Information Processing Letters 115 (2015) 425–430

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Using variable automata for querying data graphs

Domagoj Vrgoč [a,b,*]

[a] University of Edinburgh, Scotland, United Kingdom
[b] Pontificia Universidad Católica de Chile, Vicuna Mackenna 4860, Edificio San Agustin, Macul, Santiago, Chile

ARTICLE INFO

Article history:
Received 5 June 2014
Received in revised form 27 October 2014
Accepted 28 October 2014
Available online 20 November 2014
Communicated by Jef Wijsen

Keywords:
Databases
Query languages
Variable automata

ABSTRACT

Thus far query languages for graphs databases that, in addition to navigating the structure of a graph, also consider data values encountered along the paths they traverse, seem to exhibit somewhat dual behaviour in terms of the efficiency of their query evaluation problem. Namely, their combined complexity is either tractable, or are at least PSPACE-hard. In this paper we show how to use the model of variable automata to get a query language with intermediate (NP-complete) combined complexity of query evaluation. Since variable automata are incomparable in terms of expressive power with previously studied querying mechanisms for data graphs we also show how to join their capabilities with the ones of previously used languages without an increase in the complexity of query evaluation, thus getting the best of both worlds.

1. Introduction

Querying graph databases has become an important topic in the database community, fuelled by applications such as social networks, biological databases and the Semantic Web. There are now several vendors offering graph database systems [4,12] and a growing body of research literature on the subject (for a survey see [1]). In all of these applications the data is naturally modelled by graphs, with nodes representing entities in the database and edges representing various connections these entities can form. For example if we are describing a social network it is natural to represent users by nodes, with edges symbolizing the connection between two users, such as friends, co-workers, relatives and so on. In this model a node carries information about a specific user in the usual (attribute name, attribute value) format, where the name of the attribute is drawn from a finite alphabet of labels, while the attribute value comes from an infinite domain. For example we can store information about user's name, phone number, etc. Furthermore, since nodes can form different types of connections, it is usual to assign labels to the edges connecting them, as well as some additional information such as the time of the edge creation, or how the edge was modified.

Over the years several querying mechanisms for graph data have been developed, both for navigating graphs and for dealing with the stored data, and their evaluation properties were studied in detail. Most notable among these are *regular path queries*, or RPQs [3] and their extensions with conjunction and two-way navigation [2], or the ability to define more complex graph patterns [13]. All of them have in common the fact that they query the graph structure without the ability to access data values stored in the nodes. More recently languages that in addition to topology also consider data values have been studied [9,11]. For both these classes of languages one of the main concern is the efficiency of their *query evaluation* – that is the problem of checking, given a data graph, a query and a tuple of nodes, if this tuple belongs to the answer of the query on this given graph. This problem is often referred to as the *combined complexity* of query evaluation. When the query itself is fixed and not considered as part of the input we are talking about *data complexity*. By now the consensus

* Corresponding author at: Pontificia Universidad Católica de Chile.
E-mail address: domagojvrgoc@gmail.com.

http://dx.doi.org/10.1016/j.ipl.2014.10.019
0020-0190/© 2014 Elsevier B.V. All rights reserved.

is that while navigational languages can be designed with very low combined complexity in mind, for languages that mix topological properties with data features the problem is either tractable, or at least PSPACE-hard.

Indeed, it appears that models that use memory, such as register automata and their expression equivalent [11], make the query evaluation PSPACE-hard, while XPath-based approaches bring the complexity down to PTIME, but lose the ability to store values into separate memory locations. However, the panorama of languages that mix topology and data is far from being completely understood and it therefore makes sense to look for other query formalisms that might lead to languages with lower complexity of query evaluation while still retaining some of the desirable properties related to manipulation of memory locations.

One such model that was not considered previously for querying graphs is that of variable automata. These were originally introduced in [6] to reason about words over (countable) infinite alphabets, but here we show how they can also be used to define a graph query language with NP-complete combined complexity. We also show that data complexity remains NL-complete, matching the bound for RPQs. Furthermore, since variable automata are incomparable in terms of expressive power with the well established model of register automata, we show how the two can be joined together to get a graph querying formalism whose evaluation complexity (both data and combined) does not exceed that of register automata, while at the same time giving them more expressive power.

Remark. Note that some of the results presented here were announced previously in [10].

Organization. We review notation in Section 2. In Section 3 we introduce variable automata and show how they can be used to query graph databases, while in Section 4 we extend register automata in a way that subsumes properties definable by variable automata. We conclude in Section 5.

2. Preliminaries

Let Σ be a finite alphabet and $\mathcal{D}$ a countable infinite set of data values.

Definition 2.1. A data graph (over Σ) is pair $G = (V, E)$, where

- V is a finite set of nodes;
- $E \subseteq V \times \Sigma \times \mathcal{D} \times V$ is a set of edges where each edge contains a label from Σ and a data value from $\mathcal{D}$.

We write $V(G)$ and $E(G)$ to denote the set of nodes and edges of G, respectively. An edge e from a node u to a node u' is written in the form $(u, \binom{a}{d}, u')$, where $a \in \Sigma$ and $d \in \mathcal{D}$. We call a the label of the edge e and d the data value of the edge e. We write $\mathcal{D}(G)$ to denote the set of data values in G. A sample data graph is given in Fig. 1.

A path from a node v to a node v' in G is a sequence $\pi = v_1 \binom{a_1}{d_1} v_2 \binom{a_2}{d_2} v_3 \binom{a_3}{d_3} \cdots v_n \binom{a_n}{d_n} v_{n+1}$ such that each $(v_i, \binom{a_i}{d_i}, v_{i+1})$ is an edge for each $i \leq n$, and $v_1 = v$ and $v_{n+1} = v'$.

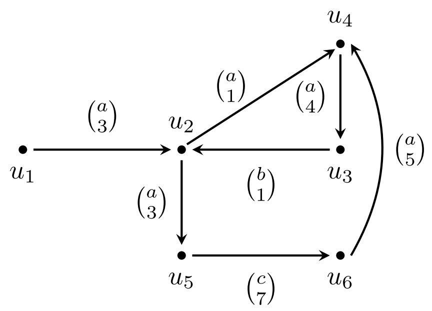

Fig. 1. A graph database with data values.

Each path π defines a *data word* $w(\pi) = \binom{a_1}{d_1}\binom{a_2}{d_2}\binom{a_3}{d_3}\cdots\binom{a_n}{d_n}$. Data words are commonly studied in XML literature [5], where they are used to describe paths in XML trees. We use them in a similar manner to describe paths in data graphs.

Remark Note that we use the model where both labels and data values appear in the edges. Several different approaches have been used in the past, for example with data values in the nodes and labels on edges [11], or both labels and data values in nodes and edges [12], but it is easily shown that all of these variations are essentially equivalent [14]. Our choice is dictated by the ease of notation primarily, as it identifies paths with data words.

3. Variable automata as a graph language

In this section we show how to use the model of variable automata introduced in [6] to query graph databases. These automata can be viewed as less procedural than register automata [8]; in fact they can be seen as NFAs with a guess of values to be assigned to variables, with the run of the automaton verifying correctness of the guess. Originally they were defined on words over infinite alphabets [6], but it is straightforward to extend them to the setting of data words. In what follows we define variable automata as a query language, give examples of some queries one can ask using them and show that their query evaluation problem can be solved in NP-time.

We begin by defining variable automata formally.

Definition 3.1. Let Σ be a finite alphabet and $\mathcal{D}$ a countable infinite domain of data values. We will also assume that we have a countable set V of variables. A *variable finite automaton* (or VFA for short) over $\Sigma \times \mathcal{D}$ is a pair $\mathcal{A} = (\Gamma, A)$, where A is an NFA over the alphabet $\Sigma \times \Gamma$, and $\Gamma = C \cup X \cup \{\star\}$ such that:

- $C \subseteq \mathcal{D}$ is a finite set of data values called *constants*
- $X \subseteq V$ is a finite set of *bound variables*, and
- $\star$ is a symbol for the *free variable*.

Next we define when a VFA accepts a data word $w = w_1 w_2 \ldots w_n \in (\Sigma \times \mathcal{D})^*$. For each letter $u = \binom{a}{d}$ in $\Sigma \times \mathcal{D}$,