



LANGUAGE that YIELDS MEDICAL PROCESS HANDLING

# SYNTAX

May 22, 2017

Rickard Verner Hultgren *rihu0003@student.umu.se*

|     |                      |   |
|-----|----------------------|---|
| 1   | Introduction         | 2 |
| 2   | Variable name        | 2 |
| 3   | Specifications       | 3 |
| 3.1 | Property             | 3 |
| 3.2 | Sub-variable name    | 3 |
| 3.3 | Sub-variable content | 3 |
| 4   | Content              | 3 |
| 4.1 | Event content        | 4 |
| 4.2 | Factor content       | 4 |
| 5   | Datatypes            | 4 |

© This work by Rickard Verner Hultgren is licensed under a Creative Commons Attribution 3.0 Unported License. For the code and pseudocode in the document applies this free-BSD license:

---

Copyright (c) 2017, Rickard Verner Hultgren  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the software and documentation are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the FreeBSD Project.

---

# 1 Introduction

The possibilities of treating diseases increase in a rapid speed. Consequently both diagnostics and treatment become more complex. The purpose of LYMPHA is to give clarity to those complex reasonings. LYMPHA is a logical language for formulating medical algorithms. The language can be applied as *clinical decision support system*.

The clinical work flow is dealing with series of events effecting the patients condition and evaluations of the condition. In LYMPHA this is put as *statements* divided by `->` building up a *serie*. A serie ends with semicolon as follows:

*statement -> statement -> statement ;*

CODE (REGEX)

```
[^;]*(?=:;)
```

CODE (PYTHON)

```
[x.strip() for x in serie.split("->")]
```

A statement has either value of true (1) if it is executed, or false (0). A statement is further divided into these parts:

*variable name ( specification ) = content*

CODE (REGEX)

```
(\w*[a-zA-Z]\w*\.\.)(^[^\)]*(?=\s*))?\s*?=\s*?(^[^;]*(?=:;))
```

Events effecting the condition includes ethymology, diagnostics and treatment. E.g. a bone fracture that is examined and then treated. Some medical procedures are both diagnostical and treatment. Nevertheless these are events, that are semanticly separated from the evaluation of data from an event. The evaluation does oftne consist of sub-evaluation that will become factors in the main-evaluation: Therefore there are these two *datatypes* in LYMPHA: *events* and *factors*.

## 2 Variable name

Variable names are one word with at least one letter. The last character of the name indicates what datatype the variable has.

- *Events*; Always end with a full stop (.).
- *Factors*; Always end with a questionmark (?).

Even though both datatypes have a structure of an element as explained above, the parts looks a little bit different.

### 3 Specifications

A specification is a sub-variable. It is built as follow:

*datatype* [ *sub-variable name* ] = *content*

CODE (REGEX)

```
(\\w*[a-zA-Z]\\w*\\)\\[^[^\\]]*?(?=\\)?\\s*?=\\s*?(\\w*[a-zA-Z]\\w*\\)
```

Here follows how structurize the parts:

#### 3.1 Property

These properties are used:

| <i>Datatype</i> | <i>Meaning</i>     |
|-----------------|--------------------|
| L               | length             |
| L2              | surface area       |
| L3              | volume             |
| M               | mass               |
| N               | mole               |
| -T              | time elapsed       |
| %s              | string             |
| R               | other real numbers |

#### 3.2 Sub-variable name

These is the name or unit of the data.

#### 3.3 Sub-variable content

This is the value. All strings has quotationmarks at the beginning and at the end.

### 4 Content

These is the part of a statement that separates factors from events. The goal of a factor content is to evaluate data. The goal of event content is to describe sub-events.

## 4.1 Event content

This is one serie of events and factors separted with commas ,:

*{ event , factor , event }*

CODE (PYTHON)

```
[x.strip() for x in content.split(',')]
```

## 4.2 Factor content

This is an evaluation of other sub-factors. Sub events are not allowed to be included in factor content.

*tipping point*      *relational operator* | *{ sub-factor , sub-factor }* |

CODE (REGEX)

```
(\d)\s*?(==|>|<|>=|<=)\s*?{(\w*[a-zA-Z]\w*)}
```

Valid rational operators

| <i>relational operator</i> | <i>read as</i>                   |
|----------------------------|----------------------------------|
| <i>==</i>                  | if and only if ( $\equiv$ )      |
| <i>&gt;</i>                | greater than                     |
| <i>&gt;=</i>               | greater than or equal ( $\geq$ ) |
| <i>&lt;</i>                | less than                        |
| <i>&lt;=</i>               | less than or equal ( $\leq$ )    |
| <i>!=</i>                  | not equal to ( $\neq$ )          |

## 5 Datatypes

The goal of the syntax is to build asignments for the two datatypes. Here are the type members for each datatype:

- Events; Always end with a full stop (.).
  - *string name*
  - *list specifications*
  - *list content*
- Factors; Always end with a questionmark (?).
  - *string name*
  - *list specifications*
  - *list content*
  - *int tipping point*
  - *int relational operator*