MATH ::= MATH MATH ‘+’ | MATH MATH ‘*’ | “name” | “number” | ARRAY | ε
CONV ::= “int” BODY | “float” BODY
ARRAY ::=‘[’ “forall” “name” “in” “number” ‘.’’.’ ”number” ‘]’ | “name” ‘[’ ” number” ‘]’

Τελική Γραμματική

S ={PROGRAM}
T={“start”, “end”, “print”, “name”, “number”, ‘+’, ‘*’, “int”, “float”,”forall”, “in”, ‘[’, ‘]’, ‘.’}
N={PROGRAM, MATH ,BODY, CONV, PRINT, ARRAY}
P=
PROGRAM ::= “start” “name” BODY “end”
BODY ::= “(” BODY BODY “)” BODY | PRINT | MATH | CONV | ARRAY| ε
PRINT ::= “print” BODY
MATH ::= MATH MATH ‘+’ | MATH MATH ‘*’ | “name” | “number” | ARRAY | ε
CONV ::= “int” BODY | “float” BODY
ARRAY ::=‘[’ “forall” “name” “in” “number” ‘.’’.’ ”number” ‘]’ | “name” ‘[’ ” number” ‘]’

Κώδικας

Flex

```
#include <stdlib.h>
#include <string.h>
int line = 1;
%}

NZDIGITS [1-9]
DIGITS [0-9]
LETTER [a-zA-Z]
NAME {LETTER}({LETTER}|{DIGITS}|"_")*
FLOAT {DIGITS}+\.{DIGITS}+
INT  ({NZDIGITS}|0)+
NUMBER {INT}|{FLOAT}
newline \n|\x0A|\x0D\x0A
%%

"start"   {return T_START;}
"end"     {return T_END;}
"print"   {return T_PRINT;}
"forall"  {return T_FORALL;}
"in"      {return T_IN;}

"int" {yylval.tokentype = type_integer; return T_type;}
"float" {yylval.tokentype = type_real; return T_type;}

"(" {return '(';}
")" {return ')';}
"[" {return '[';}
"]" {return ']';}
"." {return '.';}

"+" {return '+';}
"*" {return '*';}


{NAME}    {yylval.lexical = strdup(yytext); return T_NAME;}
{NUMBER}  {yylval.lexical = strdup(yytext); return T_NUMBER;}
{newline}  { line++;}
```

```
[ \t] { /* nothing */ }
. {
        printf("Lexical Analysis: Unexpected String! :: %s. in line %d.
\n",yytext,yylineno); }

%%
```

Bison

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Flex Declarations
/* Just for being able to show the line number were the error occurs.*/
extern int line;
extern FILE *yyout;
int yylex();
/* Error Related Functions and Macros*/
int yyerror(const char *);
int no_errors;
/* Error Messages Macros*/
#define ERR_VAR_DECL(VAR,LINE) fprintf(stderr,"Variable :: %s on line %d.
",VAR,LINE); yyerror("Var already defined")
#define ERR_VAR_MISSING(VAR,LINE) fprintf(stderr,"Variable %s NOT declared,
n line %d.",VAR,LINE); yyerror("Variable Declation fault.")

// Type Definitions and JVM command related Functions
#include "jvmLangTypesFunctions.h"
// Symbol Table definitions and Functions
#include "symbolTable.h"
/* Defining the Symbol table. A simple linked list. */
ST_TABLE_TYPE symbolTable;
#include "codeFacilities.h"

%}
/* Output informative error messages (bison Option) */
%define parse.error verbose

%union{
```

```
  char *lexical;
  struct {
          ParType type;
          char * place;} se;
 RelationType relopIndex;
 struct {
     NUMBER_LIST_TYPE trueLbl;
     NUMBER_LIST_TYPE falseLbl;
   } condLabels;

}

/* Token declarations and their respective types */

%token <lexical> T_NAME
%token <lexical> T_NUMBER

%token '('
%token ')'
%token '['
%token ']'
%token '.'

%token <tokentype> T_type
%token T_start "start"
%token T_end "end"
%token T_forall "forall"
%token T_print "print"
%token T_in "in"
%token T_NAME "name"
%token T_NUMBER "number"

%left '+'
%left '*'

%type<se> MATH  // add types for non-terminal symbols

%%
program: "start" T_NAME {create_preample($2); symbolTable=NULL; }
                        BODY "end"
```

```
                    {insertINSTRUCTION("return");
    insertINSTRUCTION(".end method\n");}
      ;

BODY: /* empty */
      |'(' BODY BODY ')' BODY {/* nothing */}
      | ARRAY {/* nothing */}
      | PRINT {/* nothing */}
      | CAST  {/* nothing */}
      | MATH  {/* nothing */}
      ;


PRINT:/* empty */
      |"print" BODY
      {if (!lookup(symbolTable,UNKNOWN) )
{ERR_VAR_MISSING(UNKNOWN,line);}
       $$ = lookup_type(symbolTable,UNKNOWN);}
      ;

CONV: "int" BODY //check for int and float if T_NAME/T_NUM are already of the
right type, if not change its type
     |"float" BODY
     ;

MATH: /* empty */
      | MATH MATH '+' {$$.type = typeDefinition($1.type,$2.type);
                    insertOPERATION($$.type, "add");}
      | MATH MATH '*' {$$.type = typeDefinition($1.type,$2.type);
                    insertOPERATION($$.type, "mul");}
      | T_NUMBER {($$.type = type_integer; int x; x = atoi($1); pushInteger(x);)}

      | T_NAME {if (!($$.type = lookup_type(symbolTable,$1)))
            {ERR_VAR_MISSING($1,line);}
            insertLOAD($$.type,
            lookup_position(symbolTable,$1));};
      |ARRAY
      ;

ARRAY:  /* empty */
```

```
      |"[" "forall" T_NAME "in" T_NUMBER ".""." T_NUMBER "]" //if
T_NUMBER T_type != int then error
      {if ($$.type != type_integer; }
      | T_NAME '[' T_NUMBER ']' {if (!($$.type lookup_type(symbolTable,$1)))
            {ERR_VAR_MISSING($1,line);}
            insertLOAD($$.type,
            lookup_position(symbolTable,$1));};
      ;

%%

/* The usual yyerror */
int yyerror (const char * msg)
{
  fprintf(stderr, "PARSE ERROR: %s.on line %d.\n ", msg,line);
  no_errors++;
}

/* Other error Functions*/
/* The lexer... */
#include "jvmLExp.lex.c"

/* Main */
int main(int argc, char **argv ){

   ++argv, --argc;  /* skip over program name */
   if ( argc > 0 && (yyin = fopen( argv[0], "r")) == NULL)
    {
      fprintf(stderr,"File %s NOT FOUND in current directory.\n Using stdin.
\n",argv[0]);
      yyin = stdin;
    }
   if ( argc > 1) {yyout = fopen(argv[1], "w");}
   else {
      fprintf(stderr,"No second argument defined. Output to screen.\n\n");
      yyout = stdout;
    }

    // Calling the parser
   int result = yyparse();
```

```
  fprintf(stderr,"Errors found %d.\n",no_errors);
  if (no_errors == 0)
     {print_int_code(yyout);}
  fclose(yyout);
  /// Need to remove even empty file.
  if (no_errors != 0 && yyout != stdout) {
    remove(argv[1]);
     fprintf(stderr,"No Code Generated.\n");}
  print_symbol_table(symbolTable); /* uncomment for debugging. */

 return result;
}
```