Programming Languages (LP) - Haskell Lab Assignment
Querying Small Property Graphs
Q1 2021–2022

Nowadays many graph data base management systems are based on the *property graph data model*. The goal of this assignment is to implement this abstraction and to evaluate some queries against this kind of graphs.

Definition 1 (Property graph [1])**.** A property graph G is a tuple $(V, E, \rho, \lambda, \sigma)$, where:

1. V is a finite set of vertices (or nodes).
2. E is a finite set of edges such that V and E have no elements in common.
3. $\rho : E \to (V \times V)$ is a total function.
   Intuitively $\rho(e) = (v1, v2)$ indicates that e is a directed edge from node v_1 to node v_2 in G.
4. $\lambda : (V \cup E) \to Lab$ is a total function with Lab a set of labels.
   Intuitively,
   (a) If $v \in V$ and $\lambda(v) = \ell$, then ℓ is the label of node v in G.
   (b) If $e \in E$ and $\lambda(e) = \ell$, then ℓ is the label of edge e in G.
5. $\sigma : (V \cup E) \times Prop \to Val$ is a partial function with $Prop$ a finite set of properties and Val a set of values.
   Intuitively,
   (a) If $v \in V$, $p \in Prop$ and $\sigma(v, p) = s$, then s is the value of property p for node v in the property graph G.
   (b) If $e \in E$, $p \in Prop$ and $\sigma(e, p) = s$, then s is the value of property p for edge e in the property graph G.

Remark 1. The set of values, Val, can be numbers, strings, dates, booleans. When representing the type Val, it must be defined in terms of the basic types in Haskell: Int, Double, String and Bool. Additionally, consider having the Date type.

Remark 2. The function σ is a partial function, thus vertices and edges can have undefined properties. If a property $p \in Prop$ is undefined for a vertex v (resp. an edge e), we write $\sigma(v, p) = \perp$ (resp. $\sigma(e, p) = \perp$). You must choose a representation in Haskell suitable for dealing with this partial function.

Example 1. *Figure 1 depicts (part of) a very simple social property graph* $G = (V, E, \rho, \lambda, \sigma)$. *This example is useful for visualizing the key concepts in Definition 1.*

The components of the social property graph G *are:*

$V = \{n_1, n_2, n_3, n_4, n_5\}$

$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

$Lab = \{\mathsf{Person}, \mathsf{Post}, \mathsf{Tag}, \mathsf{likes}, \mathsf{dislikes}, \mathsf{hasFollowers}, \mathsf{knows}, \mathsf{date}\}$.

$Prop = \{\mathsf{name}, \mathsf{firstName}, \mathsf{lastName}, \mathsf{country}, \mathsf{gender}, \mathsf{date}, \mathsf{content}, \mathsf{language}\}$.

$$\rho(e_1) = (n_1, n_2) \quad \rho(e_2) = (n_2, n_1) \quad \rho(e_3) = (n_3, n_1) \quad \rho(e_4) = (n_4, n_3)$$
$$\rho(e_5) = (n_1, n_4) \quad \rho(e_6) = (n_2, n_5) \quad \rho(e_7) = (n_2, n_4)$$

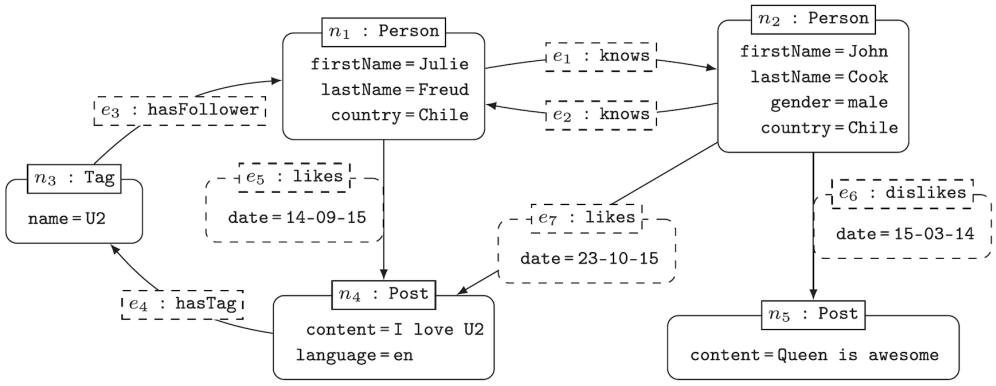

Figure 1: Part of a social property graph. Example borrowed from [1].

$\lambda(n_1) = \mathsf{Person}$ $\lambda(n_2) = \mathsf{Person}$ $\lambda(n_3) = \mathsf{Tag}$ $\lambda(n_4) = \mathsf{Post}$
$\lambda(n_5) = \mathsf{Post}$

$\lambda(e_1) = \mathsf{knows}$ $\lambda(e_2) = \mathsf{knows}$ $\lambda(e_3) = \mathsf{hasFollower}$ $\lambda(e_4) = \mathsf{hasTag}$
$\lambda(e_5) = \mathsf{likes}$ $\lambda(e_6) = \mathsf{dislikes}$ $\lambda(e_7) = \mathsf{likes}$

$\sigma(n_1, \mathsf{firstName}) = \mathsf{Julie}$
$\sigma(n_1, \mathsf{lastName}) = \mathsf{Freud}$
$\sigma(n_1, \mathsf{country}) = \mathsf{Chile}$
$\sigma(n_2, \mathsf{firstName}) = \mathsf{John}$
$\sigma(n_2, \mathsf{lastName}) = \mathsf{Cook}$
$\sigma(n_2, \mathsf{Country}) = \mathsf{Chile}$
$\sigma(n_2, \mathsf{gender}) = \mathsf{male}$
$\sigma(n_3, \mathsf{name}) = \mathsf{U2}$
$\sigma(n_4, \mathsf{content}) = \mathsf{I\ love\ U2}$
$\sigma(n_4, \mathsf{lenguage}) = \mathsf{en}$
$\sigma(n_5, \mathsf{content}) = \mathsf{Queen\ is\ awesome}$
$\sigma(e_5, \mathsf{date}) = 14 - 09 - 15$
$\sigma(e_6, \mathsf{date}) = 15 - 03 - 14$
$\sigma(e_7, \mathsf{date}) = 23 - 10 - 15$

Notice that, among other undefinitions of the function σ, we have $\sigma(n_1, \mathsf{gender}) = \perp$, $\sigma(n_5, \mathsf{language}) = \perp$.

1 Property Graphs in Haskell

Define and implement a representation of property graphs, PG, in Haskell. The representation must be suitable for implementing the functions below and to evaluate the queries specified in Section 2 against a given property graph.

1. $populate : String \times String \times String \times String \to PG$
   For instance, *populate*(`"rhoFile.pg","lambdaFile.pg","sigmaFile.pg", "propFile.pg"`) creates a property graph from input files `"rhoFile.pg"`, `"lambdaFile.pg"`, `"sigmaFile.pg"` and `"propFile.pg"`. These input files are given in the following format:

`"rhoFile.pg"`	`"lambdaFile.pg"`	`"sigmaFile.pg"`	`"propFile.pg"`
$e \quad n_1 \quad n_2$	$n_1 \quad \ell$	$n_1 \quad p \quad v$	$p \quad tvalue$

   where $e \in E, n_1, n_2 \in V, \ell \in Lab, p \in Prop$, $v \in Val$ and $tvalue$ is one of the types allowed for Val, i.e. $\mathsf{Int}, \mathsf{Double}, \mathsf{String}, \mathsf{Bool}$ and Date. In Appendix A are part of the input files to populate the graph in Example 1.

2. $addEdge : PG \times E \times V \times V \to PG$

3. $defVprop : PG \times V \times \mathcal{P}(Prop \times Val) \to PG$[1].

4. $defEprop : PG \times E \times \mathcal{P}(Prop \times Val) \to PG$

   In these two functions above, the given properties for the vertex (resp. edge) must be updated using the corresponding given values.

5. $defVlabel : PG \times V \times Lab \to PG \cup Error$

6. $defElabel : PG \times E \times Lab \to PG \cup Error$

   In these two functions above, if the label is already defined for the vertex (resp. edge), an error message must be emitted. Otherwise, the label must be defined using the new value.

7. $showGraph : PG \to (V, E, Lab, Prop, \rho, \lambda, \sigma)$
   Actually, the function *showGraph* is a printing action. This is, *showGraph* must print the components of the 7-tuple in the following format:

   (a) For each vertex v in the graph, with $\lambda(v) = \ell$ and $\sigma(v, p_i) = val_i$, $i = 1, \ldots, k$, it must be listed as follows:

   $$v[\ell]\{(p_1, val_1), (p_2, val_2), \cdots, (p_k, val_k)\}$$

   (b) For each edge e in the graph, with $\lambda(e) = \ell$, $\rho(e) = (n_1, n_2)$ and $\sigma(e, p_i) = val_i$, $i = 1, \ldots, k$, it must be listed as follows: it must be listed as follows:

   $$(n_1) - e[\ell] -> (n_2)\{(p_1, val_1), (p_2, val_2), \cdots, (p_k, val_k)\}$$

   In both cases above, only defined properties must be listed. In Appendix B can be seen the output emitted by $showGraph(G)$. G is the graph in Example 1.

[1] $\mathcal{P}(A)$ stands for the set of parts of A.

2 Querying against Property Graphs

A query against a property graph is a request for data or information regarding vertices, edges (relations), graph patterns or paths occurring in the property graph. Usually, these queries involve some constraints over the property of vertices and/or edges. In this assignment we ask you for implementing the set of functions below. These functions allow for some querying against property graphs.

1. $\sigma' : PG \times (V \cup E) \to \mathcal{P}(Prop \times Val)$
   If $\sigma(v, p) = val$ (resp. $\sigma(e, p) = val$), the pair $(p, val) \in \sigma'(G, v)$ (resp. $(p, val) \in \sigma'(G, e)$)

2. $propV : PG \times Nat \times Prop \to \mathcal{P}(V \times Val)$
   $propV(G, k, p)$ returns the k first pairs $(\lambda(v), \sigma(v, p))$, $v \in V$ such that $\sigma(v, p)$ is defined in G.

3. $propE : PG \times Nat \times Prop \to \mathcal{P}(E \times Val)$
   $propE(G, k, p)$ returns the k first pairs $(\lambda(e), \sigma(e, p))$, $e \in E$ such that $\sigma(e, p)$ is defined in G.

4. $kHops : PG \times Nat \times V \times Prop \times (Val \times Val \to Bool) \times Val \to \mathcal{P}(V \times Lab \times Val)$
   $kHops(G, k, v, p, f, x) = \{(v', \lambda(v'), \sigma(v', p)) | reach_k(v, v') \wedge f(x, \sigma(v', p))\}$
   $reach_k(v, v')$ means that there exists a k-path from v to v'.

5. $reachable : PG \times V \times V \times Lab \to Bool$
   $reachable(G, v, v', \ell) = \mathsf{True}$ if $\exists v = v_0, v_1, \ldots, v_n = v'$such that $\forall e, \rho(e) = (v_i, v_{i+1}), \lambda(e) = \ell$, $i = 0, \ldots, (n-1)$. Otherwise False
   This is, $reachable(G, v, v', \ell)$ returns True if there is a path from v to v' where all the edges in the traversal have the same label ℓ.

Remark 3. Even though you will represent the type Val according to the needs of your code, keep in mind the result of the querying functions above must be given in terms of basic types of Haskell.

3 Submissions

- Your submission must consist of a single `propgraphs.hs` file, duly commented on, making explicit the decisions you have had to make.

- Your implementation must provide a main program such that:

    1. It populates a property graph. Users must enter the names of the files containing the property graph info.
    2. It repeats a script of updates and queries against the created property graph until users decide stopping executing the script. Each update must be followed by a call to the *showGraph* function. For each query evaluated users must enter the real parameters and the result must print.

- **It is forbidden to use any Haskell module or library developed for implementing graphs**.

- Regarding programming style,

    1. Use mnemotecnic names for functions and variables.
    2. For the indentation use blank spaces instead of tabs.
    3. Code lines must not be longer than 120 columns.
    4. Comments must be clear, concise and precise.
    5. In general, take care of the legibility of your code

IMPORTANT Your input file (.pg) should be in a text file. This text file must be in unix format (line break with "\n") and not in WindowsDOS format (line break with "\r" and "\n"). We noticed problems with read operations in Haskell if the file is in WindowsDOS format. If you work on Windows, you should keep this in mind.This link may be useful to you: `https://phoenixnap.com/kb/convert-dos-to-unix`.

4 Further Explanations

- Assume the property graph given by means of the files *.pg is right. This means, it is not necessary to validate these data when populating the PG.

- Remark 3 in Section 2 means that, for instance, when showing the PG, if there is a vertex with properties represented as (name, Str Rosa) and (age, Intg 23), these properties must be shown as (name, Rosa) and (age, 23), respectively.

- Regarding functions *addEdge*, $defVlabel$ and $defElabel$: These functions are intended to be used when populating the PG. However, they also are useful for updating the graph. When they are used for updating the PG, its is possible that their parameters, vertices as well as edges, are not defined and, hence, they must be defined in the PG. Notice that in this cases, vertices/edges won't have any property defined. When updating via $defVlabel/defElabel$, if the vertice/edge has a previously defined label or the vertice/edge does not exits, an error must be reported. In particular, changing a label could generate an inconsistency in the graph. In the case of functions $defVprop$ and $defEprop$, if the vertice/edge does not exists, the PG must remain the same.

- Regarding the function *showGraph*, you could define the representation of the PG as an instance of the class Show and then, define *showGraph* in terms of showing the PG.

Appendix A Input Files Format Example

Below you can see (part of) the input files for the graph in Figure 1

"rhoFile.pg"			"lambdaFile.pg"		"sigmaFile.pg"		"propFile.pg"
$e1$	$n1$	$n2$	$n1$	Person	$n1$	firstName Julie	firstName String
$e2$	$n2$	$n1$	$n2$	Person	$n1$	lastName Freud	lastName String
$e3$	$n3$	$n1$	$n3$	Tag	$n1$	country Chile	country String
$e4$	$n4$	$n3$	$n4$	Post	$n2$	firstName John	gender String
$e5$	$n1$	$n4$	$n5$	Post	$n2$	lastName Cook	content String
$e6$	$n2$	$n5$	$e1$	knows	$n2$	gender male	date Date
$e7$	$n2$	$n4$	$e2$	knows	$\cdots$		$\cdots$
			$\cdots$		$\cdots$		$\cdots$

Appendix B Output Format Example

The result of applying showGraph(G) to the graph in Example 1 -actually, a part of the result- is the following:

$n1[\mathsf{Person}]\{(\mathsf{firstName}, \mathsf{Julie}), (\mathsf{lastName}, \mathsf{Freud}), (\mathsf{Country}, \mathsf{Chile})\}$
$n2[\mathsf{Person}]\{(\mathsf{firstName}, \mathsf{John}), (\mathsf{lastName}, \mathsf{Cook}), (\mathsf{gender}, \mathsf{Male}), (\mathsf{Country}, \mathsf{Chile})\}$
$\vdots$
$n4[\mathsf{Post}]\{(\mathsf{Content}, \mathsf{I\ love\ U2}), (\mathsf{language}, \mathsf{en})\}$
$\vdots$
$(n_4) - e4[\mathsf{hashTag}] -> (n_3)\{\ \}$
$(n_1) - e5[\mathsf{likes}] -> (n_4)\{(\mathsf{date}, 14-09-15)\}$
$(n_2) - e6[\mathsf{dislikes}] -> (n_5)\{(\mathsf{date}, 15-03-14)\}$
$\vdots$

References

[1] Angles, R., Arenas, M., Barceló, P., Hogan, A. Reutter, J. and Vrgoč, D. *Foundations of modern query languages for graph databases.* ACM Computing Surveys (CSUR). Vol. 50, Num. 5, pp. 1–40, ACM New York, NY, USA. 2017.