UPMC/master/info/APS-4I503
Syntaxe

P. Manoury

janvier 2018

1 En C

1.1 Syntaxe abstraite

Fichier : `ast.h`

```
typedef struct _expr Expr;
typedef enum _tag Tag;
typedef enum _oprim Oprim;

enum _tag {
  ASTNum, ASTId, ASTPrim
};

enum _oprim { AST_ADD, AST_SUB, AST_DIV, AST_MUL };


struct _expr {
  Tag tag;
  union {
    int num;
    char* id;
    struct {
      Oprim op;
      Expr *opand1;
      Expr *opand2;
    } binOp;
  } content;
};

Expr* newASTNum(int n);
Expr* newASTId(char* x);
Expr* newASTPrim(Oprim op, Expr* e1, Expr* e2);

#define tagOf(r) r->tag
#define getNum(r) r->content.num
#define getId(r) r->content.id
#define getOp(r) r->content.binOp.op
#define getOpand1(r) r->content.binOp.opand1
#define getOpand2(r) r->content.binOp.opand2
```

Fichier : `ast.c`

```
#include <stdlib.h>
#include <stdio.h>
#include "ast.h"

Expr* newASTNum(int v) {
  Expr* r = malloc(sizeof(*r));
  r->tag = ASTNum;
  r->content.num = v;
  return r;
}

Expr* newASTId(char* v) {
  Expr* r = malloc(sizeof(*r));
  r->tag = ASTId;
  r->content.id = v;
  return r;
}

Expr* newASTPrim(Oprim op, Expr* e1, Expr* e2) {
  Expr* r = malloc(sizeof(*r));
  r->tag = ASTPrim;
  r->content.binOp.op = op;
  r->content.binOp.opand1 = e1;
  r->content.binOp.opand2 = e2;
  return r;
}
```

1.2 Analayse lexicale

Fichier : `lexer.lex`

```
%{

#include <stdlib.h>

#include "ast.h"
#include "y.tab.h"

%}

nls "\n"|"\r"|"\r\n"
nums "-"?[0-9]+
idents [a-zA-Z][a-zA-Z0-9]*
%%

[ \t]  { /* On ignore */ }

"add"   return(PLUS);
"sub"   return(MINUS);

"mul"   return(TIMES);
```

```
"div"   return(DIV);

"("   return(LPAR);
")"   return(RPAR);

{nls}  { return(0); }

{nums}    {
  yylval.num=atoi(yytext);
      return(NUM);
    }

{idents}    {
      yylval.str=strdup(yytext);
      return(IDENT);
    }
```

1.3 Analyse grammaticale

Ficheir : `parser.y`

```
%{

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "ast.h"
#include "toProlog.h"

int yylex (void);
int yyerror (char *);

 Expr* theExpr;

%}

%token<num>  NUM
%token<str>  IDENT
%token  PLUS MINUS TIMES  DIV
%token  LPAR RPAR
%token  NL

%left PLUS MINUS
%left TIMES DIV
%left NEG

%union {
  int num;
  char* str;
```

```
  Expr* expr;
}

%type<expr> exp
%type<expr> line

%start line
%%

line: exp    { theExpr = $1; }
  ;

exp:
  NUM                       { $$ = newASTNum($1); }
| IDENT                     { $$ = newASTId($1); }
| LPAR PLUS exp  exp RPAR   { $$ = newASTPrim(AST_ADD,$3,$4); }
| LPAR MINUS exp exp RPAR   { $$ = newASTPrim(AST_SUB,$3,$4); }
| LPAR TIMES exp exp RPAR   { $$ = newASTPrim(AST_MUL,$3,$4); }
| LPAR DIV exp exp RPAR     { $$ = newASTPrim(AST_DIV,$3,$4); }
;

%%

int yyerror(char *s) {
  printf("error: %s\n",s);
  return 1;
}

int main(int argc, char **argv) {
  yyparse();
  printExpr(theExpr);
  printf("\n");
  return 0;
}
```

1.4 Termes Prolog

Ficheir : `toProlog.h`

```
void printExpr(Expr*);
```

Fichier : `toProlog.c`

```
#include <stdio.h>
#include "ast.h"

void printOp(Oprim op) {
  switch(op) {
  case AST_ADD : printf("add"); break;
  case AST_SUB : printf("sub"); break;
  case AST_MUL : printf("mul"); break;
  case AST_DIV : printf("div"); break;
  }
```

```
}

void printNum(int n) {
  printf("%d",n);
}

void printId(char* x) {
  printf("\"%s\"",x);
}

void printExpr(Expr *e) {
  switch(tagOf(e)) {
  case ASTNum : printNum(getNum(e)); break;
  case ASTId : printId(getId(e)); break;
  case ASTPrim : {
    printOp(getOp(e));
    printf("(");
    printExpr(getOpand1(e));
    printf(",");
    printExpr(getOpand2(e));
    printf(")");
    break;
  }
  }
}
```

1.5 Makefile

```
LEX_C = flex
YACC_C = yacc
GCC = gcc

toProlog: parser ast.h ast.c toProlog.c
        $(GCC) -c ast.c
        $(GCC) -c toProlog.c
        $(GCC) -o toProlog ast.o toProlog.o lex.yy.o y.tab.o -lm -ll

parser: lexer.lex parser.y
        $(YACC_C) -d parser.y
        $(LEX_C) lexer.lex
        $(GCC) -c lex.yy.c
        $(GCC) -c y.tab.c

clean:
        rm lex.yy.*
        rm y.tab.*
        rm *.o
        rm toProlog
```

2 En JAVA

2.1 Synatxe abstraite et termes prolog

Fichier : `Op.java`

```
public enum Op {
    ADD("add"), SUB("sub"), MUL("mul"), DIV("div");

    private String str;

    Op(String str) { this.str = str; }

    public String toString() {
        return this.str;
    }
}
```

Fichier : `Ast.java` (interface)

```
interface Ast {

        public String toPrologString();

}
```

Fichier : `AstNum.java`

```
public class AstNum implements Ast {

        Integer val;

        AstNum(Integer n) {
                val = n;
        }

        @Override
        public String toPrologString() {
                return (""+val);
        }

}
```

Fichier : `AstId.java`

```
public class AstId implements Ast {

        String name;

        AstId(String x) {
                name = x;
        }

        @Override
        public String toPrologString() {
```

```
                return "\""+name+"\"";
        }

}
```

Fichier : `AstPrim.java`

```
public class AstPrim implements Ast {

    Op op;
    Ast a1;
    Ast a2;

    AstPrim(Op op, Ast a1, Ast a2) {
        this.op = op;
        this.a1 = a1;
        this.a2 = a2;
    }

    public String toPrologString() {
        return op.toString()+"("+a1.toPrologString()+","+a2.toPrologString()+")";
    }

}
```

2.2 Analyse lexicale

Fichier : `lexer.lex`

```
%%

%byaccj

%{
  private Parser yyparser;

  public Yylex(java.io.Reader r, Parser yyparser) {
    this(r);
    this.yyparser = yyparser;
  }
%}

nums = -?[0-9]+
ident = [a-z][a-zA-Z0-9]*
nls  = \n | \r | \r\n

%%

/* operators */
"add"  { return Parser.PLUS; }
"sub"  { return Parser.MINUS; }
"mul"  { return Parser.TIMES; }
"div"  { return Parser.DIV; }
```

```
/* parenthesis */
"("  { return Parser.LPAR; }
")"  { return Parser.RPAR; }

/* newline */
{nls}   { return 0; } //{ return Parser.NL; }

/* float */
{nums}  { yyparser.yylval = new ParserVal(Integer.parseInt(yytext()));
          return Parser.NUM; }

{ident} { yyparser.yylval = new ParserVal(yytext());
  return Parser.IDENT;
}

/* whitespace */
[ \t]+ { }

\b     { System.err.println("Sorry, backspace doesn't work"); }

/* error fallback */
[^]    { System.err.println("Error: unexpected character '"+yytext()+"'"); return -1; }
```

2.3 Analayse grammaticale

Fichier : `parser.y`

```
%{

  import java.io.*;

%}

%token NL                    /* newline  */
%token <ival> NUM            /* a number */
%token <sval> IDENT          /* an identifier */
%token PLUS MINUS TIMES DIV  /* operators */
%token LPAR RPAR             /* parethesis */

%left MINUS PLUS
%left TIMES DIV
%left NEG          /* negation--unary minus */

%type <obj> line
%type <obj> exp
%start line
%%

line:  exp   { prog=(Ast)$1; $$=$1; }
       ;
```

```
exp:
  NUM                       { $$ = new AstNum($1); }
| IDENT                     { $$ = new AstId($1); }
| LPAR PLUS exp exp RPAR    { $$ = new AstPrim(Op.ADD,(Ast)$3,(Ast)$4); }
| LPAR MINUS exp exp RPAR   { $$ = new AstPrim(Op.SUB,(Ast)$3,(Ast)$4); }
| LPAR TIMES exp exp RPAR   { $$ = new AstPrim(Op.MUL,(Ast)$3,(Ast)$4); }
| LPAR DIV exp exp RPAR     { $$ = new AstPrim(Op.DIV,(Ast)$3,(Ast)$4); }
;

%%

  public Ast prog;

  private Yylex lexer;


  private int yylex () {
    int yyl_return = -1;
    try {
      yylval = new ParserVal(0);
      yyl_return = lexer.yylex();
    }
    catch (IOException e) {
      System.err.println("IO error :"+e);
    }
    return yyl_return;
  }


  public void yyerror (String error) {
    System.err.println ("Error: " + error);
  }


  public Parser(Reader r) {
    lexer = new Yylex(r, this);
  }
```

2.4 Makefile

```
LEX_J  = jflex
YACC_J = ~/tmp/yacc.macosx -J
JAVAC = javac

toProlog: parser Op.java ToProlog.java
        $(JAVAC) ToProlog.java

parser: parser.y lexer.lex
        $(LEX_J) lexer.lex
        $(YACC_J) parser.y

clean:
```

```
rm Parser*.java
rm Yylex.java
rm *.class
```

3 En OCAML

3.1 Syntaxe abstraite

Fichier : `lexer.mll`

```
type op = Add | Mul | Sub | Div

let string_of_op op =
  match op with
      Add -> "add"
    | Mul -> "mul"
    | Sub -> "sub"
    | Div -> "div"

let op_of_string op =
  match op with
      "add" -> Add
    | "mul" -> Mul
    | "sub" -> Sub
    | "div" -> Div

type expr =
    ASTNum of int
  | ASTId of string
  | ASTPrim of op * expr * expr
```

3.2 Analyse lexicale

Fichier : `lexer.mll`

```
{
  open Parser        (* The type token is defined in parser.mli *)
  exception Eof

}
rule token = parse
    [' ' '\t']       { token lexbuf }     (* skip blanks *)
  | ['\n' ]          { EOL }
  | ['0'-'9']+('.'['0'-'9'])? as lxm { NUM(int_of_string lxm) }
  | "add"            { PLUS }
  | "sub"            { MINUS }
  | "mul"            { TIMES }
  | "div"            { DIV }
  | '('              { LPAR }
  | ')'              { RPAR }
  | eof              { raise Eof }
```

3.3 Analyse grammaticale

Fichier : `parser.mly`

```
%{
open Ast
%}

%token <int> NUM
%token <string> IDENT
%token PLUS MINUS TIMES DIV
%token LPAR RPAR
%token EOL

%start line             /* the entry point */

%type <Ast.expr> line

%%

  line:
  expr EOL                      { $1 }
  ;
  expr:
    NUM                         { ASTNum($1) }
  | IDENT                       { ASTId($1) }
  | LPAR PLUS expr expr RPAR    { ASTPrim(Ast.Add, $3, $4) }
  | LPAR MINUS expr expr RPAR   { ASTPrim(Ast.Sub, $3, $4) }
  | LPAR TIMES expr expr RPAR   { ASTPrim(Ast.Mul, $3, $4) }
  | LPAR DIV expr expr RPAR     { ASTPrim(Ast.Div, $3, $4) }
  ;
```

3.4 Termes Prolog

Fichier : `toProlog.ml`

```
open Ast

let rec print_prolog e =
  match e with
      ASTNum n -> Printf.printf"%d" n
    | ASTId x -> Printf.printf"\"%s\"" x
    | ASTPrim(op, e1, e2) -> (
        Printf.printf"%s" (string_of_op op);
        Printf.printf"(";
        print_prolog e1;
        Printf.printf",";
        print_prolog e2;
        Printf.printf")"
      )

let _ =
  try
```

```
    let lexbuf = Lexing.from_channel stdin in
    let e = Parser.line Lexer.token lexbuf in
      print_prolog e;
      print_char '\n'
  with Lexer.Eof -> exit 0
```

3.5 Makefile

```
LEX_ML = ocamllex
YACC_ML = /usr/local/bin/ocamlyacc
OCAMLC = ocamlc

toProlog: parser toProlog.ml
        $(OCAMLC) -o toProlog ast.cmo lexer.cmo parser.cmo toProlog.ml

parser: ast.ml lexer.mll parser.mly
        $(OCAMLC) -c ast.ml
        $(LEX_ML) -o lexer.ml lexer.mll
        $(YACC_ML) -b parser parser.mly
        $(OCAMLC) -c parser.mli
        $(OCAMLC) -c lexer.ml
        $(OCAMLC) -c parser.ml

clean:
        rm -f *.cmo
        rm -f *.cmi
        rm -f toProlog
        rm -f lexer.ml
        rm -f parser.mli
        rm -f parser.ml
```