Manual for the Myrkvi language

Tumi Snær Gíslason
Instructor & mentor: Snorri Agnarsson
Compilers 2014
University of Iceland

April 22, 2014

Myrkvi is a simple programming language based on Morpho.

Contents

1 Introduction **2**

2 Usage and installation **2**

3 Syntax **3**
3.1 Primitives 3
3.1.1 Comments 3
3.1.2 Keywords 3
3.2 Grammar 3

4 Definition **6**
4.1 Values 6
4.2 Variables 6
4.3 Definition of expressions 6
4.3.1 Integers 6
4.3.2 Floating point numbers 6
4.3.3 Character 6
4.3.4 String 6
4.3.5 List 7
4.3.6 return-expression 7
4.3.7 Boolean expressions 7

4.3.8 Call expressions 7
4.3.9 Binary operations 7
4.3.10 Unary operations 7
4.3.11 if-expression 8
4.3.12 while-expression 8

5 Examples 8
5.1 Hello,world. 8
5.2 Fibonacci 8
5.3 Fizz-Buzz 9

1 Introduction

Myrkvi is a programming language made in the course *Compilers* in the spring of 2014. It's a much simpler version of the programming language Morpho[1] made by Snorri Agnarsson.

It's grammar is simple and the functionality is limited so it can only handle the most basic tasks.

2 Usage and installation

Myrkvi is written in Java using the *JFlex* and *Byaccj* tools. It communicates with Morpho by emitting Morpho assembly language which is then translated by Morpho.

The requirements for compiling and running a Myrkvi program are Java[2], Byaccj[3], JFlex.jar[4] and morpho.jar[5]

In Unix, having the requirements, you can set up the Myrkvi environment by using the *makefile*

```
> make
> make test
```

[1]http://morpho.cs.hi.is/
[2]https://www.java.com/en/download/
[3]http://byaccj.sourceforge.net/#download
[4]https://github.com/tumsgis/Myrkvi/blob/master/JFlex.jar
[5]https://github.com/tumsgis/Myrkvi/blob/master/morpho.jar

3 Syntax

3.1 Primitives

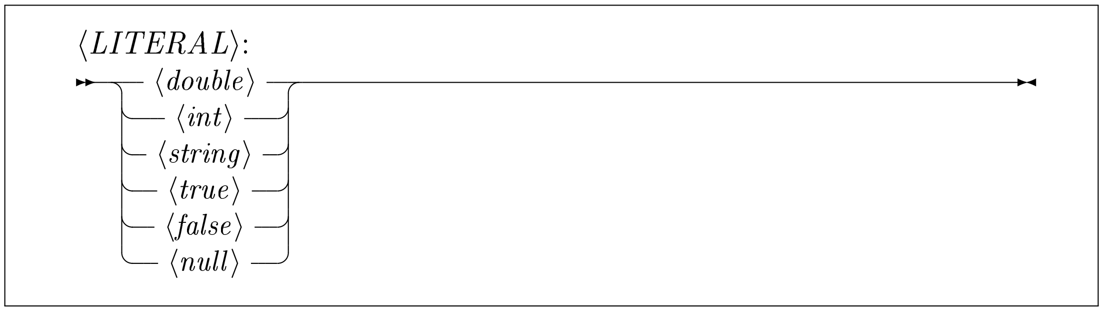
3.1.1 Comments

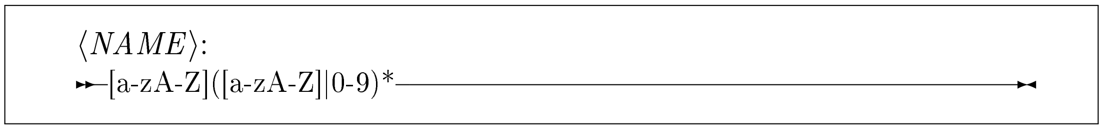
makes sure that the rest of the line is ignored.

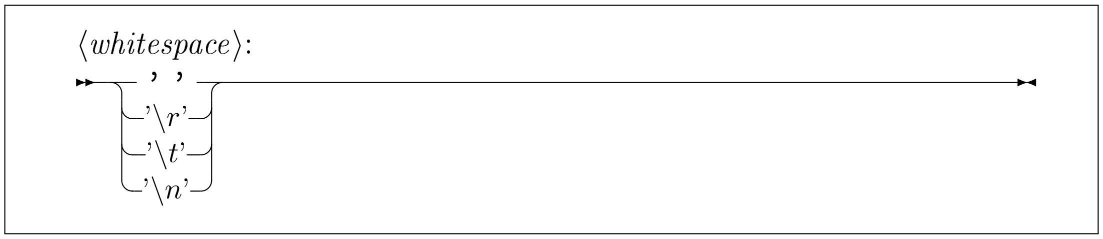
3.1.2 Keywords

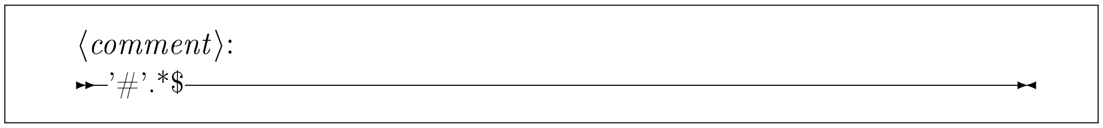
if, *elif*, *else*, *while*, *def*, *return*, *var*, *print*, *println*, *not*, *and*, *or*.

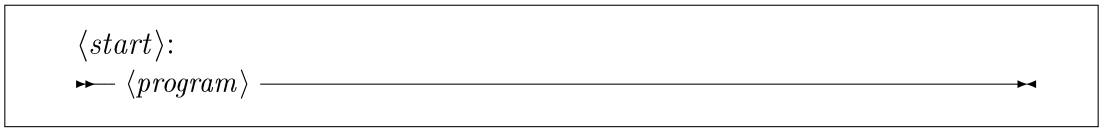
3.2 Grammar

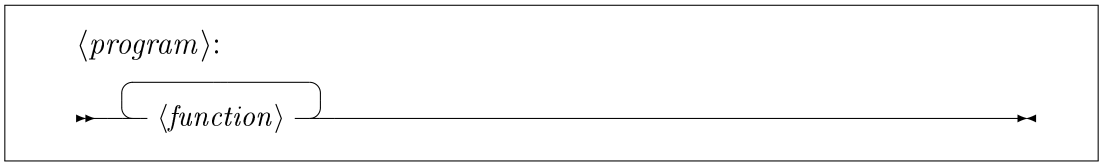


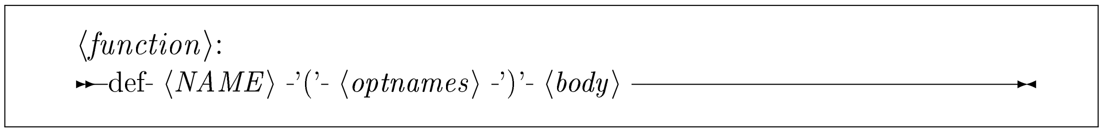


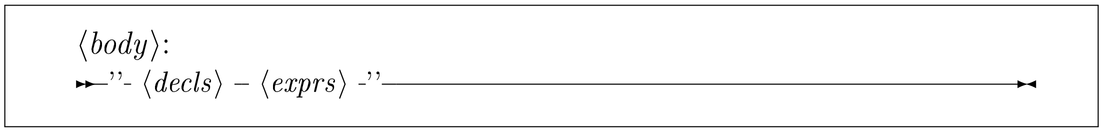


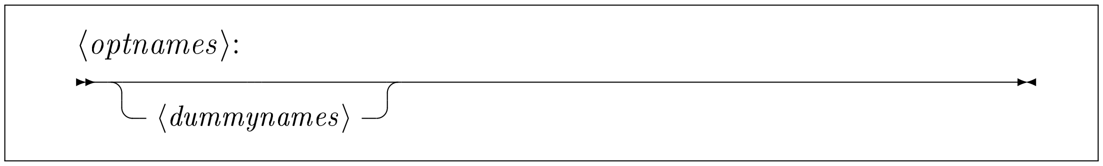


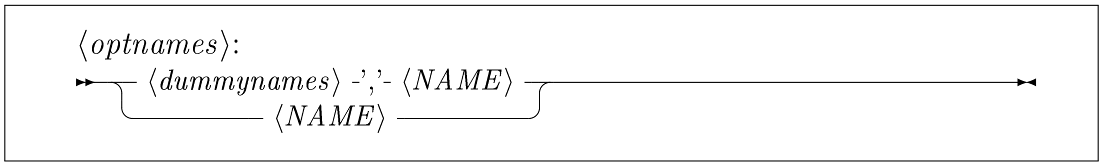


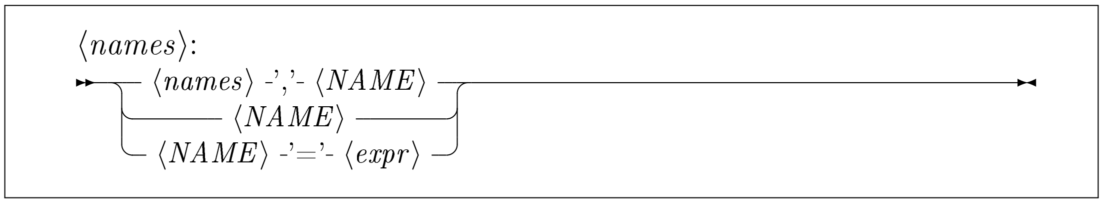

⟨program⟩:
⟨function⟩

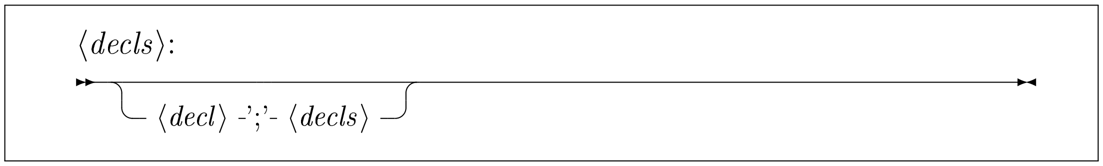

⟨function⟩:
def- ⟨NAME⟩ -'('- ⟨optnames⟩ -')'- ⟨body⟩

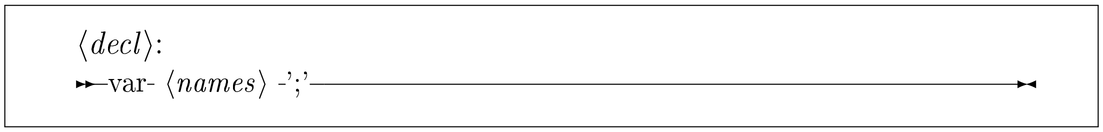

⟨body⟩:
''- ⟨decls⟩ – ⟨exprs⟩ -''


⟨optnames⟩:
⟨dummynames⟩

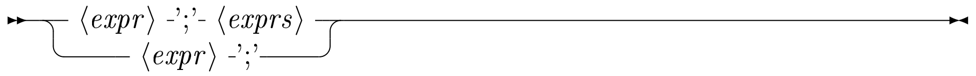

⟨optnames⟩:
⟨dummynames⟩ -','- ⟨NAME⟩
⟨NAME⟩


⟨names⟩:
⟨names⟩ -','- ⟨NAME⟩
⟨NAME⟩
⟨NAME⟩ -'='- ⟨expr⟩

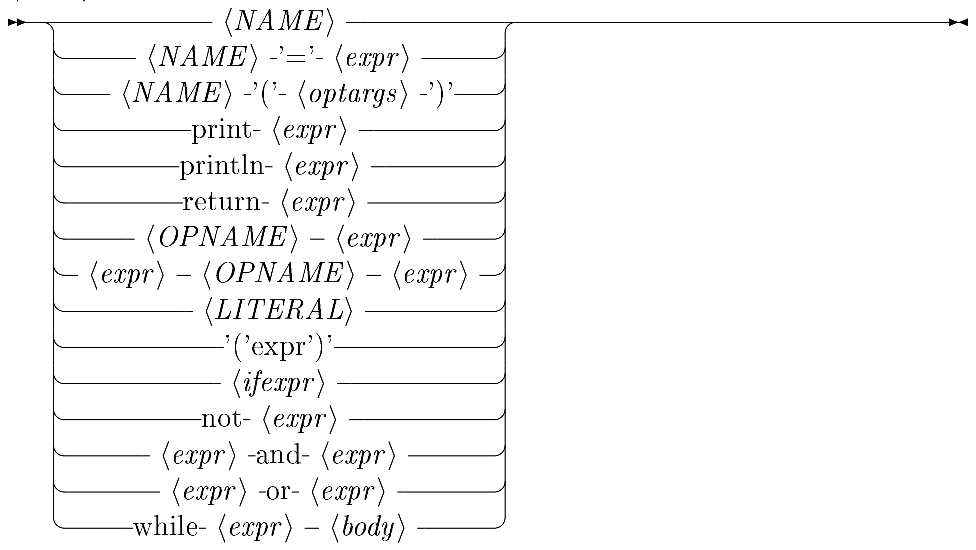

⟨decls⟩:
⟨decl⟩ -';'- ⟨decls⟩


⟨decl⟩:
var- ⟨names⟩ -';'

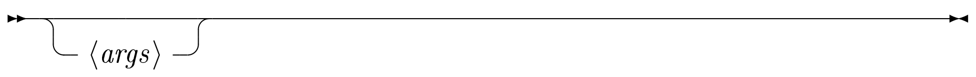
$\langle exprs \rangle$:



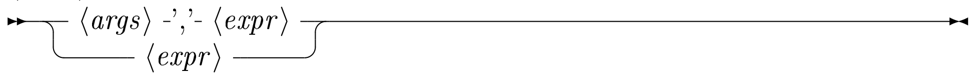
$\langle expr \rangle$:



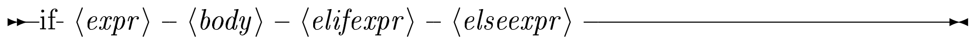
$\langle optargs \rangle$:



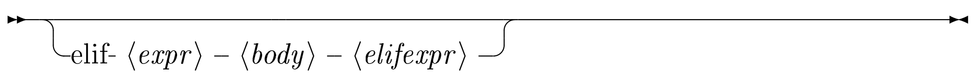
$\langle args \rangle$:

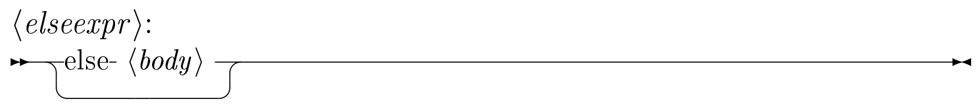


$\langle ifexpr \rangle$:



$\langle elifexpr \rangle$:





4 Definition

4.1 Values

Floating point numbers,integers,strings,*true*,*false* and *null* are all represented the same way as in Morpho, so that they're convenient to use with the Morpho assembly language. Every expression is *true* except for *false*, *null* and 0.

4.2 Variables

The *var* keyword is used to assign variables. The variables are weakly typed so for example this is a legal expression:

```
var x = 1;
x = true;
```

4.3 Definition of expressions

4.3.1 Integers

```
{0−9}+
```

4.3.2 Floating point numbers

```
{0−9}+\.{0−9}+([eE][+−]?{0−9}+)?
```

4.3.3 Character

Undefined.

4.3.4 String

```
\"([^\"\\]|\\b|\\t|\\n|\\f|\\r|\\\"|\\\'|\\\\|
(\\[0−3][0−7][0−7])|\\[0−7][0−7]|\\[0−7])∗\"
```

4.3.5 List

Undefined.

4.3.6 return-expression

return $\langle expr \rangle$
stops the activation of the current function and returns $\langle expr \rangle$. If no return-expression is in a function then the last expression of the function is returned.

4.3.7 Boolean expressions

not
A unary prefix operation,left associative, having the highest precedence of the booleans.

and
A binary operation having right associativity and the same precedence as the *or* operation.

or
A binary operation having right associativity.

Comparisons: $<,>,==,<=,>=$

All boolean expressions return either *true* or *false*.

4.3.8 Call expressions

A function can be called simply by calling its name with the right amount of parameters.

For the Morpho compiler to translate the assembly language correctly there must be a function called *main* present.

4.3.9 Binary operations

+,-,,/ and %* are all left associative and have the same precedence.

4.3.10 Unary operations

not, $+$ and -.

4.3.11 if-expression

The if-expression(*if(b)...* where b is a boolean expression) is a control sequence, better described in the grammar rules above.

4.3.12 while-expression

The while-expression(*while(b)...* where b is a boolean expression) is a control sequence, better described in the grammar rules above.

5 Examples

5.1 Hello,world.

```
def main()
{
        println "Hello,world.";
}
```

5.2 Fibonacci

```
def fibo(n)
{
        if(n < 2)
        {
                return 1;
        }
        else
        {
                return fibo(n-1) + fibo(n-2);
        };
}
```

5.3 Fizz-Buzz

```
def main()
{
        var end = 100;
        var iter = 1;

        while iter <= end
        {
                if (iter % 3 == 0) and (iter % 5 == 0)
                {
                        print "FizzBuzz ";
                }
                elif iter % 3 == 0
                {
                        print "Fizz ";
                }
                elif iter % 5 == 0
                {
                        print "Buzz ";
                }
                else
                {
                        print iter ++ " ";
                };
                iter = iter + 1;
        };
        println "";
}
```