Þýðendur
Þáttari, milliþulusmiður og lokaþulusmiður með JFlex og BYACC/J

Pétur Daníel Ámundason

March 22, 2018

NanoMorpho.byaccj skrá

```
%{
        import java.io.*;
        import java.util.*;
%}

%token<sval> LITERAL,NAME,OPNAME,ERROR,DEFINE,PRINTLN
%token<sval> OP1,OP2,OP3,OP4,OP5,OP6,OP7,OP8,OP9,OP10
%token IF,ELSE,ELSIF,WHILE,VAR
%token RETURN

%left RETURN, '='
%left OP1                       // +
%right OP2                      // -
%left OP3                       // /
%left OP4                       // *
%left OP5                       // <
%left OP6                       // >
%left OP7                       // ==
%left OP8                       // &&
%left OP9                       // ||
%left OP10                      // !=

%type <obj> program ,fundecl, expr, exprs, args, arglist, body, bodyexpr, ifrest
%type <ival> ids, idlist

%%

start                                                   /*@ \label{grammarstart} @*/
: program
  { generateProgram(name,((Vector<Object>)($1)).toArray()); }
;

program
: program fundecl
        { ((Vector<Object>)($1)).add($2); $$=$1; }
| fundecl
        { $$=new Vector<Object>(); ((Vector<Object>)($$)).add($1); }
;

fundecl
: {
   varCount = 0;
   varTable = new HashMap<String,Integer>();
  }
```

```
   NAME '(' ids ')' '{'
    VAR idlist ';'
                exprs
         '}'
  {
     $$ = new Object[]{$2,$4,$8+$4,((Vector<Object>)($10)).toArray()};
  }
;

ids
:       /* empty */     { $$=0; }
| ids ',' NAME
        { addVar($3); $$=$1+1; }
| NAME
        { addVar($1); $$+=1; }
;

idlist
:       /* empty */
        { $$=0; }
| idlist ',' NAME
        { addVar($3); $$=$1+1; }
| NAME
        { addVar($1); $$+=1; }
;

exprs
: exprs expr ';'
{ ((Vector<Object>)($1)).add($2); $$=$1; }
| expr ';'
{ $$=new Vector<Object>(); ((Vector<Object>)($$)).add($1); }
;

args
:       /* empty */
        { $$=new Vector<Object>(); }
|       arglist
;

arglist
:       arglist ',' expr
        { ((Vector<Object>)($1)).add($3); $$=$1; }
|       expr
        { $$=new Vector<Object>(); ((Vector<Object>)($$)).add($1); }
;

body
: /* empty */
        { $$=new Vector<Object>(); }
| '{' bodyexpr '}'
        { $$=((Vector<Object>)($2)).toArray(); }
;

bodyexpr
:       bodyexpr expr   ';'
        { ((Vector<Object>)($1)).add($2); $$=$1; }
| expr  ';'
        { $$=new Vector<Object>(); ((Vector<Object>)($$)).add($1); }
;

ifrest
: ELSE body
        { $$ = new Object[]{"IF3",$2}; }
| ELSIF '(' expr ')' body
        { $$ = new Object[]{"IF1",$3,$5}; }
| ELSIF '(' expr ')' body ifrest
        { $$ = new Object[]{"IF2",$3,$5,$6}; }
;
```

```
expr
        : RETURN expr
        { $$ = new Object[]{"RETURN",$2}; }
        | NAME '=' expr
                { $$ = new Object[]{"STORE",varPos($1),$3}; }
        | PRINTLN '(' expr ')'
                { $$ = new Object[]{"PRINT", $3}; }
        | expr OP1 expr
                { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
        | expr OP2 expr
                { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
        | expr OP3 expr
                { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
        | expr OP4 expr
                { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
        | expr OP5 expr
                { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
        | expr OP6 expr
                { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
        | expr OP7 expr
                { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
        | expr OP8 expr
                { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
        | expr OP9 expr
                { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
        | expr OP10 expr
                { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
        | '(' expr ')'
                { $$ = $2; }
        | NAME
                { $$ = new Object[]{"FETCH",varPos($1)}; }
        | NAME '(' args ')'
                { $$ = new Object[]{"CALL",$1, ((Vector<Object>)($3)).toArray() }; }
        | WHILE '(' expr ')' body
                { $$=new Object[]{"WHILE",$3,$5}; }
        | IF '(' expr ')' body
                { $$ = new Object[]{"IF1",$3,$5}; }
        | IF '(' expr ')' body ifrest
                { $$ = new Object[]{"IF2",$3,$5,$6}; }
        | LITERAL
                { $$ = new Object[]{"LITERAL",$1}; }
        ;

%%

static private String name;
private NanoMorphoLexer lexer;
private int varCount;
private HashMap<String,Integer> varTable;

private void addVar( String name )
{
  if( varTable.get(name) != null )
  yyerror("Variable "+name+" already exists");
  varTable.put(name,varCount++);
}

private int varPos( String name )
{
        Integer res = varTable.get(name);
        if( res == null )
                yyerror("Variable "+name+" does not exist");
        return res;
}

int last_token_read;
```

```
private int yylex()
{
        int yyl_return = -1;
        try
        {
                yylval = null;
                last_token_read = yyl_return = lexer.yylex();
                if( yylval==null )
                        yylval = new NanoMorphoParserVal(NanoMorphoParser.yyname[yyl_return
                            ]);
        }
        catch (IOException e)
        {
                emit("IO error: "+e);
        }
        return yyl_return;
}

public void yyerror( String error )
{
        emit("Error:  "+error);
        emit("Token:  "+NanoMorphoParser.yyname[last_token_read]);
        System.exit(1);
}

public NanoMorphoParser( Reader r )
{
        lexer = new NanoMorphoLexer(r,this);
}

public static void main( String args[] )
throws IOException
{
  NanoMorphoParser yyparser = new NanoMorphoParser(new FileReader(args[0]));
  name = args[0].substring(0,args[0].lastIndexOf('.'));
  yyparser.yyparse();
}

public static void emit( String s )             /*@ \label{byaccgeneratorstart} @*/
{
  System.out.println(s);
}

static void generateProgram( String name, Object[] p )
{
  emit("\""+name+".mexe\" = main in");
  emit("!{{");
  for( int i=0 ; i!=p.length ; i++ ) generateFunction((Object[])p[i]);
  emit("}}*BASIS;");
}

static void generateFunction( Object[] f )
        {
                String fname = (String)f[0];
                int count = (Integer)f[1];
    int varcount = (Integer)f[2];
                emit("#\""+fname+"[f"+count+"]\" =");
                emit("[");
    if( varcount!=0 ) emit("(MakeVal null)");
                for( int i=0 ; i!=varcount ; i++ ) System.out.println("(Push)");
                Object[] exprs = (Object[])f[3];
                for( Object e: exprs ) generateExpr((Object[])e);
                emit("];");
        }

static int nextLab = 0;
```

```
static void generateExpr( Object[] e )
{
        switch( (String)e[0] )
        {
        case "FETCH":
                        emit("(Fetch "+e[1]+")");
                        return;
        case "STORE":
                        generateExpr((Object[])e[2]); emit("(Store "+e[1]+")");
                        return;
        case "IF1":
                        {
                                        // ["IF1",cond,thenpart]
                                        int endlab = nextLab++;
                                        generateExpr((Object[])e[1]);
                                        emit("(GoFalse _"+endlab+")");
                                        generateBody((Object[])e[2]);
                                        emit("_"+endlab+":");
                                        return;
                        }
        case "IF2":
                        {
                                        // ["IF2",cond,thenpart,elsepart]
                                        int elslab = nextLab++;
                                        int endlab = nextLab++;
                                        generateExpr((Object[])e[1]);
                                        emit("(GoFalse _"+elslab+")");
                                        generateBody((Object[])e[2]);
                                        emit("(Go _"+endlab+")");
                                        emit("_"+elslab+":");
                                        generateExpr((Object[])e[3]);
                                        emit("_"+endlab+":");
                                        return;
                        }
                case "IF3":
                        {
                                        // ["IF3",elsepart]
                                        int elslab = nextLab++;
                                        int endlab = nextLab++;
                                        emit("_"+elslab+":");
                                        generateBody((Object[])e[1]);
                                        emit("_"+endlab+":");
                                        return;
                        }
        case "WHILE":
                        {
                                        int startlab = nextLab++;
                                        int endlab = nextLab++;
                                        emit("_"+startlab+":");
                                        generateExpr((Object[])e[1]);
                                        emit("(GoFalse _"+endlab+")");
                                        generateBody((Object[])e[2]);
                                        emit("(Go _"+startlab+")");
                                        emit("_"+endlab+":");
                                        return;
                        }
        case "CALL":
                        {
                                        Object[] args = (Object[])e[2];
                                        if( args.length!=0 ) generateExpr((Object[])args[0])
                                            ;
                                        for( int i=1 ; i<args.length ; i++ )
                                        {
                                                        emit("(Push)");
                                                        generateExpr((Object[])args[i]);
                                        }
                emit("(Call #\""+e[1]+"[f"+args.length+"]\" "+args.length+")");
                                        return;
```

```
                        }
        case "RETURN":
                        generateExpr((Object[])e[1]);
                        emit("(Return)");
                        return;
        case "LITERAL":
                        emit("(MakeVal "+e[1]+")");
                        return;
        case "PRINT":
                        generateExpr((Object[])e[1]);
                        emit("(Call #\""+"writeln"+"[f1]\" 1)");
                        return;
        default:
                        throw new Error("Invalid expression type: "+e[0]);
        }
}

static void generateBody( Object[] bod )
{
                for( Object e: bod )
                {
                                generateExpr((Object[])e);
                }
}
```

nanoMorpho.jflex skrá

```
import java.io.*;

%%

%public
%class NanoMorphoLexer
%unicode
%byaccj
%line
%column

%{


public static String lexeme;

public NanoMorphoParser yyparser;

public NanoMorphoLexer( java.io.Reader r, NanoMorphoParser yyparser )
{
        this(r);
        this.yyparser = yyparser;
}

%}

/* Reglulegar skilgreiningar */

/* Regular definitions */

_DIGIT=[0-9]
_FLOAT={_DIGIT}+\.{_DIGIT}+([eE][+-]?{_DIGIT}+)?
_INT={_DIGIT}+
_STRING=\"([^\"\\]|\\b|\\t|\\n|\\f|\\r|\\\"|\\\'|\\\\|(\\[0-3][0-7][0-7])
      |\\[0-7][0-7]|\\[0-7])*\"
_CHAR=\'([^\'\\]|\\b|\\t|\\n|\\f|\\r|\\\"|\\\'|\\\\|(\\[0-3][0-7][0-7])|(\\[0-7][0-7])
      |(\\[0-7]))\'
_DELIM=[={},()\[\];]
_NAME=([:letter:]|{_DIGIT})+
_OPNAME=[!%\^]+
```

```
%%

/* Lesgreiningarreglur */

{_DELIM} {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return yycharat(0);
}

{_OPNAME} {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.OPNAME;
}

{_STRING} | {_FLOAT} | {_CHAR} | {_INT} | null | true | false {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.LITERAL;
}

"+" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.OP1;
}

"-" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.OP2;
}

"/" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.OP3;
}

"*" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.OP4;
}

"<" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.OP5;
}

">" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.OP6;
}

"==" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.OP7;
}

"||" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.OP9;
}

"&&" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.OP8;
}

"!=" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.OP10;
```

```
}



"println" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.PRINTLN;
}

"return" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.RETURN;
}

"else" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.ELSE;
}

"elsif" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.ELSIF;
}

"while" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.WHILE;
}

"if" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.IF;
}

"define" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.DEFINE;
}

"var" {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.VAR;
}

{_NAME} {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.NAME;
}

";;;".*$ {
}

[ \t\r\n\f] {
}

. {
yyparser.yylval = new NanoMorphoParserVal(yytext());
return NanoMorphoParser.ERROR;
}
```

test.s skrá

```
;;; Fibonacci
fibo(n){
  var x;
  if(n < 0) {
    return n * ( 1 - 2);
  } elsif( n == 0) {
    return n * ( 1 - 2);
  } else {
    return fibo(n-1) + fibo(n-2);
  };
}

main(){
  var n;
  n = 0;
  while(n < 12){
    n = n+1;
    println(fibo(n));
  };
}
```