



LANGUAGE that YIELDS MEDICAL PROCESS HANDLING

ALGORITHM

May 23, 2017

Rickard Verner Hultgren rihu0003@student.umu.se

1	Initiate	2
2	Recording	2
3	Playing	3

© 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.

The process of interpretation LYPHA scripts consists of the following steps:

1. *Initiate*; All the objects are stored into a workig memory.
2. *Recording*; Making an index list.
3. *Playing*; The script is played a given amount of steps from a given start node.

Each of these is covered below.

1 Initiate

The goal of the syntax is to build assignments for the two datatypes. These assignments will then be put into an index list. Below are the node types for the index list:

- Head node.
 - *list next*; List of starting nodes
- Events; Always end with a full stop (.).
- *string name*
- *list specifications*
- *list content*
- *list next*
- Factors; Always end with a questionmark (?).
- *string name*
- *list specifications*
- *list content*
- *list next*
- *int tipping point*
- *int relational operator*

2 Recording

An index list or data base is made in order to store data objects. The algorithm for building an index list:

```
# 1.1 Start a list
p = ( head-of-object-list→next-object-list )
```

```
# 1.2 Make objects
if REGEX(object found) then
  object = malloc(sizeof(struct node))
  ( p→next-object-list ) = object
  p = object
end if
```

3 Playing

During this step the recorded data will be played in one of three modes:

- *Execution mode*; This mode will search for nodes/objects to execute. During each step the program compares the patient data with the script . In this way the program checks if the step is valid for execution.
- *Show mode*; This mode will show all available outcomes
- *Map mode*; A combination of the previous two.

Algorithms for each mode are presented below.

EXECUTION FUNCTION

```
LIST-OF-OBJECTS-TO-EXECUTE
for next-object in object.next-objects do
  for list-object in object-list do
    if list-object == next-object then
      pointer to list-object is added to LIST-OF-OBJECTS-TO-EXECUTE
    end if
  end for
end for
for exeobject in LIST-OF-OBJECTS-TO-EXECUTE do
  if exeobject.flow==1 then
    execute exeobject
    for subexeobject in exeobject.subobjects do
      execute subexeobject
    end for
    pointer to list-object is added to PAST-LIST-OF-OBJECTS-TO-EXECUTE
    delete exeobject-pointer in LIST-OF-OBJECTS-TO-EXECUTE
  end if
end for
for pastexeobject in PAST-LIST-OF-OBJECTS-TO-EXECUTE do
  EXECUTE FUNCITON (pastexeobject.next)
end for
```

SHOW FUNCTION

```

LIST-OF-OBJECTS-TO-EXECUTE
for next-object in object.next-objects do
  for list-object in object-list do
    if list-object == next-object then
      pointer to list-object is added to LIST-OF-OBJECTS-TO-EXECUTE
    end if
  end for
end for
for exeobject in LIST-OF-OBJECTS-TO-EXECUTE do
  execute exeobject
  for subexeobject in exeobject.subobjects do
    show subexeobject
  end for
  pointer to list-object is added to PAST-LIST-OF-OBJECTS-TO-EXECUTE
  delete exeobject-pointer in LIST-OF-OBJECTS-TO-EXECUTE
end for
for pastexeobject in PAST-LIST-OF-OBJECTS-TO-EXECUTE do
  execute exeobject
end for

```

MAP FUNCTION

```

LIST-OF-OBJECTS-TO-EXECUTE
for next-object in object.next-objects do
  for list-object in object-list do
    if list-object == next-object then
      pointer to list-object is added to LIST-OF-OBJECTS-TO-EXECUTE
    end if
  end for
end for
for exeobject in LIST-OF-OBJECTS-TO-EXECUTE do
  if exeobject.flow==1 then
    execute exeobject
    for subexeobject in exeobject.subobjects do
      map subexeobject
    end for
  end if
  pointer to list-object is added to PAST-LIST-OF-OBJECTS-TO-EXECUTE
  delete exeobject-pointer in LIST-OF-OBJECTS-TO-EXECUTE
end for
for pastexeobject in PAST-LIST-OF-OBJECTS-TO-EXECUTE do
  execute exeobject
end for

```