Weighted Expressions and DFS Tree Automata

Benedikt Bollig, Paul Gastin,
Benjamin Monmege, and Marc Zeitoun

April 2011

Research report LSV-11-08

Laboratoire Spécification & Vérification

École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Weighted Expressions and DFS Tree Automata ⋆

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

[1] LSV, ENS Cachan, CNRS & INRIA, France
firstname.lastname@lsv.ens-cachan.fr
[2] LaBRI, Univ. Bordeaux & CNRS, France

Abstract. We introduce weighted expressions, a calculus to express quantitative properties over unranked trees. They involve products and sums from a semiring as well as classical boolean formulas. We show that weighted expressions are expressively equivalent to a new class of weighted tree-walking automata. This new automata model is equipped with pebbles, and follows a depth-first-search policy in the tree.

1 Introduction

The motivation of this paper is to develop a high-level denotational formalism to express quantitative properties on trees. We would like this language to be flexible and versatile, and it should allow us to compute arithmetic expressions, possibly guarded by logical conditions written in a standard language (*e.g.*, first-order logic or XPath). Moreover, it should have an equivalent operational model.

Let us start with an example. Assume we have an XML tree representing a database storing information about car models and car parts, cf. Fig. 1. Each car model is described, among other things, by its list of car parts (Fig. 1 (*a*)). Each model of car part has an attached set of currently reported errors indicated by nodes labeled Err in Fig. 1 (*b*). One can express that there is a car model using some car part with an error by the first-order formula $\exists x, y, z, t\, \varphi(x, y, z, t)$, where

$$\varphi = [\mathsf{CModel}(x) \wedge \mathsf{Part}(y) \wedge x <_{\mathrm{v}} y] \wedge [\mathsf{PModel}(z) \wedge \mathsf{Err}(t) \wedge z <_{\mathrm{v}} t] \wedge \mathsf{Match}(y, z).$$

This is a purely logical statement, where $x <_{\mathrm{v}} y$ means that node x is an ancestor of node y, and $\mathsf{Match}(y, z)$ is a shortcut meaning that the car part at y matches the part model at z. In this paper, we abstract data away.

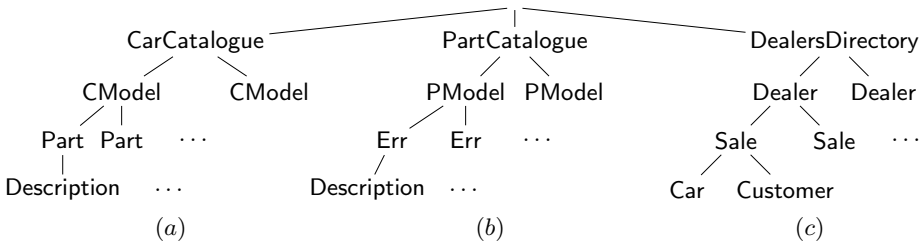

Fig. 1: Part of an XML document for the car database

One may want more accurate information, for example the total number of errors for all car models. Intuitively, this can be achieved by replacing existential quantifications with sums: $\sum_{x,y,z,t} \varphi(x, y, z, t)$, and computing in the natural semiring $(\mathbb{N}, +, \times)$ instead of the boolean semiring $(\{0, 1\}, \vee, \wedge)$.

⋆ Supported by FP7 Quasimodo, ANR-06-SETI-003 DOTS, ANR 2010 BLAN 0202 01 FREC, and ARCUS Île-de-France–Inde.

Assume now that the database also includes car dealers, see Fig. 1 (c). Each dealer records a list of performed sales. The following expression computes the maximal number of errors to be fixed per dealer:

$$\max_{d} \ \mathsf{Dealer}(d) \wedge \sum_{u} \Big[d <_{\mathrm{v}} u \wedge \mathsf{Car}(u) \wedge \sum_{x,y,z,t} \mathsf{Match}'(u,x) \wedge \varphi(x,y,z,t) \Big]. \tag{1}$$

Here, one needs to perform sums and max operations which are available in the max-plus semiring $(\mathbb{N} \cup \{-\infty\}, \max, +)$. Thus, by suitably choosing the semiring, various properties can be expressed.

In this paper, we introduce a calculus based on the above-mentioned concepts, which allows us to express quantitative properties in a declarative manner. When one considers related expressiveness and algorithmic issues, a natural question arises: is there an equivalent robust automata model? Our answer will be positive: we actually obtain a Kleene/Büchi-like correspondence, as in the boolean case. Towards a suitable automata device, we consider tree-walking automata with pebbles, for two reasons. First, weighted automata, the classical quantitative extensions of finite automata [15] are not expressive enough to encode quantitative expressions with products and sums, neither for trees, nor for words [5]. Actually, one can show that the max-plus expression above, computing the maximal number of errors to be fixed per dealer, cannot be computed by a weighted bottom-up tree automaton. Second, we are looking for a model that conforms with common query languages for trees, such as XPath or equivalent variants of first-order logic, aiming at a quantitative version of the latter and a suitable algorithmic framework. Indeed, tree-walking automata are an appropriate machine model for compiling XPath queries [3]. Moreover, enriched with pebbles, they capture exactly first-order logic with transitive closure [9]. It is the main goal of this paper to obtain a quantitative counterpart of this equivalence.

Contribution. Motivated by the above considerations, we introduce weighted expressions, a calculus to express quantitative properties over unranked trees. Contrarily to previous works (*e.g.*, [6,7]), our expressions allow for unrestricted nesting of sum and product. Weighted expressions may contain classical first-order formulas, which are convenient to describe boolean properties of a tree.

Second, we define (weighted) *pebble dfs automata* (DFSA). They can be considered as an extension of 1-way pebble automata over words, which have been studied in the context of infinite alphabets [12]. A DFSA traverses a tree according to a (not necessarily complete) depth-first-search (dfs) policy. This explains why we consider it to be 1-way. Unlike the notion of simple runs from [10], dfs traversals are a purely syntactic concept, which guarantees that the (quantitative) semantics of an automaton is well-defined. Moreover, dfs traversals are natural if we see unranked trees as a representation of XML documents, like in our example. In fact, they are closely related to the streaming of XML documents investigated in [16], where we can only visit an XML document with a single pass from the beginning to the end, and with finite memory. The dfs navigation is also natural when unranked trees represent nested words [1] (children of a node are encoded in the body of a call-return edge): a dfs navigation in the tree corresponds to a left-to-right walk in the nested word following either direct successor edges, or call-return edges.

Our main technical result states that DFSA and weighted expressions have the same expressive power. In addition to the theoretical interest that such a result enjoys, the automata-theoretic characterization of weighted expressions is the basis of an algorithmic study of quantitative properties over trees. In particular, we show that our automata can be evaluated wrt a tree in polynomial time.

Outline. The paper is structured as follows: in Section 2, we introduce our quantitative calculus of weighted expressions. Section 3 presents the automata model. In Section 4, we show that automata and weighted expressions are expressively equivalent when the latter are enriched by a quantitative extension of the transitive-closure operator. Algorithmic issues, such as the emptiness problem and complexity of evaluation are addressed in Section 5.

2 Weighted expressions

2.1 Definitions and examples

Trees. Let Σ be a finite alphabet. A finite unranked Σ-tree is a partial mapping $t : \mathbb{N}^* \rightharpoonup \Sigma$ whose domain $\mathrm{dom}(t)$ is nonempty, finite, prefix-closed, and such that $u(i+1) \in \mathrm{dom}(t)$ implies $ui \in \mathrm{dom}(t)$. Elements of $\mathrm{dom}(t)$ are called *nodes*. The size $|t|$ of t is its number of nodes. The *label* of a node u is $t(u)$. The empty sequence $\varepsilon \in \mathrm{dom}(t)$ is the *root* of t, and every maximal element in $\mathrm{dom}(t)$ is called a *leaf*. Non-leaf nodes are called *internal*. We denote by $\mathcal{T}(\Sigma)$ the set of Σ-trees. For $u, v \in \mathrm{dom}(t)$, we write

- $u \lessdot_{\mathrm{v}} v$ if $v = ui$ for some $i \in \mathbb{N}$: u is the *parent* of v and v a *child* of u.
- $u \lessdot_{\mathrm{h}} v$ if $u = wi$ and $v = w(i+1)$ with $i \in \mathbb{N}$: v is the *right sibling* of u, and u is the *left sibling* of v.

The rank $\mathsf{rk}(u)$ of u is its number of children. If $u \lessdot_{\mathrm{v}} v$ and v has no right sibling, we say that v is a *last child*. *First children* are defined symmetrically. For a binary relation $\rho \subseteq \mathrm{dom}(t) \times \mathrm{dom}(t)$, we denote by ρ^k its k-iteration, and $\rho^* = \bigcup_{k \geqslant 0} \rho^k$ its transitive reflexive closure. We let $<_{\mathrm{v}}$ and $<_{\mathrm{h}}$ be the transitive closures of $\lessdot_{\mathrm{v}}$ and $\lessdot_{\mathrm{h}}$, respectively. We define $\leqslant_{\mathrm{v}}$ and $\leqslant_{\mathrm{h}}$ as the corresponding order relations: $\leqslant_{\mathrm{v}} \stackrel{\text{def}}{=} \lessdot_{\mathrm{v}}^*$ and $\leqslant_{\mathrm{h}} \stackrel{\text{def}}{=} \lessdot_{\mathrm{h}}^*$. If $u <_{\mathrm{v}} v$, we say that u is an *ancestor* of v. For $x \in \mathrm{dom}(t)$, the *subtree of t rooted at x* is defined by $t_x(y) \stackrel{\text{def}}{=} t(xy)$.

Weighted expressions. A *semiring* is a structure $\mathbb{S} = (S, +, \times, \mathbf{0}, \mathbf{1})$ where $(S, +, \mathbf{0})$ is a commutative monoid, $(S, \times, \mathbf{1})$ is a monoid, $\times$ distributes over $+$, and $\mathbf{0}$ is absorbing for $\times$. We only consider *commutative* semirings, *i.e.*, such that $\times$ is commutative. A formal power series over trees is a mapping from $\mathcal{T}(\Sigma)$ into $\mathbb{S}$. See [11,2,14] for background on formal power series.

We fix an infinite supply of first-order variables $\mathrm{Var} = \{x, y, x_1, x_2, \ldots\}$. The set $\mathrm{FO}(\Sigma)$ (or simply FO) of first-order formulas over Σ is given by the grammar:

$$\varphi ::= P_a(x) \mid x \leqslant_{\mathrm{v}} y \mid x \leqslant_{\mathrm{h}} y \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x\, \varphi$$

where $a \in \Sigma$ and $x, y \in \mathrm{Var}$. Given a formula φ, a tree t, a finite set $\mathcal{V} \subseteq \mathrm{Var}$ containing all free variables of φ, and a $(\mathcal{V}, t)$-*assignment* as a mapping $\sigma : \mathcal{V} \to \mathrm{dom}(t)$, we write $t, \sigma \models \varphi$ if t satisfies φ under σ (where $P_a(x)$ means that the node at position x carries an a, the remaining notation being standard): see [17] for a detailed semantics. Note in particular that the semantics of φ only depends on the restriction of σ to free variables of φ in $\mathcal{V}$.

As usual, a $(\mathcal{V}, t)$-assignment $\sigma : \mathcal{V} \to \mathrm{dom}(t)$ is encoded in the alphabet $\Sigma_{\mathcal{V}} = \Sigma \times \{0,1\}^{\mathcal{V}}$. More precisely, we write a tree over $\Sigma_{\mathcal{V}}$ as a pair (t, σ) where t is the projection over Σ and σ is the projection over $\{0,1\}^{\mathcal{V}}$. Therefore, σ represents a *valid* assignment over $\mathcal{V}$ if for each variable $x \in \mathcal{V}$ the x-component of σ contains exactly one 1. In this case, we identify σ with the assignment such that for each variable $x \in \mathcal{V}$, $\sigma(x)$ is the position of the 1 in the x-component.

We write $\sigma[x \mapsto u]$ for the valuation coinciding with σ except on x, which it maps to u. We denote also by $[x \mapsto u, y \mapsto v]$ the valuation defined only on x, y, mapped to u, v, respectively. Since variables are interpreted by nodes in a tree, we will sometimes use letters $x, y, z, x_i, \ldots$ to also denote nodes (we use $u, v, \ldots$ for nodes if there is a possible confusion). We use macros $\mathsf{leaf}(x)$, $\mathsf{fc}(x)$, $\mathsf{lc}(x)$, $\mathsf{root}(x)$ checking respectively if x is a leaf, a first child, a last child, the root.

Definition 2.1. *Given a class $\mathcal{L}$ of boolean formulas, we denote by* $\mathrm{WE}(\mathcal{L})$ *the class of* weighted expressions over $\mathbb{S}$*, defined by*

$$E ::= s \mid \varphi \mid E \oplus E \mid E \otimes E \mid \bigoplus_x E \mid \bigotimes_x E \mid \circledast_{x,y}^{N\text{-dfs}} E \tag{2}$$

where $s \in \mathbb{S}$, $\varphi \in \mathcal{L}$, $x, y \in \mathrm{Var}$, and $N \in \mathbb{N}$. We let $\mathrm{WE}^-(\mathcal{L})$ be the fragment of $\mathrm{WE}(\mathcal{L})$ that does not make use of $\circledast_{x,y}^{N\text{-dfs}} E$.

The semantics of $E \in \mathrm{WE}(\mathcal{L})$, with free variables in $\mathcal{V}$, is a formal power series over $\Sigma_{\mathcal{V}}$-trees mapping each pair (t, σ), with σ a $(\mathcal{V}, t)$-assignment, to a value in S. We first define it for the fragment $\mathrm{WE}^-(\mathcal{L})$:

$$[\![s]\!](t,\sigma) = s \qquad [\![\varphi]\!](t,\sigma) = \begin{cases} \mathbf{1} & \text{if } t, \sigma \models \varphi \\ \mathbf{0} & \text{otherwise} \end{cases}$$

$$[\![E_1 \oplus E_2]\!] = [\![E_1]\!] + [\![E_2]\!] \qquad [\![E_1 \otimes E_2]\!] = [\![E_1]\!] \times [\![E_2]\!]$$

$$[\![\textstyle\bigoplus_x E]\!](t,\sigma) = \sum_{u \in \mathrm{dom}(t)} [\![E]\!](t, \sigma[x \mapsto u]) \qquad [\![\textstyle\bigotimes_x E]\!](t,\sigma) = \prod_{u \in \mathrm{dom}(t)} [\![E]\!](t, \sigma[x \mapsto u])$$

For a boolean formula $\psi \in \mathcal{L}$ and a weighted expression $E \in \mathrm{WE}(\mathcal{L})$, we define a macro $\psi \ominus E$, which is a natural generalization of the boolean implication:

$$\psi \ominus E \stackrel{\text{def}}{=} \neg\psi \oplus (\psi \otimes E).$$

Its semantics is $[\![E]\!]$ if ψ holds, and $\mathbf{1}$ otherwise.

Example 2.2. In the semiring $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$, the $\mathrm{WE}^-(\mathrm{FO})$ expression below computes the value (1) of the example of car dealers from the introduction:

$$\bigoplus_d \Big[\bigotimes_{u,x,y,z,t} \psi(d,u,x,y,z,t) \ominus 1 \Big]$$

where $\psi(d,u,x,y,z,t) = P_{\mathsf{Dealer}}(d) \wedge d <_{\mathrm{v}} u \wedge P_{\mathsf{Car}}(u) \wedge \mathsf{Match}'(u,x) \wedge \varphi(x,y,z,t)$. Observe that in the present case, $\mathbf{1}$ is the integer 0, so that if ψ holds (there is an error) then we count $[\![1]\!] = 1$, otherwise we count 0. □

Example 2.3. If we use for $\mathcal{L}$ the class of MSO-formulas, instead of FO-formulas, we can write a formula $\mathsf{even\text{-}depth}(x)$ that is satisfied by (t, σ) if and only if $\sigma(x)$ has an even depth. We can hence count in $(\mathbb{N}, +, \times)$ the number of nodes of a tree at even depth with the expression $\bigoplus_x \mathsf{even\text{-}depth}(x)$. It is well-known that we cannot express the property $\mathsf{even\text{-}depth}(x)$ in FO. In fact, we can also prove that the series $\bigoplus_x \mathsf{even\text{-}depth}(x)$ is not definable in $\mathrm{WE}^-(\mathrm{FO})$. □

We denote by $\mathrm{WE}^-(\mathrm{AP})$ the fragment of the class $\mathrm{WE}^-(\mathrm{FO})$, where AP stands for atomic propositions, and contains only the formulas $P_a(x)$, $x \leqslant_{\mathrm{v}} y$, $x \leqslant_{\mathrm{h}} y$ and their negations. This fragment is actually equivalent to the logic $\mathrm{FO}(\mathbb{S}, \Sigma)$ of [5] over words and of [6,7] over trees (ranked and unranked). Contrary to previous work, we cleanly separate the purely logical part from the constructs intended to perform computations in the semiring. Not only this makes the formalism more intuitive, but this also allows more flexibility to study expressiveness. For instance, one can investigate how the underlying logic influences the computational power.

Lemma 2.4. $\mathrm{WE}^-(\mathrm{AP}) = \mathrm{WE}^-(\mathrm{FO})$.

Proof. The inclusion $\mathrm{WE}^-(\mathrm{AP}) \subseteq \mathrm{WE}^-(\mathrm{FO})$ is clear. To prove the other direction, we have to prove that we can express disjunctions, conjunctions, existential quantifications and universal quantifications using the weighted operators. We adapt the construction of unambiguous formulas of [5], for the tree case. First, we define in $\mathrm{WE}^-(\mathrm{AP})$ an expression $x \leqslant y$ by

$$x \leqslant y \stackrel{\text{def}}{=} x \leqslant_{\mathrm{v}} y \oplus \bigoplus\nolimits_{z,z'} (z \leqslant_{\mathrm{v}} x \otimes z <_{\mathrm{h}} z' \otimes z' \leqslant_{\mathrm{v}} y)$$

For every tree t and every assignment $\sigma : \mathcal{V} \to \mathrm{dom}(t)$, we have $[\![x \leqslant y]\!](t,\sigma) \in \{\mathbf{0},\mathbf{1}\}$ and the binary relation over $\mathrm{dom}(t)$ made of pairs (u,v) such that $t, [x \mapsto u, y \mapsto v] \models x \leqslant y$ is a total order: we can check that this total order follows the dates of first visit in a depth-first-search of the tree.

Thanks to the order $\leqslant$, we can simulate disjunction by $\oplus$ and existential quantification by $\bigoplus_x$. More precisely, for each FO-formula φ, we effectively construct two weighted expressions $E^+_\varphi, E^-_\varphi \in \mathrm{WE}^-(\mathrm{AP})$ such that for every tree t and every $\sigma : \mathcal{V} \to \mathrm{dom}(t)$ we have $[\![E^+_\varphi]\!](t,\sigma), [\![E^-_\varphi]\!](t,\sigma) \in \{\mathbf{0},\mathbf{1}\}$,

$$[\![E^+_\varphi]\!](t,\sigma) = \mathbf{1} \iff t,\sigma \models \varphi \qquad \text{and} \qquad [\![E^-_\varphi]\!](t,\sigma) = \mathbf{1} \iff t,\sigma \not\models \varphi\,.$$

We define these expressions by structural induction on φ, removing ambiguity by picking the leftmost witness of the formulas (in particular in disjunction and existential quantification):

- if $\varphi \in \mathrm{AP}$ then $E^+_\varphi \stackrel{\text{def}}{=} \varphi$ and $E^-_\varphi \stackrel{\text{def}}{=} \neg\varphi$ (with $\neg\neg\psi = \psi$ by convention);
- $E^+_{\varphi\vee\psi} \stackrel{\text{def}}{=} E^+_\varphi \oplus (E^-_\varphi \otimes E^+_\psi)$ and $E^-_{\varphi\vee\psi} \stackrel{\text{def}}{=} E^-_\varphi \otimes E^-_\psi$;
- $E^+_{\neg\varphi} \stackrel{\text{def}}{=} E^-_\varphi$ and $E^-_{\neg\varphi} \stackrel{\text{def}}{=} E^+_\varphi$;
- $E^+_{\exists x\varphi} \stackrel{\text{def}}{=} \bigoplus_x E^+_\varphi \otimes \bigotimes_y [x \leqslant y \oplus (y < x \otimes E^-_\varphi)]$ and $E^-_{\exists x\varphi} \stackrel{\text{def}}{=} \bigotimes_x E^-_\varphi$. □

We now motivate and describe the full $\mathrm{WE}(\mathcal{L})$ calculus, including the new operator $\circledast^{N\text{-dfs}}_{x,y} E$. In the boolean semiring, the calculus $\mathrm{WE}^-(\mathrm{FO})$ coincides with first-order logic. Increasing its expressiveness can be done either by introducing second-order quantification, or by using a transitive closure operator. As shown in [5], second-order quantification is much too powerful in the weighted setting, with regard to any tractable computational model. We therefore enrich $\mathrm{WE}^-(\mathcal{L})$ with a weighted version of transitive closure. To explain intuitively what we want, consider a tree $t \in \mathcal{T}(\Sigma)$ and an expression $E \in \mathrm{WE}^-(\mathcal{L})$ with two free variables x and y. This defines a weighted graph G with set of vertices $V = \mathrm{dom}(t)$, set of edges $F = \{(x,y) \in V^2 \mid [\![E]\!](x,y) \neq \mathbf{0}\}$ and costs defined by $\lambda(x,y) = [\![E]\!](x,y)$. We would like to define a new weighted operator to compute, given vertices $u, v \in V$, the sum of the weights of paths leading from u to v in G (where the weight of a path is the product of the weight of its edges). Depending on the chosen semiring, this operator would capture several natural algorithmic properties of G: in $(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$, it computes the minimal costs of paths connecting u and v. In $(\mathbb{R}_{\geqslant 0}, +, \times, 0, 1)$, for suited E, we can interpret its outcome as the probability for reaching v starting from u. In $(\mathbb{N} \cup \{+\infty, -\infty\}, \max, \min, -\infty, +\infty)$, we obtain the maximal capacity of a path between u and v.

Let us formally define this weighted transitive closure $\circledast^{N\text{-dfs}}_{x,y} E$ computing such *path costs* in trees. Let E be a weighted expression with at least two free variables x and y (in the following, we will denote it by $E(x,y)$): we allow other free variables that we call parameters, and we consider two fresh variables x' and y' that are not parameters. We first define the *path cost* $[\circledast_{x,y} E](x',y')$ as a new expression whose semantics is

$$[\![\circledast_{x,y} E]\!](t, \sigma[x' \mapsto u, y' \mapsto v]) = \sum_{u=u_0,u_1,\ldots,u_{k-1},u_k=v} \prod_{i=0}^{k-1} [\![E]\!](t, \sigma[x \mapsto u_i, y \mapsto u_{i+1}]) \qquad (3)$$

with σ an assignment of the parameters of E, and where the sum ranges over all sequences $u = u_0, u_1, \ldots, u_{k-1}, u_k = v$ of nodes (with $k \geqslant 0$). Remark that in the boolean semiring, this definition coincides with the reflexive and transitive closure of a binary relation (see *e.g.*, [13]). For example, we can define the predicate $\mathsf{even\text{-}depth}(x')$ of Example 2.3 with the path cost $[\circledast_{x,y}\, y \lessdot_{\mathrm{v}}^{2} x](x', \varepsilon)$. Note that, in the semiring $(\mathbb{N}, +, \times, 0, 1)$, this expression always evaluates to 0 or 1, and $[\![\mathsf{even\text{-}depth}(x')]\!](t, [x' \mapsto u])$ evaluates to 1 iff u has even depth in the tree t. Hence, the weighted expression $\bigoplus_x \mathsf{even\text{-}depth}(x)$ computes the series defined in Example 2.3.

In general however, the previous sum ranges over an infinite set of sequences, so it may not be well defined. We therefore want to restrict it to a finite subset of all node sequences. One way to achieve this is to ensure *progression* in the node sequence: we shall enforce the nodes to belong to a depth-first-search path (dfs path) in the tree. To do so, let us first clarify the notion of path. To describe paths, we use $\Gamma = \{\uparrow, \swarrow, \rightarrow\}$ as alphabet of directions. Intuitively, $\uparrow$ means "go to the father", $\swarrow$ means "go to the first child", and $\rightarrow$ means "go to the next sibling". We call word over Γ a *path*. Note that a path just represents a way to navigate in a tree, but does not hold information about actual nodes in a tree.

Given a Σ-tree t and $u, v \in \mathrm{dom}(t)$, we write $u \xrightarrow{\delta} v$ if the word $\delta \in \Gamma^*$ leads from node u to node v in the tree: this is defined inductively on the length of δ by $u \xrightarrow{\varepsilon} u$ for every node u, and for every $d \in \Gamma$, $\delta \in \Gamma^*$ and $u, v \in \mathrm{dom}(t)$:

$$u \xrightarrow{d\delta} v \iff \exists w \in \mathrm{dom}(t), \quad w \xrightarrow{\delta} v \text{ and } \begin{cases} w \lessdot_{\mathrm{v}} u & \text{if } d = \uparrow, \\ u \lessdot_{\mathrm{v}} w \text{ and } \mathsf{fc}(w) & \text{if } d = \swarrow, \\ u \lessdot_{\mathrm{h}} w & \text{if } d = \rightarrow. \end{cases}$$

A *dfs path* is then simply a word of Γ^* that does not contain the factor $\uparrow\swarrow$. If δ is a dfs path such that $u \xrightarrow{\delta} v$ for $u, v \in \mathrm{dom}(t)$, then the length of δ is at most $2|t|$, since it is not possible to visit more than twice each node of the tree.

Given $u, v \in \mathrm{dom}(t)$, let $\mathsf{sp}(u, v) \in \Gamma^*$ be the shortest path from u to v in the directed graph with $\mathrm{dom}(t)$ as set of vertices, and edges naturally defined by the three possible moves $\uparrow$, $\rightarrow$ and $\swarrow$. More formally, we define it inductively by

$$\mathsf{sp}(u, v) = \begin{cases} \varepsilon & \text{if } u = v \\ \swarrow \cdot \mathsf{sp}(u.0, v) & \text{if } u <_{\mathrm{v}} v \\ \rightarrow^{j} \cdot \mathsf{sp}(u', v) & \text{if } u \lessdot_{\mathrm{h}}^{j} u' \leqslant_{\mathrm{v}} v \\ \uparrow \cdot \mathsf{sp}(u'', v) & \text{otherwise, with } u'' \lessdot_{\mathrm{v}} u\ . \end{cases}$$

For example, for the tree of Fig. 9, $\mathsf{sp}(15, 7) = \uparrow\uparrow\swarrow\rightarrow\swarrow$ (which is not a dfs path).

We now present the well-defined version of path cost used in this paper. Let $N > 0$ and let E be a weighted expression, with two free variables x and y. Define the bdfs-cost (bounded dfs path cost) $[\circledast_{x,y}^{N\text{-dfs}} E](x', y')$ as a new expression with semantics defined in (3) where the sum ranges now over all sequences $u = u_0, u_1, \ldots, u_{k-1}, u_k = v$ of nodes (with $k \geqslant 0$) such that, for $\delta_i \stackrel{\text{def}}{=} \mathsf{sp}(u_i, u_{i+1})$:

(*i*) $\delta = \delta_0 \cdots \delta_{k-1}$ is a dfs path and δ_i is nonempty for all $0 \leqslant i < k$,
(*ii*) for all $0 \leqslant i < k$, the length of δ_i is *at most* N.

By the above, (*i*) implies $|\delta| \leqslant 2|t|$. Since no δ_i can be empty also by (*i*), we have $k \leqslant 2|t|$ and the sum and product in (3) are finite: the semantics is well-defined. The additional restriction (*ii*) limits the computational power of this operator.

To shorten notation, we write $[\![F]\!](t, u, v)$, or $[\![F]\!](u, v)$ if t is understood, instead of $[\![F]\!](t, [x \mapsto u, y \mapsto v])$. We write $\circledast_{x,y}^{\text{bdfs}} E$ if the bound N is understood.

It is easy to check that if $\mathsf{sp}(u,v)$ is a nonempty dfs path, it has a very simple form, described by the following regular expression over Γ:

$$\text{sp-dfs} ::= \uparrow^+ \ \cup \ \uparrow^*\!\rightarrow(\swarrow \cup \rightarrow)^* \ \cup \ \swarrow(\swarrow \cup \rightarrow)^*.$$

For $N > 0$, we let N-sp-dfs be the set of words in sp-dfs of length at most N: hence the sum in (3) ranges in fact over all sequences $u = u_0, u_1, \ldots, u_{k-1}, u_k = v$ of nodes (with $k \geqslant 0$) such that $\delta_i = \mathsf{sp}(u_i, u_{i+1}) \in N$-sp-dfs for each $0 \leqslant i < k$ and $\delta = \delta_0 \cdots \delta_{k-1}$ is a dfs path.

Example 2.5. Consider XML files with tags in a finite alphabet Σ, representing actions. A *controller* receives such an XML file as a stream [16]. At each point, it can decide to visit the subtree of the current node of the stream (in the tree associated to the streamed document), or decide to skip that subtree. That way, the execution of a controller generates a trace, *i.e.*, a word of actions read during the execution. The execution corresponds actually to a dfs path in the tree. We may be interested only in controllers ensuring some additional properties, *e.g.*, such that the underlying trace reaches at least one leaf tagged `Goal` in the tree.

In the weighted setting, we can equip every action of Σ with a cost. For instance, in the semiring $(\mathbb{Z} \cup \{-\infty\}, \max, +, -\infty, 0)$, every action can be seen either as an energy consumption (negative cost) or as an energy refill (positive cost). Denote by energy such a mapping $\Sigma \to \mathbb{Z}$. Given a tree t, we are interested in the best possible controller ensuring property `Goal`, *i.e.*, the maximum, among all controllers ensuring property `Goal`, of the remaining energy at the end of the underlying trace. We can first design a weighted expression $\mathsf{energy}(x)$ that computes the cost of the action at node x: $\mathsf{energy}(x) \stackrel{\text{def}}{=} \bigoplus_{a \in \Sigma} P_a(x) \otimes \mathsf{energy}(a)$. We denote by $E(x,y)$ the weighted expression

$$\big[\big((x \lessdot_{\mathrm{v}} y \wedge \mathsf{fc}(y)) \vee x \lessdot_{\mathrm{h}} y\big) \otimes \mathsf{energy}(y)\big] \oplus \big[(\mathsf{lc}(x) \wedge y \lessdot_{\mathrm{v}} x) \otimes 0\big].$$

Consider $F(x',y') = [\circledast^{\text{1-dfs}}_{x,y} E(x,y)](x',y')$. Sequences $u_0, u_1, \ldots$ occurring in (3) to compute F have to satisfy $0 < |\mathsf{sp}(u_i, u_{i+1})| \leqslant N = 1$ by *(i)* and *(ii)*. Hence $u_i \xrightarrow{d_i} u_{i+1}$ for some direction $d_i \in \Gamma$. By definition of E, if $d_i \in \{\swarrow, \rightarrow\}$ (meaning a first visit of u_{i+1} along δ defined in *(i)*), then E compiles for this move the cost $\mathsf{energy}(u_{i+1})$, whereas if $d = \nwarrow$ (last visit), it compiles 0. Therefore, the optimal value we are interested in is computed by $\bigoplus_x (F(\varepsilon,x) \otimes P_{\texttt{Goal}}(x) \otimes \mathsf{leaf}(x) \otimes F(x,\varepsilon))$ (by convention, we take into account the energy of a node when visiting it for the first time, and the energy of the root is irrelevant).

We can also want to find the worst case, *i.e.*, to compute the minimum, among all controllers, of the remaining energy at the end of the underlying trace. For that, we simply change the semiring to $(\mathbb{Z} \cup \{+\infty\}, \min, +, +\infty, 0)$. Then, the expression $F(\varepsilon,\varepsilon)$ is appropriate. □

Example 2.6. The boolean predicate $\text{depth}(x') \equiv_\ell k$ stating that the depth of node x' is k modulo ℓ can be defined by the WE(FO) expression $\bigoplus_{y'} \varepsilon \lessdot_{\mathrm{v}}^{k} y' \otimes [\circledast^{\text{bdfs}}_{x,y} \ y \lessdot_{\mathrm{v}}^{\ell} x](x',y')$. Indeed, it is easy to verify that the semantics of this expression can only evaluate to $\mathbf{0}$ or $\mathbf{1}$ and defines exactly the predicate $\text{depth}(x') \equiv_\ell k$. The $\circledast^{\text{bdfs}}_{x,y}$ follows a dfs and is bounded since the shortest path between nodes x and y is the word $\uparrow^\ell$. Note that, these jumps would not be bounded if we replace the move $\uparrow$ by a move $\nwarrow$ in the alphabet Γ, *i.e.*, if we enforce the up moves to appear only at a last child. □

Remark 2.7. We allow $k = 0$ in (3). Hence, when $u = v$, the empty path is possible and the associated product is empty and evaluates to the unit $\mathbf{1}$. But there may be other paths in the sum visiting the subtree of node $u = v$. Hence, $[\![\circledast^{N\text{-dfs}}_{x,y} E]\!](u,u)$ is $\mathbf{1}$ plus the sum over the nonempty paths ($k > 0$) visiting the subtree of u.

2.2 First properties of the bdfs-costs

In this section, we study some useful properties of bdfs-costs that will be used in Section 4.

Lemma 2.8. *Let $x_0, x_1, \dots, x_k$, and let $\delta_i = \mathsf{sp}(x_i, x_{i+1})$ for $i = 0, \dots, k-1$. If $\delta_0\delta_1\cdots\delta_{k-1}$ is a dfs path, then $\mathsf{sp}(x_0, x_k) \in \text{sp-dfs} \cup \{\varepsilon\}$.*

Proof. If $k = 0$, then $\mathsf{sp}(x_0, x_0) = \varepsilon$. If $k = 1$, then $\mathsf{sp}(x_0, x_1) = \delta_0$ is a dfs-path by assumption, so it is in sp-dfs $\cup \{\varepsilon\}$ by definition of sp-dfs. For $k \geqslant 2$, we prove by induction on k that if $\delta = \delta_0\delta_1\cdots\delta_{k-1}$ is a dfs path, then:

(*a*) $\mathsf{sp}(x_0, x_k) \in \text{sp-dfs} \cup \{\varepsilon\}$, and

(*b*) if $\mathsf{sp}(x_0, x_k)$ ends with $\uparrow$, then so does δ_{k-1}.

For $k = 2$, let $x = x_0$, $y = x_1$ and $z = x_2$. If $\mathsf{sp}(x, y)\mathsf{sp}(y, z)$ is a dfs path, then $\uparrow\swarrow$ does not occur in $\mathsf{sp}(x, y)\mathsf{sp}(y, z)$. In consequence, we have $\mathsf{sp}(x, y), \mathsf{sp}(y, z) \in \text{sp-dfs}$. We split sp-dfs in its three components $\text{sp-dfs}_1 = \uparrow^+$, $\text{sp-dfs}_2 = \uparrow^*\rightarrow(\swarrow \cup \rightarrow)^*$ and $\text{sp-dfs}_3 = \swarrow(\swarrow \cup \rightarrow)^*$. Since $\mathsf{sp}(x, y) \in \text{sp-dfs}$ and $\mathsf{sp}(y, z) \in \text{sp-dfs}$, we have nine cases and we can easily check that

	$\mathsf{sp}(y,z) \in \text{sp-dfs}_1$	$\mathsf{sp}(y,z) \in \text{sp-dfs}_2$	$\mathsf{sp}(y,z) \in \text{sp-dfs}_3$
$\mathsf{sp}(x,y) \in \text{sp-dfs}_1$	$\mathsf{sp}(x,z) \in \text{sp-dfs}_1$	$\mathsf{sp}(x,z) \in \text{sp-dfs}_2$	impossible
$\mathsf{sp}(x,y) \in \text{sp-dfs}_2$	$\mathsf{sp}(x,z) \in \text{sp-dfs}_1 \cup \text{sp-dfs}_2$	$\mathsf{sp}(x,z) \in \text{sp-dfs}_2$	$\mathsf{sp}(x,z) \in \text{sp-dfs}_2$
$\mathsf{sp}(x,y) \in \text{sp-dfs}_3$	$\mathsf{sp}(x,z) \in \text{sp-dfs}_1 \cup \text{sp-dfs}_3 \cup \{\varepsilon\}$	$\mathsf{sp}(x,z) \in \text{sp-dfs}_2 \cup \text{sp-dfs}_3$	$\mathsf{sp}(x,z) \in \text{sp-dfs}_3$

The case $\mathsf{sp}(x, y) \in \text{sp-dfs}_1$ and $\mathsf{sp}(y, z) \in \text{sp-dfs}_3$ is impossible, since otherwise $\mathsf{sp}(x, y)\mathsf{sp}(y, z)$ would contain $\uparrow\swarrow$ as a factor, contradicting the hypothesis. Hence, $\mathsf{sp}(x, z) \in \text{sp-dfs} \cup \{\varepsilon\}$, and we see that if $\mathsf{sp}(x, z)$ ends with $\uparrow$ (*i.e.*, belongs to sp-dfs_1), then so does $\mathsf{sp}(y, z)$.

Let now $k > 2$ and a dfs path $\delta = \delta_0\delta_1\cdots\delta_{k-1}$. We claim that $\mathsf{sp}(x_0, x_{k-1})\mathsf{sp}(x_{k-1}, x_k)$ is a dfs path. Indeed, δ is a dfs path, so are its factors $\delta_0\delta_1\cdots\delta_{k-2}$ and δ_{k-1}. By induction, $\mathsf{sp}(x_0, x_{k-1}) \in \text{sp-dfs} \cup \{\varepsilon\}$ is also a dfs path. Now, if $\mathsf{sp}(x_0, x_{k-1})$ ends with $\uparrow$, then by induction so does δ_{k-2}, and since δ is a dfs path, δ_{k-1} does not start with $\swarrow$. This proves the claim. Using the already proved case $k = 2$, we finally obtain (*a*) and (*b*). ◻

Corollary 2.9. *Let $E(x, y)$ be a weighted expression. If $[\![\circledast_{x,y}^{N\text{-dfs}} E]\!](x', y') \neq \mathbf{0}$, then $\mathsf{sp}(x', y') \in \text{sp-dfs} \cup \{\varepsilon\}$.*

Proof. By definition, we have

$$[\![\circledast_{x,y}^{N\text{-dfs}} E]\!](x', y') = \sum_{x'=x_0,x_1,\dots,x_{k-1},x_k=y'} \prod_{i=0}^{k-1} [\![E]\!](x_i, x_{i+1})$$

where the sum ranges over all sequences $x' = x_0, x_1, \dots, x_{k-1}, x_k = y'$ of nodes (with $k \geqslant 0$) such that $\delta_i \stackrel{\text{def}}{=} \mathsf{sp}(x_i, x_{i+1}) \in N\text{-sp-dfs}$ for each $0 \leqslant i < k$, and $\delta = \delta_0\cdots\delta_{k-1}$ is a dfs path. If $[\![\circledast_{x,y}^{N\text{-dfs}} E]\!](x', y') \neq \mathbf{0}$, there exists at least one such sequence $x' = x_0, x_1, \dots, x_{k-1}, x_k = y'$. By Lemma 2.8, we get $\mathsf{sp}(x', y') \in \text{sp-dfs} \cup \{\varepsilon\}$. ◻

Notice that some properties of the reflexive and transitive closure operator in the boolean setting are not true anymore in the weighted case. For example, in the standard setting, it holds that $E^* = \mathbf{1} + EE^*$. This does not always generalize to bdfs-cost expressions: the expressions $F = [\![\circledast_{x,y}^{N\text{-dfs}} E]\!](x', z')$ and $F' = (x' = z') \oplus \bigoplus_{y'} E(x', y') \otimes [\![\circledast_{x,y}^{N\text{-dfs}} E]\!](y', z')$ are not equivalent in general, even if the sum ranges over those y' which are connected to x' by an N-sp-dfs. The intuition would be that we might split the bdfs-cost of F at some intermediary node y' along paths from x' to z'. The main difference is that the last step from x' to y' in F' may be $\uparrow$, while the first step of the bdfs-cost in F' may be $\swarrow$. This is forbidden by the global constraint (no factor $\uparrow\swarrow$) on the bdfs-cost of F.

To remedy this issue, we define a subclass of weighted expressions that will verify this property. We say that an expression $E(x,y)$ is *progressing* if for all x, y, z

$$[\![E]\!](x,y) \neq \mathbf{0} \neq [\![E]\!](y,z) \implies \mathsf{sp}(x,y)\mathsf{sp}(y,z) \text{ is a dfs path} \tag{4}$$

Lemma 2.10. *Let $E(x,y)$ be a progressing expression. Then for all x, x', y, y', z, z', we have*

1. $[\![\circledast_{x,y}^{N\text{-dfs}} E]\!](z,z) = \mathbf{1}$,
2. $[\circledast_{x,y}^{N\text{-dfs}} E](x',z') \equiv (x'=z') \oplus \bigoplus_{y' \mid \mathsf{sp}(x',y') \in N\text{-sp-dfs}} E(x',y') \otimes [\circledast_{x,y}^{N\text{-dfs}} E](y',z')$.

Proof. 1. Consider $[\![\circledast_{x,y}^{N\text{-dfs}} E]\!](z,z)$. Let $z = u_0, u_1, \dots, u_{k-1}, u_k = z$ be a sequence of nodes (with $k \geqslant 0$) such that $\delta_i \stackrel{\text{def}}{=} \mathsf{sp}(u_i, u_{i+1}) \in N$-sp-dfs for each $0 \leqslant i < k$ and the factor $\uparrow\swarrow$ does not occur in $\delta = \delta_0 \cdots \delta_{k-1}$. If $k = 1$, then $\delta_0 = \mathsf{sp}(z,z) = \varepsilon \notin N$-sp-dfs, which contradicts the assumption. If $k > 1$ then $\delta_0 \in \swarrow(\swarrow + \rightarrow)^*$ and $\delta_{k-1} \in \uparrow^+$. Hence, the factor $\uparrow\swarrow$ occurs in $\delta_{k-1}\delta_0 = \mathsf{sp}(u_{k-1}, z)\mathsf{sp}(z, u_1)$. By (4) we deduce that $[\![E]\!](u_{k-1}, z) \times [\![E]\!](z, u_1) = \mathbf{0}$. Therefore, $\prod_{0 \leqslant i < k} [\![E]\!](u_i, u_{i+1}) = \mathbf{0}$. By Remark 2.7, we know that this product is $\mathbf{1}$ when $k = 0$. Hence, $[\![\circledast_{x,y}^{N\text{-dfs}} E]\!](z,z) = \mathbf{1}$.
2. By distributivity, we obtain from (3)

$$[\![\circledast_{x,y}^{N\text{-dfs}} E]\!](x',z') = [\![x'=z']\!] + \sum_{y'} \Big([\![E]\!](x',y') \times \sum_{\substack{y'=x_1,x_2,\dots,\\ x_{k-1},x_k=z'}} \prod_{1 \leqslant i < k} [\![E]\!](x_i, x_{i+1}) \Big)$$

where $x_0 = x'$, $x_1 = y'$, $x_2, \dots, x_{k-1}$, $x_k = z'$ is a sequence of nodes such that $\delta_i = \mathsf{sp}(x_i, x_{i+1}) \in N$-sp-dfs for each $0 \leqslant i < k$ and $\delta_0 \cdots \delta_{k-1}$ is a dfs path. If $\delta_0 \cdots \delta_{k-1}$ is not a dfs path, there exists $0 \leqslant j < k-1$ such that $\delta_j\delta_{j+1}$ is not a dfs path. Since E is progressing, by (4), we obtain $[\![E]\!](x_j, x_{j+1}) \times [\![E]\!](x_{j+1}, x_{j+2}) = \mathbf{0}$, and finally $\prod_{0 \leqslant i < k} [\![E]\!](x_i, x_{i+1}) = \mathbf{0}$. Hence, we can remove this constraint and we obtain the desired formula. ◻

3 Weighted navigational automata

In the semiring $(\mathbb{N}, +, \times)$, we can compute the tree series that maps a tree t of size n to 2^{n^2} with the expression $E \stackrel{\text{def}}{=} \bigotimes_x \bigotimes_y 2$. However, as noticed in [5], the classical model of weighted automata cannot recognize such a series (even over words). We will therefore add features to classical navigational automata in order to be able to deal with $\mathrm{WE}^-(\mathrm{FO})$ (and even $\mathrm{WE}(\mathrm{FO})$).

Before defining an operational model to compute the weight of a tree, we first recall the model of pebble tree-walking automata (PA) [8,3]. Then, we extend PA to a weighted setting, considering only simple (*i.e.*, non-looping) runs. Next, we give sufficient syntactical restrictions, which lead to a new class of *1-way* PA, which we call *pebble depth-first-search automata* (DFSA).

Definition 3.1 (Pebble tree-walking automaton). *Let $r \geqslant 0$. An r-*PA *is a tuple $\mathcal{A} = (Q, \Sigma, I, \Delta)$ such that*

- *Q is a finite set of states,*
- *Σ is a finite alphabet,*
- *$I \subseteq Q$ is the set of initial states,*
- *$\Delta \subseteq Q \times \Sigma \times 2^D \times 2^{\{1,\dots,r\}} \times M \times Q$ is the set of transitions, where $D = \{\swarrow, \nwarrow, \rightarrow, \leftarrow\}$ is the set of* directions *and $M = D \cup \{\mathit{accept}, \mathit{drop\text{-}reset}, \mathit{resume\text{-}lift}\}$ is the set of* moves.

Intuitively, a run of an r-PA navigates in a tree. The special moves in $M \setminus D$ are used to add a new pebble on the current head position, to remove the last dropped pebble, or to accept. A 0-PA is a classical tree-walking automaton.

Formally, let t be a tree in $\mathcal{T}(\Sigma)$. Define the *type* $\tau(u)$ of a node $u \in \mathrm{dom}(t)$ as the set of directions in $\{\swarrow, \nwarrow, \rightarrow, \leftarrow\}$ available at u in t: $\swarrow \in \tau(u)$ iff u has a child, $\nwarrow \in \tau(u)$ iff u is a last child, $\rightarrow \in \tau(u)$ iff u has a right sibling, and $\leftarrow \in \tau(u)$ iff u has a left sibling. Note that $\nwarrow$ and $\rightarrow$ are mutually exclusive.

A word $\pi = u_1 \cdots u_k \in \mathrm{dom}(t)^{\leqslant r}$, with $k \leqslant r$ and $u_i \in \mathrm{dom}(t)$ for every $1 \leqslant i \leqslant k$, encodes the positions of the dropped pebbles: pebble $\ell \in \{1, \ldots, k\}$ is currently on node u_ℓ. Given $u \in \mathrm{dom}(t)$ and such a word π, we define $\mathsf{Peb}(\pi, u)$ to be the set of pebbles currently dropped on u: $\mathsf{Peb}(u_1 \cdots u_k, u) = \{\ell \in \{1, \ldots, k\} \mid u_\ell = u\}$.

A *configuration* of an r-PA $\mathcal{A}$ over the tree t is a triple (q, u, π) with $q \in Q$, $u \in \mathrm{dom}(t)$, and $\pi \in \mathrm{dom}(t)^{\leqslant r}$. A sequence $\rho = (q_m, u_m, \pi_m)_{0 \leqslant m \leqslant n}$ of configurations is said to be a *run* if there exists a sequence of moves $(d_m)_{0 \leqslant m \leqslant n-1}$ such that, for every $0 \leqslant m \leqslant n-1$, both $(q_m, t(u_m), \tau(u_m), \mathsf{Peb}(\pi_m, u_m), d_m, q_{m+1}) \in \Delta$ and one of the following holds:

$$\left.\begin{array}{l}
1.\ d_m = \swarrow \text{ and } u_{m+1} = u_m 0 \text{ and } \pi_{m+1} = \pi_m; \\
2.\ d_m = \nwarrow \text{ and } u_m = u_{m+1} i \text{ and } \nwarrow \in \tau(u_m) \text{ and } \pi_{m+1} = \pi_m; \\
3.\ d_m = \rightarrow \text{ and } u_m \lessdot_{\mathrm{h}} u_{m+1} \text{ and } \pi_{m+1} = \pi_m; \\
4.\ d_m = \leftarrow \text{ and } u_{m+1} \lessdot_{\mathrm{h}} u_m \text{ and } \pi_{m+1} = \pi_m; \\
5.\ d_m = \text{drop-reset and } u_{m+1} = \varepsilon \text{ and } \pi_{m+1} = \pi_m u_m; \\
6.\ d_m = \text{resume-lift and } u_m = \varepsilon \text{ and } \pi_{m+1} u_{m+1} = \pi_m; \\
7.\ d_m = \text{accept and } u_m = u_{m+1} = \varepsilon \text{ and } \pi_{m+1} = \pi_m = \varepsilon.
\end{array}\right\} \quad (5)$$

At each moment of the run, the head of the PA points to the position u_m of the current configuration. Note that dropping a pebble resets the head of the PA to the root of the tree ($u_{m+1} = \varepsilon$ in 5 above), whereas lifting the last dropped pebble may only occur at the root ($u_m = \varepsilon$ in 6) and restores the head of the PA to the position where the last pebble was dropped ($\pi_{m+1} u_{m+1} = \pi_m$ in 6). Finally, the accept move (7) is only applied at the root with no dropped pebbles. It is easy to verify that cases 1 to 7 of (5) are disjoint, so that if ρ is a run, there is a unique sequence of moves (d_m) that verifies (5).

A run $\rho = (q_m, u_m, \pi_m)_{0 \leqslant m \leqslant n}$ is *accepting* if $u_0 = \varepsilon$, $q_0 \in I$, $\pi_0 = \varepsilon$ and the last move d_{n-1} is an accept move.

Example 3.2. Assume that Σ is the union (disjoint or not) of two alphabets Σ_i and Σ_ℓ. Call a tree *well-labeled* if each internal node is labeled by a letter in Σ_i and each leaf is labeled by a letter in Σ_ℓ. The *frontier* of such a tree is the word in Σ_ℓ^+ obtained by concatenating the labels of all leaves, from left to right. Given a regular language $L \subseteq \Sigma_\ell^*$, consider the language of all well-labeled trees whose frontiers belong to L. It is folklore that for ranked trees, this tree language is recognizable by a bottom-up tree automaton. For unranked trees, there is a 0-PA (*i.e.*, a tree-walking automaton) that recognizes this tree language. If L is recognized by the deterministic finite automaton $\mathcal{B} = (P, \Sigma_\ell, p_0, P_f, \delta)$, with P a finite set of states, $p_0 \in P$ the initial state, $P_f \subseteq P$ the set of final states, and $\delta : P \times \Sigma_\ell \to P$ the (deterministic and complete) transition function, we define a 0-PA $\mathcal{A} = (Q, \Sigma, I, \Delta)$ with $Q = P \times \{1, 3\} \cup \{q_f\}$ (we use extra

bits of information 1 and 3 to be consistent with further developments), $I = \{(p_0, 1)\}$ and

$$\begin{aligned}\Delta = &\big\{((p,1), a, \tau, \swarrow, (p,1)) \mid p \in P \wedge a \in \Sigma_i \wedge \swarrow \in \tau\big\}\\ &\cup \big\{((p,1), a, \tau, \rightarrow, (\delta(p,a),1)) \mid p \in P \wedge a \in \Sigma_\ell \wedge \swarrow \notin \tau \wedge \rightarrow \in \tau\big\}\\ &\cup \big\{((p,1), a, \tau, \nwarrow, (\delta(p,a),3)) \mid p \in P \wedge a \in \Sigma_\ell \wedge \swarrow \notin \tau \wedge \nwarrow \in \tau\big\}\\ &\cup \big\{((p,3), a, \tau, \rightarrow, (p,1)) \mid p \in P \wedge \rightarrow \in \tau\big\}\\ &\cup \big\{((p,3), a, \tau, \nwarrow, (p,3)) \mid p \in P \wedge \nwarrow \in \tau\big\}\\ &\cup \big\{((p,1), a, \emptyset, \mathrm{accept}, q_f) \mid p \in P \wedge a \in \Sigma_\ell \wedge \delta(p,a) \in P_f\big\}\\ &\cup \big\{((p,3), a, \{\swarrow\}, \mathrm{accept}, q_f) \mid p \in P_f\big\}.\end{aligned}$$

As no pebbles are used, we omit the $2^{\{1,\dots,r\}}$ part of the transitions. Intuitively, if the tree is well-formed, we visit each internal node of the tree exactly twice, and each leaf exactly once. At first visit (extra bit 1), we verify that the label follows the type requirements for the tree to be well-formed. Moreover, when leaving a leaf, we update the P-component of the state, to simulate the computation of the deterministic automaton $\mathcal{B}$. At second visit (extra bit 3), we simply pass through the nodes following a depth-first-search of the tree. The last two sets of transitions are used to accept the tree, depending on whether the tree is reduced to its root or not. □

We extend PA by adding weights to transitions. A weighted r-PA over the semiring $\mathbb{S}$ (r-wPA($\mathbb{S}$)) is a tuple $\mathcal{A} = (Q, \Sigma, I, \Delta, \mathsf{weight})$ such that (Q, Σ, I, Δ) is an r-PA and weight is a function that maps each transition of Δ to a value in S. The weight of a run $\rho = (q_m, u_m, \pi_m)_{0 \leqslant m \leqslant n}$ on a tree t with induced $(d_m)_{0 \leqslant m \leqslant n-1}$ is

$$\mathsf{weight}(\rho) = \prod_{0 \leqslant m \leqslant n-1} \mathsf{weight}(q_m, t(u_m), \tau(u_m), \mathsf{Peb}(\pi_m, u_m), d_m, q_{m+1}) \ .$$

The semantics of $\mathcal{A}$ over a tree t is $[\![\mathcal{A}]\!](t) = \sum_\rho \mathsf{weight}(\rho)$ where the sum is over *simple* accepting runs over t, *i.e.*, which do not use twice the same configuration. As there are only finitely many simple accepting runs on a finite tree, the semantics is well defined.

Example 3.3. (continued from Example 2.3) We design a 0-wPA over the semiring $(\mathbb{N}, +, \times, 0, 1)$ computing the number of nodes at even depth. Each accepting run computes the weight 1: its shape is an even succession of $\swarrow$ moves, interleaved with $\rightarrow$ moves, followed by a non-deterministic choice to stop exploration and reset the head to the root. Hence we use non-determinism to count the number of positions where it is possible to stop the exploration, which are exactly nodes at even depth. In Fig. 2, we depict the automaton, where every transition has weight 1. □

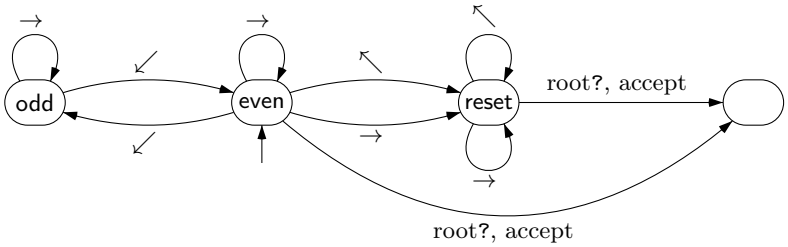

Fig. 2: 0-wPA computing the number of nodes at even depth

We now introduce a syntactic restriction, ensuring the simplicity of runs of a PA. Basically, we restrict to *1-way* runs following a *depth-first-search* (dfs) traversal of the tree. Without pebbles, it suffices to disable the left moves $\leftarrow$ and to forbid reversals $\nwarrow\swarrow$ in the walk, which we can enforce by remembering whether the last move was $\nwarrow$ or not. We will see below what is the suitable restriction in the presence of pebbles.

We show through examples that, even when restricted to dfs runs, (weighted) PA can check/compute interesting properties/values. First, the 0-PA presented in Examples 3.2 and 3.3 fulfill the dfs restriction. We show in the following example that we can check with a 1-PA using only dfs runs the validity of a boolean formula using or and and gates of arbitrary arity. This is a classical example on unranked trees requiring one pebble even for PA which are not restricted to dfs runs.

Example 3.4. We follow ideas of [3, Example 2]. Over the alphabet $\Sigma = \{\mathsf{ff}, \mathsf{tt}, \mathsf{or}, \mathsf{and}\}$, every unranked tree with leafs labeled by ff or tt, and internal node labeled by or or and represents a boolean sentence (with unranked boolean connectives "or" and "and"): these trees are called well-formed. Using Example 3.2, there exists a 0-PA that recognizes exactly the set of well-formed trees.

We describe in Fig. 3 a 1-PA recognizing a set of trees such that its restriction to well-formed trees contains exactly all those trees that represent valid sentences (node labels are explained in Example 3.10 below). It is convenient to use in such examples new moves which are compositions of basic ones. First, we allow the move $\uparrow$, which is obtained by taking move $\rightarrow$ until the last child is reached and then taking move $\nwarrow$. Second, the move drop, whose semantics is to drop a pebble at the current position without resetting the head to the root as in drop-reset. The drop move can be simulated easily: first, make a drop-reset, and then search the last dropped pebble, *e.g.*, with a depth-first-search of the tree. Similarly, we can simulate a resume-lift occurring at an arbitrary node by first walking up to the root of the tree with $\uparrow^*$ and then applying resume-lift.

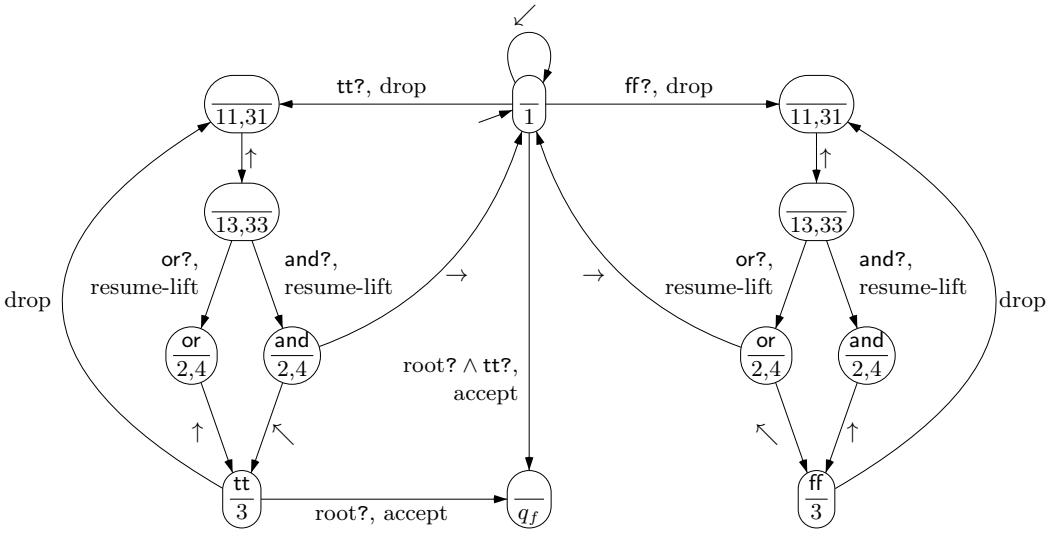

Fig. 3: 1-PA recognizing well-formed trees representing valid sentences

The automaton of Fig. 3 starts by moving down to the leftmost leaf. Then, it drops a pebble on the current leaf and, depending on its label (tt or ff), enters one of the 2 dual gadgets of the automaton. Each gadget moves up to check the label (or or and) of the father of the marked node. We use this information to determine how to continue the computation: either we have

enough information to know the truth value of the father and we enter the corresponding state (tt or ff) or we need to visit the next child and we re-enter the initial state. □

We describe now what we mean by 1-way PA. To ensure simplicity in the presence of pebbles, we have to disable the left moves ← and to forbid reversals $\nwarrow\swarrow$ as previously, but we also have to forbid consecutive resume-lift/drop-reset and to remember whether the last move before a drop-reset was $\nwarrow$ or not, so that after the corresponding resume-lift we may forbid $\swarrow$ if needed.

Formally, for $r \geqslant 0$, we define the *universal* r-PA $\mathcal{A}_r^{\mathrm{dfs}} = (Q_r, \{1\}, \Delta_{\mathrm{dfs}})$ with $Q_r = \{q_f\} \cup (\{1,3\}^{\leqslant r} \cdot \{1,2,3,4\})$ and $\Delta_{\mathrm{dfs}} = \Delta_{\nwarrow} \cup \Delta_{\swarrow} \cup \Delta_{\rightarrow} \cup \Delta_{\mathrm{drop}} \cup \Delta_{\mathrm{lift}} \cup \Delta_{\mathrm{accept}}$, where

$$\begin{aligned}
\Delta_{\nwarrow} &= \{(q.e, \nwarrow, q.3) \mid e \in \{1,2,3,4\}\} & \Delta_{\swarrow} &= \{(q.e, \swarrow, q.1) \mid e \in \{1,2\}\} \\
\Delta_{\rightarrow} &= \{(q.e, \rightarrow, q.1) \mid e \in \{1,2,3,4\}\} & \Delta_{\mathrm{drop}} &= \{(q.e, \text{drop-reset}, q.e.1) \mid e \in \{1,3\}\} \\
\Delta_{\mathrm{lift}} &= \{(q.e.f, \text{resume-lift}, q.(e+1)) \mid e \in \{1,3\}, f \in \{1,2,3,4\}\} \\
\Delta_{\mathrm{accept}} &= \{(e, \text{accept}, q_f) \mid e \in \{1,2,3,4\}\}.
\end{aligned}$$

Since they are useless, we omit in Δ_{dfs} the components $\Sigma \times 2^D \times 2^{\{1,\dots,r\}}$ of transitions. Note that the ← move is disabled, that a $\nwarrow$ move cannot be immediately followed by a $\swarrow$ move, and that a resume-lift move cannot be immediately followed by a drop-reset move. Moreover, if we see moves from M as inputs, the automaton $\mathcal{A}_r^{\mathrm{dfs}}$ is deterministic.

Lemma 3.5. *Let t be a tree and let $(d_m)_{0 \leqslant m \leqslant n-1}$ with $n > 1$ be a sequence of moves in $\{\swarrow, \rightarrow, \nwarrow\}$ leading from some node $u \in \mathrm{dom}(t)$ back to itself. If the sequence contains no reversal,* i.e.*, if there are no $m < n-1$ with $d_m = \nwarrow$ and $d_{m+1} = \swarrow$, then $d_0 = \swarrow$ and $d_{n-1} = \nwarrow$.*

Proof. By induction on n (for every node u). □

As a corollary of Lemma 3.5, one can show that each run of $\mathcal{A}_0^{\mathrm{dfs}}$ follows a dfs exploration of the tree, possibly cutting some subtrees, and every node is visited at most twice. To handle pebbles, the state of $\mathcal{A}_r^{\mathrm{dfs}}$ is a stack of height at most $r+1$. The top of the stack records whether, in the dfs, we are currently in a descending phase (1 or 2), and whether a pebble has just been lifted (2 or 4).

Fact 3.6. *Along any run of $\mathcal{A}_r^{\mathrm{dfs}}$, the length of the word representing the current state is exactly one plus the number of pebbles currently dropped.*

We use this fact to prove that $\mathcal{A}_r^{\mathrm{dfs}}$ has a useful deletion property.

Lemma 3.7. *Let $\rho = (q_m, u_m, \pi_m)_{0 \leqslant m \leqslant n}$ be a run of $\mathcal{A}_r^{\mathrm{dfs}}$ over some tree t. Assume that there exist k and ℓ such that $k+1 < \ell < n$, $\pi_k = \pi_\ell$ and π_k is a strict prefix of π_m for all $k < m < \ell$. Then the concatenation of the sequences $(q_m, u_m, \pi_m)_{0 \leqslant m \leqslant k}$ and $(q_m, u_m, \pi_m)_{\ell+1 \leqslant m \leqslant n}$ is still a run of $\mathcal{A}_r^{\mathrm{dfs}}$ over t.*

Proof. Let $(d_m)_{0 \leqslant m \leqslant n-1}$ be the sequence defined by (5) for ρ. We have to prove that the configurations (q_k, u_k, π_k) and $(q_{\ell+1}, u_{\ell+1}, \pi_{\ell+1})$ are *compatible*.

First, since π_k is a strict prefix of π_{k+1}, we have $\pi_{k+1} = \pi_k u_k$, $d_k = \text{drop-reset}$, $q_k = q.e$ with $e \in \{1,3\}$. Similarly, as π_ℓ is a strict prefix of $\pi_{\ell-1}$, we have $d_{\ell-1} = \text{resume-lift}$ and we deduce $q_\ell = q.(e+1)$ using Fact 3.6. As $e+1 \in \{2,4\}$, we get $d_\ell \neq \text{drop-reset}$ from the definition of Δ_{dfs}. From $(q_\ell, d_\ell, q_{\ell+1}) \in \Delta_{\mathrm{dfs}}$ we deduce that $(q_k, d_\ell, q_{\ell+1}) \in \Delta_{\mathrm{dfs}}$.

Next, as $\pi_k = \pi_\ell$ is a strict prefix of every π_m for $k < m < \ell$ and $\pi_{k+1} = \pi_k u_k$, we obtain $\pi_{\ell-1} = \pi_k u_k$. Using $d_{\ell-1} = \text{resume-lift}$ we finally get $u_\ell = u_k$. Since ρ is a run, d_ℓ is *the* move from $(q_\ell, u_\ell, \pi_\ell)$ to $(q_{\ell+1}, u_{\ell+1}, \pi_{\ell+1})$ according to (5). We deduce that d_ℓ is also *the* move from (q_k, u_k, π_k) to $(q_{\ell+1}, u_{\ell+1}, \pi_{\ell+1})$ according to (5). This proves the statement. □

Proposition 3.8. *All runs of $\mathcal{A}_r^{\mathrm{dfs}}$ are simple.*

Proof. By contradiction, assume that there is a non simple run $\rho = (q_m, u_m, \pi_m)_{0 \leqslant m \leqslant n}$ over a tree t, whose length may be chosen minimal so that $(q_0, u_0, \pi_0) = (q_n, u_n, \pi_n)$. Also assume that $|\pi_0|$ is minimal among the non simple runs of minimal length. Let $(d_m)_{0 \leqslant m \leqslant n-1}$ be the sequence of moves defined by (5).

As a first case, suppose that $\pi_0 = \pi_n = \varepsilon$. If $\pi_k = \varepsilon$ for all $0 \leqslant k \leqslant n$, then by Lemma 3.5, we have $d_0 = \swarrow$ and $d_{n-1} = \nwarrow$. Therefore, $q_0 \in \{1, 2\}$ and $q_n = 3$, a contradiction with $q_0 = q_n$. So let k be minimal such that $\pi_{k+1} \neq \varepsilon$, and let then $\ell > k+1$ be minimal with $\pi_\ell = \varepsilon = \pi_k$.

- If $\ell < n$, we obtain using Lemma 3.7 a shorter non simple run, contradicting the choice of ρ.
- If $\ell = n$, then $u_k = \pi_{k+1} = \pi_{n-1} = u_n$. As $u_0 = u_n$, the sequence of moves $d_0, \dots, d_{k-1}$ is a path in t from node u_0 to node $u_k = u_0$. Moreover, this sequence does not contain any drop-reset nor resume-lift (by minimality of k), nor consecutive $\nwarrow$ and $\swarrow$ moves (by definition of Δ_{dfs}). If $k \geqslant 1$, then by Lemma 3.5, $d_0 = \swarrow$ and $d_{k-1} = \nwarrow$. By definition of Δ_{dfs}, we get $q_0 \in \{1, 2\}$ and $q_k = 3$. By minimality of ℓ, we have $d_{\ell-1} =$ resume-lift and $q_\ell = q_k + 1 = 4$, so $q_n = 4 \neq q_0$, a contradiction. Therefore, $k = 0$ (the sequence $d_0, \dots, d_{k-1}$ is empty). Since $d_0 =$ drop-reset, $q_0 \in \{1, 3\}$ and since $d_{n-1} =$ resume-lift, $q_n \in \{2, 4\}$, again in contradiction with $q_0 = q_n$.

As a second case, suppose that $\pi_0 = \pi_n \neq \varepsilon$.

- If there is no k such that $\pi_k = \varepsilon$, then for every $m \in \{0, \dots, n\}$ we have $\pi_m = u\pi'_m$ and $q_m = qq'_m$. Then the sequence $\rho' = (q'_m, u_m, \pi'_m)_{0 \leqslant m \leqslant n}$ is a run which is non simple, a contradiction with the minimality of $|\pi_0|$.
- Otherwise, let $k > 0$ be minimal with $\pi_k = \varepsilon$ and $k \leqslant \ell < n$ be maximal with $\pi_\ell = \varepsilon$. Necessarily, we have $d_{k-1} =$ resume-lift and $d_\ell =$ drop-reset. Since Δ_{dfs} ensures that a resume-lift move cannot be immediately followed by a drop-reset move, we have $k < \ell$. By minimality of k, we have $\pi_0 = u_k\pi'_0$ and $q_0 = (q_k - 1)q'_0$ with $q_k - 1 \in \{1, 3\}$ so $q_k \in \{2, 4\}$. By maximality of ℓ, we have $\pi_n = u_\ell\pi'_n$ and $q_n = q_\ell q'_n$. Since $\pi_0 = \pi_n$ and $q_0 = q_n$, we get $u_k = u_\ell$ and $q_k - 1 = q_\ell$.
  Therefore, $k + 1 < \ell$ (one cannot have simultaneously $\ell = k + 1$, $u_\ell = u_k$ and $\pi_\ell = \pi_k$). Hence for all $k < m < \ell$ we have $\pi_m = \varepsilon$, otherwise we could find $k', \ell' \in \{k, \dots, \ell\}$ fulfilling the hypotheses of Lemma 3.7 to shorten the run ρ, contradicting its minimality.
  In conclusion, the sequence of moves $d_k, \dots, d_{\ell-1}$ that goes from node u_k to node $u_\ell = u_k$ satisfies the assumptions of Lemma 3.5, so $d_k = \swarrow$ and $d_{\ell-1} = \nwarrow$. From $d_k = \swarrow$ and $q_k \in \{2, 4\}$, we conclude that $q_k = 2$. From $d_{\ell-1} = \nwarrow$, we conclude that $q_\ell = 3$. This contradicts $q_k - 1 = q_\ell$. □

Synchronizing an r-PA with $\mathcal{A}_r^{\mathrm{dfs}}$ yields a tree-walking automaton, which in particular has no $\nwarrow$ immediately followed by a $\swarrow$, and no drop-reset immediately followed by a resume-lift. We call the resulting automaton a *1-way* PA, also called a *depth-first-search automaton* (DFSA). We do not formally describe the notion of synchronization, but we define DFSA by using a notion of simulation.

Definition 3.9. *Let $r \geqslant 0$. An r-pebble depth-first-search automaton (r-*DFSA*) is an r-*PA *$\mathcal{A} = (Q, \Sigma, I, \Delta)$ for which there exists a relation $R \subseteq Q \times Q_r$ that verifies*

1. *for every $q \in I$, $(q, 1) \in R$;*
2. *for every $(p, p') \in R$ and for every $(p, a, \tau, P, d, q) \in \Delta$, there exists $q' \in Q_r$ such that $(q, q') \in R$ and $(p', d, q') \in \Delta_{\mathrm{dfs}}$.*

Example 3.10. (continued from Example 3.4) The 1-PA that recognizes the set of well-formed trees such that the underlying boolean sentence is valid is in fact a 1-DFSA: in Fig. 3, the bottom part of the nodes contains the states of Q_r associated with each state of Q in the relation R. In particular, the special move drop can still be simulated by a DFSA, using a dfs exploration after resetting the automaton, to find the last dropped pebble. □

Proposition 3.8 can be used in particular to prove that all accepting runs of a DFSA are *simple*.

Lemma 3.11. *Let $\mathcal{A}$ be an r-PA such that there exists a relation $R \subseteq Q \times Q_r$ verifying the previous properties. Then every accepting run of $\mathcal{A}$ is simple.*

Proof. By contradiction, let $\rho = (q_m, u_m, \pi_m)_{0 \leqslant m \leqslant n}$ be an accepting, non simple run of $\mathcal{A}$: there exist $0 \leqslant i < n$ and $k \geqslant 1$ such that $(q_i, u_i, \pi_i) = (q_{i+k}, u_{i+k}, \pi_{i+k})$. By a pumping argument, one constructs an accepting run $\tilde{\rho}$ of $\mathcal{A}$ that follows ρ until index i, then repeats $|Q_r|$ times the nonempty part $(q_m, u_m, \pi_m)_{i+1 \leqslant m \leqslant i+k}$ of ρ, and finally follows the remaining part of ρ until the accepting state. Write $\tilde{\rho} = (\tilde{q}_m, \tilde{u}_m, \tilde{\pi}_m)_{0 \leqslant m \leqslant K}$. By construction,

$$(\tilde{u}_i, \tilde{\pi}_i) = (\tilde{u}_{i+k}, \tilde{\pi}_{i+k}) = (\tilde{u}_{i+2k}, \tilde{\pi}_{i+2k}) = \cdots = (\tilde{u}_{i+|Q_r|k}, \tilde{\pi}_{i+|Q_r|k}). \tag{6}$$

Using the simulation relation R, one constructs inductively from $\tilde{\rho}$ a run $\rho' = (q'_m, \tilde{u}_m, \tilde{\pi}_m)_{0 \leqslant m \leqslant K}$ of $\mathcal{A}_r^{\mathrm{dfs}}$ on the same tree, that follows the same path as $\tilde{\rho}$ in that tree. By the pigeonhole principle, there are two identical states in the sequence $(q'_{i+\ell k})_{0 \leqslant \ell \leqslant |Q_r|}$, hence by (6), the run ρ' of $\mathcal{A}_r^{\mathrm{dfs}}$ is not simple. This contradicts Proposition 3.8. □

Definition 3.12. *Let $r \geqslant 0$. A weighted r-*DFSA *over the semiring $\mathbb{S}$ (r-*wDFSA$(\mathbb{S})$*) is an r-*wPA$(\mathbb{S})$ *$\mathcal{A} = (Q, \Sigma, I, \Delta, \mathsf{weight})$ such that (Q, Σ, I, Δ) is an r-*DFSA*.*

Previous lemma shows that the semantics of wDFSA does not require the restriction to simple runs: for an r-wDFSA $\mathcal{A}$, we have $[\![\mathcal{A}]\!](t) = \sum_{\rho \text{ accepting run over } t} \mathsf{weight}(\rho)$.

Remark 3.13. Our syntactic enforcement of simplicity to PA depends on the choice of $\mathcal{A}_r^{\mathrm{dfs}}$. Other choices are possible, *e.g.*, dfs walks from right to left. Our choice is mainly motivated by the analogy with the word case and the streaming of XML documents.

Example 3.14. (continued from Example 2.5) In $(\mathbb{Z} \cup \{-\infty\}, \max, +, -\infty, 0)$, a 0-wDFSA $\mathcal{A}$ recognizing the series defined by $\bigoplus_x (F(\varepsilon, x) \otimes P_{\texttt{Goal}}(x) \otimes \mathsf{leaf}(x) \otimes F(x, \varepsilon))$ is shown in Fig. 4, where the weights energy mean that we compute the weight regarding the label of the current node (since costs are computed after leaving a node, there are energy weights even on some $\nwarrow$ transitions).

To find the best controller reaching a `Goal` at a leaf, taking additionally into account more properties and energy costs at the moment where the `Goal` is visited, we can drop a pebble at a `Goal` node, and compute a weight generated by a wDFSA $\mathcal{B}$, as shown in Fig. 5. Notice also that if a weighted expression $G(x)$ recognizes the series defined by $\mathcal{B}$, then the weighted expression $\bigoplus_x (F(\varepsilon, x) \otimes P_{\texttt{Goal}}(x) \otimes G(x) \otimes \mathsf{leaf}(x) \otimes F(x, \varepsilon))$ defines the desired series. □

Example 3.15. Let $\Sigma = \{\mathsf{and}, \mathsf{neg}, \alpha\}$. Consider boolean formulas with connectors and (unranked) and neg (unary). Letter α stands for a variable. An unranked tree is now well-formed if nodes labeled by α are exactly its leaves, and nodes labeled by neg have a single child. A well-formed Σ-tree represents a formula in which distinct leaves correspond to distinct propositional variables. The 1-wDFSA of Fig. 6 counts the number of valuations of the variables that make the formula true, where $\mathcal{A}$ is a 0-wDFSA that can *see* a fixed pebble and computes 2^n with n

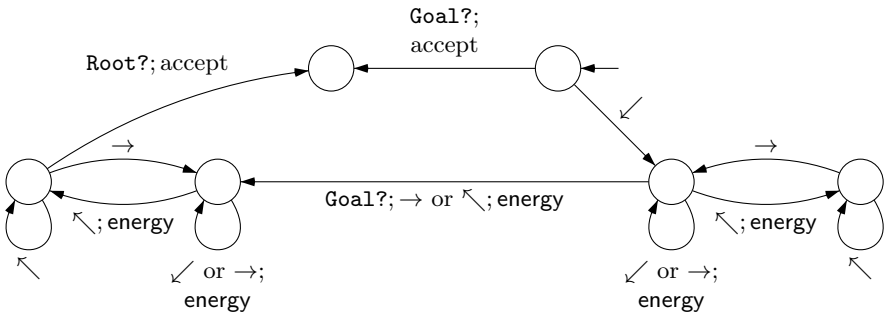

Fig. 4: 0-wDFSA computing the best controller reaching `Goal` at a leaf

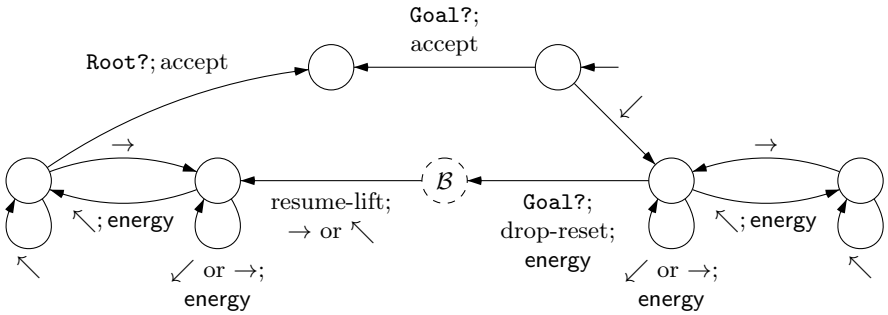

Fig. 5: 1-wDFSA computing the best controller reaching a `Goal` tag (revisited)

the number of leaves of the subtree of the node carrying the pebble. When omitted, the weight of a transition is 1. Only the self loop labeled `neg?` has a weight (-1) different from 1.

Note that if $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_k$ has pairwise distinct variables, the number of valid valuations of φ is the product of the numbers of valid valuations of all formulas φ_i. Thanks to distributivity, this product can be computed by *linking* the computations for the children of the corresponding `and` node. Similarly, the number of valid valuations of $\neg\psi$ is $2^n - d$ where n is the number of variables in ψ (*i.e.*, the number of leaves of the underlying tree) and d the number of valid valuations of ψ. The value $2^n - d$ is computed using nondeterminism from the initial state: runs starting with the -1 weighted transition compute $-d$, while runs starting by the `neg?`, drop transition compute 2^n, thanks to the 0-wDFSA $\mathcal{A}$. □

4 Expressiveness result

This section will be devoted to the proof of the main theorem of this paper, that relates weighted expressions in WE(FO) to depth-first-search automata.

Theorem 4.1. *The class of tree series definable in* WE(FO) *is exactly the class of tree series recognizable by* wDFSA*.*

4.1 From WE(FO) to wDFSA

We prove first that every tree series definable by a WE(FO) expression can be recognized by a wDFSA. We will describe wDFSA using a high-level pseudo-code. We use simple tests, *e.g.*, `a?` to check the letter of the current position, `Root?` or `LastChild?` to check whether the current position is the root or a last child, and "$x = 1$" to check whether the x-component of a $\Sigma_{\mathcal{V}}$-tree contains a 1. Finally `Reset` moves the head of the automaton back to the root of the tree.

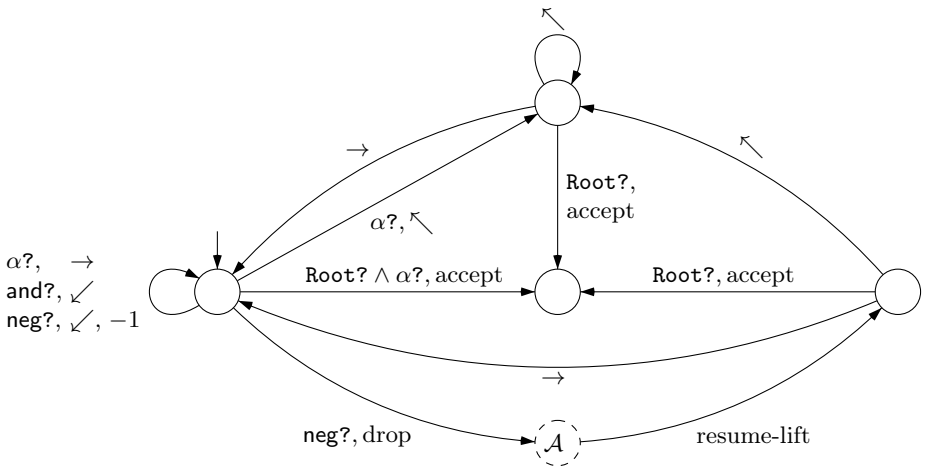

Fig. 6: 1-wDFSA computing the number of valid valuations

In the following, we assume that we only deal with valid assignments. If we deal with expressions without free variables, this suffices to directly yield the result. Otherwise, a simple 0-wDFSA recognizes exactly the series mapping a tree to **1** if it represents a valid assignment, and to **0** otherwise. In Fig. 7, we give the 0-wDFSA checking the valid assignment with one free variable x. Observe that the automaton essentially consists of two identical parts, each of which performs a sequence of "next" moves in the dfs traversal of the tree: the $x = 1$, $x \neq 1$ tests are performed while leaving a node just after its first visit. Note also that by definition of 0-wDFSA, the two accept transitions can only be performed at the root.

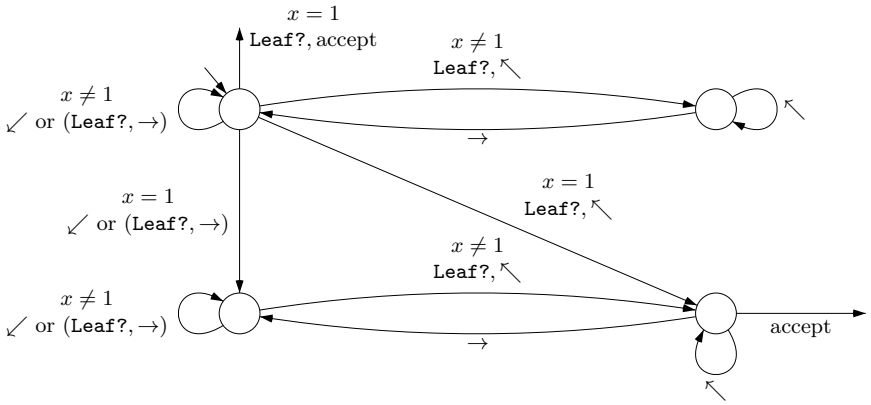

Fig. 7: 0-wDFSA that checks validity of assignment with one free variable x

A way to explain this automaton is to add a macro next to move to the next node in the dfs traversal, which is simply the first child for an internal node, and, for a leaf, we move up $\nwarrow$ until a right move $\rightarrow$ is possible (we assume that next applied to the very last leaf leads to the root): in a Conditional XPath-like syntax, next can be written $\swarrow \cup \texttt{Leaf?} \cdot \nwarrow^* \cdot (\texttt{Root?} \cup \rightarrow)$. Then, a 0-DFSA following a complete dfs of the tree can be written `Repeat` next `Until Root?`. The automaton is simply a synchronized product of this automaton with an automaton that only checks that the sequence of read letters during a run contains exactly one 1 in the x-component.

For several free variables, combining such automata using the construction below for $\otimes$ produces the desired wDFSA.

First, the atomic propositions are implemented by

$$
\begin{array}{ll}
P_a(x): & \texttt{Repeat } (\swarrow \text{ or } \rightarrow) \texttt{ Until } a? \wedge x = 1; \texttt{ Reset} \\
x <_{\mathrm{v}} y: & \texttt{Repeat } (\swarrow \text{ or } \rightarrow) \texttt{ Until } x = 1; \swarrow; \\
 & \texttt{Repeat } (\swarrow \text{ or } \rightarrow) \texttt{ Until } y = 1; \texttt{ Reset} \\
\neg(x <_{\mathrm{v}} y): & \texttt{Repeat } (\swarrow \text{ or } \rightarrow) \texttt{ Until } y = 1; \\
 & \texttt{Repeat } \rightarrow^*; \nwarrow; \neg(x = 1) \texttt{ Until Root?} \\
x <_{\mathrm{h}} y: & \texttt{Repeat } (\swarrow \text{ or } \rightarrow) \texttt{ Until } x = 1; \rightarrow; \\
 & \texttt{Repeat } \rightarrow \texttt{ Until } y = 1; \texttt{ Reset} \\
\neg(x <_{\mathrm{h}} y): & \texttt{Repeat } (\swarrow \text{ or } \rightarrow) \texttt{ Until } x = 1; \\
 & \texttt{While } \neg\texttt{LastChild? Do } \rightarrow; \neg(y = 1) \texttt{ Od; Reset}
\end{array}
$$

and $\neg P_a(x)$ is equivalent to $\bigoplus_{b \neq a} P_b(x)$. One can check that these pieces of pseudo-code can indeed be written as wDFSA. For instance, the wDFSA implementing $x <_{\mathrm{v}} y$ is drawn in Fig. 8 (a).

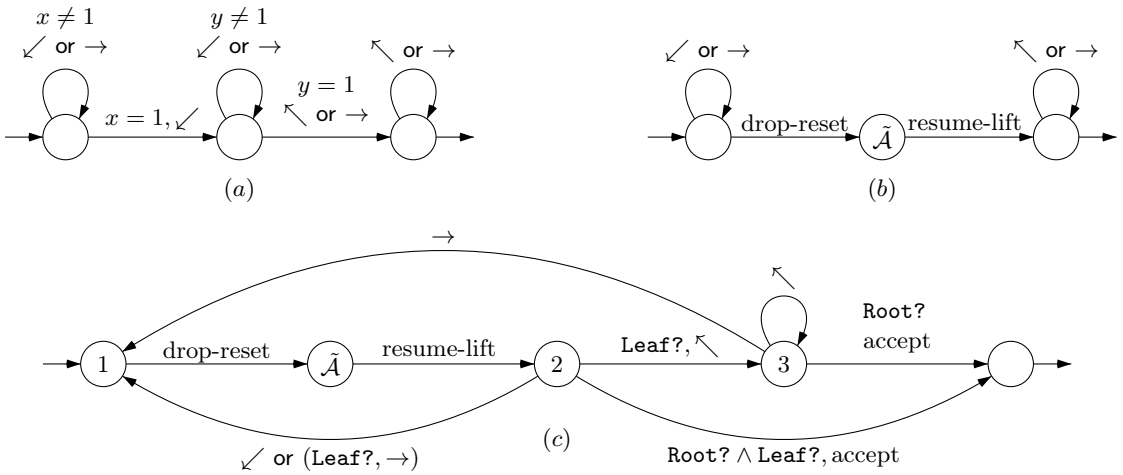

Fig. 8: Automata constructions (a) for $x <_{\mathrm{v}} y$, (b) for $\bigoplus_x E$, and (c) for $\bigotimes_x E$

Since WE(FO) = WE(AP) (simple corollary of Lemma 2.4), it remains to deal with $\oplus$, $\otimes$, $\bigoplus_x$, $\bigotimes_x$ and $\circledast^{\mathrm{bdfs}}_{x,y} E$. Let $E, E_1, E_2 \in \mathrm{WE(FO)}$ define tree series recognized respectively by r-wDFSA $\mathcal{A}$, $\mathcal{A}_1$, and $\mathcal{A}_2$. As usual, $E_1 \oplus E_2$ is recognized by the disjoint union of $\mathcal{A}_1$ and $\mathcal{A}_2$. For the product $E_1 \otimes E_2$, notice that we cannot as usual use a synchronized product of the two automata, as in navigational automata, the two automata may follow inconsistent walks in the trees: even in their one-way version, one automaton could cut one subtree whereas the other one must visit it. Instead, we use one additional pebble:

$$\mathcal{A}_1 \otimes \mathcal{A}_2 : \quad \text{drop-reset; } \mathcal{A}_1\text{; resume-lift; } \swarrow\text{; drop-reset; } \mathcal{A}_2\text{; resume-lift; } \texttt{Reset}$$

if the tree is not reduced to its root. Otherwise we use one of the automata, depending of the letter of the root: $\texttt{Leaf?} \wedge a?; [\![\mathcal{A}_1]\!](a) \times [\![\mathcal{A}_2]\!](a)$.

The automaton $\bigoplus_x \mathcal{A}$ for $\bigoplus_x E$, represented in Fig. 8 (b), chooses non deterministically, following a dfs, a position x in the tree, drops a pebble on this position, launches $\mathcal{A}$ (interpreting

the free variable x as the position of the last dropped pebble, the automaton is written $\tilde{\mathcal{A}}$ in the figure), and, when the computation of $\mathcal{A}$ ends, it lifts the pebble and resets the head to the root.

For the product $\bigotimes_x E$, we use the automaton depicted in Fig. 8 (c), that drops the pebble exactly once at every node of the tree. To do so, it visits the whole tree remembering at every step if the current node is visited for the first time (states labeled 1 and 2) or for the second time (state labeled 3): the pebble is only dropped when visiting a node for the first time. Using the next move introduced previously, we can write it in pseudo-code by

$$\texttt{Repeat}\ \text{drop-reset};\ \tilde{\mathcal{A}};\ \text{resume-lift};\ \text{next};\ \texttt{Until Root?}$$

Finally the tree series defined by $[\circledast_{x,y}^{N\text{-dfs}} E](x', y')$ is recognizable by an $(r+2)$-wDFSA $\mathcal{B}$. We use two fresh pebbles to mark positions of variables x and y in automaton $\mathcal{A}$. We first find non deterministically the variable x' encoded in the alphabet. After dropping the pebble on this position x, we resume and come back to this very same node. We now guess the next factor δ_i satisfying (*i*) and (*ii*), store it in the states of $\mathcal{B}$ (which is possible since $|\delta_i| \leqslant N$) and *follow* this path with the automaton: notice that this may need an unbounded number of steps because of the direction $\uparrow$ which is not allowed in the automata, but that we can simulate with $\rightarrow^*\nwarrow$. We finally reach node y on which we drop the second pebble. After resuming, we can simulate a run of automaton $\mathcal{A}$ where the variables x and y are marked by the pebbles. At the end of the computation, we resume and lift both pebbles (first that in position y, then that in position x). The automaton can then follow the same path δ_i from x to reach again position y. Note that we actually reach the same node y from x following the stored path δ_i. Dropping the first pebble on this position now, we can continue the computation of the bdfs-cost, until we reach position y'. Using pseudo-code, we can describe the automaton by

$$\begin{aligned}
&\texttt{While}\ \neg x' = 1\ \texttt{Do}\ \text{next};\\
&\texttt{Repeat}\ \ \texttt{Guess}\ \delta \in N\text{-sp-dfs};\ \texttt{Store}\ \delta;\ \text{drop};\ \texttt{Follow}\ \delta;\\
&\qquad\quad \text{drop-reset};\ \mathcal{A};\ \text{resume-lift};\ \texttt{Reset};\ \text{resume-lift};\ \texttt{Follow}\ \delta;\\
&\texttt{Until}\ y' = 1;\ \texttt{Reset}
\end{aligned}$$

Notice that the translation from a weighted expression E in WE(AP) yields an automaton with a number of states linear in the size of E (the size of E is the common size of all derivation trees of the grammar (2) needed to derive E), and exponential in the maximal size of bounds N occurring in the bdfs-costs of E.

The rest of this section is devoted to the proof of the remaining inclusion of Theorem 4.1, from automata to expressions. We have to solve three main difficulties. The last one will be to deal with pebbles. For this we will use a rather easy induction. The main problems occur already for a 0-wDFSA $\mathcal{A}$. First, we will show how to compute with a bdfs-cost operation $\circledast^{\text{bdfs}}$ the sum of the weights of all runs of $\mathcal{A}$ following a *complete* dfs in the tree, that does not skip any subtree. In a second phase, we will consider all dfs runs, including those that cut some subtrees.

4.2 From 0-wDFSA to expressions: the case of no-cut dfs runs

Let $\mathcal{A} = (Q, \Sigma, I, \Delta, \mathsf{weight})$ be a 0-DFSA. We show in this section how to construct from $\mathcal{A}$ a WE(FO) expression E computing, for each tree t, the sum of the weights of all runs of $\mathcal{A}$ over t following the unique path performing a *complete* dfs, which goes from the root to the root in t. We first formally define walks, and the complete dfs.

Let $\Sigma_D = \{(a, \tau, d) \in \Sigma \times 2^D \times D \mid d \in \tau \text{ and } d \neq \leftarrow\}$ be the set of possible *typed moves*. Recall that $D = \{\swarrow, \rightarrow, \leftarrow, \nwarrow\}$, hence here we use $\nwarrow$ rather than $\uparrow$ as was done for the sp-dfs

expressions. Compared to paths, walks hold additional information (node labels and node types). We call *dfs walk* a walk such that for each factor $(a,\tau,d)(a',\tau',d')$, we have $dd' \neq \nwarrow\swarrow$. A *no-cut dfs walk* is a word $w \in \Sigma_D^*$ such that for each factor $(a,\tau,d)(a',\tau',d')$ of w we have $dd' \neq \nwarrow\swarrow$ and if $\swarrow \in \tau'$ and $d \neq \nwarrow$ then $d' = \swarrow$. Hence, a no-cut dfs walk cannot cut any subtree.

We write $\mathrm{walk}_w(x,y)$ if the dfs walk w leads from node x to node y in the tree. Formally, this is defined by induction on the length of w: $\mathrm{walk}_\varepsilon(x,y) \stackrel{\text{def}}{=} (x=y)$ and $\mathrm{walk}_{(a,\tau,d)w}(x,y) \stackrel{\text{def}}{=} P_a(x) \wedge \tau(x) \wedge \exists x' \big[\mathrm{mv}_d(x,x') \wedge \mathrm{walk}_w(x',y)\big]$ where $\tau(x)$ checks that the possible moves at x are exactly those in τ and $\mathrm{mv}_d(x,x')$ states that moving in direction d from x leads to x'. These two formulas can easily be expressed in FO. We finally denote the *complete* no-cut dfs walk by w^c: it is the unique no-cut dfs walk w such that $\mathrm{walk}_w(\varepsilon,\varepsilon)$.

Before explaining the proof outline, we first observe that one can use a convenient matrix notation to compute weights of runs. Define $\mathrm{Runs}(p,x,w,q,y)$ as the set of runs ρ starting from node x in state p, following the fixed dfs walk w, and reaching node y in state q. Formally, such runs are of the form $\rho = (q_0,x_0,\emptyset)\cdots(q_k,x_k,\emptyset)$ with $(q_0,x_0)=(p,x)$, $(q_k,x_k)=(q,y)$ and such that the projection of w on its third component is $d_0\cdots d_{k-1}$, where $(d_i)_{0\leqslant i\leqslant k-1}$ is defined from ρ by (5). Since the shape w is fixed, these runs only differ by their intermediate states $q_1,\dots,q_{k-1}$. In order to compute the sum of the weights of all runs in $\mathrm{Runs}(p,x,w,q,y)$ we introduce the morphism $\mu : \Sigma_D^* \to S^{Q\times Q}$ defined by $\mu(a,\tau,d)_{p,q} = \mathsf{weight}(p,a,\tau,d,q)$.

Lemma 4.2. *Let $w \in \Sigma_D^*$ be a dfs walk, let $p,q \in Q$ and let x,y be two nodes such that $\mathrm{walk}_w(x,y)$ holds. Then,*

$$\mu(w)_{p,q} = \sum_{\rho\in\mathrm{Runs}(p,x,w,q,y)} \mathsf{weight}(\rho).$$

Proof. Easy induction on the length of w. □

Proof outline and difficulties. Since in this first step we are only interested in computing the weight of the runs following the complete no-cut dfs traversal of the tree, we have to define a WE(FO) expression $F_{p,q}$ which computes $\mu(w^c)_{p,q}$. The idea is to split the complete no-cut dfs walk $w^c = w_0w_1\cdots w_{m-1}$ into factors w_i of bounded length, so that $\mu(w^c)_{p,q} = \bigoplus_{p_1,\dots,p_{m-1}} \mu(w_0)_{p_0,p_1}\otimes\cdots\otimes\mu(w_{m-1})_{p_{m-1},p_m}$ (with $p_0=p$ and $p_m=q$) can be computed with a bdfs-cost operation.

We let $Q=\{0,\dots,n-1\}$. The nodes reached after each factor w_i of the complete no-cut dfs walk will be called *split nodes*: $\varepsilon = x'_0 \xrightarrow{\delta_0} x'_1\cdots x'_{m-1} \xrightarrow{\delta_{m-1}} x'_m = \varepsilon$ where δ_i is the third component of w_i. We have to encode the states p_i at split nodes x'_i. Usually this is done by choosing split nodes whose distances from the origin are multiples of the number $n=|Q|$ of states, and using *jump nodes* x_i in the bdfs-cost iteration which are at distance p_i of x'_i.

The problem with trees is that each internal node is visited twice by the complete no-cut dfs walk w so there may be 4 distances between two nodes depending on whether we assume a first visit or a last visit for each node. For instance, a dfs walk between node 2 and node 6 of the tree in Fig. 9 could be of length $0+2+0$ or $5+2+0$ or $0+2+7$ or $5+2+7$ depending on whether we visit the subtrees of node 2 and of node 6, or not. One has to remove this ambiguity, otherwise the bdfs-cost could take into account too many paths. We do so by partitioning the (internal) nodes into two sets, depending on whether we will use them during the first or second visit. Indeed, one cannot always use jump nodes at their first visit, since otherwise from some node we may have to walk up ($\nwarrow$) an unbounded number of steps before moving right ($\rightarrow$) to the next available node, which would result in some unbounded $\delta_i \in$ sp-dfs.

The main difficulties are therefore to define split nodes and jump nodes in WE(FO), and then to ensure that each run is captured once by the bdfs-cost expression.

The $\leqslant_{\mathsf{dfs}}$ ordering. To formally define split nodes and associated jump nodes to be used in a bdfs-cost operator, we build a relation $\leqslant_{\mathrm{dfs}} \subseteq \mathrm{dom}(t) \times \mathrm{dom}(t)$ enjoying the following properties:

(i) $\leqslant_{\mathrm{dfs}}$ is a total order on $\mathrm{dom}(t)$ whose minimal element is the root ε,
(ii) $\lessdot_{\mathrm{dfs}} \stackrel{\text{def}}{=} <_{\mathrm{dfs}} \setminus <^2_{\mathrm{dfs}}$ is (FO+depth)-definable[3],
(iii) If $x \lessdot^{\ell}_{\mathrm{dfs}} y$, then $\mathsf{sp}(x, y) \in 4\ell$-sp-dfs.

Item *(i)* will allow us to define split nodes as nodes whose $\lessdot_{\mathrm{dfs}}$-distance from the root is 0 modulo n. Recall that $\mathsf{lc}(x)$ means that node x is a last child. Node x is called an *up-node* if it satisfies

$$\mathsf{up}(x) \stackrel{\text{def}}{=} \neg\mathsf{leaf}(x) \wedge \mathsf{lc}(x) \wedge (\mathrm{depth}(x) \equiv_2 1). \tag{7}$$

In a bdfs-cost expression, the first move of a no-cut dfs walk from an up-node must be $\nwarrow$: intuitively, this means that an up-node will only be used at its second visit. On the other hand, if x is not an up-node then the first move of a no-cut dfs walk from x must be $\swarrow$ if possible, or else $\rightarrow$ if possible, or else $\nwarrow$ if possible. To reflect this condition, we set

$$\underline{\mathrm{walk}}_w(x, y) = \mathrm{walk}_w(x, y) \wedge \mathrm{check}_w(x, y),$$

where $\mathrm{check}_w(x, y)$ checks that the first move of w conforms to the above requirement, and that the last move is $\nwarrow$ iff y is an up-node.

We will define a bijective *numbering* $\mathsf{nb} : \mathrm{dom}(t) \to \{0, \ldots, |t|-1\}$. Then, we define the total order $\leqslant_{\mathrm{dfs}}$ by

$$y \leqslant_{\mathrm{dfs}} z \text{ if and only if } \mathsf{nb}(y) \leqslant \mathsf{nb}(z). \tag{8}$$

Informally, the numbering is induced by the complete no-cut dfs traversal: non up-nodes are numbered during their first visit, whereas up-nodes are numbered during their second visit. The tree of Fig. 9 has its up-nodes filled, and each node is decorated with its number. The relation $\lessdot_{\mathrm{dfs}}$ is also depicted with the dashed (red) arrows (whose labels will be explained below).

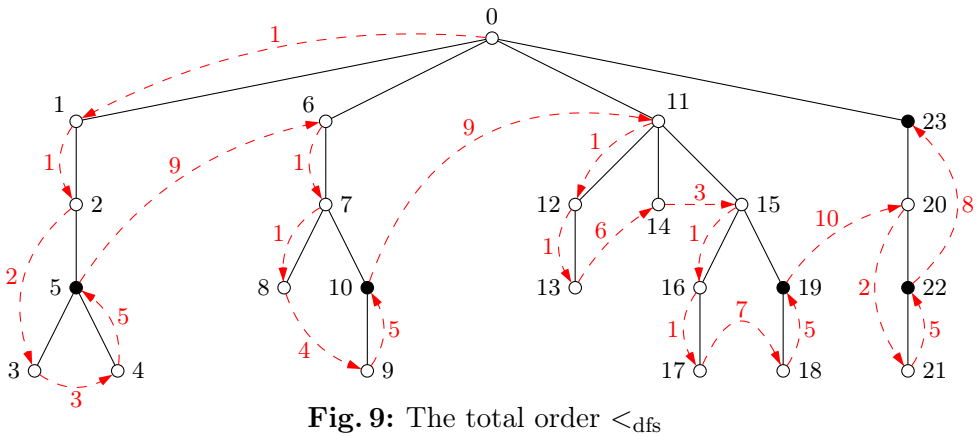

Fig. 9: The total order $<_{\mathrm{dfs}}$

Algorithm 1 Numbering all nodes of a subtree according to the $\lessdot_{\mathrm{dfs}}$ relation

1. global $c \leftarrow 0$

NUMBERING(x : node)

2. **if** $\neg\mathsf{up}(x)$ **then** $\mathsf{nb}(x) \leftarrow c; c \leftarrow c+1$ **end if**
3. **for** $i = 0$ **to** $\mathsf{rk}(x) - 1$ **do** NUMBERING($x.i$) **end for**
4. **if** $\mathsf{up}(x)$ **then** $\mathsf{nb}(x) \leftarrow c; c \leftarrow c+1$ **end if**

[3] FO+depth is the extension of FO boolean formulas with atoms $\mathrm{depth}(x) \equiv_\ell k$, for every $\ell \geqslant 1$ and $0 \leqslant k < \ell$.

Algorithm 1 computes nb when launched at the root $x = \varepsilon$. It uses a global variable c holding the next number to be assigned. The recursive calls perform a depth-first-search in the subtree rooted at x. Therefore, each node x indeed gets a unique identifier, set either at line 2 or line 4. Clearly $\mathsf{nb}(\varepsilon) = 0$. Hence, property (i) above is true. Let us explain why (ii) and (iii) also hold.

One can prove inductively that there are 3 patterns for entering and leaving a subtree of size at least 2, when running through the nodes of the tree respecting $\lessdot_{\text{dfs}}$. They are depicted in Fig. 10, where *min* and *max* denote the $\leqslant_{\text{dfs}}$-minimal and $\leqslant_{\text{dfs}}$-maximal elements of the subtree. Distinguished nodes denote the root of the subtree, its first child in (a), its last child in (b) and its last child and last grandchild in (c).

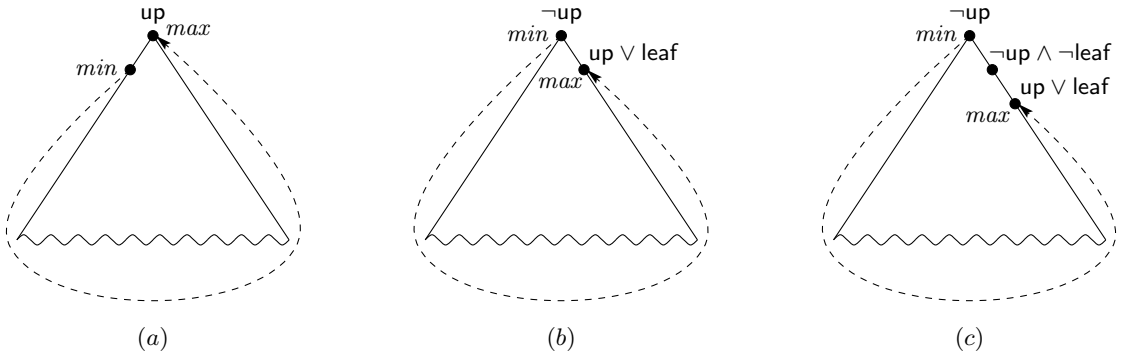

Fig. 10: Minimal and maximal elements of $\leqslant_{\text{dfs}}$ restricted to a subtree

(a) The root of the subtree is an up-node. In this case, the minimal element of the subtree is the first child of the root, and the maximal element of the subtree is the root itself.
(b) The root of the subtree is not an up-node, and its last child is either an up-node or a leaf.
(c) The root of the subtree is not an up-node, its last child y is neither an up-node nor a leaf. Note that in this case, the last child z of y is either an up-node or a leaf. Indeed, since y is not an up-node, its depth must be even, hence the depth of z is odd.

$$
\begin{aligned}
x \lessdot_{\text{dfs}} y \stackrel{\text{def}}{=}\ & \neg\mathsf{leaf}(x) \wedge \neg\mathsf{up}(x) \wedge x \lessdot_{\text{v}} y \wedge \mathsf{fc}(y) \wedge \neg\mathsf{up}(y) && (1)\\
\vee\ & \neg\mathsf{leaf}(x) \wedge \neg\mathsf{up}(x) \wedge \exists z\ (x \lessdot_{\text{v}} z \wedge \mathsf{fc}(z) \wedge \mathsf{up}(z) \wedge z \lessdot_{\text{v}} y \wedge \mathsf{fc}(y)) && (2)\\
\vee\ & \mathsf{leaf}(x) \wedge x \lessdot_{\text{h}} y \wedge \neg\mathsf{up}(y) && (3)\\
\vee\ & \mathsf{leaf}(x) \wedge \exists z\ (x \lessdot_{\text{h}} z \wedge \mathsf{up}(z) \wedge z \lessdot_{\text{v}} y \wedge \mathsf{fc}(y)) && (4)\\
\vee\ & (\mathsf{up}(x) \vee (\mathsf{leaf}(x) \wedge \mathsf{lc}(x))) \wedge \\
& [\quad y \lessdot_{\text{v}} x \wedge \mathsf{up}(y) && (5)\\
& \ \vee \exists z\ (z \lessdot_{\text{v}} x \wedge z \lessdot_{\text{h}} y \wedge \neg\mathsf{up}(y)) && (6)\\
& \ \vee \exists z, z'\ (z \lessdot_{\text{v}} x \wedge z \lessdot_{\text{h}} z' \wedge \mathsf{up}(z') \wedge z' \lessdot_{\text{v}} y \wedge \mathsf{fc}(y)) && (7)\\
& \ \vee \exists z\ (y \lessdot_{\text{v}} z \lessdot_{\text{v}} x \wedge \mathsf{lc}(z) \wedge \mathsf{up}(y)) && (8)\\
& \ \vee \exists z, z'\ (z' \lessdot_{\text{v}} z \lessdot_{\text{v}} x \wedge \neg\mathsf{up}(z) \wedge \mathsf{lc}(z) \wedge z' \lessdot_{\text{h}} y \wedge \neg\mathsf{up}(y)) && (9)\\
& \ \vee \exists z, z', z''\ (z' \lessdot_{\text{v}} z \lessdot_{\text{v}} x \wedge \neg\mathsf{up}(z) \wedge \mathsf{lc}(z) \wedge z' \lessdot_{\text{h}} z'' \wedge \mathsf{up}(z'') \wedge z'' \lessdot_{\text{v}} y \wedge \mathsf{fc}(y))\ ] && (10)
\end{aligned}
$$

Table 1: FO+depth definition of $\lessdot_{\text{dfs}}$

Using this property, one can establish the alternate (FO+depth)-definition of the successor relation $\lessdot_{\text{dfs}}$ shown in Table 1. The definition has 10 cases indicated in Fig. 9 as labels of

the dashed arrows. This expression for $\preccurlyeq_{\text{dfs}}$ shows both *(ii)* and *(iii)*. This is straightforward for *(ii)*. For *(iii)*, let $x \preccurlyeq^{\ell}_{\text{dfs}} y$: $x = x_0 \preccurlyeq_{\text{dfs}} x_1 \preccurlyeq_{\text{dfs}} \cdots \preccurlyeq_{\text{dfs}} x_\ell = y$. One may observe that a dfs path from some node x_i to its $\preccurlyeq_{\text{dfs}}$-successor x_{i+1} has at most 4 moves, the worst case being (10) where the path is $\uparrow\uparrow\rightarrow\swarrow$. Therefore, $\mathsf{sp}(x, y)$ has length at most 4ℓ. Finally, one can check that $\mathsf{sp}(x_{i-1}, x_i)\mathsf{sp}(x_i, x_{i+1}) \in$ sp-dfs, and therefore $\mathsf{sp}(x, y)$ is a dfs path by Lemma 2.8. All in all, $\mathsf{sp}(x, y) \in 4\ell$-sp-dfs.

In the following, we will use the boolean formula dfs-nb$(y) \equiv_\ell k$ stating that the $<_{\text{dfs}}$ numbering of y is k modulo ℓ. This gives us the opportunity to use expressions of WE(FO + depth + dfs-nb)[4].

If the tree t has size less or equal than n, we can write a specific expression. Hence, we suppose in the following that the tree has size greater than n. We therefore define split nodes as nodes x' such that dfs-nb$(x') \equiv_n 0$. Hence, if $m = \lceil |t|/n \rceil$ (by the previous assumption on the size of the tree, $m \geqslant 2$), the split nodes are exactly $x'_0 = \varepsilon \preccurlyeq^n_{\text{dfs}} x'_1 \preccurlyeq^n_{\text{dfs}} x'_2 \preccurlyeq^n_{\text{dfs}} \cdots \preccurlyeq^n_{\text{dfs}} x'_{m-1}$, and $x'_m = \varepsilon$. By property *(iii)* of the $<_{\text{dfs}}$ ordering, the unique words $w_0, \ldots, w_{m-1} \in \Sigma_D^+$ that are no-cut dfs walks and defined for $i \in \{0, \ldots, m-1\}$ by $\underline{\text{walk}}_{w_i}(x'_i, x'_{i+1})$ have a length bounded by $4n$.

In the following, we say that node x is the *jump node* associated to the pair (x', p) if dfs-nb$(x') \equiv_n 0$ and $x \preccurlyeq^p_{\text{dfs}} x'$ with $p \in Q$: we denote $x' = \text{split}(x)$ and $p = \text{state}(x)$. We define a boolean formula of WE(FO + depth + dfs-nb) characterizing for two consecutive jump nodes x and y the no-cut dfs walk w between the corresponding split nodes x' and y' and the encoded states $p, q \in Q$:

$$\varphi_{p,q,w}(x, y) \stackrel{\text{def}}{=} \exists x', y' \ (\underline{\text{walk}}_w(x', y') \wedge x \preccurlyeq^p_{\text{dfs}} x' \wedge y \preccurlyeq^q_{\text{dfs}} y' \wedge \text{dfs-nb}(x') \equiv_n 0 \wedge x' \preccurlyeq^n_{\text{dfs}} y')$$

Note that, given two nodes x and y there is at most one tuple p, q, w (with p, q states and w no-cut dfs walk) such that the formula $\varphi_{p,q,w}(x, y)$ holds. More precisely, $p = \text{state}(x)$, $q = \text{state}(y)$ and there exists $0 \leqslant i < m-1$ such that $x'_i = \text{split}(x)$, $x'_{i+1} = \text{split}(y)$ and $w = w_i$. Moreover, the sum of the weights of all runs starting from x'_i with state p, ending in x'_{i+1} with state q, is exactly $\mu(w_i)_{p,q}$.

Observe that when $\varphi_{p,q,w}(x, y)$ holds, the $\preccurlyeq_{\text{dfs}}$ distance between x and y is $p+n-q < 2n$, and $|w| \leqslant 4n$ (by property *(iii)* of the $<_{\text{dfs}}$ ordering and since the $\preccurlyeq_{\text{dfs}}$ distance between $x' = \text{split}(x)$ and $y' = \text{split}(y)$ is at most n). We therefore define the bdfs-cost expression

$$E \stackrel{\text{def}}{=} \circledast_{x,y}^{8n\text{-dfs}} F(x, y) \quad \text{with} \quad F(x, y) = \bigoplus_{p,q,w} \mu(w)_{p,q} \otimes \varphi_{p,q,w}(x, y),$$

where the sum ranges over all states $p, q \in Q$ and all no-cut dfs walk w of length at most $4n$. In fact, E actually has the same semantics as $\circledast_{x,y}^{N\text{-dfs}} F(x, y)d$ for $N \geqslant 8n$, since the $\preccurlyeq_{\text{dfs}}$ distance between two consecutive jump nodes is at most $2n$, and therefore, the length of any dfs path between x and y is at most $8n$.

Intuitively, the value $[\![E]\!](x, y)$ is the sum of the weights of all runs following a no-cut dfs walk of $\mathcal{A}$ starting from split(x) with state(x), ending in split(y) with state(y).

Lemma 4.3. *Let $x, y \in \text{dom}(t)$ be such that $x \leqslant_{\text{dfs}} y$ and* split(y) *exists. Let* no-cut(x, y) *be the unique no-cut dfs walk w such that* $\underline{\text{walk}}_w(\text{split}(x), \text{split}(y))$ *holds. Then,*

$$[\![E]\!](x, y) = \mu(\text{no-cut}(x, y))_{\text{state}(x),\text{state}(y)}.$$

[4] FO+depth+dfs-nb is the extension of FO+depth with one more atom [dfs-nb$(x) \equiv_\ell k$] for all $\ell \geqslant 1$, $0 \leqslant k < \ell$.

Proof. The proof is by induction on $\ell \geqslant 0$ such that $x \lessdot_{\mathrm{dfs}}^{\ell} y$. Notice first that $\varphi_{p,q,w}(x,y)$ never holds if $y \leqslant_{\mathrm{dfs}} x$ (so in particular, if $\mathsf{sp}(x,y) \notin$ sp-dfs). Note also that $F(x,y)$ is progressing: this is in fact ensured by $\mathrm{check}_w(x,y)$. Corollary 2.9 and Lemma 2.10 show that $[\![E]\!](x,y) = \mathbf{0}$ if $y <_{\mathrm{dfs}} x$, $[\![E]\!](x,x) = \mathbf{1}$, and that

$$E(x,y) \equiv (x=y) \oplus \bigoplus_{z \mid \mathsf{sp}(x,z) \in 8n\text{-sp-dfs}} F(x,z) \otimes E(z,y) \tag{9}$$

Next, we have $\text{no-cut}(x,x) = \varepsilon$. Indeed, assume that w is a nonempty dfs walk leading from x back to itself. Since a reversal $\nwarrow\swarrow$ is not allowed in w, by Lemma 3.5 its first move must be $\swarrow$ and its last move must be $\nwarrow$. But this is incompatible with $\mathrm{check}_w(x,x)$. Since $\mu(\varepsilon)$ is the identity matrix, we get $\mu(\varepsilon)_{\mathrm{state}(x),\mathrm{state}(x)} = \mathbf{1}$. This concludes the base case when $\ell = 0$.

We can also check that $[\![E]\!](x,y) = \mathbf{0}$ if $x <_{\mathrm{dfs}} y$ and $\mathrm{split}(x) = \mathrm{split}(y)$. Moreover in this case, $\text{no-cut}(x,y) = \{\varepsilon\}$ and $\mathrm{state}(x) \neq \mathrm{state}(y)$. Therefore, $\mu(\varepsilon)_{\mathrm{state}(x),\mathrm{state}(y)} = \mathbf{0}$.

Assume now that $x <_{\mathrm{dfs}} y$ and $\mathrm{split}(x) \neq \mathrm{split}(y)$. Let $x' = \mathrm{split}(x)$ and $p = \mathrm{state}(x)$. Let z' be the next split node: $x' \lessdot_{\mathrm{dfs}}^{n} z'$, and let w' be the no-cut dfs walk such that $\underline{\mathrm{walk}}_{w'}(x',z')$. We have seen above that $\varphi_{r,q,v}(x,z)$ holds if and only if $r = p$, $v = w'$ and $z \lessdot_{\mathrm{dfs}}^{q} z'$ for some $q \in Q$. In this case, we have $[\![F]\!](x,z) = \mu(w')_{p,q}$. Hence, from (9), we get

$$[\![E]\!](x,y) = \sum_{z \mid \mathrm{split}(z) = z'} [\![F]\!](x,z) \times [\![E]\!](z,y) = \sum_{z \mid \mathrm{split}(z) = z'} \mu(w')_{\mathrm{state}(x),\mathrm{state}(z)} \times [\![E]\!](z,y).$$

By induction, we obtain

$$[\![E]\!](x,y) = \sum_{z \mid \mathrm{split}(z) = z'} \Big(\mu(w')_{\mathrm{state}(x),\mathrm{state}(z)} \times \mu(w'')_{\mathrm{state}(z),\mathrm{state}(y)} \Big)$$

with w'' the no-cut dfs walk such that $\underline{\mathrm{walk}}_{w''}(z',y')$. We can then check that $\text{no-cut}(x,y) = w' \cdot w''$. Hence, we get

$$\begin{aligned} [\![E]\!](x,y) &= \sum_{z \mid \mathrm{split}(z) = z'} \Big(\mu(w')_{\mathrm{state}(x),\mathrm{state}(z)} \times \mu(w'')_{\mathrm{state}(z),\mathrm{state}(y)} \Big) \\ &= \sum_{p \in Q} \Big(\mu(w')_{\mathrm{state}(x),p} \times \mu(w'')_{p,\mathrm{state}(y)} \Big) \\ &= \mu(w'w'')_{\mathrm{state}(x),\mathrm{state}(y)} = \mu(\text{no-cut}(x,y))_{\mathrm{state}(x),\mathrm{state}(y)}. \end{aligned}$$

□

As a corollary, expression E is able to compute the sum of the weights of all runs of $\mathcal{A}$ following a no-cut dfs walk from x'_1 to x'_{m-1}. To conclude and capture the semantics of $\mathcal{A}$, it remains to write special expressions for *small* trees (of size less than n), and for the beginning and the end of complete runs of *large* trees. This is left to the reader since it will not be useful in order to understand the next stage of the proof, dealing with *may-cut* dfs walks.

The expression we constructed belongs to WE(FO + depth + dfs-nb). Note however that Example 2.6 yields a translation of the depth predicate to WE(FO). Moreover, since the successor relation $\lessdot_{\mathrm{dfs}}$ is deterministic and thanks to *(ii)*, the formula $\text{dfs-nb}(y) \equiv_\ell k$ may also be expressed by a WE(FO) expression, namely:

$$\bigoplus\nolimits_x \Big(\varepsilon \lessdot_{\mathrm{dfs}}^{k} x \otimes [\circledast_{x',y'}^{\mathrm{bdfs}}\ x' \lessdot_{\mathrm{dfs}}^{\ell} y'](x,y) \Big).$$

Therefore, expressions of WE(FO+depth+dfs-nb) can be translated into expressions of WE(FO).

4.3 From 0-wDFSA to expressions: the general case

We turn now to the second difficulty which is that runs in a 0-wDFSA may cut some subtrees. This *non-determinism* in the dfs walk is a major problem since it forbids to uniquely determine the split node x' immediately following a jump node x, as was done in Section 4.2. To stay as much as possible in the deterministic paradise, we first reduce the problem to the same question on a simpler input, namely a 0-wDFSA which is syntactically allowed to cut subtrees *only at special nodes*. Let $M = 2n + 2$ (recall that $n = |Q|$). We say that node x is *special* if it satisfies

$$\mathsf{sn}(x) \stackrel{\text{def}}{=} \mathrm{root}(x) \vee (\mathrm{depth}(x) \equiv_M 0 \wedge \exists y\ (x \lessdot^M_{\mathrm{dfs}} y \wedge x <_{\mathrm{v}} y))$$

where the last part ensures that the subtree of x contains at least M nodes. Since M is even, a special node is never an up-node.

Reduction to a simpler automaton. From the 0-wDFSA $\mathcal{A}$, we construct an equivalent 0-wDFSA $\mathcal{B} = (Q', I', \Delta', \mathsf{weight}')$ which may only cut subtrees at special nodes. It uses an extra counter $c \in \{0, \dots, 2M\}$ to store the depth of a current node with respect to its ancestor x at which $\mathcal{A}$ would have cut the subtree, which $\mathcal{B}$ could not do if x was not special. In its *normal mode* ($c = 0$) $\mathcal{B}$ inherits the weighted transitions of $\mathcal{A}$ provided they do not cut a subtree at a non special node. If $\mathcal{A}$ wants to cut a subtree at some non special node x, we switch to the *cut mode* ($c > 0$) of $\mathcal{B}$ by moving down to the first child of x and setting $c = 1$. Then, $\mathcal{B}$ performs a dfs which cuts at and only at special nodes. Counter c is incremented with $\swarrow$ moves and decremented with $\nwarrow$ moves, so that it keeps the depth from x. This cut mode is deterministic, has weight $\mathbf{1}$, and stops when c is 0 again, *i.e.*, when we reach back node x. Note that, to distinguish between the first and second visit of x by $\mathcal{B}$, we need a flag remembering whether the last move was $\nwarrow$ or not. Then, we execute in $\mathcal{B}$ the weighted cut transition of $\mathcal{A}$. Note that this is not a cut transition in $\mathcal{B}$ since the last move of the cut mode is $\nwarrow$. If t is a Σ-tree, we can construct a $(\Sigma \times \{0,1\})$-tree t_{sn} where the $\{0,1\}$ component is set to 1 at node u iff u is a special node. Then we can prove that for every tree t, $[\![\mathcal{A}]\!](t) = [\![\mathcal{B}]\!](t_{\mathsf{sn}})$. We let n' be the number of states of $\mathcal{B}$ and we identify Q' with $\{0, \dots, n'-1\}$. This yields a canonical embedding of $Q = \{0, \dots, n-1\}$ in Q'.

A *may-cut* dfs walk is like a no-cut dfs walk, but it may cut subtrees of special nodes it encounters, and only those subtrees. To this aim, we add sn in the typing information of a node. Formally, a may-cut dfs walk is a word w over $\Sigma'_D = \{(a, \tau, d) \in \Sigma \times 2^{D \cup \{\mathsf{sn}\}} \times D \mid d \in \tau \text{ and } d \neq \leftarrow\}$ such that for each factor $(a, \tau, d)(a', \tau', d')$ of w we have $dd' \neq \nwarrow\swarrow$ and if $\swarrow \in \tau'$ and $d \neq \nwarrow$ and $\mathsf{sn} \notin \tau'$ then $d' = \swarrow$. Every run not following a may-cut dfs walk has weight 0 in $\mathcal{B}$.

We may visit the subtree of a special node x' only when $\mathcal{B}$ is in its normal mode, *i.e.*, when $c = 0$. In this case, x' will be a split node (both for its first visit and its last visit) and we only have to encode the corresponding states of $\mathcal{A}$, *i.e.*, from $Q = \{0, \dots, n-1\}$. To encode the first visit of x' with state $p \in Q$, we jump at node x satisfying

$$\mathrm{begin}_p(x', x) \stackrel{\text{def}}{=} \mathsf{sn}(x') \wedge x' \lessdot^{p+1}_{\mathrm{dfs}} x.$$

To encode the second visit of x' with state q, we jump at node x satisfying the formula

$$\mathrm{end}_q(x, x') \stackrel{\text{def}}{=} \mathsf{sn}(x') \wedge \exists z\ (x \lessdot^q_{\mathrm{dfs}} z \wedge \mathsf{lc}(z) \wedge x' \lessdot_{\mathrm{v}} z).$$

For $p \in Q' \setminus Q = \{n, \dots, n'-1\}$, the formulas $\mathrm{begin}_p(x', x)$ and $\mathrm{end}_p(x, x')$ are set to false. A node x is *reserved* if it satisfies the formula $\mathrm{res}(x) \stackrel{\text{def}}{=} \exists x' \bigvee_p (\mathrm{begin}_p(x', x) \vee \mathrm{end}_p(x, x'))$. Note that special nodes are not reserved.

A node x is *linked* with a special node x' if x' is the closest ancestor of x which is special:

$$\mathrm{link}(x,x') \stackrel{\text{def}}{=} \mathsf{sn}(x') \wedge x' <_{\mathrm{v}} x \wedge \neg\exists y\ x' <_{\mathrm{v}} y <_{\mathrm{v}} x \wedge \mathsf{sn}(y).$$

Note that a special node is not linked to itself. When not entering or leaving the subtree of a special node (cases begin_p and end_q above), automaton $\mathcal{B}$ stays in the set of nodes linked to some special node x'. The restriction of the $\leqslant_{\mathrm{dfs}}$ ordering to such areas induces a new *special dfs* ordering defined by

$$x \leqslant_{\mathrm{sdfs}} y \stackrel{\text{def}}{=} x \leqslant_{\mathrm{dfs}} y \wedge \exists x'\ \mathrm{link}(x,x') \wedge \mathrm{link}(y,x')$$

which is a disjoint union of total orders. The *special dfs successor* $\lessdot_{\mathrm{sdfs}} \stackrel{\text{def}}{=} <_{\mathrm{sdfs}} \setminus <^2_{\mathrm{sdfs}}$ describes steps of runs which are neither entering nor leaving the subtree of a special node. We can get an FO+depth definition of $\lessdot_{\mathrm{sdfs}}$ by replacing in the definition of $\lessdot_{\mathrm{dfs}}$ from Table 1 each occurrence of $\mathsf{leaf}(x)$ by $(\mathsf{leaf}(x) \vee \mathsf{sn}(x))$ and by adding the constraint $\neg\mathsf{sn}(z)$ to cases (6)-(10).

When we are not entering or leaving the subtree of a special node, the automaton $\mathcal{B}$ performs a no-cut dfs walk. We have to define additional split nodes, and to encode states of $\mathcal{B}$ at these split nodes. To this aim, we define the sdfs-nb of a node x as the integer k such that if $\mathrm{link}(x,x')$ holds and x'' is the $\lessdot_{\mathrm{dfs}}$-successor of the special node x', then $x'' \lessdot^k_{\mathrm{sdfs}} x'$. We can express the formula $\text{sdfs-nb}(x) \equiv_\ell k$ in $\mathrm{WE}(\mathrm{FO} + \mathrm{depth})$:

$$(\text{sdfs-nb}(x) \equiv_\ell k) \stackrel{\text{def}}{=} \bigoplus_{x',x'',z} \Big(\mathsf{sn}(x') \otimes x' \lessdot_{\mathrm{dfs}} x'' \lessdot^k_{\mathrm{sdfs}} z \otimes [\circledast^{\mathrm{bdfs}}_{y',z'}\ y' \lessdot^\ell_{\mathrm{sdfs}} z'](z,x)\Big).$$

A *must-cut* dfs walk is like a no-cut dfs walk except that it *must* cut all subtrees of special nodes it encounters, and only those subtrees. Formally, a must-cut dfs walk is a word w over Σ'_D such that for each factor $(a,\tau,d)(a',\tau',d')$ of w, if $\mathsf{sn} \in \tau'$ then $d \neq \nwarrow$, and if $\swarrow \in \tau'$ then $d' = \swarrow$ iff ($d \neq \nwarrow$ and $\mathsf{sn} \notin \tau'$). Hence, a must-cut dfs walk is trapped in the subtree of a special node considering all the lower special nodes as leaves.

We must now update the definition of $\underline{\mathrm{walk}}_w(x,y)$. We define now $\mathrm{swalk}_w(x,y)$, as the same formula than $\mathrm{walk}_w(x,y)$, assuming now that $\tau(x)$ also checks $\mathsf{sn}(x)$ (resp. $\neg\mathsf{sn}(x)$) if $\mathsf{sn} \in \tau$ (resp. $\mathsf{sn} \notin \tau$). Then, we define $\underline{\mathrm{swalk}}_w(x,x',y,y') \stackrel{\text{def}}{=} \mathrm{swalk}_w(x',y') \wedge \mathrm{check}_w(x,x',y,y')$ assuming that x' (resp. y') is the split node associated with the jump node x (resp. y). Finally, in the formula $\mathrm{check}_w(x,x',y,y')$:

- if x' is not a special node, it checks that the first letter of w conforms to $\mathrm{check}_w(x',y')$;
- if y' is not a special node, it checks that the last letter of w conforms to $\mathrm{check}_w(x',y')$;
- if x' is a special node, it checks that w starts with a $\swarrow$ move iff $\mathrm{begin}_p(x',x)$ holds for a given state $p \in Q$;
- if y' is a special node, it checks that w ends with a $\nwarrow$ move iff $\mathrm{end}_q(y,y')$ holds for a given state $q \in Q$.

Decomposing runs of $\mathcal{B}$. As in the no-cut dfs case, every node x will define at most one pair (x',p) with x' a split node and p a state:

- either $x' <_{\mathrm{dfs}} x$ and x' is a special node with p uniquely defined by $\mathrm{begin}_p(x',x)$ or $\mathrm{end}_p(x,x')$ (in this case, x is a reserved node),
- or $x \leqslant_{\mathrm{dfs}} x'$, and p is uniquely defined by $x \lessdot^p_{\mathrm{sdfs}} x'$ (in this case, x is not a reserved node).

Let $\ell = n + n'$. The boolean formula $\varphi_{p,q,w}(x,y)$ characterizing, for two consecutive jump nodes x and y, the must-cut dfs walk w between the corresponding split nodes x' and y' and

the encoded states $p, q \in Q'$ is now written

$$\varphi_{p,q,w}(x,y) \stackrel{\text{def}}{=} \exists x', x'', y' \ \underline{\text{swalk}}_w(x,x',y,y') \\ \wedge \begin{pmatrix} \text{begin}_p(x',x) \wedge x' \lessdot_{\text{dfs}} x'' \\ \vee \neg\text{res}(x) \wedge x \lessdot^p_{\text{sdfs}} x' \wedge \text{sdfs-nb}(x') \equiv_\ell 0 \wedge x' = x'' \\ \vee \text{end}_p(x,x') \wedge x' = x'' \end{pmatrix} \tag{10}$$

$$\wedge \begin{pmatrix} \text{begin}_q(y',y) \wedge \bigvee_{k \leqslant \ell} x'' \lessdot^k_{\text{sdfs}} y' \\ \vee \neg\text{res}(y') \wedge y \lessdot^q_{\text{sdfs}} y' \wedge x'' \lessdot^\ell_{\text{sdfs}} y' \\ \vee \text{end}_q(y,y') \wedge \neg\exists z \ (x'' \lessdot^\ell_{\text{sdfs}} z \wedge \neg\text{res}(z)) \end{pmatrix} \tag{11}$$

The expression E is given as in the no-cut case, but with μ the matrix associated with $\mathcal{B}$ and $\varphi_{p,q,w}$ the previous formula:

$$E \stackrel{\text{def}}{=} \circledast^{\text{bdfs}}_{x,y} F(x,y) \qquad \text{with} \qquad F(x,y) \stackrel{\text{def}}{=} \bigoplus_{p,q,w} \mu(w)_{p,q} \otimes \varphi_{p,q,w}(x,y)$$

where the sum ranges over all states $p, q \in Q'$ and all must-cut dfs walks w of length at most $4(\ell + n - 1) + 2$.

It is not a priori straightforward that the dfs-cost expression defining E is actually bounded. For $\varphi_{p,q,w}(x,y)$ to hold, one of the 3 cases defining the possible position of x wrt x' (resp. of y wrt y') must hold. By inspecting these 9 cases, one can check that the constraint on the length of w yields a bound $M' = 4(n' - 1 + \ell + n) < 8\ell$ on the length of a dfs-path between two consecutive jump nodes x, y in the bdfs-cost expression of E. One can prove that the worst case occurs when y' is a special node, x is not, $x \lessdot^{n'-1}_{\text{sdfs}} x' = x''$, and $\text{begin}_{n-1}(y', y)$.

Let us explain the intuition behind the formula $\varphi_{p,q,w}$: it is used inside a bdfs-cost operator to control the sequences $u_0, \ldots, u_k$ that we want to use as jump nodes in the definition (3) of the bdfs-cost, and to control the position of associated split nodes. Sequences such that there is an index i making $\varphi_{p,q,w}(u_i, u_{i+1})$ false for all p, q, w will be assigned a null contribution. The intuition is that an arbitrary run of $\mathcal{A}$ has a corresponding run in $\mathcal{B}$, and the bdfs-cost formula explains how such a run can be decomposed into successive, jointed must-cut walks.

Note first that by definition of res, given $p, q \in Q'$, the formulas $\text{begin}_p(x,x')$, $\neg\text{res}(x)$ and $\text{end}_q(x',x)$ are mutually exclusive, so only one disjunct of the conjunct (10) may hold. One may check that the same is true for $\text{begin}_p(y,y')$, $\neg\text{res}(y')$ and $\text{end}_q(y',y)$, even if this does not directly follow from the definition, since we use $\text{res}(y')$ instead of $\text{res}(y)$ in the conjunct (11) of $\varphi_{p,q,w}$. This makes again the disjuncts of the conjunct (11) mutually exclusive.

The three disjuncts in the conjunct (10) of $\varphi_{p,q,w}$ explain how some jump node $x = u_i$ is connected to the next jump node $y = u_{i+1}$, that is, how the segment of run between u_i and u_{i+1} evolves from u_i. Call *special area* the set of nodes linked to a given special node. The case $\text{begin}_p(x',x)$ corresponds to the case where the run of $\mathcal{B}$ enters some special area after jump node x. The case $\neg\text{res}(x)$ corresponds to that where the run stays inside the same special area. Finally, the case $\text{end}(x,x')$ corresponds to the case where the run exits the current special area, going up to the parent area.

Again, one can check that given two nodes x and y, there is at most one tuple (p,q,w) validating $\varphi_{p,q,w}(x,y)$, and therefore that x', y' are uniquely defined. We denote again x' as $\text{split}(x)$ and p as $\text{state}(x)$. Finally, let $\text{may-cut}(x,y)$ be the set of may-cut dfs walks $w \in \Sigma_D^*$ such that $\underline{\text{swalk}}_w(x, \text{split}(x), y, \text{split}(y))$ holds.

Intuitively, the value $[\![E]\!](x,y)$ is the sum of the weights of all the runs following a may-cut dfs walk of $\mathcal{A}$ starting from $\text{split}(x)$ with state $\text{state}(x)$, ending in $\text{split}(y)$ with state $\text{state}(y)$.

Lemma 4.4. *Let $x, y \in \text{dom}(t)$ be such that $x <_{\text{dfs}} y$ and $\text{split}(x)$, $\text{split}(y)$ exist. Then,*

$$[\![E]\!](x,y) = \sum_{w \in \text{may-cut}(x,y)} \mu(w)_{\text{state}(x),\text{state}(y)} \ .$$

Proof. The proof goes again by induction over ℓ such that $x \lessdot^{\ell}_{\text{dfs}} y$. As previously, we can check that $F(x,y)$ is progressing. Hence, Corollary 2.9 and Lemma 2.10 show that $[\![E]\!](x,x) = \mathbf{1}$, $[\![E]\!](x,y) = \mathbf{0}$ if $y <_{\text{dfs}} x$ and

$$E(x,y) \equiv (x=y) \oplus \bigoplus\nolimits_z F(x,z) \otimes E(z,y) \tag{12}$$

where the sum ranges over all z such that $\mathsf{sp}(x,z) \in M'$-sp-dfs. As in the complete case, we have may-cut$(x,x) = \{\varepsilon\}$. In consequence, for the base case $\ell = 0$, we can conclude as in the complete case that the statement holds.

We can also check that $[\![E]\!](x,y) = \mathbf{0}$ if $x <_{\text{dfs}} y$ and split$(x) =$ split(y). Moreover in this case, may-cut$(x,y) = \{\varepsilon\}$ since split$(x) =$ split(y). Since $x <_{\text{dfs}} y$, we have state$(x) \neq$ state(y). Therefore, $\mu(\varepsilon)_{\text{state}(x),\text{state}(y)} = \mathbf{0}$ and the statement of the lemma holds.

Assume now that $x <_{\text{dfs}} y$ and split$(x) \neq$ split(y). Let $x' =$ split(x) and $p =$ state(x). Define x'' as in the definition of $\varphi_{p,q,w}$: if x' is not special or if $\text{end}_p(x,x')$, then $x'' = x'$; if $\text{begin}_p(x',x)$, then x'' is the $\lessdot_{\text{dfs}}$ successor of x'.

Let $z'_1 <_{\text{dfs}} \ldots <_{\text{dfs}} z'_{m-1}$ be the sequence of all special nodes such that for all $1 \leqslant i \leqslant m-1$, $x'' \lessdot^{k}_{\text{sdfs}} z'_i$ for a $k \leqslant \ell$. Observe that all these nodes are linked with the same special node. By definition of M', all their corresponding begin-reserved nodes are accessible from x by a word $\delta \in M'$-sp-dfs. Let u' be the node defined by $x'' \lessdot^{\ell}_{\text{sdfs}} u'$. Considering the cases of (11) relating y and y' in $\varphi_{p,q,w}(x,y)$, we distinguish three cases for u'.

1. Suppose that u' is neither a reserved node nor a special node, and let $z'_m = u'$. This situation is described in Fig. 11. One can show that $\varphi_{\text{state}(x),q,w}(x,z)$ holds only if split$(z) = z'_i$ for

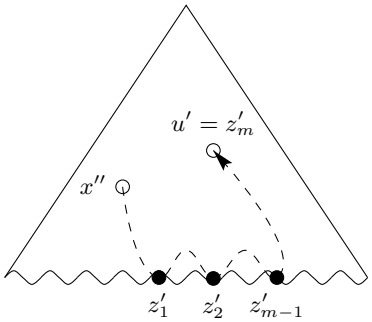

Fig. 11: $z'_1, \ldots, z'_{m-1}, z'_m$ are all nodes which may be chosen as a split node after x', and z'_m is the only one which is not a special node.

some $0 \leqslant i \leqslant m$, $q =$ state(z) and $\underline{\text{swalk}}_w(x,x',z,z'_i)$ holds. Let $W =$ may-cut(x,y). For $1 \leqslant j \leqslant m-1$, let W_j be the set of walks of W that cut all subtrees of $z'_1, \ldots, z'_{j-1}$ but visit the subtree of z'_j, and let W_m be the set of walks of W that cut all subtrees of $z'_1, \ldots, z'_{m-1}$. We have $W = W_1 \cup \cdots \cup W_{m-1} \cup W_m$. Moreover, every walk in W_j, $1 \leqslant j \leqslant m$, starts with a common prefix w_j, namely the must-cut dfs walk from x' to z'_j.

Let us write b(x), or m(x) or e(x), if $\text{begin}_{\text{state}(x)}(\text{split}(x),x)$, or $x \lessdot^{\text{state}(x)}_{\text{sdfs}} \text{split}(x)$, or $\text{end}_{\text{state}(x)}(x, \text{split}(x))$ holds, respectively.
From (12), we obtain

$$[\![E]\!](x,y) = \sum_{i=1}^{m-1} \sum_{\substack{z \text{ s.t. b}(z) \\ \text{split}(z)=z'_i}} [\![F]\!](x,z) \times [\![E]\!](z,y) + \sum_{\substack{z \text{ s.t. m}(z) \\ \text{split}(z)=z'_m}} [\![F]\!](x,z) \times [\![E]\!](z,y)$$

Moreover the union $W_1 \cup \cdots \cup W_{m-1} \cup W_m$ is a partition of W. Using the induction hypothesis for every z such that $x <_{\text{dfs}} z <_{\text{dfs}} y$, we obtain

$$\begin{aligned}[\![E]\!](x,y) = &\sum_{i=1}^{m-1} \sum_{\substack{z \text{ s.t. b}(z) \\ \text{split}(z)=z'_i}} \mu(w_i)_{p,\text{state}(z)} \times \sum_{w \in \text{may-cut}(z,y)} \mu(w)_{\text{state}(z),\text{state}(y)} \\ &+ \sum_{\substack{z \text{ s.t. m}(z) \\ \text{split}(z)=z'_m}} \mu(w_m)_{p,\text{state}(z)} \times \sum_{w \in \text{may-cut}(z,y)} \mu(w)_{\text{state}(z),\text{state}(y)} \\ = &\sum_{i=1}^{m} \sum_{w \in W_i} \mu(w)_{\text{state}(x),\text{state}(y)} = \sum_{w \in \text{may-cut}(x,y)} \mu(w)_{\text{state}(x),\text{state}(y)}\end{aligned}$$

2. Suppose now that u' is a special node. As $x'' \lessdot^{\ell}_{\text{sdfs}} u'$, we have $u' = z'_{m-1}$. Compared with the situation of Fig. 11, the split nodes z'_{m-1} and z'_m should now be merged. Using again the sets $W_1, \ldots, W_{m-1}, W_m$ previously defined, we obtain from (12)

$$[\![E]\!](x,y) = \sum_{i=1}^{m-1} \sum_{\substack{z \text{ s.t. b}(z) \\ \text{split}(z)=z'_i}} [\![F]\!](x,z) \times [\![E]\!](z,y) + \sum_{\substack{z \text{ s.t. m}(z) \\ \text{split}(z)=z'_{m-1}}} [\![F]\!](x,z) \times [\![E]\!](z,y)$$

If $\text{split}(y) \neq z'_{m-1}$, the union $W_1 \cup \cdots \cup W_{m-1} \cup W_m$ is again a partition of W, and we can conclude as in the previous case. Otherwise, $\text{split}(y) = z'_{m-1}$, and the sets W_{m-1} and W_m are identical because the walks must end in the special node z'_{m-1}. If $\text{b}(y)$ holds, then

$$\sum_{\substack{z \text{ s.t. m}(z) \\ \text{split}(z)=z'_{m-1}}} [\![F]\!](x,z) \times [\![E]\!](z,y) = 0$$

because $[\![E]\!](z,y) = 0$ for all z such that $z \neq y$ and $\text{split}(z) = \text{split}(y)$. For the same reason, if $\text{m}(y)$ holds, then

$$\sum_{\substack{z \text{ s.t. b}(z) \\ \text{split}(z)=z'_{m-1}}} [\![F]\!](x,z) \times [\![E]\!](z,y) = 0$$

In both cases, we can conclude as previously.

3. Otherwise, the node u' such that $x'' \lessdot^{\ell}_{\text{sdfs}} u'$ is a reserved node (and necessarily an end_q-reserved node). We then define z'_m as the unique special node such that $\text{link}(u', z'_m)$. On Fig. 11, the node z'_m is now the root of the considered subtree. Consider as above the sets of walks $W_1, \ldots, W_{m-1}, W_m$, which form a partition of W. From (12), we obtain

$$[\![E]\!](x,y) = \sum_{i=1}^{m-1} \sum_{\substack{z \text{ s.t. b}(z) \\ \text{split}(z)=z'_i}} [\![F]\!](x,z) \times [\![E]\!](z,y) + \sum_{\substack{z \text{ s.t. e}(z) \\ \text{split}(z)=z'_m}} [\![F]\!](x,z) \times [\![E]\!](z,y)$$

We conclude as previously by distributivity and using induction hypothesis. □

Finally, given a tree with at least M nodes (for small trees we may write a specific WE(FO) expression), the sum of the weights of all runs of $\mathcal{B}$ starting in $p \in Q$ at the root, ending in $q \in Q$ at the root and following *all* may-cut dfs walks is computed with the expression

$$F_{p,q} = \bigoplus\nolimits_{x,y} \text{begin}_p(\varepsilon, x) \otimes E(x,y) \otimes \text{end}_q(y, \varepsilon) \tag{13}$$

To fully relate the semantics of automaton $\mathcal{A}$ with the semantics of $F_{p,q}$, it remains to compute only the weight of the accepting runs. Hence the expression defined by

$$\bigoplus_{\substack{p\in I, q\in Q\\ q'\in Q}} F_{p,q} \otimes [\mathsf{weight}(q, t(\varepsilon), \mathrm{root} \wedge \mathsf{leaf}, \emptyset, \mathrm{accept}, q') \oplus \mathsf{weight}(q, t(\varepsilon), \mathrm{root} \wedge \neg\mathsf{leaf}, \emptyset, \mathrm{accept}, q')]$$

computes exactly $[\![\mathcal{A}]\!](t)$.

4.4 From r-wDFSA to expressions

It remains to deal inductively with pebbles. As previously, we do not consider initial states and accept transitions for the moment. Let $\mathcal{A}$ be an r-DFSA over the alphabet Σ. If $r = 0$, we have previously shown the result. Otherwise, a run of $\mathcal{A}$ can intuitively be decomposed as the interleaving of a run of a 0-DFSA with some drop-reset/resume-lift factors and some runs of an $(r-1)$-DFSA $\mathcal{A}^1$ from the root to the root over the alphabet $\Sigma \times \{0,1\}$ (the second composant is used to encode the position of the first dropped pebble). We can formaly define the automaton $\mathcal{A}^1$ by $(Q', \Sigma \times \{0,1\}, \Delta', \mathsf{weight}')$ with $Q' \subseteq Q$ the set of states $q \in Q$ such that there exists $p = e.p' \in Q_r$ with $e \in \{1,2\}$, $p' \in \{1,2,3,4\}^+$ and $(q,p) \in R$ (with R a fixed relation used in the definition of a r-DFSA), and

$$\begin{aligned}\Delta' =&\{(q,(a,0),\tau,P,d,q') \mid q,q' \in Q', (q,a,\tau,P_{+1},d,q') \in \Delta\}\\ &\cup \{(q,(a,1),\tau,P,d,q') \mid q,q' \in Q', (q,a,\tau,P_{+1}\cup\{1\},d,q') \in \Delta\}\end{aligned}$$

where $P_{+1} = \{k+1 \mid k \in P\}$. Using the simulation R, we can easily show that $\mathcal{A}^1$ is a $(r-1)$-DFSA.

Using induction hypothesis, we have an expression $F^1_{s_1,s_2}$ to compute the sum of the weights of all runs of $\mathcal{A}^1$ starting at the root with state s_1, ending at the root with state s_2. The only $\Sigma\times\{0,1\}$-trees that we will use are those where exactly one 1 appears in the second composant. A well-known technique consists in encoding this unique node of the tree in a first-order variable x. Hence, we can transform formula $F^1_{s_1,s_2}$ over $\Sigma \times \{0,1\}$ into a formula $\tilde{F}^1_{s_1,s_2}(x)$ over Σ by replacing every atomic formula $P_{(a,0)}(y)$ by $P_a(y) \wedge \neg(x=y)$ and every atomic formula $P_{(a,1)}(y)$ by $P_a(y) \wedge (x=y)$. Then, for a Σ-tree t and a node $x \in \mathrm{dom}(t)$, $[\![\tilde{F}^1_{s_1,s_2}(x)]\!](t) = [\![F^1_{s_1,s_2}]\!](t,x)$, if we denote by (t,x) the $\Sigma\times\{0,1\}$-tree t' of domain $\mathrm{dom}(t)$ and $t'(u) = (t(u),0)$ if $u \neq x$ and $t'(x) = (t(x),1)$. In particular, $[\![\tilde{F}^1_{s_1,s_2}(x)]\!](t)$ computes exactly the sum of the weights of the runs of $\mathcal{A}$ from the root with state s_1 to the root with state s_2, and with the first pebble being permanently dropped on node x of the tree t.

Then, the behavior of $\mathcal{A}$ with initial state $p' \in Q$ and final state $q' \in Q$ can be computed by expression (13) replacing $\mu(w)_{p,q}$ (in formula $E(x,y)$) by $\tilde{\mu}(\mathrm{split}(x), w)_{p,q}$, so that recursive calls made to $\mathcal{A}^1$ by dropping pebble 1 are taken into account. We let $\tilde{\mu}(x,\varepsilon)$ be the identity matrix, and we define inductively

$$\begin{aligned}\tilde{\mu}(x,(a,\tau,d)w)_{p,q} \stackrel{\text{def}}{=} \bigoplus\nolimits_{p'} \Big[\Big(\mu(a,\tau,d)_{p,p'} \oplus \bigoplus\nolimits_{p''} \mu^{\mathrm{drop}}(a,\tau,x)_{p,p''} \otimes \mu(a,\tau,d)_{p'',p'}\Big)\\ \otimes \mathrm{mv}_d(x,x') \otimes \tilde{\mu}(x',w)_{p',q}\Big]\end{aligned}$$

where $\mu^{\mathrm{drop}}(a,\tau,x)_{s_0,s_3}$ is defined by

$$\bigoplus\nolimits_{s_1,s_2} \mu(a,\tau,\text{drop-reset})_{s_0,s_1} \otimes \tilde{F}^1_{s_1,s_2}(x) \otimes \mu(t(\varepsilon),\tau(\varepsilon),\text{resume-lift})_{s_2,s_3}$$

if x is not the root, otherwise the last weight $\mu(t(\varepsilon),\tau(\varepsilon),\text{resume-lift})_{s_2,s_3}$ should be replaced by the weight $\mathsf{weight}(s_2, t(\varepsilon), \tau(\varepsilon), \{1\}, \text{resume-lift}, s_3)$. In fact, we have omitted the computation of the last call to $\mathcal{A}$ when returning to the root of t and dropping pebble 1 a last time, but this can be added in the same time than the final accept transition.

5 Emptiness and Evaluation

A natural problem in the context of verification is emptiness, which in the weighted setting asks whether, for a given r-wDFSA $\mathcal{A}$, there exists a tree t such that $[\![\mathcal{A}]\!](t) \neq \mathbf{0}$. It is easy to check that this problem is decidable for positive semirings. However, the problem becomes undecidable over general semirings, even for 1-way pebble word automata with 2 pebbles.

In the context of XML documents, an important problem is evaluation of an automaton wrt. a given tree. More precisely, given a tree t and an r-wDFSA $\mathcal{A} = (Q, I, F, \Delta, \mathsf{weight})$, we are interested in computing $[\![\mathcal{A}]\!](t)$. To do so, we inductively calculate $[\![\mathcal{A}]\!]^\pi_e(u) \in \mathbb{S}^{Q\times Q}$ for $u \in \mathrm{dom}(t)$, $e \in \{1,2,3,4\}$, and $\pi \in \mathrm{dom}(t)^{\leqslant r}$. Intuitively, $[\![\mathcal{A}]\!]^\pi_e(u)_{p,q}$ is the sum of weights of paths that go from u to ε, starting in p, dfs phase e and with pebbles π, and ending in q and pebbles π. Let $\mu(a,\tau,T,d) \in \mathbb{S}^{Q\times Q}$ be given by $\mu(a,\tau,T,d)_{p,q} = \mathsf{weight}(p,a,\tau,T,d,q)$ if $(p,a,\tau,T,d,q) \in \Delta$, and $\mu(a,\tau,T,d)_{p,q} = \mathbf{0}$, otherwise. We set

$$[\![\mathcal{A}]\!]^\pi_e(u) = \begin{cases} \overline{\mathbf{0}} & \\ + \overline{\mathbf{1}} & \text{if } u = \varepsilon \\ + \mu(t(u),\tau(u),\mathsf{Peb}(\pi,u),\swarrow) \times [\![\mathcal{A}]\!]^\pi_1(u0) & \text{if } e \in \{1,2\} \text{ and } u0 \in \mathrm{dom}(t) \\ + \mu(t(u),\tau(u),\mathsf{Peb}(\pi,u),\rightarrow) \times [\![\mathcal{A}]\!]^\pi_1(u') & \text{if } u \lessdot_{\mathrm{h}} u' \in \mathrm{dom}(t) \\ + \mu(t(u),\tau(u),\mathsf{Peb}(\pi,u),\nwarrow) \times [\![\mathcal{A}]\!]^\pi_3(u') & \text{if } u' \lessdot_{\mathrm{v}} u \text{ and } u \text{ has no right sibling} \\ + \begin{pmatrix} \mu(t(u),\tau(u),\mathsf{Peb}(\pi,u),\text{drop-reset}) \times [\![\mathcal{A}]\!]^{\pi u}_1(\varepsilon) \\ \times \mu(t(u),\tau(u),\mathsf{Peb}(\pi u,u),\text{resume-lift}) \times [\![\mathcal{A}]\!]^\pi_{e+1}(u) \end{pmatrix} & \text{if } e \in \{1,3\} \text{ and } |\pi| < r \end{cases}$$

Then, $[\![\mathcal{A}]\!](t) = \sum_{(p,q)\in I\times Q} ([\![\mathcal{A}]\!]^\varepsilon_1(\varepsilon) \times \mu(t(\varepsilon),\tau(\varepsilon),\emptyset,\text{accept}))_{p,q}$ which, using dynamic programming, can be computed in time $\mathcal{O}(|Q|^3 \cdot n^{r+1} \cdot (r+1))$. Here, $\mathcal{O}(|Q|^3)$ is due to matrix multiplication, and $\mathcal{O}(n^{r+1}(r+1))$ is the number of entries $[\![\mathcal{A}]\!]^\pi_e(u)$ that have to be determined. We obtain the following result:

Theorem 5.1. *Evaluation of an r-wDFSA with set of states Q wrt. a tree with n nodes is $\mathcal{O}(|Q|^3 \cdot n^{r+1} \cdot (r+1))$.*

6 Conclusion and further work

We defined a high-level language to express quantitative properties that cannot be captured by existing notions of weighted automata on trees. Moreover, we provided a new type of weighted tree-walking automaton with pebbles whose runs are constrained to a 1-way, dfs-like navigation. We showed that these automata are powerful enough to compute expressions of our quantitative language. Conversely, the semantics of such automata can be encoded in the language by adding a "bounded dfs path cost" operator. The main result (Theorem 4.1) states that, actually, the expressiveness of weighted dfs automata exactly matches that of weighted expressions with bdfs-cost.

An open question we would like to tackle is the following. On words, it is well-known that one-way pebble and two-way pebble automata are equally expressive, a result extended in [4] to the weighted setting. It is therefore relevant to compare the expressive power of wDFSA with that of pebble tree-walking automata. This question makes sense even in the boolean case. It would be interesting to capture tree-walking automata by wDFSA, trading off addition of extra pebbles against one-way, structured, and more manageable runs.

References

1. R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56:16:1–16:43, May 2009.
2. J. Berstel and Ch. Reutenauer. *Noncommutative rational series with applications*, volume 137 of *Encyclopedia of Mathematics & Its Applications*. Cambridge, 2011.
3. M. Bojańczyk. Tree-walking automata. In *LATA'08*, volume 5196 of *LNCS*, pages 1–17. Springer, 2008.
4. B. Bollig, P. Gastin, B. Monmege, and M. Zeitoun. Pebble weighted automata and transitive closure logics. In *ICALP'10*, volume 6199 of *LNCS*, pages 587–598. Springer, 2010.
5. M. Droste and P. Gastin. Weighted automata and weighted logics. *Theor. Comp. Sci.*, 380(1-2):69–86, 2007.
6. M. Droste and H. Vogler. Weighted tree automata and weighted logics. *Theoretical Computer Science*, 366(3):228–247, 2006.
7. M. Droste and H. Vogler. Weighted logics for unranked tree automata. *Theory Comput. Syst.*, 48(1):23–47, 2011.
8. J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In *Jewels Are Forever, Contributions to Theoretical Computer Science in Honor of Arto Salomaa*, pages 72–83. Springer, 1999.
9. J. Engelfriet and H. J. Hoogeboom. Automata with nested pebbles capture first-order logic with transitive closure. *Log. Meth. in Comput. Sci.*, 3, 2007.
10. Z. Fülöp and L. Muzamel. Weighted tree-walking automata. *Acta Cybernetica*, 19(2):275–293, 2009.
11. W. Kuich, H. Vogler, and M. Droste, editors. *Handbook of Weighted Automata*. EATCS Monographs in Theoret. Comput. Sci. Springer, 2009.
12. F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5:403–435, July 2004.
13. F. Neven and Th. Schwentick. On the power of tree-walking automata. In Springer, editor, *ICALP*, volume 1853 of *LNCS*, pages 547–560, 2000.
14. J. Sakarovitch. *Éléments de théorie des automates*. Vuibert, 2003.
15. M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
16. L. Segoufin and V. Vianu. Validating streaming XML documents. In *PODS '02*, pages 53–64. ACM, 2002.
17. W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, pages 389–455. Springer, 1997.