Handbók fyrir NanoMorpho

Pétur Daníel Ámundason

26. apríl 2018

Efnisyfirlit

1 Inngangur

Enn einn þýðandi NanoMorpho Einföld útgáfa af Morpho. NanoMorpho er bálkmótað mál.

2 Notkun og uppsetning

Uppsetning Notast skal við Git til þess að sækja þýðandan.

Git $ Git clone https://Github.com/rutep/NanoMorpho.Git
Þegar náð hefur verið í skrár þá eru þær þrjár skrár. **NanoMorphoLexer.java** sem er lesgreinir, **NanoMorphoParser.java** og **NanoMorphoParserVal.java** sem sjá um þáttun og milli þulusmíði.

Morpho Morphoskrárnar eru til þess að geta þýtt **NanoMorpho** skrár yfir á Morpho smalarmál og smalarmálið yfir í keyrslu hæfa Morphoskrá. Þegar fyrrnefnda er búið þá er hægt að keyra forritið í gegnum Morpho.

makefile Hér fyrir neðan sést makefile skrá sem hægt er að nota til þess að auðvelda sér þýðingu og keyrslu **NanoMorpho** forrits. Ef búnaður er ekki til staðar til þess að nota makefile skrá þá er hægt að keyra beint skipanir á skipunarlínu ef java þýðandi og Morpho er upp sett.

- $ javac NanoMorphoLexer.java NanoMorphoParser.java NanoMorphoParserVal.java
- Skipuninn að ofan þýðir java skrár
- $ java NanoMorphoParser test.s > test.masm
- Býr til Morpho smalarmáls skrá test.masm
- $ Morpho -c test.masm
- $ Morpho test
- Býr til test.mexem skrá og svo keyrir forrit

```
1  SHELL=/usr/bin/env /bin/bash
2
3  parser: NanoMorphoParser.java NanoMorphoParserVal.java NanoMorphoLexer
       .java
4  javac NanoMorphoLexer.java NanoMorphoParser.java NanoMorphoParserVal.
       java
5
6  test: NanoMorphoParser.class NanoMorphoParserVal.class
7  java NanoMorphoParser test.s > test.masm
8
9  compile: test.masm
10 Morpho -c test.masm
11
12 run: test.mexe
13 Morpho test
```

3 Málfræði

3.1 Frumeiningar málsins

Samhengislaust mál Línubil og línu endingar hafa enga merkingu fyrir þíðandan og væri hægt að skrifa heilu forritin í einni línu með engum línubilum.

3.1.1 Athugasemd

Dæmi um athugasemd ;;; Hunsað af þýðanda Þýðandinn mun hunsa allt sem kemur á eftir athugarsemdar í þeirri línu sem hún er.

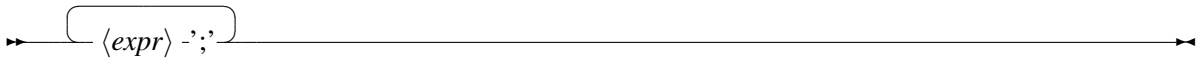
3.1.2 Lykilorð

Lykilorðin í þýðandanum eru:
else, elsif, false, if, null, return, true, var, while, println

3.2 Mállýsing

⟨*idlist*⟩:

⟨*exprs*⟩:

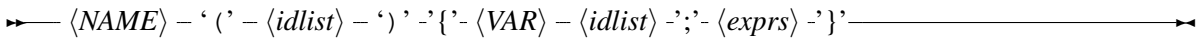


3.2.1 Forrit

⟨*program*⟩:

3.2.2 Föll

⟨*function*⟩:

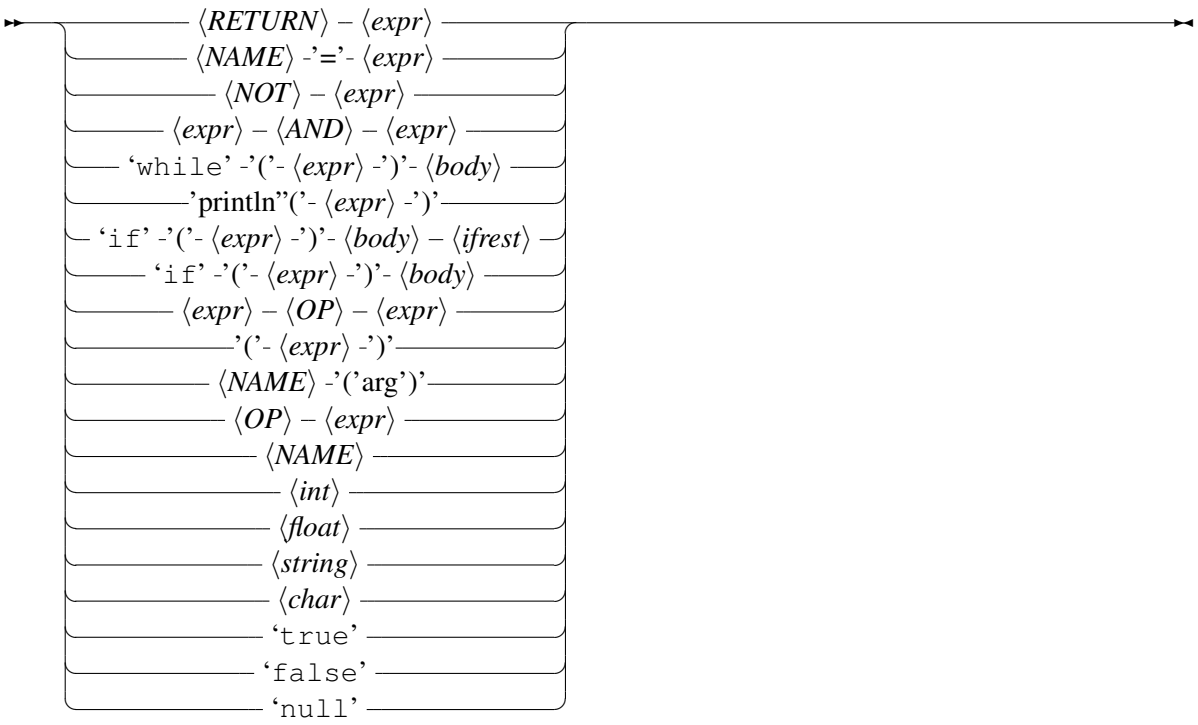


3.2.3 Stofnar

⟨*body*⟩:

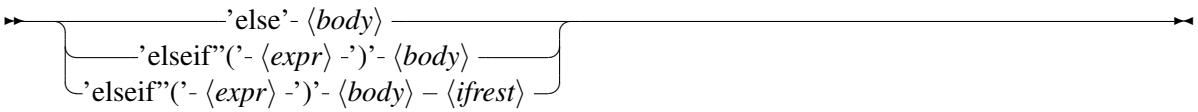
3.2.4 Segðir

⟨*expr*⟩:



⟨*arg*⟩:

⟨*ifrest*⟩:



4 Merking málsins

NanoMorpho skiptis upp í að það er eitt main fall sem keyrt er og úr því er hægt að kalla á önnur skilgreind föll.

```
1  foo(){
2    var a;
3    a = 1;
4    return a;
5  }
6
7  main(){
8    var a;
9    a = foo();
10 }
```

4.1 Breytur

Fyrsta gildisveitinginn sem á sér stað er þegar breyta er búinn til , þá mun hún innihalda null gildið. Seinna meir í stofni falls er hægt að gefa henni nýtt gildi. Notkunardæmi er hér fyir neðan. Breytunöfn verða hafa a.m.k. einn bókstaf í sér og geta innihaldið tölur.

```
1  main(){
2    var a;  ;;; a = null
3    a = 0;  ;;; a = 0
4  }
```

4.2 Merking segða

4.2.1 null-segð

Þjónar þeim tilgangi að sýna fram á að breyta hafi ekki verið gefið neitt gildi.

4.2.2 true-false segð

Boolean gildi true notað í boolean reikningi.

4.2.3 false-segð

Boolean gidli false notað í boolean reikningi.

4.2.4 Heiltölusegð

Heiltölu gildi.

4.2.5 Fleytitölusegð

Fleititölu gildi.

4.2.6 Stafsegð

Stafsegðar gildi ’s’.

4.2.7 Strengsegð

Strengsegðar gildi "Strengsegð".

4.2.8 return-segð

Return skilar gildi úr falli. Notkunardæmi:

```
1  foo(){
2    return 1; ;;; Skilar 1 ef kallað er á
3  }
```

4.2.9 Röksegðir

Röksegðir eru or ||, and &&, !=, >=, <= og == .
Þær eru að mestu leiti reiknaðar frá vinstri til hægri. Notkunardæmi:

```
1        ;;; gefur
2  1 != 2      ;;; true
3  true || false     ;;; true
4  true && true    ;;; true
5  1 >= 1      ;;; true
6  1 <= 1      ;;; true
7  1 == 1      ;;; true
```

4.2.10 Kallsegð

Fall eru annað hvort úr basis eða gert af notenda. Fall tekur inn breytur sem viðföng og reikningur úr þeim er frá vinstri til hægri. Dæmi um slíkt fall sem notendi getur gert og notkun:

```
1  foo(a,b,s){
2    var t;
3    return 1;
4  }
5
6  main(){
7    var x;
8    x = foo(1+1,2-5,-3);  ;;; foo fall skilar einum
9  }
```

4.2.11 Tvíundaraðgerðir

Tvíundaraðgerðir eru á infix formi.

4.2.12 Einundaraðgerðir

Einundaraðgerðir hafa hæsta forgang.

4.2.13 if-segð

Hér er dæmi um notkun á if elsif else segðum. Ef s1 er satt þá body1. Ef s1 ósatt og s2 satt þá body2. Ef s1 og s2 ósatt þá body3.

```
1  main(){
2    var s1,s2,s3;
3    if(s1){
4      ;;; body1
5    } elsif (s2) {
6      ;;; body2
7    } else {
8      ;;; body3
9    };
10 }
```

4.2.14 while-segð

while-segðin mun alltaf keyrast ef ástandið s1 helst satt. Dæmi um while-segð. Í gefni while-segð ef röksegðini s1 helst sönn þá mun fyrir hvert stef í while lykkju i hækka um einn.

```
1  main(){
2  var i;
3  i = 0;
4  while(s1){
5    i = i + 1;
6  };
7  }
```

4.3 Föll og forrit

Hér er dæmi um heilt NanoMorpho forrit. Þetta forrit reiknar fyrstu 12 Fibonacci tölurnar.

```
1  ;;; Fibonacci
2    fibo(n){
3        var ;
4        if(n < 0) {
5          return n * -1;
6        } elsif( n == 0) {
7          return n * ( 1 - 2);
8        } else {
9          return fibo(n-1) + fibo(n-2);
10       };
11   }
12   main(){
13       var n;
14       n = 0;
15       while(n < 12){
16           n = n+1;
17           println(fibo(n)); }; }
```

5 Byjaccj skrá

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


%token<sval> LITERAL,NAME,OPNAME,ERROR,PRINTLN
%token<sval> OP1,OP2,OP3,OP4,OP5
%token IF,ELSE,ELSIF,WHILE,VAR,UNOP
%token RETURN
%type <sval> op

%right RETURN, '='
%right OR
%right AND
%right NOT
%left OP1
%left OP2
%left OP3
%left OP4
%left OP5
%right UNOP

%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 ';'
```

```
45        exprs
46      '}'
47      {
48        $$ = new Object[]{$2,$4,$8+$4,((Vector<Object>)($10)).toArray()
             };
49      }
50    ;
51
52  ids
53    : /* empty */    { $$=0; }
54    | ids ',' NAME   { addVar($3); $$=$1+1; }
55    | NAME           { addVar($1); $$+=1; }
56    ;
57
58
59  idlist
60    : /* empty */        { $$=0; }
61    | idlist ',' NAME    { addVar($3); $$=$1+1; }
62    | NAME               { addVar($1); $$+=1; }
63    ;
64
65  exprs
66    : exprs expr ';'     { ((Vector<Object>)($1)).add($2); $$=$1; }
67    | expr ';'           { $$=new Vector<Object>(); ((Vector<Object>)($$)
        ).add($1); }
68    ;
69
70  args
71    : /* empty */        { $$=new Vector<Object>(); }
72    | arglist
73    ;
74
75  arglist
76    : arglist ',' expr   { ((Vector<Object>)($1)).add($3); $$=$1; }
77    | expr               { $$=new Vector<Object>(); ((Vector<Object>)($$)
        ).add($1); }
78    ;
79
80  body
81    : '{' bodyexpr '}'   { $$=((Vector<Object>)($2)).toArray(); }
82    ;
83
84  bodyexpr
85    : bodyexpr expr ';'    { ((Vector<Object>)($1)).add($2); $$=$1; }
86    | expr  ';'            { $$=new Vector<Object>(); ((Vector<Object>)(
        $$)).add($1); }
87    ;
88
89  ifrest
90    : ELSE body                              { $$ = new Object[]{"IF3",$2};
```

```
      }
91   | ELSIF '(' expr ')' body              { $$ = new Object[]{"IF1",$3,
       $5}; }
92   | ELSIF '(' expr ')' body ifrest       { $$ = new Object[]{"IF2",$3,
       $5,$6}; }
93   ;
94
95 op: OP1 | OP2 | OP3 | OP4 | OP5 ;
96
97 expr
98   : RETURN expr
99     { $$ = new Object[]{"RETURN",$2}; }
100  | NAME '=' expr
101    { $$ = new Object[]{"STORE",varPos($1),$3}; }
102  | NOT expr
103      { $$ = new Object[]{"NOT",$2}; }
104  | expr AND expr
105      { $$ = new Object[]{"CALL",$1,$3}; }
106  | expr OR expr
107      { $$ = new Object[]{"CALL",$1,$3}; }
108  | PRINTLN '(' expr ')'
109    { $$ = new Object[]{"PRINT", $3}; }
110  | expr OP1 expr
111    { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
112  | expr OP2 expr
113    { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
114  | expr OP3 expr
115    { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
116  | expr OP4 expr
117    { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
118  | expr OP5 expr
119    { $$ = new Object[]{"CALL",$2,new Object[]{$1,$3}}; }
120  | '(' expr ')'
121    { $$ = $2; }
122  | NAME
123    { $$ = new Object[]{"FETCH",varPos($1)}; }
124  | NAME '(' args ')'
125    { $$ = new Object[]{"CALL",$1, ((Vector<Object>)($3)).toArray() };
         }
126  | WHILE '(' expr ')' body
127    { $$=new Object[]{"WHILE",$3,$5}; }
128  | IF '(' expr ')' body
129    { $$ = new Object[]{"IF1",$3,$5}; }
130  | IF '(' expr ')' body ifrest
131    { $$ = new Object[]{"IF2",$3,$5,$6}; }
132  | LITERAL
133    { $$ = new Object[]{"LITERAL",$1}; }
134  | op expr %prec UNOP
135      { $$ = new Object[]{"CALL",$1,new Object[]{$2}}; }
136  ;
```

```
137
138 %%
139
140 static private String name;
141 private NanoMorphoLexer lexer;
142 private int varCount;
143 private HashMap<String,Integer> varTable;
144
145 private void addVar( String name )
146 {
147   if( varTable.get(name) != null )
148   yyerror("Variable "+name+" already exists");
149   varTable.put(name,varCount++);
150 }
151
152 private int varPos( String name )
153 {
154   Integer res = varTable.get(name);
155   if( res == null )
156     yyerror("Variable "+name+" does not exist");
157   return res;
158 }
159
160 int last_token_read;
161
162 private int yylex()
163 {
164   int yyl_return = -1;
165   try
166   {
167     yylval = null;
168     last_token_read = yyl_return = lexer.yylex();
169     if( yylval==null )
170       yylval = new NanoMorphoParserVal(NanoMorphoParser.yyname[
            yyl_return]);
171   }
172   catch (IOException e)
173   {
174     emit("IO error: "+e);
175   }
176   return yyl_return;
177 }
178
179 public void yyerror( String error )
180 {
181   emit("Error:  "+error);
182   emit("Token:  "+NanoMorphoParser.yyname[last_token_read]);
183   System.exit(1);
184 }
185
```

```
186 public NanoMorphoParser( Reader r )
187 {
188   lexer = new NanoMorphoLexer(r,this);
189 }
190
191 public static void main( String args[] )
192 throws IOException
193 {
194   NanoMorphoParser yyparser = new NanoMorphoParser(new FileReader(args
        [0]));
195   name = args[0].substring(0,args[0].lastIndexOf('.'));
196   yyparser.yyparse();
197 }
198
199 public static void emit( String s )   /*@ \label{byaccgeneratorstart}
     @*/
200 {
201   System.out.println(s);
202 }
203
204 static void generateProgram( String name, Object[] p )
205 {
206   emit("\""+name+".mexe\" = main in");
207   emit("!{{");
208   for( int i=0 ; i!=p.length ; i++ ) generateFunction((Object[])p[i]);
209   emit("}}*BASIS;");
210 }
211
212 static void generateFunction( Object[] f )
213   {
214     String fname = (String)f[0];
215     int count = (Integer)f[1];
216     int varcount = (Integer)f[2];
217     emit("#\""+fname+"[f"+count+"]\" =");
218     emit("[");
219     if( varcount!=0 ) emit("(MakeVal null)");
220     for( int i=0 ; i!=varcount ; i++ ) System.out.println("(Push)");
221     Object[] exprs = (Object[])f[3];
222     for( Object e: exprs ) generateExpr((Object[])e);
223     emit("];");
224   }
225
226 static int nextLab = 0;
227
228 static void generateExpr( Object[] e )
229 {
230   switch( (String)e[0] )
231   {
232   case "FETCH":
233       emit("(Fetch "+e[1]+")");
```

```
234         return;
235     case "STORE":
236         generateExpr((Object[])e[2]); emit("(Store "+e[1]+")");
237         return;
238     case "IF1":
239         {
240             // ["IF1",cond,thenpart]
241             int endlab = nextLab++;
242             generateExpr((Object[])e[1]);
243             emit("(GoFalse _"+endlab+")");
244             generateBody((Object[])e[2]);
245             emit("_"+endlab+":");
246             return;
247         }
248     case "IF2":
249         {
250             // ["IF2",cond,thenpart,elsepart]
251             int elslab = nextLab++;
252             int endlab = nextLab++;
253             generateExpr((Object[])e[1]);
254             emit("(GoFalse _"+elslab+")");
255             generateBody((Object[])e[2]);
256             emit("(Go _"+endlab+")");
257             emit("_"+elslab+":");
258             generateExpr((Object[])e[3]);
259             emit("_"+endlab+":");
260             return;
261         }
262       case "IF3":
263         {
264             // ["IF3",elsepart]
265             generateBody((Object[])e[1]);
266             return;
267         }
268     case "WHILE":
269         {
270             int startlab = nextLab++;
271             int endlab = nextLab++;
272             emit("_"+startlab+":");
273             generateExpr((Object[])e[1]);
274             emit("(GoFalse _"+endlab+")");
275             generateBody((Object[])e[2]);
276             emit("(Go _"+startlab+")");
277             emit("_"+endlab+":");
278             return;
279         }
280     case "CALL":
281         {
282             Object[] args = (Object[])e[2];
283             if( args.length!=0 ) generateExpr((Object[])args[0]);
```

```
284             for( int i=1 ; i<args.length ; i++ )
285             {
286                 emit("(Push)");
287                 generateExpr((Object[])args[i]);
288             }
289             emit("(Call #\""+e[1]+"[f"+args.length+"]\" "+args.length+")
                    ");
290             return;
291         }
292     case "RETURN":
293         generateExpr((Object[])e[1]);
294         emit("(Return)");
295         return;
296     case "LITERAL":
297         emit("(MakeVal "+e[1]+")");
298         return;
299     case "NOT":
300         // ["NOT",expr]
301         generateExpr((Object[])e[1]); emit("(Not)");
302         return;
303     case "PRINT":
304         generateExpr((Object[])e[1]);
305         emit("(Call #\""+"writeln"+"[f1]\" 1)");
306         return;
307     default:
308         throw new Error("Invalid expression type: "+e[0]);
309     }
310 }
311 
312 static void generateBody( Object[] bod )
313 {
314     for( Object e: bod )
315     {
316         generateExpr((Object[])e);
317     }
318 }
```

6 Jflex skrá

```
1  import java.io.*;
2  
3  %%
4  
5  %public
6  %class NanoMorphoLexer
7  %unicode
8  %byaccj
9  %line
10 %column
11 
```

```
%{


public static String lexeme;

public NanoMorphoParser yyparser;

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

static int priority( String opname )
{
  switch( opname.charAt(0) )
  {
  case '|':
    return 1;
  case '&':
    return 2;
  case '!':
  case '=':
  case '<':
  case '>':
    return 3;
  case '+':
  case '-':
    return 4;
  case '*':
  case '/':
  case '%':
    return 5;
  default:
    throw new Error("Invalid opname");
  }
}

%}

/* 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);
}


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

"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;
}

```

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

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

"&&" {
  return NanoMorphoParser.AND;
}

"||" {
  return NanoMorphoParser.OR;
}

"!" {
  return NanoMorphoParser.NOT;
}

{_OPNAME} {
  yyparser.yylval = new NanoMorphoParserVal(yytext());
  switch( yytext().charAt(0) )
  {
  case '|':
    return NanoMorphoParser.OP1;
  case '&':
    return NanoMorphoParser.OP2;
  case '!':
  case '=':
  case '<':
  case '>':
    return NanoMorphoParser.OP3;
  case '+':
  case '-':
    return NanoMorphoParser.OP4;
  case '*':
  case '/':
  case '%':
    return NanoMorphoParser.OP5;
  default:
    throw new Error("Invalid operation name");
  }
}


";;;".*$ {
}
```

```
160
161 [ \t\r\n\f] {
162 }
163
164 . {
165 yyparser.yylval = new NanoMorphoParserVal(yytext());
166 return NanoMorphoParser.ERROR;
167 }
```

7 Prófun

Hér er forritið sem prófað var:

```
1  ;;; Fibonacci
2  fibo(n){
3    var ;
4    if(n < 0) {
5      return n * -1;
6    } elsif( n == 0) {
7      return n * ( 1 - 2);
8    } else {
9      return fibo(n-1) + fibo(n-2);
10   };
11 }
12
13 main(){
14   var n;
15   n = 0;
16   while(n < 12){
17     n = n+1;
18     println(fibo(n));
19   };
20
21   while(n != 0) {
22     n = n - 1;
23   }
24
25   println(n + 2 * -1 / -2);
26   println( true || false && false);
27
28 }
```

Niðurstaðan úr prófun

```
C:\Users\Pétur PC\Desktop\Skoli\Compilers\NeðanSækinÞýðandi>morpho -c test.masm
Reused 1565 out of 2180 operations, 615 operation objects used.
Reuse ratio is 72%

C:\Users\Pétur PC\Desktop\Skoli\Compilers\NeðanSækinÞýðandi>morpho test
1
1
2
3
5
8
13
21
34
55
89
144
1
true
```