A Formalism for Graph Databases and its Model of Computation

Juan Reutter and Tony Tan

University of Edinburgh

Abstract. Graph databases are directed graphs in which the edges are labeled with symbols from a finite alphabet. In this paper we introduce a logic for such graphs in which the domain is the set of edges. We compare its expressiveness with the standard logic in which the domain the set of vertices. Furthermore, we introduce a robust model of computation for such logic, the so called graph pebble automata.

1 Introduction

The study of graph structured data has received much attention lately, due to numerous applications in areas such as biological networks [12, 15], social networks [17], and the semantic Web [9]. The common database model proposed by such applications is most commonly denoted as *graph databases*, in which nodes are objects, and edge labels define relationships between those objects [2].

For querying graph structured data, one normally wishes to specify certain types of paths between nodes. Most common examples of these queries are conjunctive regular path queries [1, 14, 6, 3]. Those querying formalisms have been thoroughly studied, and their algorithmic properties are more or less understood. On the other hand, there has been much less work devoted on other formalisms other than graph reachability patterns, say, for example, the integrity constraints such as labels with unique names, typing constraints on nodes, functional dependencies, domain and range of properties. See, for instance, the survey [2] for more examples of integrity constraints.

Our intention is to study formalisms for such graph databases which is capable of expressing these integrity constraints, while at the same time still feature manageable model checking properties. Obviously such formalisms depend on how the underlying directed graph of the databases are represented in the first place.

The standard representation of directed graphs is simply a set of nodes, together with a binary relation on these nodes to represent the edge among them. The labeling of the edges is represented as a function from the edges to the finite alphabet of symbols. We call such representation the *vertex* representation.

Another less common way to represent directed graphs is to take the edges as the domain, together with some well defined binary relations on these edges to indicate how two edges intersect. The labeling of the edges is represented as a set of unary predicates on the domain. We call such representation the *edge* representation.

In the first part of our paper we propose a vocabulary for the edge representation, which we call $\mathbb{E}$-vocabulary. We call $\mathbb{V}$-vocabulary the vocabulary for the vertex representation. We study the expressive power of $\mathbb{E}$ and compare it to $\mathbb{V}$-vocabulary. In this respect our contributions are the following.

– The logic that we propose for edge representation is robust, in the sense that for each graph database in the vertex representation, there exists a unique (up to isomorphism) graph database in the edge representation that have the same underlying directed graph. Vice versa, for each graph database in the edge representation, there exists a unique (up to isomorphism) graph database in the vertex representation that have the same underlying directed graph.
– Next, we turn our attention to expressivity of $\mathbb{E}$-vocabulary. For first-order logic (FO) we show that it is equivalent to $\mathbb{V}$-vocabulary. On the other hand, for the existential monadic second-order logic ($\exists$MSO), as well as monadic second-order logic (MSO), the $\mathbb{E}$-vocabulary is more expressive than the $\mathbb{V}$-vocabulary. That is, there are $\exists$MSO and MSO sentences in $\mathbb{E}$-vocabulary that cannot be expressed in sentences in $\mathbb{V}$-vocabulary in $\exists$MSO and MSO logics, respectively.

In the second part of our paper we introduce a notion of automata for graph databases. We follows the direction in [19] by defining pebble automata for directed graphs.

Pebble automata was initially introduced for words over finite alphabet in [8]. Later it was extended words over infinite alphabets in [16]. Roughly speaking, a k pebble automaton, in short k-PA, is a finite state automaton equipped with k pebbles. The pebbles are placed on/lifted from the input word in the stack discipline – first in last out – and are intended to mark positions in the input word. One pebble can only mark one position and the most recently placed pebble serves as the head of the automaton. The automaton moves from one state to another depending on the equality tests among data values in the positions currently marked by the pebbles, as well as, the equality tests among the positions of the pebbles.

Later in [19] the connection between graphs and pebble automata was initially introduced. The main idea in [19] is that a word of even length over an *infinite* alphabet can be viewed as a directed graph, hence pebble automata for words over infinite alphabets can be viewed as a model of computation for directed graphs.

In this paper we extend this connection to the case of graph databases, i.e., directed graphs in which edges are labeled with symbols from a finite alphabet Σ. Some of the results in this paper are the following.

1. We define the notion of k pebble graph automata, or in short k-PA, for graph databases.
2. Every first-order sentences of quantifier rank k over graph databases can be simulated by k-PA.
3. We demonstrate the robustness of pebble automata by showing the equivalence between *two-way alternating* k-PA and *one-way deterministic* k-PA. This result settles a question raised in [16]. It was first spelled in [19] for words over infinite alphabet, but no formal proof has been given until now.
   This robustness immediately implies that the class of families of directed graphs captured by k-PA is closed under boolean operations.

We also note that almost all results in [19] can be carried over to the case of graph databases, including the fact that reachability from the source node s to the target node t can be checked by k-PA if and only if the distance from s to t is less than or equal to 2^k. This fact, together with item (1) above, yields the fact that reachability can be

expressed by first-order sentence of quantifier rank k if and only if the distance between source and target nodes is less than or equal to 2^k. As the proof is non standard, in the sense that we do not use the standard Ehrenfeucht-Fraïssé approach which is commonly used in most finite definability results, it is worth to mention that pebble automata can be a potentially useful tool to prove definability results in first-order logic over graph databases.

Related work. Closely related to our work is Courcelles work [5], which appears to be the first ones that suggest including the graph edges as part of the domain. The results and definitions here do not follow from [5]. The first reason is that the logic introduced by Courcelle is essentially two sorted logic. That is, the domain consists of two kinds of elements: the vertices and the edges. Whereas, the logic that we define here has only the edges as the domain. Thus, the logic is defined with different vocabulary than ours. The second reason is that it has not been shown that every structure defined in the vocabulary in [5] is indeed a directed graph. It is not clear at all in the first place why it is true. We prove in this paper that indeed such is the case.

Later on in the paper [10] monadic second-order logic was introduced for abstract matroids, which are extensions of graphs. It was shown in [10] that many results in [5] also hold in this setting. However, the emphasis in [10] is decidability issue for satisfaction problem. So naturally it only considers the family of matroids with bounded *branch width*, the analog of *tree width* for graphs. While in our paper we are more interested in a model of computation for graph databases that feature manageable model checking properties.

Another work related to ours is the work in [4]. In that paper two models of computation for directed graphs are introduced, the so called V-automata and E-automata. In brief, given an input directed graph G, a V-automaton marks the vertices of G with symbols from finite alphabet. The decision to accept G or not depends on this labeling. E-automata operate in the same manner, except that they mark the edges, instead of vertices. It is shown in [4] that V-automata are weaker than E-automata.

These models, the V- and E-automata, are incomparable to our graph pebble automata. On one side, E-automata are capable of simulating μ-calculus on directed graphs, but they are not closed under negation. On the other side, our graph pebble automata are capable of simulating the whole first-order logic on directed graphs, closed under all boolean operations.

Organization. This paper is organized as follows. In Section 2 we define the vocabularies $\mathbb{V}$ and $\mathbb{E}$. Then in Section 3 we define the notion of *structural equivalent*, the notion to compare two structures from $\mathbb{V}$ and $\mathbb{E}$ logics. In Section 4 we compare the expressive power between $\mathbb{V}$ and $\mathbb{E}$ logics. We introduce graph pebble automata in Section 5. We then extend all previous definitions to the labeled edges graphs in Section 6. Finally we conclude with a future direction for our work in Section 7.

2 Representation for graph databases

Graph databases are usually defined as finite edge-labeled directed graphs [2]. In this paper, in order to keep the presentations simple, we shall work only with unlabeled

directed graphs. We will explain how to extend these results for the case of labeled graphs in Section 6.

In what follows, we state two representations for graph databases. The first is the standard one, where a directed graph is just a set of vertices equipped with a binary relation on the vertices. We will denote its vocabulary by $\mathbb{V}$.

The second one is our proposed representation for directed graphs where the edges are the domain. We will denote its vocabulary by $\mathbb{E}$.

The vocabulary $\mathbb{V}$. The vocabulary $\mathbb{V}$ simply consists of one binary predicate E. We denote by $\mathsf{STRUCT}[\mathbb{V}]$ the set of structures of $\mathbb{V}$, which are simply directed graphs. A $\mathbb{V}$-structure is a structure in $\mathsf{STRUCT}[\mathbb{V}]$.

We will usually write $G = (V(G), E(G))$ for structures in $\mathsf{STRUCT}[\mathbb{V}]$, where $V(G) = \mathsf{Dom}(G)$ is the domain and $E(G)$ is the binary relation on the elements in $V(G)$.

The atomic formula in the logic $\mathbb{V}$ is either $x = y$ or $E(x, y)$. The meaning of $E(x, y)$ is simply $(x, y) \in E$. The first-order logic $\mathsf{FO}[\mathbb{V}]$ is obtained by closing the atomic formulas under the Boolean connectives and first-order quantification over V. The logic $\mathsf{MSO}[\mathbb{V}]$, which stands for monadic second-order, is obtained by adding quantification over unary predicates on the domain. If the unary predicates quantifications are all existential, then we denote it by $\exists\mathsf{MSO}[\mathbb{V}]$. A $\mathbb{V}$-sentence is a sentence using the vocabulary $\mathbb{V}$. A sentence φ defines a set of directed graphs via $\mathcal{G}(\varphi) := \{G \mid G \models \varphi\}$.

For the sake of presentation, we only consider graphs $G \in \mathsf{STRUCT}[\mathbb{V}]$ in which there is no isolated vertices and there is no self loop.

The vocabulary $\mathbb{E}$. Intuitively, rather than viewing a directed graph $G = (V, E)$ as a set V of vertices and E a binary relation on V, we take E as the domain and define some relations among the elements in E.

Let u and v be two vertices and e be an edge from u to v. What we mean by the *head* of e is the vertex v, while the *tail* of e is the vertex u. Now the vocabulary $\mathbb{E}$ consists of the binary relations $\mathsf{HeadHead}$, $\mathsf{HeadTail}$ and $\mathsf{TailTail}$ on the directed edges, where the intentions of each predicate are as follows.

- $\mathsf{TailTail}(e_1, e_2)$ means that the tails of e_1 and e_2 are the same.
- $\mathsf{HeadHead}(e_1, e_2)$ means that the heads of e_1 and e_2 are the same.
- $\mathsf{HeadTail}(e_1, e_2)$ means that the head of e_1 is the tail of e_2.

As above, $\mathsf{STRUCT}[\mathbb{E}]$ denotes the set of all structures of $\mathbb{E}$ and an $\mathbb{E}$-structure is a structure in $\mathsf{STRUCT}[\mathbb{E}]$. We assume that the structures in $\mathsf{STRUCT}[\mathbb{E}]$ satisfy the following axioms.

$E1$. Both $\mathsf{HeadHead}$ and $\mathsf{TailTail}$ are equivalence relations.
$E2$. If $\mathsf{HeadHead}(e_1, e_2)$ and $\mathsf{HeadTail}(e_1, e_3)$, then $\mathsf{HeadTail}(e_2, e_3)$.
$E3$. If $\mathsf{TailTail}(e_1, e_2)$ and $\mathsf{HeadTail}(e_3, e_1)$, then $\mathsf{HeadTail}(e_3, e_2)$.
$E4$. If $\mathsf{HeadTail}(e_1, e_3)$ and $\mathsf{HeadTail}(e_2, e_3)$, then $\mathsf{HeadHead}(e_1, e_2)$.
$E5$. If $\mathsf{HeadTail}(e_3, e_1)$ and $\mathsf{HeadTail}(e_3, e_2)$, then $\mathsf{TailTail}(e_1, e_2)$.
$E6$. If $\mathsf{HeadHead}(e_1, e_2)$ and $\mathsf{TailTail}(e_1, e_2)$, then $e_1 = e_2$.

$E7$. For all e, $\neg\mathsf{HeadTail}(e, e)$.

The purpose of axioms $E1$–$E5$ are for consistency, that is, the structures in $\mathsf{STRUCT}[\mathbb{E}]$ are really graphs in the ordinary sense of graphs as structures in $\mathsf{STRUCT}[\mathbb{V}]$. (See Proposition 2 below.) Axiom $E6$ does not allow multiple edges, whereas Axiom $E7$ does not allow self-loop. Axioms $E6$ and $E7$ are not essential, but they will be useful for our convenience in the presentation.

As usual, $\mathsf{FO}[\mathbb{E}]$, $\mathsf{MSO}[\mathbb{E}]$ and $\exists\mathsf{MSO}[\mathbb{E}]$ denote the classes of first-order, monadic second-order and existential monadic second-order sentences in the logic $\mathbb{E}$. An $\mathbb{E}$-sentence is a sentence using the vocabulary $\mathbb{E}$.

We will usually write $\mathcal{E}$ to denote the elements in $\mathsf{STRUCT}[\mathbb{E}]$ and $\mathsf{Dom}(\mathcal{E})$ to denote the domain of $\mathcal{E}$. A sentence φ in $\mathbb{E}$-logic defines a set of $\mathbb{E}$-structures via

$$\mathcal{G}(\varphi) := \{\mathcal{E} \mid \mathcal{E} \models \varphi\}.$$

3 The equivalence between edge and vertex representations

In this section we will show that both the edge and the vertex representations essentially denote the same class of objects.

Definition 1. *Let $G \in \mathsf{STRUCT}[\mathbb{V}]$ and $\mathcal{G} \in \mathsf{STRUCT}[\mathbb{E}]$. We say that G and $\mathcal{E}$ are* structurally equivalent *if there exists a 1-1 mapping $\xi : E(G) \to \mathsf{Dom}(\mathcal{E})$ such that for all $(v_1, v_2), (v_2, v_3), (v_1, v_3) \in E(G)$ and $e_1, e_2 \in \mathsf{Dom}(\mathcal{E})$,*

1. *$\xi(v_1, v_2) = e_1$ and $\xi(v_2, v_3) = e_2$ if and only if $\mathsf{HeadTail}(e_1, e_2)$;*
2. *$\xi(v_1, v_2) = e_1$ and $\xi(v_1, v_3) = e_2$ if and only if $\mathsf{TailTail}(e_1, e_2)$; and*
3. *$\xi(v_1, v_3) = e_1$ and $\xi(v_2, v_3) = e_2$ if and only if $\mathsf{HeadHead}(e_1, e_2)$.*

The 1-1 mapping ξ is called a $(\mathbb{V}, \mathbb{E})$-isomorphism.

In other words, if G and $\mathcal{E}$ are structurally equivalent, then they essentially denote the same underlying directed graph. The following proposition states that this notion is robust.

Proposition 1.

(a) *Let G be a $\mathbb{V}$-structure and $\mathcal{E}_1$, $\mathcal{E}_2$ be $\mathbb{E}$-structures. If both $\mathcal{E}_1$ and $\mathcal{E}_2$ are structurally equivalent to G, then $\mathcal{E}_1$ and $\mathcal{E}_2$ are isomorphic.*

(b) *Let G_1, G_2 be $\mathbb{V}$-structures and $\mathcal{E}$ be a $\mathbb{E}$-structure. If both G_1 and G_2 are equivalent to $\mathcal{E}$, then G_1 and G_2 are isomorphic.*

Moreover, the following proposition shows that both edge and vertex representations are equivalent, in the sense that each graph stored using the standard vertex representation can be coded as a graph under the edge representation, and vice versa.

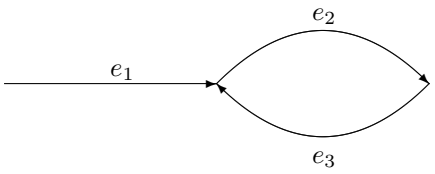
Proposition 2.
1. *For every $\mathbb{V}$-structure G, there exists a unique (up to isomorphism) $\mathbb{E}$-structure $\mathcal{E}$ which is structurally equivalent to G.*
2. *For every $\mathbb{E}$-structure $\mathcal{E}$, there exists a unique (up to isomorphism) $\mathbb{V}$-structure G structurally equivalent to $\mathcal{E}$.*

We do not state the full proof, but rather give an example of how the edge to vertex translation works.

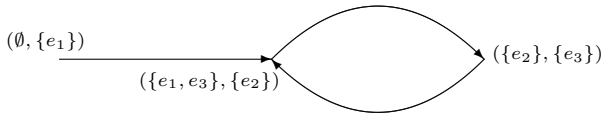
Example 1. Let $\mathcal{E}$ be an $\mathbb{E}$-structure, where

- $\mathsf{Dom}(\mathcal{E}) = \{e_1, e_2, e_3\}$;
- $\mathsf{HeadHead} = \{(e_1, e_1), (e_2, e_2), (e_3, e_3), (e_1, e_3), (e_3, e_1)\}$;
- $\mathsf{TailTail} = \{(e_1, e_1), (e_2, e_2), (e_3, e_3)\}$;
- $\mathsf{HeadTail} = \{(e_1, e_2), (e_2, e_3), (e_3, e_2)\}$.

The following picture well illustrates the structure of $\mathcal{E}$:



We can get a $\mathbb{V}$-structure $G = (V(G), E(G))$ equivalent to $\mathcal{E}$ as follows. Let $\mathcal{H}$ be the equivalent classes of $\mathsf{HeadHead}$ and $\mathcal{T}$ the equivalent classes of $\mathsf{TailTail}$, i.e. $\mathcal{H} = \{\{e_1, e_3\}, \{e_2\}\}$ and $\mathcal{T} = \{\{e_1\}, \{e_2\}, \{e_3\}\}$. Then we define $G = (V(G), E(G))$ as follows. The set of vertices is $V(G) = \mathcal{H} \times \mathcal{T}$, and $((H_1, T_1), (H_2, T_2)) \in E(G)$ if and only if $T_1 \cap H_2 \neq \emptyset$. It is depicted as follows.



4 Vertex and edge representations and their logics

In this section we will study the relation between the expressive power of logics using vertex or edge vocabularies. We need the following definition. For a set $\mathcal{A} \subseteq \mathsf{STRUCT}[\mathbb{V}]$, we define $\mathsf{Equiv}^{\mathbb{E}}(\mathcal{A})$ as the set of $\mathbb{E}$-structures which are equivalent to the structures in $\mathcal{A}$. Formally,

$$\mathsf{Equiv}^{\mathbb{E}}(\mathcal{A}) = \{\mathcal{E} \in \mathsf{STRUCT}[\mathbb{E}] \mid \mathcal{E} \text{ is structurally equivalent to some } G \in \mathcal{A}\}.$$

Vice versa, for a set $\mathcal{B} \subseteq \mathsf{STRUCT}[\mathbb{E}]$, we define

$$\mathsf{Equiv}^{\mathbb{V}}(\mathcal{B}) = \{G \in \mathsf{STRUCT}[\mathbb{V}] \mid G \text{ is structurally equivalent to some } \mathcal{E} \in \mathcal{B}\}.$$

By Proposition 1, it is immediate that for every sets $\mathcal{A} \subseteq \mathsf{STRUCT}[\mathbb{V}]$ and $\mathcal{B} \subseteq \mathsf{STRUCT}[\mathbb{E}]$,

$$\mathcal{A} = \mathsf{Equiv}^{\mathbb{V}}(\mathsf{Equiv}^{\mathbb{E}}(\mathcal{A})) \text{ and } \mathcal{B} = \mathsf{Equiv}^{\mathbb{E}}(\mathsf{Equiv}^{\mathbb{V}}(\mathcal{B}))$$

From this we immediately get that $\mathcal{A} = \mathsf{Equiv}^{\mathbb{V}}(\mathcal{B})$ if and only if $\mathcal{B} = \mathsf{Equiv}^{\mathbb{E}}(\mathcal{A})$.

Now we introduce the notion of $(\mathbb{V}, \mathbb{E})$-*equivalent*, the logical version of Definition 1.

Definition 2. *A $\mathbb{V}$-sentence φ and an $\mathbb{E}$-sentence ψ are* $(\mathbb{V}, \mathbb{E})$-equivalent *if $\mathcal{G}(\varphi) = \mathsf{Equiv}^{\mathbb{V}}(\mathcal{G}(\psi))$, or equivalently, $\mathcal{G}(\psi) = \mathsf{Equiv}^{\mathbb{E}}(\mathcal{G}(\varphi))$.*

Using the notion of $(\mathbb{V}, \mathbb{E})$-*equivalent*, we can now compare the expressive power between vertex and edge representations. Our first proposition shows that the edge representation is as least as expressive as the vertex representation:

Proposition 3. *Let $\mathcal{L}$ in $\{\mathsf{FO}, \exists\mathsf{MSO}, \mathsf{MSO}\}$. Then, for every sentence $\varphi \in \mathcal{L}[\mathbb{V}]$, there exists a sentence $\psi \in \mathcal{L}[\mathbb{E}]$ such that φ and ψ are $(\mathbb{V}, \mathbb{E})$-equivalent*

The proof is pretty straightforward, thus omitted.

The natural question is whether the converse holds, that is, whether for every sentence using the edge representation we can find an equivalent sentence using the vertex representation. As we show below, it turns out that this is not true even for $\exists\mathsf{MSO}$ sentences, nor if the full power of MSO is allowed.

Theorem 1. *1. There exists a sentence $\psi \in \exists\mathsf{MSO}[\mathbb{E}]$ such that for all sentence $\varphi \in \exists\mathsf{MSO}[\mathbb{V}]$, ψ and φ are not $(\mathbb{V}, \mathbb{E})$-equivalent.*

2. There exists a sentence $\psi \in \mathsf{MSO}[\mathbb{E}]$ such that for all sentence $\varphi \in \mathsf{MSO}[\mathbb{V}]$, ψ and φ are not $(\mathbb{V}, \mathbb{E})$-equivalent.

Proof. We begin with the $\exists\mathsf{MSO}$ case. The idea is to use the fact that (s, t)-reachability in directed graph is not expressible in $\exists\mathsf{MSO}[\mathbb{V}]$ (see, for example, [13, Theorem 7.16]).

For this we need to add two constants s and t to both $\mathbb{V}$- and $\mathbb{E}$-vocabularies, denoting the source and target vertices respectively. The interpretation of the constants s and t in $\mathbb{V}$-structures are the source and the target vertices, while their interpretation in $\mathbb{E}$-structures are two edges: one whose tail is the source vertex, and the other whose head is the target vertex.

We define the following class of $\mathbb{V}$-structures consists of directed graphs in which there is a path from s to t.

$$\mathcal{R}_{\mathbb{V}} = \left\{ G \in \mathsf{STRUCT}[\mathbb{V}] \;\middle|\; \begin{array}{l} \text{there are } v_1, \ldots, v_k \text{ s.t. } v_1 = s \text{ and } v_k = t \text{ and} \\ \text{for each } i = 1, \ldots, k-1, (v_i, v_{i+1}) \in E(G) \end{array} \right\}$$

It can be readily seen that the class $\mathsf{Equiv}^{\mathbb{E}}(\mathcal{R}_{\mathbb{V}})$ is expressible in $\exists\mathsf{MSO}[\mathbb{E}]$ in the following sentence. There exists a set P such that

- there is an edge y in P such that $\mathsf{TailTail}(y, s)$ holds;
- there is an edge y in P such that $\mathsf{HeadHead}(y, t)$ holds;
- for every edge y in P where $\neg\mathsf{HeadHead}(y, t)$, there is an edge z in P such that $\mathsf{HeadTail}(y, z)$ holds.

This immediately implies that $\exists\mathsf{MSO}[\mathbb{E}]$ is strictly more expressive than $\exists\mathsf{MSO}[\mathbb{V}]$. This proves the first case of the theorem.

The proof for the second case goes along the same lines, this time using the fact that directed graph hamiltonicity (i.e., whether a graph is hamiltonian) is not expressible in $\mathsf{MSO}[\mathbb{V}]$ (see, for example, [13, Corollary 7.24]). On the other hand, directed graph hamiltonicity can be expressed in the following $\mathsf{MSO}[\mathbb{E}]$ sentence. There exists a set U such that

- every two edges in U are connected (can be expressed as in the proof above); and
- for every edge x, x is adjacent to some edge y in U (either $\mathsf{HeadHead}(x, y)$, $\mathsf{TailTail}(x, y)$, or $\mathsf{HeadTail}(x, y)$ holds);

□

Next, we compare the two representations for the case of first-order logic. It turns out that the edge and vertex representations are equivalent if one disallows second-order quantification. Moreover, we also show that this transformation involves only a slight increase in quantifier rank.

Proposition 4. *For every sentence $\psi \in \mathsf{FO}[\mathbb{E}]$, there exists a sentence $\varphi \in \mathsf{FO}[\mathbb{V}]$ such that they are $(\mathbb{V}, \mathbb{E})$-equivalent and $\mathsf{qr}(\varphi) = 2\mathsf{qr}(\psi)$.*

With respect to the vertex to edge transformation, the following is immediate from the proof of proposition 3

Corollary 1. *For every sentence $\varphi \in \mathsf{FO}[\mathbb{V}]$, there exists a sentence $\psi \in \mathsf{FO}[\mathbb{E}]$ such that φ and ψ are $(\mathbb{V}, \mathbb{E})$-equivalent and $\mathsf{qr}(\psi) = \mathsf{qr}(\varphi) + 1$.*

5 Graph pebble automata

In this section we define pebble automata for directed graphs. It is based on the idea of pebble automata (PA) for words over infinite alphabet [16]. Let $\mathfrak{D}$ be a set of infinite symbols. We assume that the nodes in the directed graphs always come from $\mathfrak{D}$.

Briefly the way graph PA with k pebbles works as follows. If G is a directed graph, and $(a_1, b_1), \ldots, (a_n, b_n)$ are the edges in $E(G)$, then we feed a sequence $w = \binom{a_1}{b_1} \cdots \binom{a_n}{b_n}$ into graph k-PA. The pebbles are numbered from 1 to k. The automaton starts the computation with only pebble k on the sequence w. The pebbles are placed on/lifted from w in the stack discipline according to the strict order of the pebbles: Pebble i can be placed only when pebbles $i + 1, \ldots, k$ are above the sequence w. Each pebble is intended to mark one position in w and the smallest numbered pebble on w, or, equivalently the most recently placed pebble, serves as the head of the automaton. The automaton moves from one state to another depending on whether the edges read by the pebbles satisfy the $\mathsf{HeadHead}$, $\mathsf{TailTail}$, $\mathsf{HeadTail}$ relations.

Definition 3. *A* two-way alternating graph k-pebble automaton*, (in short graph k-PA) is a system $\mathcal{A} = \langle Q, q_0, F, \mu \rangle$, where*

- *Q, $q_0 \in Q$, $U \subseteq Q$ and $F \subseteq Q$ are a finite set of* states*, the* initial state*, the set of* universal states *and the set of* final states*, respectively; and*
- *μ is a finite set of transitions of the form $\alpha \to \beta$ such that*

- α *is of the form*

$$(i, P, V_{00}, V_{10}, V_{01}, V_{11}, q)$$

, where $i \in \{1, \ldots, k\}$*,* $P, V_{00}, V_{10}, V_{01}, V_{11} \subseteq \{i+1, \ldots, k\}$*, and*
- β *is of the form* $(q, \texttt{act})$*, where* $q \in Q$ *and*

$$\texttt{act} \in \{\texttt{left}, \texttt{right}, \texttt{place-pebble}, \texttt{lift-pebble}\}.$$

Given a sequence of edges $w = \binom{a_1}{b_1} \cdots \binom{a_n}{b_n}$, a *configuration of* $\mathcal{A}$ *on* $\triangleleft w \triangleright$ is a triple $[i, q, \theta]$, where $i \in \{1, \ldots, k\}$, $q \in Q$ and $\theta : \{i, i+1, \ldots, k\} \to \{0, 1, \ldots, n, n+1\}$. The function θ defines the position of the pebbles and is called the *pebble assignment*. The symbols in the positions 0 and $n+1$ are $\triangleleft$ and $\triangleright$, respectively.

The *initial* configuration is $\gamma_0 = [k, q_0, \theta_0]$, where $\theta_0(k) = 0$ is the *initial* pebble assignment. A configuration $[i, q, \theta]$ with $q \in F$ is called an *accepting* configuration.

A transition $(i, P, V_{00}, V_{01}, V_{10}, V_{11}, p) \to \beta$ *applies to a configuration* $[j, q, \theta]$, if

(1) $i = j$ and $p = q$,
(2) $P = \{l > i : \theta(l) = \theta(i)\}$,
(3.a) $V_{00} = \{l > i : a_{\theta(l)} = a_{\theta(i)}\}$,
(3.b) $V_{10} = \{l > i : b_{\theta(l)} = a_{\theta(i)}\}$,
(3.c) $V_{10} = \{l > i : a_{\theta(l)} = b_{\theta(i)}\}$, and
(3.d) $V_{11} = \{l > i : b_{\theta(l)} = b_{\theta(i)}\}$.

A transition $(i, P, V_{00}, V_{01}, V_{10}, V_{11}, p) \to \beta$ *applies to a configuration* $[j, q, \theta]$, if conditions (1)–(3) above hold.

We define the transition relation $\vdash_{\mathcal{A}}$ as follows: $[i, q, \theta] \vdash_{\mathcal{A}} [i', q', \theta']$, if there is a transition $\alpha \to (p, \texttt{act}) \in \mu$ that applies to $[i, q, \theta]$ such that $q' = p$, for all $j > i$, $\theta'(j) = \theta(j)$, and

- if $\texttt{act} = \texttt{left}$, then $i' = i$ and $\theta'(i) = \theta(i) - 1$,
- if $\texttt{act} = \texttt{right}$, then $i' = i$ and $\theta'(i) = \theta(i) + 1$,
- if $\texttt{act} = \texttt{lift-pebble}$, then $i' = i + 1$,
- if $\texttt{act} = \texttt{place-pebble}$, then $i' = i - 1$, $\theta'(i-1) = 0$ and $\theta'(i) = \theta(i)$.

As usual, we denote the reflexive, transitive closure of $\vdash_{\mathcal{A}}$ by $\vdash^*_{\mathcal{A}}$.

The acceptance criteria is based on the notion of *leads to acceptance* below. For every configuration $\gamma = [i, q, \theta]$,

- if $q \in F$, then γ leads to acceptance;
- if $q \in U$, then γ leads to acceptance if and only if for all configurations γ' such that $\gamma \vdash \gamma'$, γ' leads to acceptance;
- if $q \notin F \cup U$, then γ leads to acceptance if and only if there is at least one configuration γ' such that $\gamma \vdash \gamma'$, and γ' leads to acceptance.

A sequence of edges $\binom{a_1}{b_1} \cdots \binom{a_n}{b_n}$ is accepted by $\mathcal{A}$, if the initial configuration γ_0 leads to acceptance. The language $L(\mathcal{A})$ consists of all sequence of edges accepted by $\mathcal{A}$. Obviously, the sequence w induces a set of directed edges G_w as explain in the beginning of this section.

We have presented here the notion of *alternating* graph PA, since it is easier to work with for our purposes. However, it is not difficult to define instead the notion of *deterministic* graph PA. The next theorem shows that this choice is without loss of generality, as both models are equivalent.

Theorem 2. *1. For each $k \geq 1$, two-way non-deterministic graph k-PA and one-way deterministic graph k-PA have the same recognition power.*
2. For each $k \geq 1$, graph k-PA languages are closed under boolean operation.

Next, we introduce the relationship between graph PA and First Order logic.

Theorem 3. *For every FO $\mathbb{E}$-sentence ψ, there exists a graph k-PA $\mathcal{A}_\psi$ such that $k = \mathsf{qr}(\psi)$ and $L(\mathcal{A}) = \mathcal{G}(\psi))$.*

Proof. The proof is an adaptation of similar result in [19]. First, by Theorem 2, PA_k is closed under boolean operations. Let $\varphi = Qx_k\psi(x_k)$ where $Q \in \{\forall, \exists\}$ and $\psi(x_k)$ is of quantifier rank $k-1$.

The proof is by straightforward induction on k. A k-PA $\mathcal{A}$ iterates pebble k through all possible positions in the input. On each iteration, the automaton $\mathcal{A}$ recursively calls a $(k-1)$-PA $\mathcal{A}'$ that accepts the language $L(\psi(x_k))$, treating the position of pebble k as the assignment value for x_k.

- If $Q = \forall$, then $\mathcal{A}$ accepts w if and only if $\mathcal{A}'$ accepts on all iterations.
- If $Q = \exists$, then $\mathcal{A}$ accepts w if and only if $\mathcal{A}'$ accepts on at least one iteration.

□

Notice that Theorem 3 is optimal in the sense that all k pebbles are needed. More precisely, it is possible to adapt the proof of [19] to show that for every $k \geq 2$ there exists an FO $\mathbb{E}$-sentence ψ, with $k = \mathsf{qr}(\psi)$, and such that $L(\mathcal{A}) \neq \mathcal{G}(\psi))$ for every graph PA $\mathcal{A}$ using less than k pebbles.

6 When the edges are labeled with symbols from finite alphabet

In the usual graph databases setting the edges are labeled with symbols from a fixed finite alphabet. Each symbol can be viewed as a unary predicate on the edges.

In this section we extend the vocabularies $\mathbb{V}$ and $\mathbb{E}$ with unary predicates on the edges, which we called *extended* $\mathbb{V}$ and $\mathbb{E}$ vocabularies. We also extend the definition of graph pebble automata for edges labeled with symbols from a fixed alphabet.

In the following we let Σ be a fixed finite alphabet.

Extended $\mathbb{V}$ logic. The vocabulary for the extended $\mathbb{V}$ logic consists of σ for each $\sigma \in \Sigma$, where each σ is a binary predicate on the domain. We denote by $\mathbb{V}^*$ the extended $\mathbb{V}$ logic.

An *extended $\mathbb{V}$-structure* is a tuple $G = (V, \{\sigma\}_{\sigma\in\Sigma})$ such that V is the domain of nodes and the sets $\{\sigma\}_{\sigma\in\Sigma}$ are disjoints. Intuitively, each relation σ denotes the set of edges which are labeled with the symbol $\sigma \in \Sigma$. Since no edge can be labeled with two different symbols, the sets $\{\sigma\}_{\sigma\in\Sigma}$ are disjoint.

Extended $\mathbb{E}$ logic. The vocabulary consists of $\mathsf{HeadHead}, \mathsf{HeadTail}, \mathsf{TailTail}, \{\sigma\}_{\sigma\in\Sigma}$, where each $\sigma \in \Sigma$ is unary predicate on the domain. We denote by $\mathbb{E}^*$ the extended $\mathbb{E}$ logic.

An *extended $\mathbb{E}$-structure* is a tuple $\mathcal{E} = (U, \mathsf{HeadHead}, \mathsf{HeadTail}, \mathsf{TailTail}, \{\sigma\}_{\sigma\in\Sigma})$, where U is the domain of edges, the relations $\mathsf{HeadHead}, \mathsf{HeadTail}, \mathsf{TailTail}$ on U are defined as before, and each $\sigma \in \Sigma$ is a unary predicate on U.

It is straightforward to show that all results on the vocabularies $\mathbb{V}$ and $\mathbb{E}$ still hold for the extended logics $\mathbb{V}^*$ and $\mathbb{E}^*$. In the following we will elaborate this point more precisely.

Definition 4. *Let G be an $\mathbb{V}^*$ structure and $\mathcal{E}$ an $\mathbb{E}^*$ structure. We say that G and $\mathcal{E}$ are* structurally equivalent *if there exists a 1-1 mapping $\xi : E(G) \to \mathsf{Dom}(\mathcal{E})$ such that for all $(v_1, v_2), (v_2, v_3), (v_1, v_3) \in \bigcup_{\sigma\in\Sigma} \sigma$ and $e_1, e_2 \in \mathsf{Dom}(\mathcal{E})$,*

1. *for each $\sigma \in \Sigma$, $(v_1, v_2) \in \sigma$ if and only if $\xi(v_1, v_2) \in \sigma$;*
2. *$\xi(v_1, v_2) = e_1$ and $\xi(v_2, v_3) = e_2$ if and only if $\mathsf{HeadTail}(e_1, e_2)$;*
3. *$\xi(v_1, v_2) = e_1$ and $\xi(v_1, v_3) = e_2$ if and only if $\mathsf{TailTail}(e_1, e_2)$; and*
4. *$\xi(v_1, v_3) = e_1$ and $\xi(v_2, v_3) = e_2$ if and only if $\mathsf{HeadHead}(e_1, e_2)$.*

The 1-1 mapping ξ is called a $(\mathbb{V}^*, \mathbb{E}^*)$-isomorphism.

Theorem 4. *1. Let $\mathcal{L}$ in $\{\mathsf{FO}, \exists\mathsf{MSO}, \mathsf{MSO}\}$. Then, for every sentence $\varphi \in \mathcal{L}[\mathbb{V}^*]$, there exists a sentence $\psi \in \mathcal{L}[\mathbb{E}^*]$ such that φ and ψ are $(\mathbb{V}^*, \mathbb{E}^*)$-equivalent*

2. *There exists a sentence $\psi \in \exists\mathsf{MSO}[\mathbb{E}^*]$ such that for all sentence $\varphi \in \exists\mathsf{MSO}[\mathbb{V}^*]$, ψ and φ are not $(\mathbb{V}^*, \mathbb{E}^*)$-equivalent.*
3. *There exists a sentence $\psi \in \mathsf{MSO}[\mathbb{E}^*]$ such that for all sentence $\varphi \in \mathsf{MSO}[\mathbb{V}^*]$, ψ and φ are not $(\mathbb{V}^*, \mathbb{E}^*)$-equivalent.*

Next we define a graph pebble automata with unary predicates on the edges. It is also pretty much straightforward extension of Definition 3. In this case the input is of the form: $\begin{pmatrix}\sigma_1\\a_1\\b_1\end{pmatrix}\cdots\begin{pmatrix}\sigma_n\\a_n\\b_n\end{pmatrix} \in \Sigma \times \mathfrak{D} \times \mathfrak{D}$, where $\sigma_i \in \Sigma$ is the label of the edge (a_i, b_i).

The transitions are of the form: $(i, \sigma, P, V_{00}, V_{10}, V_{01}, V_{11}, p) \to (q, \mathtt{act})$. It is straightforward to show that all the results in the previous section can be adapted for such graph pebble automata. More precisely,

Theorem 5. *1. For PA with unary predicates, for each $k \geq 1$, two-way non-deterministic graph k-PA and one-way deterministic graph k-PA have the same recognition power.*

2. *For each $k \geq 1$, graph k-PA (with unary predicates) languages are closed under boolean operation.*
3. *For every FO $\mathbb{E}^*$-sentence ψ, there exists a graph k-PA $\mathcal{A}_\psi$ with unary predicates such that $k = \mathsf{qr}(\psi)$ and $L(\mathcal{A}) = \mathcal{G}(\psi)$).*

7 Future directions

We would like to apply our logics and graph pebble automata in a more application oriented settings. Also, it is well known that the emptiness problem for graph pebble automata is undecidable. One direction that we would like to pursue is to charaterize a subclass of pebble automata, for which the emptiness problem is decidable. We also would like to define and study similar logics for matroid and extend the graph pebble automata for abstract matroid.

Acknowledgments: We thank the anonymous referees for many helpful comments. Partial support provided by EPSRC grant G049165 and FET-Open Project FoX, grant agreement 233599.

References

1. S. Abiteboul, P. Buneman, D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kauffman, 1999.
2. R. Angles, C. Gutiérrez. Survey of graph database models. *ACM Comput. Surv.* 40(1): (2008).
3. P. Barceló, C. Hurtado, L. Libkin, P. Wood. Expressive languages for path queries over graph-structured data. In *PODS* 2010.
4. D. Berwanger, D. Janin. Automata on Directed Graphs: Edge Versus Vertex Marking. In *ICGT* 2006.
5. B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, 1997.
6. I. Cruz, A. Mendelzon, P. Wood. A graphical query language supporting recursion. In *SIGMOD* 1987.
7. R. Fagin, L. J. Stockmeyer, and M. Y. Vardi. On monadic NP vs. monadic co-NP. *Info. and Comp.*, 120(1):78–92, 1995.
8. N. Globerman and D. Harel. Complexity results for multi-pebble automata and their logics. In *ICALP* 1994.
9. C. Gutierrez, C. Hurtado, A. Mendelzon. Foundations of semantic web databases. In *PODS* 2004.
10. P. Hliněný. Branch-width, parse trees, and monadic second-order logic for matroids. *J. Comb. Theory*, Ser. B 96(3): 325–351 (2006)
11. R. Ladner, R. Lipton and L. Stockmeyer. Alternating Pushdown and Stack Automata. *SIAM Journal of Comp.* 13(1): 135–155, 1984.
12. U. Leser. A query language for biological networks. *Bioinformatics* 21 (suppl 2) (2005), ii33–ii39.
13. L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.
14. A. O. Mendelzon, P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24(6):1235–1258, 1995.
15. R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, U. Alon. Network motifs: simple building blocks of complex networks. *Science* 298(5594) (2002), 824–827.
16. F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM ToCL*, 5(3):403–435, 2004.
17. R. Ronen and O. Shmueli. SoQL: a language for querying and creating data in social networks. In *ICDE* 2009.
18. T. Schwentick. On Winning Ehrenfeucht Games and Monadic NP. *Ann. Pure Appl. Logic*, 79(1), 61–92, 1996.
19. T. Tan. Graph reachability and pebble automata over infinite alphabets. In *LICS* 2009.
20. G. Turán. On the definability of properties of finite graphs. *Discrete Mathematics*, 49(3):291–302, 1984.