Jumping Machine Implementation On Computer

• Student Name Lixuan Dai
• Student ID 1926198
• Supervisor Name Paul Blain Levy

• Project Category/Topic: Theory

Project Aim:

• Goal

Jumping machine is a kind of method that using graph view to indicate the program execution. As its name says, jumping machine considers the calling and returning of a function as a jump and it can embody these operations. In this project we try to implement a jumping machine program on the computer, which can transform the given code to the jumping machine graphical syntax.

• Significance:

It can be used as a good pedagogical tool for programmers to understand the execution procedure of program.

• Relevance:

The relative theory contents are listed as following:

Lamda calculus

It can be a good pedagogical tool for programmers to understand the execution procedure of program.

The basic grammar of jumping machine is from lamda calculus, and call-by-value(CBV). and call-by-name (CBN) variants is also involved.

Visual programming

The program is GUI based, and it will also refer to graph drawing since it need display the jumping semantic trace on the screen.
Program execution mechanism
The jumping machine is used to show the software execution process, so understanding how it works can help a lot.

Related work:

• Jumping Semantics For Call-By-Push-Value [1] gives the definition of the jumping machine, and it also shows how the jumping machine works on paper.

• Besides, the jumping machine is based on a new concept called call-by-push-value, which is introduced in detail in Call-by-push-value [2]

Project Objectives/Deliverables:

In this project, I will implement a program. This program should have a complete GUI for Human-Computer-Interaction. Besides, it should achieve the following functions:

Graphical syntax builder

1. The program could allow user to build the graphical traces using JK machine syntax or Jumping syntax. User can input the code and drag corresponding shapes to the canvas.
2. The program could allow user to arrange the trace. User can adjust the place of those shapes and lines to make it looks good and clear.

Graphical syntax executer

3. For a complete jumping syntax trace, the program can display it step by step, which can clearly show the fetch-decode-execute cycle of the trace.
4. For a complete CK machine trace, the program can transform it to jumping syntax.
5. The program will also display the state of the stack and the binding relationship of values.

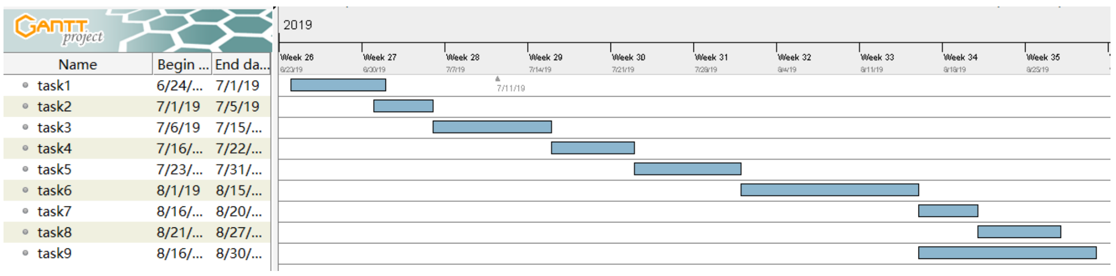
Methodology:

I determine to use Java to implement this program, because the GUI and graph drawing can be easily achieved by JavaFX, which is quite powerful on these fields. Besides, I have some experience on programming by Java, so I think it may be a good choice.

Project plan:

The project can be divided into several tasks, and the Gantt chart will be given as following.

Task 1. Theorical research
Task 2. Complete a simple GUI
Task 3. Complete the graphical syntax builder module
Task 4. Try to execute simple graphical syntax on computer (without conditional branches).
Task 5. Try to achieve the transform between CK syntax and jumping syntax (without conditional branches).
Task 6. Complete the graphical syntax executer module.
Task 7. Test and debug.
Task 8. Optimization (make the GUI more beautiful and add extra functions).
Task 9. Report writing.



Name	Begin ...	End da...
task1	6/24/...	7/1/19
task2	7/1/19	7/5/19
task3	7/6/19	7/15/...
task4	7/16/...	7/22/...
task5	7/23/...	7/31/...
task6	8/1/19	8/15/...
task7	8/16/...	8/20/...
task8	8/21/...	8/27/...
task9	8/16/...	8/30/...

Figure 1. Gantt chart for summer project

Risks and contingency plan:

Risk

There are some difficulties existed in this project:

1. As a quite new field, there are few reference materials and similar researches.
2. It is easy to implement the jumping machine on paper, but it is challenging to implement on the computer, which require a high level of programing skill.

Besides, transforming the code into the jumping semantics is also difficult in this project, especially if there are conditional branches in the code. Therefore, I decided to try to solve simple tasks which do not include branches.

Contingency plan

Due to the risks, it is important to prepare the contingency plan. There are two contingency plans.

1. If the time is not enough to achieve all the functions of this program, I will at least finish the first module: graphical syntax builder, which can allow user to build and arrange the graphical trees, and then try to make it more
2. Another contingency plan is to change the project to a theorical one, which is based on the paper [1]. I finished the exercise about transformation between JK syntax and jumping sytnax on the paper, and may do some deep research on that.

Reference

[1] P. B. Levy. Jumping Semantics For Call-By-Push-Value. O. Danvy and H. Thielecke eds., Proceedings, KAZAM Workshop, Birmingham, U.K., June 2005, University of Birmingham technical report CSR-06-10 (appeared 2006), pages 27-40.
[2] P. B. Levy. Call-By-Push-Value. Semantic Structures in Computation. Kluwer, 2004