Cympha™

Language that Yields Medical Process HAndling

ALGORITHM

May 23, 2017

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

The process of interpretation LYMPHA 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 coverd below.

1 Initiate

The goal of the syntax is to build asignments for the two datatypes. These asignments 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 exeboject-pointer in LIST-OF-OBJECTS-TO-EXECUTE
    end if
  end for
  for pastexeobject in PAST-LIST-OF-OBJECTS-TO-EXECUTE do
    EXECUTE FUNCITON (pastexeobect.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 exeboject-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 exeboject-pointer in LIST-OF-OBJECTS-TO-EXECUTE
end for
for pastexeobject in PAST-LIST-OF-OBJECTS-TO-EXECUTE do
  execute exeobject
end for
```