Logical Characterization of Weighted Pebble Walking Automata

Benedikt Bollig[1], Paul Gastin[1], Benjamin Monmege[2] and Marc Zeitoun*[3]

[1]LSV, ENS Cachan, CNRS, Inria, Cachan, France, bollig,gastin@lsv.ens-cachan.fr
[2]Université Libre de Bruxelles, Belgium, mailto:benjamin.monmege@ulb.ac.be
[3]Bordeaux University, France, mz@labri.fr

May 18, 2014

Abstract

Weighted automata are a conservative quantitative extension of finite automata that enjoys applications, e.g., in language processing and speech recognition. Their expressive power, however, appears to be limited, especially when they are applied to more general structures than words, such as graphs. To address this drawback, weighted automata have recently been generalized to weighted pebble walking automata, which proved useful as a tool for the specification and evaluation of quantitative properties over words and nested words. In this paper, we establish the expressive power of weighted pebble walking automata in terms of transitive closure logic, lifting a similar result by Engelfriet and Hoogeboom from the Boolean case to a quantitative setting. This result applies to general classes of graphs, including all the aforementioned classes.

1 Introduction

Automata are a universal tool for a variety of computational tasks such as parsing, verification, evaluation, etc. To that effect, most automata models restrict the power and resources of a Turing machine according to their respective field of application. At the same time, automata may process mathematical structures of various kinds, including words, trees, and, more generally, labeled graphs.

However, as soon as we leave the well-studied and robust territory of finite automata over words, we are quickly treading on a fragile equilibrium between decidability, complexity, and closure properties. Moreover, the naturalness and robustness of an automata model and, finally, its success in applications seems intrinsically linked with algebraic and logical characterizations. For example, the Büchi-Elgot-Trakhtenbrot theorem [5, 11, 25], stating that the regular word languages are captured by monadic second-order logic, "explains", to a certain extent, the success of finite automata and its numerous fields of application. In fact, many prominent and important automata models enjoy a logical characterization (cf. [10]).

A by now classical result by Engelfriet and Hoogeboom states that Boolean (nondeterministic) pebble graph-walking automata have the same expressive power as first-order logic with positive transitive closure [12, 13]. We actually aim at an extension of that result:

*Supported by ANR 2010 BLAN 0202 01 FREC

It is the goal of this paper to lift the Engelfriet-Hoogeboom Theorem to a quantitative *setting.*

Quantitative reasoning has indeed become a necessity in various applications, and it has arrived in classical fields such as database theory and formal verification. This naturally asks for quantitative extensions of automata models that allow for the transfer of tools and methods from the Boolean framework to more fine-grained analyses. Weighted automata constitute an important extension of finite word automata [9]. However, while they have applications in quite diversified domains such as language processing and speech recognition, they appear to be limited when processing more general structures such as pictures. In fact, their natural counterpart in terms of weighted tiling systems is not able to compute 2^{n^2} for a given graph with n vertices.

Weighted pebble walking automata (wPWA) have been introduced to address the limited expressive power of weighted automata. They have proved useful for the specification and evaluation of quantitative properties over words [4, 17], nested words [3], and, more generally, graphs [20]. Like their Boolean counterpart, wPWA traverse a graph along its edges. Thus, they follow a more "algorithmic" paradigm than classical weighted automata and tiling systems. A major role is played by the pebbles, which equip automata with a means to find their way around a graph. While they already increase expressiveness in the Boolean setting, they are essential for the specification of *quantitative* properties: they can compute the aforementioned 2^{n^2}. Note that, however, wPWA and tiling systems are in general incomparable (cf. Section 3).

In this paper, we establish the expressive power of *weighted* pebble walking automata, in the spirit of the result by Engelfriet and Hoogeboom:

Weighted pebble (graph-)walking automata are expressively equivalent to weighted first-order logic with (positive) transitive closure.

In the proof, we (have to) use quite different techniques than those developed in [12, 13]. Like in [13], the effective[1] translation from logic to automata requires that the class of graphs at hand be *searchable*, i.e., there is a deterministic way of visiting each node, independently of the graph. Note that this captures natural classes like words, trees, nested words, and pictures. The translation of automata into logic, on the other hand, is more subtle in the quantitative setting. Actually, our proof requires to first check a property of the class of graphs, which we call *zonable*. Then, on zonable classes of graphs, we show a generic exponential translation of automata into logic. Interestingly, this new notion unifies (in a general setting of graphs) techniques used by Thomas [23] and by Matz [19] who show that one existential second-order quantifier is enough in *Boolean* EMSO logic over words and, respectively, pictures.

Outline of the paper In Section 2, we fix some notational conventions and recall standard notions such as graphs, (Boolean) walking automata, and first-order logic. Section 3 naturally extends walking automata with weights, allowing for a quantitative evaluation of graphs. Section IV introduces the logic. Expressive equivalence of automata and logic is established in Section 5 (from logic to automata) and Section 6 (from automata to logic). Missing proofs can be found in [20].

[1] Our construction can be done in *linear time* for the first-order fragment.

2 Preliminaries

Graph Structures Following [24, 13, 15], we consider various structures such as words, trees, pictures, etc. as graphs of bounded degree, with labeled vertices and edges. So, we fix a finite set A of *vertex labels*, and another finite set D of *edge labels* or directions.

Definition 1. An (A, D)-graph (or simply a graph, if A and D are clear from the context) is a tuple $G = (V, (E_d)_{d\in D}, \lambda, \iota)$ where (i) V is a nonempty and finite set of vertices; (ii) for all $d \in D$, $E_d \subseteq V \times V$ is an *irreflexive relation*, describing the d-edges of the graph, which is *deterministic and codeterministic*, i.e., for all $v \in V$, there exists at most one vertex $v' \in V$ such that $(v, v') \in E_d$ and at most one vertex $v'' \in V$ such that $(v'', v) \in E_d$; (iii) $\lambda\colon V \to A$ is a vertex-labeling function; (iv) $\iota \in V$ is an initial vertex. The set of all graphs with vertex labels A and directions D is denoted by $\mathcal{G}(A, D)$.

We let $E = \bigcup_{d\in D} E_d$ denote the set of *edges* of G. Moreover, $E^{-1} = \bigcup_{d\in D} E_d^{-1}$ is the set of *reverse edges*, where $E_d^{-1} = \{(v, v') \mid (v', v) \in E_d\}$. We will implicitly assume that a reverse edge from E_d^{-1} is labeled with the *dual* of d, which is denoted by d^{-1}. Accordingly, we set $E_{d^{-1}} = E_d^{-1}$. We assume that D and $D^{-1} = \{d^{-1} \mid d \in D\}$ are disjoint.

An (undirected) path in such a graph is a finite nonempty sequence $(v_k)_{0\leqslant k\leqslant n}$ of vertices such that successive vertices are linked with an edge, i.e., for every $0 \leqslant k < n$, $(v_k, v_{k+1}) \in E \cup E^{-1}$. Such a path is said to be a path of length n from v_0 to v_n. For a subset D' of $D \cup D^{-1}$, a D'-path is a path such that every edge is labeled by a direction in D': with the previous notation, $(v_k, v_{k+1}) \in \bigcup_{d\in D'} E_d$, for all $0 \leqslant k < n$. A $\{d\}$-path is also called a d-path for conciseness.

We shall only consider connected graphs, i.e., such that for all vertices v, v', there is a path from v to v'. We may therefore equip every graph $G = (V, (E_d)_{d\in D}, \lambda, \iota)$ with a distance function $\mathsf{dist}\colon V \times V \to \mathbb{N}$ with $\mathsf{dist}(v, v')$ being the length of a shortest path from v to v'.

Example 1. The versatile notion of graphs defined above allows us to consider various classical structures as subsets of $\mathcal{G}(A, D)$ for suitable choices of D:

Finite words over alphabet A may be described as a subset of $\mathcal{G}(A, \{\rightarrow\})$ where $E_\rightarrow$ is the direct successor relation of the linear order. Actually, $\mathcal{G}(A, \{\rightarrow\})$ consists of linear graphs encoding finite words and of cycles (which do not represent finite words). Other graphs are ruled out by our assumptions. We often let $\leftarrow = \rightarrow^{-1}$.

Binary trees may similarly be described as a subset of the class $\mathcal{G}(A, \{\downarrow_1, \downarrow_2\})$: here $E_{\downarrow_1}$ (resp., $E_{\downarrow_2}$) relates a vertex to its first child (resp., second child).

Nested words may be described as a subset of $\mathcal{G}(A, \{\rightarrow, \curvearrowright\})$ (cf. [1] for definitions of nested words): $\curvearrowright$-edges relate a call vertex to its matched return. Apart from their own interest, nested words can also be seen as an indirect way to handle unranked trees. Notice that $\mathcal{G}(A, \{\rightarrow, \curvearrowright\})$ also contains graphs which are not nested words. In particular, vertices may have both an outgoing $\curvearrowright$-edge and an ingoing $\curvearrowright$-edge.

Pictures (see [16], for instance) are 2-dimensional finite rectangular grids. For the sake of simplicity, we consider black-and-white pictures in this article. These can be represented as a subset (denoted $\mathfrak{G}_{\mathrm{pict}}$ in the following) of $\mathcal{G}(\{\mathsf{b}, \mathsf{w}\}, \{\rightarrow, \uparrow\})$ with vertex set $\{0, 1, \ldots, k\} \times \{0, 1, \ldots, \ell\}$ $(k, \ell \in \mathbb{N})$. Nodes are labeled by either b (black) or w (white), and edges by $\rightarrow$ or $\uparrow$. We let $\leftarrow = \rightarrow^{-1}$ and $\downarrow = \uparrow^{-1}$. We have a $\rightarrow$-edge (resp., a $\uparrow$-edge) from (i, j) to (i', j') if $i' = i + 1$ and $j' = j$ (resp., $i' = i$ and $j' = j + 1$). The initial vertex is $(0, 0)$.

Mazurkiewicz traces or *message sequence charts* may also be considered in a similar way. □

Walking Automata and Searchable Classes of Graphs We will later consider a class of weighted automata that are essentially walking automata enriched with pebbles and weights (cf. Section 3). We recall here the classical definition of walking automata.

Definition 2. A walking automaton is given by a tuple $\mathcal{A} = (Q, A, D, I, \Delta, F)$ where Q is a finite set of states, A is a finite alphabet, D is a finite set of directions, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, and Δ is a finite subset of $Q \times \text{Guards} \times A \times \text{Actions} \times Q$ where $\text{Actions} = D \cup D^{-1} \cup \{\mathsf{stay}\}$ is the set of actions and Guards is the set of guards α defined by

$$\alpha ::= \top \mid \text{init?} \mid \neg\text{init?} \mid d? \mid \neg d? \mid \alpha \wedge \alpha$$

with $d \in D \cup D^{-1}$.

Fix a graph $G = (V, (E_d)_{d\in D}, \lambda, \iota) \in \mathcal{G}(A, D)$ and a walking automaton $\mathcal{A} = (Q, A, D, I, \Delta, F)$. We denote $G, v \models \alpha$ the satisfaction of guard α by G at vertex $v \in V$: for atomic guards, $G, v \models d?$ holds if there exists $v' \in V$ with $(v, v') \in E_d$ and $G, v \models \text{init?}$ holds if $v = \iota$.

A *configuration* of $\mathcal{A}$ over G is a pair (q, v) with $q \in Q$ and $v \in V$. A *run* of $\mathcal{A}$ over G is a sequence

$$(q_0, v_0) \xrightarrow{\alpha_0,\gamma_0} (q_1, v_1) \xrightarrow{\alpha_1,\gamma_1} \cdots \xrightarrow{\alpha_{M-1},\gamma_{M-1}} (q_M, v_M)$$

where the (q_m, v_m) are configurations, $\alpha_m \in$ Guards, and $\gamma_m \in$ Actions, such that any two consecutive configurations are related by an actual transition of the automaton: formally, for all $0 \leqslant m < M$, $(q_m, \alpha_m, \lambda(v_m), \gamma_m, q_{m+1}) \in \Delta$ is a transition such that $G, v_m \models \alpha_m$, and either $\gamma_m \in D \cup D^{-1}$ and $(v_m, v_{m+1}) \in E_{\gamma_m}$, or $\gamma_m = \mathsf{stay}$ and $v_m = v_{m+1}$. Moreover, this run is said to be *from vertex* v_0. It is *accepting* if $q_0 \in I$ and $q_M \in F$. Finally, G is accepted by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ over G from vertex ι.

A walking automaton $\mathcal{A}$ is *deterministic* if (i) I is a singleton, and (ii) for every distinct transitions $(q, \alpha_1, a, d_1, q_1)$ and $(q, \alpha_2, a, d_2, q_2)$, the guard $\alpha_1 \wedge \alpha_2$ is unsatisfiable[2] over $\mathcal{G}(A, D)$. In this paper, we consider particular classes of graphs, namely *searchable classes of graphs*, inspired by [13].[3]

Definition 3. A class of graphs $\mathfrak{G} \subseteq \mathcal{G}(A, D)$ is *searchable* if there exists a deterministic walking automaton $\mathcal{A}_\mathfrak{G} = (Q, A, D, \{q_i\}, \Delta, \{q_o\})$ such that q_o has no outgoing transition and for every graph $G = (V, (E_d)_{d\in D}, \lambda, \iota) \in \mathfrak{G}$, there exists a linear order $\leqslant$ over V, with minimal element ι, such that for every vertex $v \in V$, $\mathcal{A}_\mathfrak{G}$ admits an accepting run over G from vertex v ending either at the direct successor of v for $\leqslant$, if this successor exists, or at ι if it does not exist. We call $\mathcal{A}_\mathfrak{G}$ a *guide* for the class $\mathfrak{G}$.

Clearly, the classes of finite words and nested words are searchable, by following the natural linear order of these graphs. An example of a guide for both classes is given in Fig. 1. Notice that, in graphical representations of automata, we omit guards of the form $\top$ and actions of the form stay. The class of pictures is also searchable: one possible linear order is to read each line of the picture from left to right, running through each line from bottom to top. Hence the automaton simply

[2]This can be effectively checked, since a guard α is unsatisfiable if, and only if, α contains in the list of its conjuncts both $d?$ and $\neg d?$ for a given direction $d \in D$, or both init? and $\neg$init?.

[3]Whereas in [13], *strong pebbles* are used to search graphs, we stick to walking automata that do not use pebbles. This is for the sake of simplicity of our paper, even though the definition and the results could easily be extended to walking automata with *weak pebbles*, as we define and use them in the weighted setting (see Section 3).

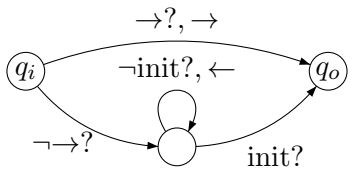


Figure 1: Guide for the searchable class of finite (nested) words

does a right move whenever it is possible; otherwise, it goes to the next line and comes back to its beginning. Binary trees are also searchable, witnessed by a guide that performs a depth-first-search [2]. Not all classes of graphs are searchable, however. For example, the class of planar graphs is known to be not searchable [21].

Boolean First-order Logic We recall the classical definition of first-order logic over graphs. Let us fix infinite supplies of first-order variables $\mathrm{Var} = \{x, y, z, t, x_1, x_2, \ldots\}$. The set $\mathrm{FO}(A, D)$ (or simply FO if A and D are clear from the context) of first-order formulae over A and D is given by the grammar:

$$\begin{aligned}\varphi ::= \top \mid (x = y) \mid \mathsf{init}(x) \mid P_a(x) \mid R_d(x, y) \mid R_d^+(x, y) \mid \\ \neg\varphi \mid \varphi \vee \varphi \mid \exists x\, \varphi\end{aligned}$$

where $a \in A$, $d \in D$ and $x, y \in \mathrm{Var}$.

For $\varphi \in \mathrm{FO}(A, D)$, the set of free variables of φ is denoted by $\mathrm{Free}(\varphi)$. If $\mathrm{Free}(\varphi) = \emptyset$, then φ is called a *sentence*.

Let $G = (V, (E_d)_{d\in D}, \lambda, \iota) \in \mathcal{G}(A, D)$ be a graph. A *valuation* for G is a partial function σ from Var to V. For $x \in \mathrm{Var}$ and $v \in V$, $\sigma[x \mapsto v]$ denotes the valuation that maps x to v and, otherwise, coincides with σ.

Suppose that the domain of σ contains $\mathrm{Free}(\varphi)$. As usual, the satisfaction relation $G, \sigma \models \varphi$ is defined by induction over the formula φ. In particular:

$$\begin{aligned}&G, \sigma \models \mathsf{init}(x) \text{ if } \sigma(x) = \iota \\ &G, \sigma \models R_d(x, y) \text{ if } (\sigma(x), \sigma(y)) \in E_d \\ &G, \sigma \models R_d^+(x, y) \text{ if there is a } d\text{-path of positive length} \\ &\qquad\qquad\qquad\qquad \text{from } \sigma(x) \text{ to } \sigma(y)\,.\end{aligned}$$

Note that the semantics of φ only depends on the restriction of σ to the free variables of φ.

In the following, we will use several macros in order to keep formulae readable and of reasonable size. For example, we let $R_d^\star(x, y)$ be a shorthand for $(x = y) \vee R_d^+(x, y)$.

Example 2. In pictures, we may express with a formula of FO that a vertex y is located at the North-East of x: $\exists t\; R_\rightarrow^+(x, t) \wedge R_\uparrow^+(t, y)$. We may also test that x and y are related by a $\rightarrow$-path containing only black vertices with the formula $R_\rightarrow^\star(x, y) \wedge \forall t\big(\neg(R_\rightarrow^\star(x, t) \wedge R_\rightarrow^\star(t, y)) \vee P_{\mathsf{b}}(t)\big)$. Similarly, we may then test that z is a vertex inside the rectangle defined by the bottom-left corner x and top-right corner y with a formula $\text{in-rect}(x, y, z)$. Finally, we may combine those formulae to express the fact that x and y define a white rectangle surrounded by a black frame with a formula $\text{bw-rect}(x, y)$.

Semirings A *semiring* is a set $\mathbf{S}$ equipped with two binary operations $+$ and $\times$, and two designated neutral elements 0 and 1 such that $(\mathbf{S}, +, 0)$ is a commutative monoid, $(\mathbf{S}, \times, 1)$ is a monoid, $\times$ distributes over $+$ and 0 is a zero for $\times$. In the following, semirings are represented as tuples $(\mathbf{S}, +, \times, 0, 1)$.

In this paper, we consider only *continuous* semirings: intuitively, these are semirings in which sums of infinite families are always well-defined, and such that these sums can be approximated by finite partial sums. Formally, a semiring $\mathbf{S}$ is *continuous* if every family $(s_i)_{i \in I}$ of elements of $\mathbf{S}$ over an arbitrary indexed set I is summable to some element in $\mathbf{S}$, denoted $\sum_{i \in I} s_i$ and called the sum of the family, such that the following conditions are satisfied: (i) $\sum_{i \in \emptyset} s_i = 0$, $\sum_{i \in \{j\}} s_i = s_j$ and $\sum_{i \in \{j,k\}} s_i = s_j + s_k$; (ii) if $I = \bigcup_{j \in J} I_j$ is a partition, then $\sum_{j \in J} \left(\sum_{i \in I_j} s_i \right) = \sum_{i \in I} s_i$; (iii) the relation $\leqslant$ defined over $\mathbf{S}$ by $s_1 \leqslant s_2$ if $s_2 = s_1 + s$ for some $s \in \mathbf{S}$, for every $s_1, s_2 \in \mathbf{S}$, is a partial order, called the *natural order* of the semiring; (iv) the sum $\sum_{i \in I} s_i$ is the least upper bound with respect to $\leqslant$ of the finite sums $\sum_{i \in J} s_i$ where J ranges over all finite subsets of I, for every family $(s_i)_{i \in I}$ in $\mathbf{S}$. For more background about semirings, see [9]. In the following, we fix a continuous semiring $(\mathbf{S}, +, \times, 0, 1)$.

Example 3. Examples of continuous semirings are the Boolean semiring $(\{0,1\}, \vee, \wedge, 0, 1)$, the positive real semiring $(\mathbb{R}^+ \cup \{+\infty\}, +, \times, 0, 1)$, the tropical semiring $(\mathbb{Z} \cup \{+\infty, -\infty\}, \min, +, +\infty, 0)$, the arctic semiring $(\mathbb{Z} \cup \{+\infty, -\infty\}, \max, +, -\infty, 0)$, the fuzzy semiring $([0,1], \min, \max, 1, 0)$, or the semiring $(2^{\Sigma^\star}, \cup, \cdot, \emptyset, \{\varepsilon\})$ of languages over a finite alphabet Σ.

3 Weighted Pebble Walking Automata

In this section, we extend the walking automata introduced in Section 2 with pebbles and weights. A pebble allows an automaton to mark some vertices in the graph so that they can later be retrieved. We will see in Example 4 how to use this feature to check whether some framed white square (as opposed to a white rectangle in Example 2) is contained in a picture. To cope with pebbles, guards on transitions are enriched with the ability to check which pebbles are dropped on the current vertex. Moreover, two new actions are provided in order to drop a new pebble on the current vertex or to lift the last dropped pebble and resume the computation at the position of the pebble. Notice that drop/lift actions follow a stack policy, meaning that there is a single stack containing positions where pebbles are currently placed in the graph: a drop consists in pushing on top of the stack a new pebble and its position, whereas a lift consists in popping the top of the stack.

We generalize the Boolean semantics of automata to a quantitative semantics by assigning weights to transitions. These weights are taken from a semiring $\mathbf{S}$. The weight of a run is the product of the weights of the transitions used along that run. Non-determinism is resolved with sum: the semantics of an automaton over a graph G is the sum of the weights of the accepting runs on G, starting from its initial vertex. Since walking automata may loop in a graph, there might be infinitely many accepting runs, which explains why we assume our semirings to be continuous.

Definition 4. A *weighted pebble walking automaton* (wPWA) over the semiring $\mathbf{S}$ is a tuple $\mathcal{A} = (Q, A, D, \mathrm{Peb}, I, F, K, \Delta, \mathrm{wt})$ where Q is a finite set of states, A is a finite alphabet, D is a finite set of directions, Peb is a finite set of pebble names, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, $K \geqslant 0$ is the number of pebbles that may be dropped on the graph at any time of the computation, Δ is a finite subset of $Q \times \mathrm{Guards} \times A \times \mathrm{Actions} \times Q$, and $\mathrm{wt} \colon \Delta \to \mathbf{S}$

gives the weights of transitions. The number K is called the *height* of $\mathcal{A}$. The size of $\mathcal{A}$, denoted by $|\mathcal{A}|$, is defined as $|Q| + K + |\Delta|$.

The set of actions used in Δ is given as $\text{Actions} = D \cup D^{-1} \cup \{\mathsf{stay}\} \cup \{\mathsf{drop}_x \mid x \in \text{Peb}\} \cup \{\mathsf{lift}\}$, and the set Guards is defined by

$$\alpha ::= \top \mid \text{init?} \mid \neg\text{init?} \mid d? \mid \neg d? \mid x? \mid \neg x? \mid \alpha \wedge \alpha$$

with $d \in D \cup D^{-1}$ and $x \in \text{Peb}$.

Through its navigation in the graph, the wPWA $\mathcal{A}$ may *drop* a pebble named x with the action drop_x and later *lift* a pebble with the action lift. Only the last dropped pebble may be lifted (stack policy). Hence, lift actions need not be labeled with a pebble name.

A guard may check with $x?$ whether a pebble named x is dropped on the current vertex (by looking for this information in the entire stack). A valuation over Peb is a mapping $\sigma\colon \text{Peb} \to V$, which describes an assignment of pebbles to vertices. Then, satisfaction of guards is denoted by $G, \sigma, v \models \alpha$. For the atomic guards, we have: $G, \sigma, v \models x?$ if $\sigma(x) = v$; $G, \sigma, v \models d?$ if there exists $v' \in V$ with $(v, v') \in E_d$; and $G, v \models \text{init?}$ if $v = \iota$.

A *configuration* of a wPWA $\mathcal{A}$ over a graph $G = (V, (E_d)_{d\in D}, \lambda, \iota) \in \mathcal{G}(A, D)$ and a valuation $\sigma\colon \text{Peb} \to V$ is a tuple (q, π, v) composed of a state q of Q, a stack $\pi \in (\text{Peb} \times V)^{\leqslant K}$ of length at most K containing pairs of pebble names and vertices (the top of the stack is on the right of the word π), and a vertex $v \in V$. The stack π records the positions of currently dropped pebbles.

Pebble names are *reusable*, meaning that several pebbles with the same name may be dropped on the graph at a given moment of the computation. Hence, a pebble name may occur several times in π, but only its topmost occurrence is visible. However, when this topmost occurrence will be lifted, the previous one will become visible again.[4] At any moment of the computation, each pebble name of Peb is mapped to a single vertex: either the one defined in the initial valuation σ, or the vertex holding the last pebble with this name dropped. Having this in mind, we associate with every stack π and valuation σ, an updated valuation $\sigma[\pi]\colon \text{Peb} \to V$ by $\sigma[\varepsilon] = \sigma$, and $\sigma[\pi(x, v)] = \sigma[\pi][x \mapsto v]$. Notice in particular that guard $x?$ tests whether the visible occurrence of pebble x is dropped on the current vertex.

A run is described as an alternating sequence of configurations and transitions. Formally, a *run* of $\mathcal{A}$ over a graph G and a valuation σ is a finite sequence

$$\rho = (q_0, \pi_0, v_0) \xrightarrow{\alpha_0,\gamma_0} (q_1, \pi_1, v_1) \xrightarrow{\alpha_1,\gamma_1} \cdots (q_M, \pi_M, v_M)$$

such that, for each $0 \leqslant m < M$, we have $G, \sigma[\pi_m], v_m \models \alpha_m$, the tuple $\delta_m = (q_m, \alpha_m, \lambda(v_m), \gamma_m, q_{m+1})$ is a transition from Δ, and one of the following holds:

- $\gamma_m \in D \cup D^{-1}$, $(v_m, v_{m+1}) \in E_{\gamma_m}$, and $\pi_{m+1} = \pi_m$;
- $\gamma_m = \mathsf{stay}$, $v_{m+1} = v_m$, and $\pi_{m+1} = \pi_m$;
- $\gamma_m = \mathsf{drop}_x$, $v_{m+1} = v_m$, and $\pi_{m+1} = \pi_m(x, v_m)$;
- $\gamma_m = \mathsf{lift}$ and $\pi_m = \pi_{m+1}(x, v_{m+1})$ for some $x \in \text{Peb}$.

[4] Notice that our notion of reusability is slightly different from the notion of invisibility introduced in [14].

The weight of ρ is the product of the weights of its transitions: $\mathrm{wt}(\rho) = \prod_{0 \leqslant m < M} \mathrm{wt}(\delta_m)$.

The third case, dealing with drop transitions, stays on the current vertex, and pushes on the stack the pair composed of the name x of the pebble and the vertex where it is dropped. Notice that this transition is enabled only if the size of the current stack is less than K. The fourth case, dealing with lift transitions, pops the top of the stack (which must be nonempty), and moves to the vertex v_{m+1} where the last pebble was dropped.[5]

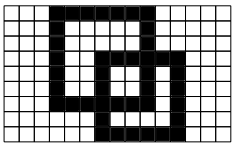
We are interested in runs that start at some vertex v, in state q with empty stack, and end in some vertex v', in state q' with empty stack: we let $[\![\mathcal{A}_{q,q'}]\!](G, \sigma, v, v')$ be the sum of the weights of all runs over (G, σ) from configuration (q, ε, v) to configuration (q', ε, v'). Since we consider continuous semirings, this possibly infinite sum is well-defined.

Finally, the semantics of $\mathcal{A}$ over a pair (G, σ), denoted by $[\![\mathcal{A}]\!](G, \sigma)$, maps every graph G and valuation σ to an element of the continuous semiring $\mathbf{S}$:

$$[\![\mathcal{A}]\!](G, \sigma) = \sum_{q \in I, q' \in F, v \in V} [\![\mathcal{A}_{q,q'}]\!](G, \sigma, \iota, v)\,.$$

In the following, the sum $\sum_{v \in V} [\![\mathcal{A}_{q,q'}]\!](G, \sigma, \iota, v)$ is denoted by $[\![\mathcal{A}_{q,q'}]\!](G, \sigma)$.

Example 4 (Pebble walking automaton on pictures)**.** A *frame* in a picture is a white square surrounded by rows and columns of black pixels. We aim at computing the biggest size of frames, where the size is defined as the white area inside the frame. The following picture has a single frame, which is of size 4.



Since we need to count and maximize, we opt for the continuous semiring $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$. We give in Fig. 2 a wPWA $\mathcal{A}$ computing this size. For readability, we do not write $\top$-guards, we omit the letter to mean that the transition is allowed for both `b` (black) and `w` (white), and we omit the weight if it is 0. We use two pebbles x and y in order to mark the lower-left and upper-right corners of a frame in the picture. At first, $\mathcal{A}$ non-deterministically chooses a position to drop x. It then follows a path in the picture that alternates $\rightarrow$- and $\uparrow$-edges, until non-deterministically dropping y. This ensures that the rectangle defined by x and y is actually a square.

Then, $\mathcal{A}$ scans each vertex of the square defined by (x, y), checking that the inside is white and the border is black. It also computes 1 for each inner vertex. This is done by starting from y, going to the left, until non-deterministically stopping to check that the current vertex is above x. We then lift y (going back to it, as defined in the semantics), and drop y again on the pixel just below. We continue this process until we check the last line of the zone. We finally accept after lifting both pebbles. Each accepting run computes the size of the chosen zone, by summing (since the product of the semiring is integer addition) 1 for every position inside this rectangle. The non-determinism is resolved by taking the maximum of the weights of all the accepting runs, which proves that

[5] We are using *weak* pebbles since a lift transition leads the automaton to the vertex where the lifted pebble was. An alternative semantics, called *strong*, allows the automaton to lift a pebble while staying on the current vertex, see e.g., [2]. In the Boolean setting, strong pebbles are shown to have the same expressive power as weak pebbles [2]. We do not know whether this is the case in our weighted setting.

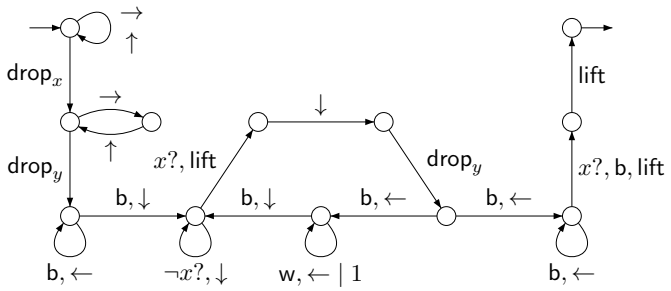


Figure 2: Weighted pebble walking automaton $\mathcal{A}$ computing the size of the largest frame in a picture.

$\mathcal{A}$ computes the size of the biggest framed square. Notice that there are several accepting runs that check the same frame – since, e.g., in the initial state, we allow the automaton to follow any sequence of $\rightarrow$ and $\uparrow$ actions. This is not an issue as the semiring is idempotent.

Remark. In the Boolean semiring, wPWA and graph acceptors [24] (that use *global tilings* as runs) are incomparable. Over binary trees, pebble tree-walking automata are known to be strictly less expressive than classical top-down automata [2]. Reciprocally, Thomas showed in [24] that there is a picture language that cannot be recognized by graph acceptors, but that can be defined by first-order formulae; by Theorem 5 (or the result of [13]), it follows that this language can be recognized by a graph-walking automaton.

We give here some results involving wPWA, in particular concerning their evaluation and emptiness problem. We refer to [20] for proofs of these theorems. We give the complexities in terms of scalar operations, namely sums, products and star operations, that is $s^\star = \sum_{n \geqslant 0} s^n$ for $s \in \mathbf{S}$ (which exists since semiring $\mathbf{S}$ is continuous).

Theorem 1. *We can compute the semantics* $[\![\mathcal{A}]\!](G, \sigma)$ *of a* wPWA $\mathcal{A} = (Q, A, D, \mathrm{Peb}, I, F, K, \Delta, \mathrm{wt})$*, over a graph* $G = (V, (E_d)_{d \in D}, \lambda, \iota)$ *and a valuation* σ*, with (letting* $p = |\mathrm{Peb}|$*)*

- $\mathcal{O}(|Q| \times |V|^{p+1})$ *scalar star operations,*
- $\mathcal{O}((p+1) \times |Q| \times |\Delta| \times |V|^{p+2} + |Q|^3 \times |V|^{p+3})$ *scalar sums and products.*

Notice in particular that the complexity does not directly depend on the size K of the stack used in the automaton. It has to be noticed that better algorithms exist for specialized classes of graphs. For example, over words, binary trees or nested words, it is possible to drop the previous complexity to

- $\mathcal{O}(|Q| \times |V|^{p+1})$ scalar star operations,
- $\mathcal{O}((p+1) \times |Q|^3 \times |V|^{p+1})$ scalar products, and
- $\mathcal{O}((p+1) \times |Q|^3 \times |V|^{p+1} + |\Delta| \times V^{p+1})$ scalar sums.

Next, we focus on the emptiness problem, by using logical arguments. Monadic second-order logic has been extensively studied over graphs, in particular the decidability status of the satisfiability problem over certain classes of graphs. In [7], it was shown that for a class $\mathfrak{G}$ of graphs of bounded

degree, this problem is decidable if $\mathfrak{G}$ is MSO definable (i.e., there exists a formula φ of MSO such that for all graphs G, we have $G \in \mathfrak{G}$ iff G verifies φ) and has bounded *clique-width*. By encoding an accepting run of a wPWA into MSO, using transformations of [13], this allows us to obtain the following theorem:

Theorem 2. *Let $\mathfrak{G}$ be an MSO definable class of (A, D)-graphs with bounded clique-width, and $\mathbf{S}$ be a continuous semiring without zero divisors (i.e., $s_1 \times s_2 = 0$ implies $s_1 = 0$ or $s_2 = 0$). The following problem is decidable:*

Input: *A* wPWA $\mathcal{A}$ *over* $\mathbf{S}$ *(and A and D).*
Question: *Do there exist $G \in \mathfrak{G}$ and a valuation σ such that $[\![\mathcal{A}]\!](G, \sigma) \neq 0$?*

Notice that, because of the assumption on continuity and zero divisors of the semiring, this result is indeed purely Boolean, namely on the existence of an accepting run in the automaton. Because of the closure by negation of MSO, it also holds that universality (namely to know whether every graph $G \in \mathfrak{G}$ and valuation σ verifies $[\![\mathcal{A}]\!](G, \sigma) \neq 0$) of wPWA is decidable. From [22], that studies emptiness and universality of pebble tree-walking automata in the Boolean setting, we know that these problems are K-EXPTIME-hard already in this setting. As a side remark, notice that these decidability results do not apply on pictures, since this class of graphs has unbounded clique-width.

4 Weighted Logic over Graphs

Our aim is to provide a denotational way to describe quantitative properties of graph models. This has first been achieved for words by Droste and Gastin [8], who have introduced weighted logics whose expressive power is equivalent to that of weighted automata. The syntax of [8] is purely quantitative, though Boolean connectives can be expressed indirectly. As it may be somewhat confusing to interpret purely logical formulae in a weighted manner, we slightly modify the original syntax, as already presented in [3], and later in [18], by clearly separating the Boolean and the quantitative parts: our weighted logic consists of a Boolean first-order kernel augmented with quantitative operators (addition, multiplication, sum and product, and weighted transitive closure). Thus, our language allows us to test explicitly for Boolean properties, *and* to perform computations.

Definition 5. The set of weighted first order formulae with bounded transitive closure, denoted by wFOTC, is defined by

$$\Phi ::= s \mid \varphi \,?\, \Phi : \Phi \mid \Phi \oplus \Phi \mid \Phi \otimes \Phi \mid \bigoplus_x \Phi \mid \bigotimes_x \Phi \mid \mathrm{TC}^N_{x,y}\Phi$$

with $s \in \mathbf{S}$, $\varphi \in \mathrm{FO}$, $x, y \in \mathrm{Var}$ and $N \in \mathbb{N} \setminus \{0\}$.

We denote again by $\mathrm{Free}(\Phi)$ the set of free variables of formula $\Phi \in$ wFOTC. The semantics $[\![\Phi]\!]$ of Φ maps a pair (G, σ), composed of a graph $G \in \mathcal{G}(A, D)$ and a valuation σ whose domain contains $\mathrm{Free}(\Phi)$, to a value in $\mathbf{S}$. The semantics of all constructs but $\mathrm{TC}^N_{x,y}\Phi$ is defined inductively in Table 1. If $\mathbf{S}$ is not commutative, we assume that the products follow a specified order over V (for instance, the order used to search the graph, in case we are considering a searchable class of graphs). As for the Boolean case, notice that the semantics $[\![\Phi]\!]$ does not depend on the assignment of variables not in $\mathrm{Free}(\Phi)$.

We now explain the bounded weighted transitive closure operator $\mathrm{TC}^N_{x,y}$. There is such an operator for each integer $N > 0$. For a formula $\Phi(x, y)$ with at least two free variables x and y,

$$
\begin{aligned}
[\![s]\!](G,\sigma) &= s \\
[\![\varphi \mathbin{?} \Phi_1 : \Phi_2]\!](G,\sigma) &= \begin{cases} [\![\Phi_1]\!](G,\sigma) & \text{if } G,\sigma \models \varphi \\ [\![\Phi_2]\!](G,\sigma) & \text{otherwise} \end{cases} \\
[\![\Phi_1 \oplus \Phi_2]\!](G,\sigma) &= [\![\Phi_1]\!](G,\sigma) + [\![\Phi_2]\!](G,\sigma) \\
[\![\Phi_1 \otimes \Phi_2]\!](G,\sigma) &= [\![\Phi_1]\!](G,\sigma) \times [\![\Phi_2]\!]G,\sigma) \\
[\![\textstyle\bigoplus_x \Phi]\!](G,\sigma) &= \sum_{v \in V} [\![\Phi]\!](G,\sigma[x \mapsto v]) \\
[\![\textstyle\bigotimes_x \Phi]\!](G,\sigma) &= \prod_{v \in V} [\![\Phi]\!](G,\sigma[x \mapsto v])
\end{aligned}
$$

Table 1: Semantics of wFO($\mathcal{L}$)

we first define the (unbounded) weighted transitive closure $\mathrm{TC}_{x,y}\Phi$. It has the same free variables as Φ. We may denote it $[\mathrm{TC}_{x,y}\Phi](x,y)$ to stress the existence of two special free variables x and y in $\mathrm{TC}_{x,y}\Phi$: we may further write $[\mathrm{TC}_{x,y}\Phi](x',y')$ to change the name of these free variables. Its semantics, denoted by $[\![[\mathrm{TC}_{x,y}\Phi](x',y')]\!](G,\sigma)$, is defined by

$$
\sum_{\substack{v_0,v_1,\ldots,v_m\ (m>0) \\ \sigma(x')=v_0,\sigma(y')=v_m}} \prod_{0 \leqslant k \leqslant m-1} [\![\Phi]\!](G,\sigma[x \mapsto v_k, y \mapsto v_{k+1}]). \tag{1}
$$

Here, the sum ranges over all $m > 0$ and all sequences $(v_k)_{0 \leqslant k \leqslant m}$ of vertices of the graph G with $v_0 = \sigma(x')$ and $v_m = \sigma(y')$. Notice that this sum is infinite, and hence requires the semiring to be continuous in order for the sum to be well-defined.

The semantics of the *bounded* transitive closure operator $\mathrm{TC}^N_{x,y}$ restricts the sequences of (1) so that $\mathsf{dist}(v_k, v_{k+1}) \leqslant N$ for all $k \in \{0,\ldots,m-1\}$. That is,

$$
\mathrm{TC}^N_{x,y}\Phi = \mathrm{TC}_{x,y}(\mathsf{dist}(x,y) \leqslant N \mathbin{?} \Phi : 0)\,.
$$

where $\mathsf{dist}(x,y) \leqslant N$ is the FO-formula that tests that x, y are interpreted by two nodes at distance at most N.

Example 5 (Pictures, contd.)**.** We write a formula of wFOTC over $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ computing the size of the largest frame in a picture (as defined in Example 4). Let $R_{\nearrow}(x,y) = \exists z[R_{\rightarrow}(x,z) \wedge R_{\uparrow}(z,y)] \mathbin{?} 0 : -\infty$. Then, formula $\mathrm{Square}(x,y) = \mathrm{TC}^2_{x,y} R_{\nearrow}(x,y)$ evaluates to 0 if the rectangle defined by x, y is a square, and to $-\infty$ otherwise: notice the use of a weighted transitive closure, which is necessary since this property cannot be expressed in FO. A formula computing the size of the biggest square frame is then:

$$
\textstyle\bigoplus_x \bigoplus_y \Big[\big(\text{bw-rect}(x,y) \mathbin{?} 0 : -\infty\big) \otimes \mathrm{Square}(x,y) \otimes \bigotimes_z \big(\text{in-rect}(x,y,z) \mathbin{?} 1 : 0\big) \Big]\,.
$$

The semantics of Table 1 immediately translates to an algorithm computing the semantics of an input formula. For instance, one can compute $[\![\bigoplus_x \Phi]\!](G,\sigma)$ by first inductively computing the values $[\![\Phi]\!](G,\sigma[x \mapsto v])$ for all the finitely many vertices v of G, and by then summing these values.

To evaluate a bounded weighted transitive closure, that is, to compute $[\![\mathrm{TC}^N_{x,y}\Phi]\!](G,\sigma)$, one first inductively computes the $\mathbf{S}^{V\times V}$-matrix whose (v,w)-entry is equal to $[\![\mathsf{dist}(x,y)\leqslant N?\Phi{:}0]\!](G,\sigma[x\mapsto v, y\mapsto w])$. The semantics $[\![\mathrm{TC}^N_{x,y}\Phi]\!](G,\sigma)$ is then given by the matrix $M^\star$, which exists since $\mathbf{S}$ is continuous, and is computable, e.g., by Conway matrix identity (see [6] and [20, Lem. 5.1], for a precise complexity result).

5 From Logic to Automata

This section shows that formulae of wFOTC can be efficiently translated into equivalent weighted pebble walking automata. The complexity of the translation is linear in the size of the formula when the bounded transitive closure is not used. For the general case, the complexity also depends on the maximal bound of transitive closures. Formally, given a wFOTC-formula Φ, we denote by $\mathrm{tcb}(\Phi)$ the maximal transitive closure bound occurring in Φ. The size $|\Phi|$ of a formula is defined as usual, the only specific case being $|\mathrm{TC}^N_{x,y}\Phi| = |\Phi| + N + 1$. This is justified since we can express with an FO formula of size $\mathcal{O}(N)$ that the distance between two vertices is bounded by N.

Theorem 3. *Let $\mathfrak{G}$ be a searchable class of graphs. For every formula Φ of* wFOTC*, there exists a weighted pebble walking automaton $\mathcal{A}_\Phi$, whose set* Peb *of pebbles is the set of variables occurring in Φ, such that for every graph $G\in\mathfrak{G}$ and every valuation σ over* Peb*, $[\![\mathcal{A}_\Phi]\!](G,\sigma) = [\![\Phi]\!](G,\sigma)$. The size of $\mathcal{A}_\Phi$ is $|\mathcal{A}_\Phi| = \mathcal{O}(|\Phi|)\cdot 2^{\mathcal{O}(\mathrm{tcb}(\Phi))}$.*

Remark. Notice that, here and in the following, we will use σ both as a valuation for logical formulae, and for initial valuation of pebbles in a wPWA. In particular, we suppose here that σ is a total mapping over Peb for $[\![\mathcal{A}_\Phi]\!](G,\sigma)$ to be defined, whereas $[\![\Phi]\!](G,\sigma)$ could be defined if σ is only set on free variables of Φ.

Proof. Let $\mathcal{A}_\mathfrak{G}$ be the guide of the searchable class $\mathfrak{G}$.

The proof is by induction over the formulae in wFOTC. In order to achieve a *linear* translation, the most difficult part is to deal with the Boolean fragment. So, we start by constructing, for every φ of FO, a weighted pebble walking automaton $\mathcal{B}^\varphi$ having one initial state q^{in} and two final states $\checkmark_\varphi$ and ✗_φ such that for every graph $G\in\mathfrak{G}$ and valuation σ whose domain contains the free variables of φ, the semantics satisfies:

$$[\![\mathcal{B}^\varphi_{q^{\mathrm{in}},\checkmark_\varphi}]\!](G,\sigma) = \begin{cases} 1 & \text{if } G,\sigma\models\varphi \\ 0 & \text{otherwise,}\end{cases}$$

$$[\![\mathcal{B}^\varphi_{q^{\mathrm{in}},\text{✗}_\varphi}]\!](G,\sigma) = \begin{cases} 0 & \text{if } G,\sigma\models\varphi \\ 1 & \text{otherwise.}\end{cases}$$

All transitions of $\mathcal{B}^\varphi$ will have weight 1, so each run also has weight 1. But the semantics of $\mathcal{B}^\varphi$ is still quantitative, summing the weights of all accepting runs. In order to achieve the desired semantics, $\mathcal{B}^\varphi$ should be *unambiguous*, which is the main difficulty. More precisely, for every pair (G,σ), automaton $\mathcal{B}^\varphi$ should have exactly one run from its initial state to either $\checkmark_\varphi$ or ✗_φ, depending on whether $G,\sigma\models\varphi$ or not.

For formula $\varphi = \mathsf{init}(x)$, the automaton checks that the initial vertex carries x with a guard $x?$, and jumps in state $\checkmark_\varphi$, or checks the guard $\neg x?$ and jumps in state ✗_φ. For $\varphi = P_a(x)$, the automaton scans the input graph using the guide $\mathcal{A}_\mathfrak{G}$, searching for a vertex labeled a and satisfying

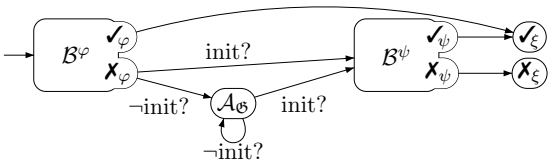


Figure 3: Weighted pebble walking automaton for disjunction $\xi = \varphi \vee \psi$

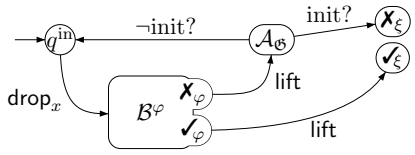


Figure 4: Weighted pebble walking automaton for $\xi = \exists x\ \varphi$

the guard $x?$ to jump in state $\checkmark_\varphi$. If no vertex verifying this condition is found before the guide reaches back vertex ι, we jump in state $\times_\varphi$. Similarly, for formula $x = y$, we check the existence of a vertex verifying guard $x? \wedge y?$. For the atomic formula $\varphi = R_d(x, y)$, the automaton scans the input graph, searching for a vertex holding variable x, then it checks that there is a d-edge that reaches a vertex holding variable y, in which case it jumps in $\checkmark_\varphi$, otherwise it jumps in $\times_\varphi$.

The construction for formula $\varphi = R_d^+(x, y)$ needs more care since the automaton $\mathcal{B}^\varphi$ must compute both φ and its negation. To do so, $\mathcal{B}^\varphi$ follows the guide to reach the vertex holding variable x. Then, it follows the unique d-path until either it reaches a vertex carrying y, in which case it jumps in $\checkmark_\varphi$, or it reaches a vertex with no d-successor, or it reaches back x. In the last two cases, it jumps in $\times_\varphi$. The correctness relies on the fact that our graphs are finite, and both deterministic and co-deterministic with respect to directions.

For negation $\xi = \neg\varphi$, we simply exchange the roles of states $\checkmark_\varphi$ and $\times_\varphi$ in $\mathcal{B}^\varphi$. The construction for disjunction $\xi = \varphi \vee \psi$ is described in Fig. 3 (a stay action is assumed on transitions where no action is shown). In this picture, as well as the following ones, when the guide $\mathcal{A}_\mathfrak{G}$ is called, we suppose it is from its initial state q_i to its final state q_o. We start by computing φ and jump in $\checkmark_\xi$ if φ is satisfied (i.e., if we arrived in $\checkmark_\varphi$); otherwise (i.e., if we arrived in $\times_\varphi$), we reset to the initial vertex of the graph using the guide, and check formula ψ. Finally, the construction for existential quantification $\xi = \exists x\ \varphi$ is described in Fig. 4. During a run of this automaton, a pebble x is successively dropped on every vertex of the graph using the guide. Whenever pebble x is dropped, we start $\mathcal{B}^\varphi$ to check whether the vertex holding pebble x satisfies φ, in which case we go to $\checkmark_\xi$. If we reach back the initial vertex, then this means that we have not found any vertex satisfying φ, so we jump in $\times_\xi$.

It remains to prove that the class of pebble weighted automata is closed under the weighted constructs of wFOTC. First, the constant $s \in \mathbf{S}$ is recognized by a wPWA having a single stay transition with weight s from its initial state to its final state. We obtain automaton $\mathcal{A}_{\varphi?\Phi_1:\Phi_2}$ by considering automaton $\mathcal{B}^\varphi$ and adding a stay transition from $\checkmark_\varphi$ (resp. $\times_\varphi$) to the initial state of $\mathcal{A}_{\Phi_1}$ (resp. $\mathcal{A}_{\Phi_2}$).

Let $\mathcal{A}_1$, $\mathcal{A}_2$ be two wPWA. Closure under $\oplus$ is as usual obtained using the disjoint union of automata $\mathcal{A}_1$ and $\mathcal{A}_2$. The wPWA that recognizes the Hadamard product which maps every graph G and valuation σ to $[\![\mathcal{A}_1]\!](G, \sigma) \times [\![\mathcal{A}_2]\!](G, \sigma)$ consists of three phases. First, it simulates the automaton $\mathcal{A}_1$ until it reaches one of its final states; then, using the guide, it goes back to the initial

vertex of the graph; finally, it simulates the automaton $\mathcal{A}_2$ and exits in a final state of $\mathcal{A}_2$.

From a wPWA $\mathcal{A}$ of height K computing $[\![\Phi(x)]\!]$, we construct two wPWA of height $K+1$ computing respectively $[\![\bigoplus_x \Phi]\!]$ and $[\![\bigotimes_x \Phi]\!]$. The automaton for $\bigoplus_x \Phi$ follows the guide, and non-deterministically (before reaching vertex ι for the second time) drops pebble x on *some* vertex in the graph. Then, it goes back to the initial vertex using the guide, and simulates automaton $\mathcal{A}$ until it reaches one of its final states where it lifts the pebble and accepts. Notice that since the guide visits each vertex exactly once, the automaton computes exactly the semantics of $\bigoplus_x \Phi$. For $\bigotimes_x \Phi$, the new automaton drops successively pebble x on *every* vertex of the input graph (again using the guide). Whenever it drops pebble x, it goes back to the initial vertex, simulates $\mathcal{A}$ until it reaches a final state, where it lifts the pebble and proceeds to the next vertex.

Finally, assume that we have a wPWA $\mathcal{A}$ of height K which evaluates some wFOTC formula $\Phi(x,y)$. We construct a wPWA $\mathcal{A}'$ of height $K+2$ which evaluates $[\mathrm{TC}^M_{x,y}\Phi](x',y')$. Following the guide, $\mathcal{A}'$ moves to the vertex $v_{x'}$ holding the free variable x'. It drops pebble x. Whenever $\mathcal{A}'$ drops pebble x on some vertex v_x, it enters some special state q_{drop_x}. It then moves to some guessed vertex v_y at a distance less than or equal to M, remembering the sequence of directions it used to reach v_y. It drops pebble y and goes back to the initial vertex with the guide. In order to compute the transitive closure in an *unambiguous* way, the automaton now checks that the path it followed from v_x to v_y is the *minimal* one amongst those that lead from v_x to v_y (minimality is with respect to the lexicographic order over the sequences of directions, supposing a fixed order over $D \cup D^{-1}$). To check this, $\mathcal{A}'$ explores deterministically the (finite) set of sequences of directions of length at most M in the lexicographic order: it stops the first time it reaches vertex v_y and rejects the current run if the sequence of directions is not the one recorded before. Now, this sequence is flushed from the memory in order to reduce the size of the automaton. $\mathcal{A}'$ then simulates $\mathcal{A}$ resulting in the evaluation of $\Phi(v_x, v_y)$. When $\mathcal{A}$ reaches a final state, $\mathcal{A}'$ uses the guide to return to vertex v_y. Then, $\mathcal{A}'$ non-deterministically moves to vertex v_x where pebble x was dropped remembering the sequence of directions it followed. Again, in order to simulate the transitive closure unambiguously, it checks that the guessed sequence of directions is the minimal one in the lexicographic order. $\mathcal{A}'$ lifts pebble y and pebble x, and then returns to vertex v_y by using the memorized sequence of directions. If y' is held by the current vertex, $\mathcal{A}'$ enters a final state. Also – even if y' is held by the current vertex – $\mathcal{A}'$ drops pebble x again in order to continue the evaluation of the transitive closure.

We analyse now the complexity of our construction. First, the guide is independent of the input formula, so its size is constant. Constructions for all operators except the transitive closure add a constant number of states and transitions. For the transitive closure, we store a fixed number of paths of lengths at most $\mathrm{tcb}(\Phi)$, hence the additional number of states and transitions is in $2^{\mathcal{O}(\mathrm{tcb}(\Phi))}$. □

Our proof strongly relies on the fact that the transitive closure operator is bounded. Otherwise, we do not know how to achieve the same result, unless one is allowed to use *strong pebbles* (see [2]) instead of weak ones. However, our proof of the other translation, that we present in Theorem 5, does not work for strong pebbles: indeed, it is an open problem to know the expressiveness of weighted strong pebble walking automata.

Remark. Notice that it would have been possible to add some other atomic Boolean formulae φ and still get the result of Theorem 3, as soon as we have an unambiguous automaton $\mathcal{B}^\varphi$ computing simultaneously φ and its negation. For instance, with the class of binary trees, we may add a Boolean formula $x \preceq y$ stating that x is an ancestor of y, i.e., that there exists a $\{\downarrow_1, \downarrow_2\}$-path from

x to y. The unambiguous automaton for this formula first moves to vertex holding x, and then executes a depth-first search of the subtree rooted in x until either it finds y and accepts, or reaches back x and rejects.

6 From Automata to Logic

This section is devoted to the translation of automata back into logic. Mainly, the aim is to show robustness of the automaton model we exhibited.

Proofs of the next theorems are partly based on ideas present in [23] and [13]. The idea is to encode runs of automata into a formula using transitive closures. However, unlike in the Boolean setting, it is not sufficient to be able to simulate *at least one* run for every accepted structure. The simulation must, in addition, be faithful with respect to the weights. This new problem is even more interesting in the case of graphs.

We split the proof into two parts. First, we define the notion of zonable classes of graphs, and prove that all graph classes we are interested in are zonable. Then, abstracting the technical difficulties inherent of this definition, we start from a zonable class of graphs and prove that we can construct a formula equivalent to a given automaton, for all graphs of this class.

6.1 Zonable Classes of Graphs

Zonability is a combinatorial notion that we introduce to cut a graph into small *zones*, however wide enough to encode each edge relating two distinct zones into one of these two zones. This zone decomposition will be used later to encode runs of a weighted automaton navigating in the graphs. The idea for this encoding is to abstract the behavior of the automaton when it walks inside a zone, and to consider jumps through edges relating two distinct zones. We also need the zonability to be uniformly computable by a fixed formula for the class. As mentioned in Ex. 5, Boolean first-order formulae are too weak, hence we use the power of the weighted transitive closure to compute zones. However, we require the formulae to be *unambiguous* in the sense that their semantics maps every possible pair (G, σ) to either 0 or 1.

To introduce zonability, we need some more notation. Let $G = (V, (E_d)_{d\in D}, \lambda, \iota)$ be a graph and $\sim \subseteq V \times V$ be an equivalence relation on V. An equivalence class of $\sim$ is called a *zone* (of $\sim$). We say that a zone $Z \subseteq V$ has a *diameter bounded by* $M \in \mathbb{N}$ if, for all $v, v' \in Z$, there exists a path from v to v' of length at most M that uses only vertices in Z. An edge $(v, v') \in E \cup E^{-1}$ such that $v \not\sim v'$ is called a *wire*. Note that, due to reversed edges, (v, v') is a wire iff so is (v', v). The set of wires is denoted by $\mathcal{W}_\sim$. Let $\mathfrak{G} \subseteq \mathcal{G}(A, D)$ be a class of graphs and $N \in \mathbb{N}$ be a natural number.[6] Consider a tuple $(M, \mathsf{zone}(z, z'), (\mathsf{enc}_n(z, z', x))_{0\leqslant n<N})$ where $M \in \mathbb{N}$ is a bound, $\mathsf{zone}(z, z')$ is an unambiguous wFOTC-formula with free variables z and z', and $\mathsf{enc}_n(z, z', x)$ are unambiguous wFOTC-formulae with free variables z, z', and x. The tuple is called a *zoning* of $\mathfrak{G}$ for N if, for every graph $G = (V, (E_d)_{d\in D}, \lambda, \iota) \in \mathfrak{G}$, there exist an equivalence relation $\sim$ over V whose equivalence classes have a diameter bounded by M, and an injective total function $enc\colon \mathcal{W}_\sim \times \{0, \dots, N-1\} \to V$ satisfying the following, for all $v, v', u \in V$ and all $n \in \{0, \dots, N-1\}$:

- $v \sim v'$ iff $[\![\mathsf{zone}(z, z')]\!](G, [z \mapsto v, z' \mapsto v']) = 1$,

[6] N is the number of states of the automaton to be translated into the logic.

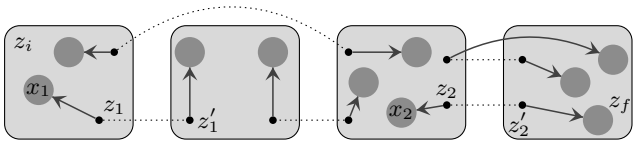


Figure 5: Zones and wires of a graph.

- $\big[\,(v,v') \in \mathcal{W}_\sim$ and $enc((v,v'),n) = u\,\big]$ iff $[\![\mathsf{enc}_n(z,z',x)]\!](G,[z \mapsto v, z' \mapsto v', x \mapsto u]) = 1$, and
- if $(v,v') \in \mathcal{W}_\sim$, then the vertex $enc((v,v'),n)$ is in the same zone as v or v'.

The mapping *enc* permits to encode each pair of wire and integer in $\{0,\dots,N-1\}$ to a vertex of the graph in an injective way. Since wires relate two distinct zones of the graph, note that they can be characterized by the following unambiguous formula:

$$\mathsf{wire}(z,z') = \bigoplus_x \mathsf{enc}_0(z,z',x)$$

Definition 6. A class $\mathfrak{G} \subseteq \mathcal{G}(A,D)$ of graphs is called *zonable* if, for every number $N \in \mathbb{N}$, there is a zoning of $\mathfrak{G}$ for N.

Fig. 5 depicts, schematically, the zones and wires of a graph that are induced by a zoning. Zones are depicted with rounded rectangles. Wires are shown with dashed lines, and the gray disk linked to a vertex v belonging to a wire (v,v') depicts the set of vertices $enc((v,v'),n)$, for $n \in \{0,\dots,N-1\}$. Note that, in principle, disks could overlap two distinct zones. Variables $z_i, z_1, x_1, \dots$ will be explained in the proof of Theorem 5.

Notice that a zone contains at most $(2|D|)^M$ vertices, since its diameter is bounded by M and all graphs of $\mathfrak{G}$ are assumed to have degree at most $2|D|$.

Let us give a non-exhaustive list of zonable classes of graphs.

Theorem 4. *The following classes of graphs are zonable: words, binary trees, nested words, and pictures.*

Proof. In this article, we only sketch the case of trees and pictures.

In binary trees, we consider zones to be subtrees with roots at depth 0 modulo $2N$ and of height between $2N$ and $4N-1$. In particular, every subtree having a root at depth 0 modulo $2N$ which has height less than $2N$ is not a zone and hence belongs to the zone which is above it. Since each zone has a height bounded by $4N-1$, its diameter is bounded by $M = 2 \times (4N-1) - 1$. Notice that the depth of a node modulo $2N$ is computable by a wFOTC-formula, as it suffices to check the length modulo $2N$ of the *unique* path leading from the root to the node, which can be done by the formula $\mathsf{depth}(z) \equiv k \mod 2N$ defined by

$$\bigoplus_{z',z''} \big(R^k_\uparrow(z,z') \wedge \mathsf{init}(z'') \quad ? \quad (R^{2N}_\uparrow)^*(z',z'') \quad : \quad 0\big)$$

where $R^k_\uparrow(z,z')$ means that z is at depth k in the subtree rooted at z' and $(R^{2N}_\uparrow)^*(z',z'')$ means that z' is at depth a multiple of $2N$ in the subtree rooted at z''. These formulae can be written as follows. First, $R_\uparrow(x,y) = R_{\downarrow_1}(y,x) \vee R_{\downarrow_2}(y,x)$. Then, for $\ell \geqslant 1$ we let $R^\ell_\uparrow(x_0,x_\ell) =$

$\exists\, x_1, \ldots, x_{\ell-1} \bigwedge_{0<i\leqslant \ell} R_\uparrow(x_{i-1}, x_i)$ and $R^0_\uparrow(x,y) = (x=y)$. Finally, we use an unambiguous weighted transitive closure for $(R^{2N}_\uparrow)^*(z', z'')$ which is defined by

$$(z' = z'')\ ?\ 1 : \big[\mathrm{TC}^{2N}_{x,y}\big(R^{2N}_\uparrow(x,y)\ ?\ 1 : 0\big)\big](z', z'')\,.$$

Formulae $\mathsf{zone}(z_1, z_2)$ may easily be written by considering the unique root x of the zone hypothetically containing both z_1 and z_2: the formula must check that x is at depth 0 modulo $2N$, that its subtree has the correct depth (with an FO-formula), and that both z_1 and z_2 are in this subtree (also with an FO-formula).

Wires are edges that enter or exit the root of a zone. Hence, each root, except the root of the whole tree, is part of exactly two wires relating the root to its parent. Notice that it is not possible to encode a wire in the zone from which it exits. Indeed, a zone may be connected to an exponential number of wires – the wires at the bottom – and the zone may not contain enough nodes to encode them all. Instead, we encode a wire in the subtree of the root. Hence, each zone must encode at most $2 \times N$ pairs, which is possible as it contains at least this number of vertices (considering its height). It is easy to choose of a decodable order and to design a formula $\mathsf{enc}_n(z_1, z_2, x)$. For example, we may consider the leftmost branch of this subtree of length $2N$ (which exists considering the hypothesis on its height) and use the N first nodes to encode the entering wire, and the others for the exiting wire. These functions may easily be defined by wFOTC-formulae.

For pictures, similar ideas to cut them into zones have been used for other purposes in [19]. Notice first that if a graph has a number of vertices bounded by M – which is verifyable by a formula of FO – then we can consider that there is a single zone containing all the vertices, and hence no wires. Therefore, we only consider pictures having at least $M+1$ vertices. Zones of such pictures will be square subpictures of width $4N$, except the zones on the right and the top of the pictures that may be a little larger. The largest zone is the one on the top right corner which can have width and height bounded above by $8N-1$. Hence, each zone has a diameter bounded by $M = 2 \times (8N-1) - 1$. This can be tested using modulo computations.

For $d \in D$, define inductively the FO formulae $R^0_d(x,y,z) = (x=y)$ and for $k \geqslant 0$, $R^{k+1}_d(x,y,z) = (x \neq z) \wedge \exists t\, R_d(x,t) \wedge R^k_d(t,y,z)$: hence, $R^k_d(x,y,z)$ states that x and y are related by a d-path of length k composed of vertices different from z (possibly except of vertex y). Then, for $\ell > 0$ and $0 \leqslant m < \ell$, we let $\mathsf{dist}_d(x,y) \equiv m \mod \ell$ be the formula

$$\bigoplus\nolimits_z R^m_d(x,z,y) \otimes \big((z=y) \oplus [\mathrm{TC}^{\ell}_{x_1,x_2}((x_1 \neq y) \wedge R^{\ell}_d(x_1,x_2,y))](z,y)\big)\,.$$

It maps (G,σ) to 1 or 0, according to the fact that the shortest d-path connecting $\sigma(x)$ and $\sigma(y)$ (if such a path exists) is of length m modulo ℓ. The purpose of tests $x \neq z$ in $R^k_d(x,y,z)$, and test $x_1 \neq y$ in the previous formula, is to ensure that we stop the first time y is reached, thus yielding 0 or 1.

Let $\Phi_d(x,y) = [\mathsf{dist}_d(x,y) \equiv 0 \mod 4N]$ and $\Psi(x,y) = \bigoplus_z \bigoplus_{0\leqslant k,\ell<4N} R^k_{\rightarrow}(x,z) \otimes R^{\ell}_{\uparrow}(z,y)$. Forgetting, for the sake of simplicity, about the zones on the right and on top, we obtain as formula $\mathsf{zone}(z_1, z_2)$:

$$\bigoplus\nolimits_{x,y} \Phi_{\rightarrow}(\mathsf{init}, x) \otimes \Phi_{\uparrow}(x,y) \otimes \Psi(y, z_1) \otimes \Psi(y, z_2).$$

Notice that in this formula, y denotes the position at the lower left corner of the zone containing z_1 and z_2. Each zone (except the larger ones) has at most $4 \times 4N$ wires for which each of the N states

must be encoded, and we will encode it in the zone from which the wire exits: hence each zone must have at least $4 \times 4N \times N = (4N)^2$ positions available which is exactly the case. Fixing some total order between the wires, it is easy to design a formula $\mathsf{enc}_n(z_1, z_2, x)$ for every $n \in \{0, \dots, N-1\}$. For example, we may consider a partitioning of each zone into 4 disjoint rectangles of height N each reserved for the wires of one border of the zone.

The proof for words was considered in [4], with different notations. The difficult extension to nested words can be found in [20]. In particular, notice that we cannot use the zone partitioning of the underlying word: indeed, since there are nesting edges, each zone may then be wired to too many zones, and it would then be impossible to encode all the wires. □

6.2 Logical Characterization of Automata

Theorem 5. *Let $\mathfrak{G} \subseteq \mathcal{G}(A, D)$ be a zonable class of graphs. Then, from every weighted pebble walking automaton $\mathcal{A}$, we can construct a formula $\Phi_{\mathcal{A}}$ of* wFOTC *with free variables included in* Peb *(the set of pebbles used by $\mathcal{A}$), which is equivalent to $\mathcal{A}$ over $\mathfrak{G}$, i.e., such that for every graph $G \in \mathfrak{G}$ and valuation σ over* Peb*, we have $[\![\mathcal{A}]\!](G, \sigma) = [\![\Phi_{\mathcal{A}}]\!](G, \sigma)$.*

Proof. The proof goes by induction on the height K of $\mathcal{A}$. We start with the case of $K = 0$. We construct the formula $\Phi_{\mathcal{A}}$ of wFOTC in several steps. First, we use an outermost transitive closure in order to make *macro steps* in the graph, and then compute the weight of the runs in-between macro steps with nested transitive closures. The main difficulty in this second step is that we have to use only bounded jumps in these transitive closure operators, which prevents us to mimic the proof of [13]. To overcome this difficulty, we use the zone cutting of the input graph presented before. Then, the outermost transitive closure simply jumps from one zone to an adjacent one, and the inner ones simulate the behavior of automaton $\mathcal{A}$ over a zone of bounded diameter: this ensures that every transitive closure operator performs bounded jumps only.

Let $\mathcal{A} = (Q, A, D, \mathrm{Peb}, I, F, K, \Delta, \mathrm{wt})$ and suppose $Q = \{0, \dots, N-1\}$. Using formulae $\mathsf{zone}(z_1, z_2)$ and $\mathsf{enc}_n(z_1, z_2, x)$ for $0 \leqslant n < N$, we design a formula $\Xi_{q_i,q_f}(z_i, z_f)$ that computes the weight of the runs starting from vertex z_i in state q_i and ending in vertex z_f in state q_f. Then, the semantics of the automaton $\mathcal{A}$ is obtained by considering the formula $\bigoplus_{q_i \in I, q_f \in F} \bigoplus_{z_f} \Xi_{q_i,q_f}(\mathsf{init}, z_f)$. For $\alpha \in \mathrm{Guards}$, let $\varphi_\alpha(z)$ be the FO formula that verifies the local test α over the vertex denoted by its free variable z. Let also $\varphi_{\alpha,a}(z) = \varphi_\alpha(z) \wedge P_a(z)$.

We first describe a formula $\mathsf{tr}_{q_1,q_2}(z_1, z_2)$ that computes the sum of the weights of all transitions going from vertex z_1 in state q_1 to vertex z_2 in state q_2. It is defined by

$$\bigoplus_{\substack{\delta=(q_1,\alpha,a,\gamma,q_2)\in\Delta \\ \text{s.t. } \gamma\in D\cup D^{-1}\cup\{\mathsf{stay}\}}} \big(\varphi_{\alpha,a}(z_1) \wedge R_\gamma(z_1, z_2)\big) \mathbin{?} \mathrm{wt}(\delta) : 0$$

where $R_{d^{-1}}(z_1, z_2)$ is defined as $R_d(z_2, z_1)$, and $R_{\mathsf{stay}}(z_1, z_2)$ as $z_1 = z_2$.

We will then use a transitive closure to compute the weights of the runs that remain in a given zone. This transitive closure will be bounded, since the diameter of a zone is bounded by M. More precisely, under the condition that vertices z_1 and z_2 are in the same zone Z, formula $\Phi_{q_1,q_2}(z_1, z_2)$ computes the sum of the weights of all nonempty runs that remain inside zone Z, going from vertex z_1 in state q_1 to vertex z_2 in state q_2. We follow the McNaughton-Yamada algorithm, i.e., we generate formulae $\Phi^R_{q_1,q_2}(z_1, z_2)$ with $R \subseteq Q$, computing the sum of the weights of such nonempty runs that visit as intermediary states only states in R. Then, $\Phi_{q_1,q_2}(z_1, z_2) = \Phi^Q_{q_1,q_2}(z_1, z_2)$. We construct these

formulae by induction over R. As a base case, we have $\Phi^{\emptyset}_{q_1,q_2}(z_1,z_2) = \mathsf{zone}(z_1,z_2) \otimes \mathsf{tr}_{q_1,q_2}(z_1,z_2)$, which moreover checks that z_1 and z_2 are in the same zone. Then, if $q \notin R$, the runs from q_1 to q_2 with intermediary states in $R \cup \{q\}$ either do not visit state q or visit it at least once. Hence, we may compute their weights with formula $\Phi^{R\cup\{q\}}_{q_1,q_2}(z_1,z_2)$ defined by

$$\Phi^{R}_{q_1,q_2}(z_1,z_2) \oplus \bigoplus\nolimits_z \Phi^{R}_{q_1,q}(z_1,z) \otimes \Phi^{R}_{q,q_2}(z,z_2) \oplus \\ \bigoplus\nolimits_{z,z'} \Phi^{R}_{q_1,q}(z_1,z) \otimes [\mathrm{TC}^{M}_{x,y}\Phi^{R}_{q,q}(x,y)](z,z') \otimes \Phi^{R}_{q,q_2}(z',z_2).$$

Finally, formula $\Xi_{q_i,q_f}(z_i,z_f)$ initializes the run and uses a transitive closure to jump from a zone to the following one (see Fig. 5 where variables appear). Formula $\mathsf{enc}_q(z_1,z_2,x)$ permits to encode the fact that state q was reached *just before* jumping through the wire (z_1,z_2), and that this situation is encoded in vertex x. Hence, $\Xi_{q_i,q_f}(z_i,z_f)$ is defined by

$$\bigoplus_{x_1,x_2} \big(\bigoplus\nolimits_{z_1,z_1'} \bigoplus\nolimits_{q_1\in Q} \mathsf{enc}_{q_1}(z_1,z_1',x_1) \otimes \Phi_{q_i,q_1}(z_i,z_1)\big) \\ \otimes [\mathrm{TC}^{3M}_{y_1,y_2}\Psi](x_1,x_2) \otimes \Big[\bigoplus\nolimits_{z_2,z_2'} \bigoplus\nolimits_{q_2,q_2'\in Q} \\ \mathsf{enc}_{q_2}(z_2,z_2',x_2) \otimes \mathsf{tr}_{q_2,q_2'}(z_2,z_2') \otimes \Phi_{q_2',q_f}(z_2',z_f)\Big]$$

with $\Psi(y_1,y_2)$ the formula

$$\bigoplus_{z_1,z_1',z_2,z_2'} \bigoplus_{q_1,q_1',q_2\in Q} \Big[\mathsf{enc}_{q_1}(z_1,z_1',y_1) \otimes \mathsf{tr}_{q_1,q_1'}(z_1,z_1') \\ \otimes \mathsf{enc}_{q_2}(z_2,z_2',y_2) \otimes \Phi_{q_1',q_2}(z_1',z_2)\Big] .$$

Since the formula $\Phi_{\_,\_}$ remains inside a zone, an additional transition $\mathsf{tr}_{\_,\_}$ has to be performed in Ψ and Ξ_{q_i,q_f} to jump through the corresponding wires.

Notice that variables x and y in formula Ψ must be at a distance bounded by $3M$ since each zone has a diameter M and a position encoding a wire must be in one of the two zones adjacent to this wire. Formula $\Psi(x,y)$ simply makes the first step jumping through the wire, before computing the weight of all the runs staying inside the zone using formula Φ_{q_1',q_2}. The correctness of this construction relies on the fact that semirings are distributive: indeed, we merge the whole set of runs into subsets visiting the same sequence of wires.

We now sketch the proof when the height of $\mathcal{A}$ is $K+1 \geqslant 0$. We apply the same construction as before, simply enriching the formula $\mathsf{tr}_{q_1,q_2}(z_1,z_2)$ so that it also deals with pebbles. The new formula is

$$\mathsf{tr}_{q_1,q_2}(z_1,z_2) \oplus (z_1 = z_2)\,?\,\mathsf{droplift}_{q_1,q_2}(z_1) : 0$$

where $\mathsf{droplift}_{q_1,q_2}(z)$ computes the sum of the weights of runs starting from z in state q_1 by dropping a pebble and run until the pebble is lifted and the control returns to z in state q_2.

By induction, for every states q_1' and q_2', we have a formula $\Xi^{u}_{q_1',q_2'}(z_i,z_f)$ that computes the sum of the weights of the runs from vertex z_i in state q_1' to vertex z_f in q_2', with pebble u dropped on vertex z_i, and visiting only configurations with a stack of size at most K. Then, define $\mathsf{droplift}_{q_1,q_2}(z)$ as

$$\bigoplus_{\substack{\delta_1=(q_1,\alpha_1,a_1,\mathsf{drop}_u,q_1')\\ \delta_2=(q_2',\alpha_2,a_2,\mathsf{lift},q_2)}} \bigoplus\nolimits_{z'} \ (\varphi_{\alpha_1,a_1}(z) \wedge \varphi_{\alpha_2,a_2}(z'))\,? \\ \mathrm{wt}(\delta_1) \otimes \Xi^{u}_{q_1',q_2'}(z,z') \otimes \mathrm{wt}(\delta_2) : 0\,. \qquad \square$$

We now discuss the complexity of the previous proof translating a weighted pebble walking automaton $\mathcal{A}$ into a formula $\Phi_{\mathcal{A}}$ of wFOTC. Let us show that $\Phi_{\mathcal{A}}$ has size $2^{\mathcal{O}(|\mathcal{A}|)}$. This will hold under some reasonable assumptions on the zonability of the class of graphs we consider, namely that there exists a function $f\colon \mathbb{N} \to \mathbb{N}$, of growth at most exponential (i.e., $f(N) = 2^{\mathcal{O}(N)}$), such that for every N, we have $M \leqslant f(N)$, and formulae $\mathsf{zone}(z_1, z_2)$ and $\mathsf{enc}_n(z, z', x)$ are of size at most $f(N)$. This assumption holds for all the classes considered in Theorem 4.

To prove that $\Phi_{\mathcal{A}}$ has size $2^{\mathcal{O}(|\mathcal{A}|)}$, we start again with the case $K = 0$. Formula $\mathsf{tr}_{q_1,q_2}(z_1, z_2)$ has size $\mathcal{O}(|\mathcal{A}|)$ (notice that the quantified sum appearing in this formula, and the following ones, is indeed a shortcut for a long sequence of binary sums, the length of which is the number of transitions of the automaton). To compute the size of formula $\Phi_{q_1,q_2}(z_1, z_2)$, let us first notice that $\Phi^{\emptyset}_{q_1,q_2}(z_1, z_2)$ has size $2^{\mathcal{O}(|\mathcal{A}|)}$ (since it uses the formula $\mathsf{zone}(z_1, z_2)$). Then, the size of $\Phi^{R\cup\{q\}}_{q_1,q_2}(z_1, z_2)$ is given as the addition of M (that is $2^{\mathcal{O}(|\mathcal{A}|)}$) and 6 times the size of formulae of the form $\Phi^{R}_{q,q'}(z, z')$. By induction, it is then possible to prove that formulae $\Phi^{R}_{q,q'}(z, z')$ have size $6^{|R|} \times 2^{\mathcal{O}(|\mathcal{A}|)}$, so that $\Phi_{q_1,q_2}(z_1, z_2) = \Phi^{Q}_{q_1,q_2}(z_1, z_2)$ is of size $6^{|Q|} \times 2^{\mathcal{O}(|\mathcal{A}|)} = 2^{\mathcal{O}(|\mathcal{A}|)}$. Finally, formulae $\Psi(y_1, y_2)$ and $\Xi_{q_i,q_f}(z_i, z_f)$ are easily shown to be of size $2^{\mathcal{O}(|\mathcal{A}|)}$.

For the case $K > 0$, the only difference lies in formula $\mathsf{tr}_{q_1,q_2}(z_1, z_2)$: it is now of size $\mathcal{O}(|\mathcal{A}|)$ to which we add the size of formulae $\Xi^{u}_{q'_1,q'_2}(z, z')$ obtained at the previous layer. In particular, if these ones are of size $2^{\mathcal{O}(|\mathcal{A}|)}$, then $\mathsf{tr}_{q_1,q_2}(z_1, z_2)$ is of size $2^{\mathcal{O}(|\mathcal{A}|)}$. Notice that this does not change the size of $\Phi^{\emptyset}_{q_1,q_2}(z_1, z_2)$ with respect to the previous case $K = 0$. Hence, the rest of the computation is identical which permits to conclude by induction that $\Phi_{\mathcal{A}}$ has indeed size $2^{\mathcal{O}(|\mathcal{A}|)}$.

Because of the construction based on McNaughton-Yamada algorithm, this exponential blow-up seems difficult to avoid.

7 Conclusion

We established expressive equivalence of weighted pebble walking automata and weighted first-order logic with transitive closure. Our result holds for general classes of graphs (searchable and zonable), and for arbitrary continuous semirings. Since the latter include the non-commutative semiring $2^{\Sigma^\star}$, we obtain, as a corollary, a logical characterization of graph-to-word pebble walking transducers. Over words, the special case of wPWA *without* pebbles is expressively equivalent to transitive-closure logic with a bounded number of TC operators [4]. An interesting open problem is whether a similar result can be achieved in the case of graphs.

References

[1] R. Alur and P. Madhusudan. Adding nesting structures to words. *Journal of the ACM*, 56(3), 2009.

[2] M. Bojańczyk, M. Samuelides, T. Schwentick, and L. Segoufin. Expressive power of pebble automata. In *Proceedings of ICALP'06*, volume 4051 of *LNCS*, pages 157–168. Springer, 2006.

[3] B. Bollig, P. Gastin, and B. Monmege. Weighted specifications over nested words. In *FoSSaCS'13*, volume 7794 of *LNCS*, pages 385–400. Springer, 2013.

[4] B. Bollig, P. Gastin, B. Monmege, and M. Zeitoun. Pebble weighted automata and transitive closure logics. In *Proceedings of ICALP'10*, volume 6199 of *LNCS*, pages 587–598. Springer, 2010.

[5] J. Büchi. Weak second order logic and finite automata. *Z. Math. Logik, Grundlag. Math.*, 5:66–62, 1960.

[6] J. H. Conway. *Regular Algebra and Finite Machines*. Chapman & Hall, 1971.

[7] B. Courcelle and J. Engelfriet. A logical characterization of the sets of hypergraphs defined by hyperedge replacement grammars. *Mathematical Systems Theory*, 28(6):515–552, 1995.

[8] M. Droste and P. Gastin. Weighted automata and weighted logics. [9], chapter 5, pages 175–211.

[9] M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. EATCS Monographs in TCS. Springer, 2009.

[10] D. D'Souza and P. Shankar. *Modern Applications of Automata Theory*. World Scientific Publishing, 2011.

[11] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the AMS*, 98:21–52, 1961.

[12] J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. *Jewels are forever, contributions to Theoretical Computer Science in honor of Arto Salomaa*, pages 72–83, 1999.

[13] J. Engelfriet and H. J. Hoogeboom. Automata with nested pebbles capture first-order logic with transitive closure. *LMCS*, 3:1–27, 2007.

[14] J. Engelfriet, H. J. Hoogeboom, and B. Samwel. XML transformation by tree-walking transducers with invisible pebbles. In *Proceedings of PODS'07*, pages 63–72. ACM, 2007.

[15] I. Fichtner. *Characterizations of Recognizable Picture Series*. PhD thesis, Universität Leipzig, 2007.

[16] I. Fichtner. Weighted picture automata and weighted logics. *Theory of Computing Systems*, 48(1):48–78, 2011.

[17] P. Gastin and B. Monmege. Adding pebbles to weighted automata – easy specification & efficient evaluation. *TCS*, 2014. To appear.

[18] S. Kreutzer and C. Riveros. Quantitative monadic second-order logic. In *Proceedings of LICS'13*, 2013.

[19] O. Matz. One quantifier will do in existential monadic second-order logic over pictures. In *MFCS'98*, volume 1450 of *LNCS*, pages 751–759. Springer, 1998.

[20] B. Monmege. *Specification and Verification of Quantitative Properties: Expressions, Logics, and Automata*. Phd thesis, ENS de Cachan, 2013.

[21] H. A. Rollik. Automaten in planaren graphen. *Acta Informatica*, 13(3):287–298, 1980.

[22] M. Samuelides and L. Segoufin. Complexity of pebble tree-walking automata. In *Proceedings of the 16th International Conference on Fundamentals of Computation Theory (FCT'07)*, volume 4639 of *Lecture Notes in Computer Science*, pages 458–469. Springer, 2007.

[23] W. Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25:360–376, 1982.

[24] W. Thomas. On logics, tilings, and automata. In *Proceedings of ICALP'91*, volume 510 of *LNCS*, pages 441–453. Springer, 1991.

[25] B. A. Trakhtenbrot. Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 149:326–329, 1961.