CS 315

Programming Languages

Language Design Report

GRAFI315

Group
Doğukan Altay
Ahmet Emre Nas
Berk Erzin

A) In GRAFI315 users can create directed or undirected graph. Directed graph can be initliaze with keyword directedGraph and undirectedgraph with undirectedGraph keyword. Vertexes can have multiple properties.

Example directedGraph construction:

```
directedGraph a{
Vertex v0, v1, v2, v3;
Edge e = v0:>v1; // v0 and v1 are vertexes and the edge e goes from
                        // v0 to v1, vertex initiliazing will be shown in next chapter
Edge b = v2:>v3;


}
```

B) Example undirectedGraph:

```
undirectedGraph a{
Vertex v0, v1, v2, v3;
Edge e = v0::v1; // v0 and v1 are vertexes and the edge e goes double way
                        // vertex initiliazing will be shown in next chapter
Edge b = v2::v3;
}
```

As seen from the examples above to construct a graph edges should be initiliaze inside curly brackets

C) In GRAFI315 users initiliaze vertex properties just like edges(inside the graph brackets)

```
directedGraph a{
Vertex v0, v1, v2, v3;
v0 -> “name” = "Ahmet"; //Defining edge properties
v0 -> ”id “= 11111;
Edge e = v0:>v1; // v0 and v1 are vertexes and the edge e goes from
                        // v0 to v1, vertex initiliazing will be shown in next chapter
Edge b = v2:>v3;
}
```

D) Edges properties can initliazed with the same way with vertexes

```
directedGraph graph1{
Vertex v0;
v0->”name” = "Emre"; //Name property of v0 is a string
v0->”number” = 123; //Number property of v0 is int
Vertex v1; //v1 does not have any properties
Edge e = v0 :>v1;
e-> “order” = 1; //e has property of order which is integer
e->”name” = "CS315"; //e has property of name which is string
 }
```

E) The GRAFI315 Language supports integers, floats and strings as primitive types. The language is a staticly typed language that user should use certain keywords to define variables.

```
int num = 10;  //variable num defined as an integer and assigned to “10”
float f_num = 10.3; //variable f_num defined as a float and assigned to “10.3”
string name =  “dogukan” // variable name defined as a string and assigned to “dogukan”
```

In GRAFI315 edge and vertex properties can be assigned to collections as well. The language supports lists, sets and maps as collection types. Also language supports arbitrary nesting.

```
// Defining set properties

directedGraph setProperty{
Vertex v0, v1, v2;
Edge e = v0:>v1;
Edge b = v1:>v2;

v0 -> “id” = <12, 10, 3, 11>; // defining a int list to a property
v2 -> “names” = <“ali”, “veli”, “haydar”, “abdullah”> //defining strings as an entry to the list.
}

// Defining map properties

directedGraph mapProperty{
Vertex v0, v1, v2;
Edge e = v0:>v1;
    Edge b = v1:>v2;

    v0 -> “scores” = { “ali”: 12, “veli”: 10, “haydar”: 11, “abdullah”: 3 } // defining maps as a property
    v1-> “courses” ={“ali”: < 315, 319> , “veli”: <223, 202, 301> }; // defining nested collections. Map keys assigned to sets of integers.
}
```

```
        // Defining list properties


        directedGraph listProperty{
        Vertex v0, v1, v2;
        Edge e = v0:>v1;
                Edge b = v1:>v2;
                v0 -> “scores” = [12, 10, 11,3 ]; // initializing a list property
                v0 -> “scores”.add(0, 8); // adds “8” to the 0th index of the list, [8,12,10,11,3].

                v0 -> “scores”.remove(4); // removes 4th index element from the list,
[8,12,10,11].           int temp = v0 -> “scores”.get(4); // assigns the 4th index element of the
list to the temp variable. (Both should be the same primitive type in order to assign)
        }
```

2. The graph querying language supports:

Quaries in GRAFI315 are initliazed like the sample codes below. All queries should have a name. This name should be used

GRAFI315 support “^” as concatination, “|” as alternation and “*” as repetition for queries.

Example Code:

Query query1 = {(vertex->”name” == “S1”) ^ (edge->”name”==”emre”) ^ (vertex->”number”==21)};
Query query2 = {((vertex->”fruit”==”apple”) ^ (edge-> “path” = 1) ^ (vertex->”vegetable”==”leek”)) | { ((vertex->”fruit”==”apple”) ^ (edge-> “path” = 1) ^ (vertex->”vegetable”==”spinach”))};
Query query3 = {vertex->”name” == “S1”^ edge->”name”==”emre” ^ vertex->”number”==5+7};
Query query4 = {(vertex->”name” == “S1”^ edge->”name”==”emre” ^ vertex->”number”==5+7)*};

Examples above shows how to use concantination, alternaton and repetition. Since all the parts of queries boolean expresions it is possible to use “not” in these part of queries with the symbol “!” at the beginning

Example Code
Query query1 = {(vertex->”name” == “S1”) ^ (edge->”name”==”emre”) ^ !(vertex->”number”==21)}; /* query search for a path that has vertex name “s1” edge name “emre” and any vertex with number different than 21.*/

Queries can have function in the parts of it like hasProp(String property)

Query query1 = {vertex->hasProp(“name”) ^ (edge->”name”==”emre”) ^!(vertex->”number”==21)}; /* query search for a path that has vertex property “name” edge named “emre” and any vertex with number different than 21.*/

Quaries also support modularity. Like below
Consider the queries in the first example

Query query5 = query4^query3;

Query mod_query = (query1^query3)* | (query2^query4)*;