Querying graph databases

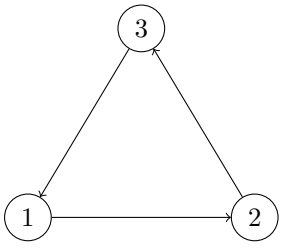
Oliver Markgraf

June 27, 2024

Problem Overview

A directed graph is a tuple $G = (V, E)$ where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. A path π is an alternating sequences of vertices $\pi := v_1 v_2 \dots v_n$ such that $v_i \in V$ and $e_j = (v_j, v_{j+1}) \in E$ for all $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, n-1\}$. Additionally, we define a set of finite attributes named `Attr`, and a function ν which maps from node and attribute to a value: $\nu : V \times \texttt{Attr} \to \texttt{Val}$ where $\texttt{Val} \subseteq \Sigma^* \cup \mathbb{R}$ is a set of possible values `Attr` can take, usually either a string or a real value (In this case Σ is all unicode letters). Figure 1 shows one example of such a graph. The set of vertices is $V = \{1, 2, 3\}$ and $E = \{(1, 2), (2, 3), (3, 1)\}$. $\texttt{Attr} = \{\text{Age, Name}\}$ and $\nu(1, \text{Age}) = 30$.

A graph query automaton $\mathcal{A} = (\mathcal{X}, Q, \Delta, q_0, F)$ with a graph G and set of `Attr`, `Val` and ν is a tuple where Q is a finite set of states, $\Delta \subseteq Q \times T(\mathcal{X}) \times Q$ is the transition function, q_0 is the initial state and F is the set of final states. The set of variables is $\mathcal{X} = \texttt{Attr} \cup \mathbb{P}$ where $\mathbb{P}$ is a set of global real valued variables and the variables from `Attr` to their corresponding type of `Val`. The transitions of the automaton are formulas of the following form, where x are variables from `Attr` that map to a string value, c_{str} is a string constant, z_i are variables from `Attr` that map to a real value or are from the set of global variables $\mathbb{P}$ or are constants in $\mathbb{R}$. All a_i are real valued coefficients and $\sim \in \{=, <, >, \geq, \leq\}$:



Attributes:

Node	Age	Name
1	30	John
2	25	Alice
3	35	Bob

Figure 1: Graph, Attributes

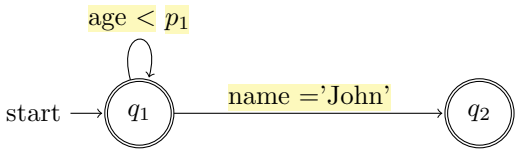


Figure 2: Automaton with 2 states and transitions labeled age $<$ 1 and name ='John'

$$\begin{aligned} \phi &::= \phi \wedge \ \phi \\ &\mid x = c_{str} \\ &\mid t_{ar} \sim t_{ar} \\ t_{ar} &::= \Sigma_i a_i z_i \end{aligned}$$

Then the *regular path query problem* is defined as follows. For $\mathcal{A}, G$, `Attr`, `Val`, ν and two vertices $s, t \in V$, does there exists a path in the graph $s = v_1 v_2 \ldots v_k = t$ such that the automaton has an accepting run, i.e., there exists a sequence of transitions in the automaton $q_0 \xrightarrow{\phi_1} q_1 \xrightarrow{\phi_2} \ldots \xrightarrow{\phi_{k-1}} q_k \in F$, such $\sigma(\phi_i)$ is satisfied for a valuation of the global variables and for each $i \in \{1, \ldots, k-1\}$, where σ is a substitution function, replacing each variable in ϕ_i from `Attr` by the value of the destination vertex in the graph, $\nu(v_{i+1})$.

Example

Consider the graph, attributes and automaton shown in Figure 1 and 2 (This corresponds to example1.json in the source code). The variable p_1 is a global variable. The graph has two different attributes, *'Age'* and *'Name'*. A path in the graph would be 1231. One path that satisfies the query is the path 1231 in the graph with the valuation of $p_1 = 36$ corresponding to the word in the automaton (25, 'Alice')(35, 'Bob')(30, 'John') where the first element in the tuple corresponds to the 'Age' value and the second element corresponds to the 'Name' value.

Input Data

The data is structured in JSON format. The information about the graph includes node connectivity via edges and the attributes associated with each node. Moreover, the automaton is part of the JSON file, encoding initial state, transitions, and final states. A parser of such a file is included in the code.

Tasks

1. **Check Path Existence in Graph:** Write a method to determine if there exists a path from a given source node to a target node in the graph.

2. **Check Path Satisfaction with Query:** Implement a method to check if there exists a path from a source node s to a target node t in the graph that satisfies the query defined by the automaton. For this do a search from the starting vertex in the graph and accumulate all formulas from the automaton path for each graph vertex and evaluate (based on the attributes). Note, that contrary to standard DFS/BFS vertices may be revisited with different path formulas.

3. **Optimize:** Instead of accumulating all formulas, one can remove redundant ones, for example if the formula $p_1 < 6 \wedge p_1 < 7$ is accumulated, then only one upper bound is necessary. In detail, can you improve the procedure from task 2 such that at most $2n$ formulas are saved per graph vertex and automaton state, where n is the number of atomic formulas in the automatons transitions.

Hints: You can use your own libraries and/or code to solve the problem. There is a rudimentary parser given in the code with some data structures and examples how to use z3 for example. First part of the task is to implement a search for the graph, suitable implementations are depth-first search or breadth-first search. You do not need to implement the second task if you have implemented the third task. Hints for the third task: work with macro states over graph vertices and automaton states, and avoid saving redundant information.