GraphQuil Project Proposal

COMS W4115

Group Members:

- Steven Weiner, Systems Architect (srw2163)
- Jon Paul, Manager (jp3144)
- John Heizelman, Language Guru (jrh2184)
- Gemma Ragozzine, Tester (gr2390)

1 Description

GraphQuil is a specialized programming language developed to manage and manipulate data stored in graphs. It is simultaneously a data-definition language as well as a data manipulation language. Users operating GraphQuil will have the ability to create vertices that represent objects, while also being able to link them together by creating directed or undirected edges.

Vertices and edges are structs that can hold as little or as much information as the user requires. The only mandatory field of a vertex or an edge is a unique ID number, which can be generated automatically or specified manually upon creation.

2 Proposed Uses

GraphQuil can not only be used to create graphs but it can also be used to manage data and also as a tool to solve popular graph theory problems (e.g. shortest path).

3 Syntax

3.1 Comments

Comments are written in the style below:

```
#this is a comment
```

Comments that span longer than a single line are written in the style below:

```
#* This is a multi-line comment
```

Each line does not need an asterisk and the end of the multi-line comment is as follows:

```
#*This is the beginning of a multi-line comment.

This is the end of a multi-line comment. *#
```

3.2 Types

- “bool”- a boolean primitive data type where 1 is equal to true and 0 equal to false

- “graph”- a system of nodes and links, an aggregate data type that is a collection of nodes, links, strings, nums, and bools
- “link”- an edge between two nodes, an aggregate data type that is a collection of strings, nums, and bools
- “list”- as a query is performed on the graph that returns multiple pieces of data, the data is by default returned as a list. A “list” also has certain types of associated operations (e.g. iteration, length)
- “node”- a vertex, an aggregate data type that is a collection of strings, nums, and bools
- “num”- an integer or float (both are indistinguishable to the user but the compiler would utilize two different num implementations, one floating point and one 2’s compliment)
- “string”- a string, a sequence of chars within single quotes

3.3 Graph Creation

#Creates an empty GRAPH structure

```
GRAPH classroom;;
```

#Set metadata of graph

```
String lang = ‘GraphQuil’;

GRAPH~language = lang;;
```

3.4 Node Creation and Insertion

NODE INTO classroom (team_name) VALUES ('GraphQuil'); -

#Populate a graph with nodes with following syntax

NODE INTO classroom (uni, first, last, year) VALUES (‘jrh2184’, ‘John’, 'Heizelman', 2016);;

NODE INTO classroom (uni, first, last, year) VALUES (‘gr2390’, ‘Gemma’, 'Ragozzine', 2016);;

NODE INTO classroom (uni, first, last, year) VALUES (‘jp3144’, ‘Jon’, 'Paul', 2015);;

NODE INTO classroom (uni, name, year) VALUES (‘srw2163’, ‘Steven Weiner', 2016);;

NOTE: all the nodes can have different data types and stored associations

3.5 Node Manipulation

UPDATE NODE classroom INSERT (nickname) VALUES ('The Pope') WHERE uni='jp3144';;

UPDATE NODE classroom SET first='2015' WHERE uni='gr2390';;

Updating a node or link can both add new data fields or change existing ones. The new node is returned with updated changes.

3.6 Edge Creation

LINK INTO classroom START team_name='GraphQuil' END uni='jrh2184' (role) VALUES ('language guru');;

LINK INTO classroom START team_name='GraphQuil' END uni='gr2390' (role) VALUES ('testing manager');;

LINK INTO classroom START team_name='GraphQuil' END uni='jp3144' (role) VALUES ('project manager');;

LINK INTO classroom START team_name='GraphQuil' END uni='srw2163' (role) VALUES ('system architect');;

these form a link from every node with team_name='GraphQuil' (could be more than one)

to the destination of every node with uni='...' (again, matches all)

here, the WHERE function is implicit after START and END

#... do the same for other project groups

3.7 Data Manipulation and Retrieval

For each case in which data is meant to be outputted or returned, our language has a similar implementation to OCaml. Instead of explicit returns, when data is meant to be outputted, the code only requires a double semicolon to end the line and that data will be the result of the function.

SELECT NODE *

FROM classroom

WHERE year='2016';;

This is a fairly simple query that selects all juniors in the class (nodes that represent students where NODE~year = 2016) and returns all data

Here the WHERE searches through NODE data within the classroom

SELECT NODE first, last

FROM classroom LINK dest

WHERE role='project manager';

ORDER BY last;;

#*

This returns the first and last name of all students who are project manager

The LINK dest after classroom indicates that the WHERE function will be used

to search for data within the links of the graph, then select the NODE data

```
from the destination, i.e. where the link goes to

NOTE: Because Steven Weiner has no first/last data, his would not be returned

to fix, change first, last to first, last, name

*#

SELECT NODE *

FROM classroom PATH

START uni='jrh214'

END uni='gr2390';;

#* The language also supports searching for nodes along a path, will return all nodes

From the START to the END, inclusive

In this case, it would return nothing, because there is no valid path. *#

LINK INTO classroom BID uni='jrh214' BID uni='jp3144' (relation) VALUES ('friends');;

LINK INTO classroom BID uni='gr2390' BID uni='jp3144' (relation) VALUES ('coworkers');;

# Supports bidirectional links using BID instead of START/END

SELECT NODE *

FROM classroom PATH

START uni='jrh214'

END uni='gr2390';;

# After adding in the necessary links, this would return data for jrh2184, jp3144, and gr2390

SELECT LINK *

FROM classroom PATH

START uni='jrh2184'

END uni='gr2390';;

# This would do the same basic thing, only returning the link data instead of nodes
```

3.8 Functions in GraphQuil

To write a function, we will be adding in some principles of functional languages. We will write basic operations on the primitives mentioned earlier (num, bool, and various String operations). To combine these operations into larger functions, a naming scheme similar to OCaml will be used within a function. Additionally, we will also support for and while loops within functions. For example, the following could be

done:

#assume grades have been added to all the students in the classroom

LET average = SELECT NODE grade FROM classroom;

sum = 0;

FOR tmp IN x (

    sum+= tmp;

)

sum / x.length;;

#notice the double semicolon at the end of the function to return average, everything else disappears in garbage collection after the completion of the function

#this would return a num representation of the average of all student's grades