Math Programming Language Documentation

Piotr Paszko

2018

Abstract

One day I've tried to teach person close to me basics of programming. After few hours I've realised, that the best method of explanation is by presenting code in mathematical syntax. Then I thought there's no easy to learn programming language, that uses syntax similar to mathematical and easy to read.

Contents

Chapter 1

Introduction

1.1 Language Assumption

MPLG have to be easy to learn for people without IT education, but with basic mathematical knowledge. The language have to provide clear and easy environment for both basic mathematical algorithms and complex engineering calculations. It have to be distributed with IDE adapted for it (providing support for all functions). It have to be executed by interpreter.

1.2 Language Goals

MPLG have to be able to meet the following goals:

1. The language have to be easy to learn for people without IT education, but with mathematical education.
2. The language have to be easy to read, with clear syntax.
3. The language does not have to be able to handle both mathematical algorithms and complex engineering calculations.
4. The language have to be distributed with IDE adapted for it.
5. The language have to provide simple library system.
6. The language have to be designed to be executed by interpreter.

1.3 What the language was not designed for

MPLG was not designed to:

1. Creating software (the language does not have to provide API interacting with system, computer components etc.).
2. Creating big and complex systems (language syntax was designed only for calculations, it's not providing testing systems, OOP etc.).

Chapter 2

Language Syntax

2.1 Introduction

Math Programming Language has a syntax similar to Python. Code blocks are defined by indentations and ":" character. Conditional instructions does not uses "()" characters etc. Language is designed to be most human friendly and clean as possible.

2.2 Code formatting

2.2.1 Operations endings

In MPLG every operation has to be ended with a new line character. Writing multiple operations in a single line will affect inability to interpret operation properly.

Example of properly writed code:

```
x = 3/4
d = 5
output c = x + d
```

2.2.2 Code blocks

In MPLG code blocks are started by a keyword with ":" character, and keeps until code indentations are bigger by 1 tabulation from a starting keyword indentation.

Example of properly writed code:

```
procedure sum a b:
        result = a + b
        return result

output c = sum 2 5
```

2.3 Variables

Variables in MPLG have a little bit different meaning than in standard programming languages. Every variable in MPLG is a set of real numbers (similar to arrays) encoded as fractions (each number consists of a dividend and divisible). Every number should be infinite in theory (that's impossible of course. But implementation need to allocate memory dynamically, according do number size). Every set can be also presented as multiple intervals. E.g variable can have value: $x = \{\frac{1}{2}, 20\} \cup (101, 220)$.

2.3.1 Declaring Variable

Variables are declared by writing name (name can not start by number and can not be same as any keyword) and assigning value. E.g:

```
x = 12
```

Variable can also be preceded by a "input" keyword (but only at same line where variables has been declared), what means that a variable is getting value from user input before code execution.

```
input x = 3/4
```

2.4 Calculations

2.4.1 Numbers presentation

Numbers in MPLG are rational numbers. That mean they're computed as a classical, mathematical fractions. It is possible to do calculations like $\frac{1}{2} + \frac{3}{4}$ and get result equal to $\frac{5}{4}$. Every number has dividend and divisible with implementation hidden from user inside interpreter. Every operations like bringing to the common denominator etc. are doing automatically, without user.

That means, this code:

```
x = 3/4 + 1/2
```

Will assign $\frac{5}{4}$ to "x" variable.