Acta Cybernetica **19** (2009) 275–293.

Weighted Tree-Walking Automata*

Zoltán Fülöp† and Loránd Muzamel†

Abstract

We define weighted tree-walking automata. We show that the class of tree series recognizable by weighted tree-walking automata over a commutative semiring K is a subclass of the class of regular tree series over K. If K is not a ring, then the inclusion is strict.

Keywords: semirings, regular tree series, weighted tree-walking automata

1 Introduction

The concept of a *tree-walking automaton* (for short: twa) was introduced in [1] for modelling syntax-directed translations from strings to strings. Recently its importance grew in XML theory, see, e.g. [23, 25, 26]. A twa A is a sequential finite-state tree acceptor with finitely many transition rules. Obeying its state-behaviour, A walks along the edges of an input tree $s \in T_\Sigma$, where Σ is the input ranked alphabet of A. Then A accepts s if there is an *accepting run on* s, i.e., a finite walk on s from the initial state to the accepting state. Tree languages recognized by twa are effectively regular. Unfortunately there is no straight proof of this fact in the literature, but it can be obtained, e.g., as the special case of the main result of [14]. However, there exists a regular tree language that cannot be recognized by any twa [4]. There are several extensions of twa which still recognize regular tree languages, such as twa with weak pebbles [13], strong pebbles [14], invisible pebbles [15], and also the alternating pebble twa of [24].

Another kind of automata in which we are interested is the weighted tree automaton (for short: wta). It is a natural generalization of the classical tree automaton [10, 19, 20]. The generalization lies in that input trees are supplied with weights taken from an underlying semiring K. In fact, each transition rule of the wta has a weight represented by an element of K. The weight of a run over an input tree $s \in T_\Sigma$ is just the (semiring) product of the transitions which take part

*The full version of a submission presented at *Weighted Automata: Theory and Applications* (Dresden University of Technology, Germany, May 13-16, 2008). The research was supported by the Hungarian Scientific Fund under Grant T 46686 and by the Fund for Teaching and Research in Informatics.

†Department of Foundations of Computer Science, University of Szeged, Árpád tér 2., H-6720 Szeged, Hungary,E-mail: {fulop,muzamel}@inf.u-szeged.hu

in that run. Then, the weight of s is the (semiring) sum of all runs over s. In this way a wta recognizes a tree series, i.e., a mapping from T_Σ to K. Tree series recognizable by wta are called *regular*, and the class of regular tree series which are recognizable by weighted tree automata over Σ and K is denoted by $\mathrm{REG}(\Sigma, K)$. Note that $\mathrm{REG}(\Sigma, \mathbb{B})$, where $\mathbb{B}$ is the Boolean semiring, is the class of recognizable tree languages [19, 20]. Wta were defined and considered in several works, see e.g. [3], [2], [8], [5], [16], [11], [22], and the survey paper [18].

In this paper we introduce the weighted version of a twa, following the idea that led from classical tree automata to wta. In a *weighted tree-walking automaton A (over Σ and K)* (for short: wtwa), every transition rule has a weight taken from the semiring K. We assume that A is non-looping, i.e., it cannot enter into an infinite cycle from the initial configuration. The *weight of a run of A on an input tree $s \in T_\Sigma$* is the product of the weights of the applied transition rules, then the *weight of s computed by A* is the sum of the weights of all the accepting runs of A on s. Since A is non-looping, it has only finitely many such accepting runs. The tree series recognized by A is $S_A : T_\Sigma \to K$, where $S_A(s)$ is the weight of s for every input tree $s \in T_\Sigma$. We denote the class of tree series which are recognizable by non-looping wtwa over Σ and K by $\mathrm{TWA}(\Sigma, K)$. Hence, wtwa with their sequential processing are alternative tools besides the classical weighted tree automata, which process trees parallelly. We note that arbitrary (i.e., maybe looping) wtwa over $\mathbb{B}$ are exactly the twa of [1].

As the main result, we show that if K is commutative, then $\mathrm{TWA}(\Sigma, K) \subseteq \mathrm{REG}(\Sigma, K)$ (Theorem 17). The proof of that the tree series recognized by a non-looping wtwa A is regular is performed according to the following steps. We encode an accepting run of A by annotating the nodes of the input tree by the rules applied in that run. Thus we obtain the concept of a *run tree* of A. Then we construct two twa such that a Boolean combination of the tree languages recognized by them turns out to be the set of run trees of A. Hence, the run trees of A form a regular tree language (Corollary 15). This implies that the tree series S that associates a run tree with the weight of the run it encodes is regular. Finally, we define an appropriate relabeling τ from the set of run trees of A to the set of input trees such that the extension of τ to tree series takes S to S_A. Since such an extension preserves regularity of tree series, we obtain that S_A is regular (Theorem 16).

Then we show that if, in addition, the semiring K is proper, i.e., is not a ring, then the inclusion is strict, i.e., $\mathrm{REG}(\Sigma, K) - \mathrm{TWA}(\Sigma, K) \neq \emptyset$. Hereby we generalize the main result of [4]. We prove this by taking a surjective homomorphism $h : K \to \mathbb{B}$. Now, if $\mathrm{TWA}(\Sigma, K) = \mathrm{REG}(\Sigma, K)$, then also $h(\mathrm{TWA}(\Sigma, K)) = \mathrm{TWA}(\Sigma, \mathbb{B}) = h(\mathrm{REG}(\Sigma, K)) = \mathrm{REG}(\Sigma, \mathbb{B})$ which contradicts the celebrated result $\mathrm{TWA}(\Sigma, \mathbb{B}) \subset \mathrm{REG}(\Sigma, \mathbb{B})$ of [4].

The paper is organized as follows. In Section 2 we introduce the necessary notions and notation. In Section 3 we define weighted tree-walking automata, then we prove our main results in Section 4. In Section 5 we summarize our results and show an open problem.

2 Definitions and notation

2.1 Sets, relations, and strings

We denote the set of nonnegative integers by $\mathbb{N}$. For every $n \in \mathbb{N}$, we let $[n] = \{1, \ldots, n\}$. The empty set is denoted by $\emptyset$.

For a set A, we denote by 2^A the power set of A and by A^* the set of *strings* over A. We denote *empty string* by ε. Sometimes we write a for a singleton $\{a\}$.

Let $\rho \subseteq A \times A$ be a binary relation. The fact that $(a, b) \in \rho$ for some $a, b \in A$ is also denoted by $a \,\rho\, b$. Moreover, the transitive closure and the reflexive, transitive closure are denoted by ρ^+ and ρ^*, respectively.

2.2 Trees and tree languages

A *ranked alphabet* is an ordered pair $(\Sigma, rank)$, where Σ is a finite, nonempty set and $rank$ is a mapping of type $\Sigma \to \mathbb{N}$. For every $k \geq 0$, we define $\Sigma^{(k)} = \{\sigma \in \Sigma \mid rank(\sigma) = k\}$. We define $maxrank(\Sigma) = \max\{rank(\sigma) \mid \sigma \in \Sigma\}$. In the sequel we drop $rank$ and write a ranked alphabet as Σ. Moreover, in the rest of the paper Σ and Δ will denote arbitrary ranked alphabets.

The set of *trees over* Σ *indexed by* A, denoted by $T_\Sigma(A)$, is the smallest set $T \subseteq (\Sigma \cup \{(,)\} \cup \{,\})^*$ such that $\Sigma^{(0)} \cup A \subseteq T$ and whenever $k \geq 1$, $\sigma \in \Sigma^{(k)}$, and $t_1, \ldots, t_k \in T$, then $\sigma(t_1, \ldots, t_k) \in T$. In case $A = \emptyset$, we write T_Σ for $T_\Sigma(A)$. Certainly, $T_\Sigma \neq \emptyset$ if and only if $\Sigma^{(0)} \neq \emptyset$. Every subset $L \subseteq T_\Sigma$ is called a *tree language*.

For every tree $s \in T_\Sigma$, we define the set $pos(s) \subseteq \mathbb{N}^*$ of *the nodes of* s as follows. We let $pos(s) = \{\varepsilon\}$ if $s \in \Sigma^{(0)}$, and $pos(s) = \{\varepsilon\} \cup \{iu \mid 1 \leq i \leq k,\ u \in pos(s_i)\}$ if $s = \sigma(s_1, \ldots, s_k)$ for some $k \geq 1$, $\sigma \in \Sigma^{(k)}$ and $s_1, \ldots, s_k \in T_\Sigma$.

Now, for a tree $s \in T_\Sigma$ and a node $u \in pos(s)$, *the label of* s *at node* u, denoted by $s(u)$, is defined in a standard way. By the *root of* s we mean the node ε. A node u of s is a *leaf* if $u1 \notin pos(s)$. Moreover, we define the parent of u, denoted by $parent(u)$, and the child number of u, denoted by $childno(u)$, as follows:
(i) if $u = \varepsilon$, then $childno(u) = 0$ and $parent(u)$ is undefined,
(ii) if $u = u'j$, where $u' \in pos(s)$ and $j \in \mathbb{N}$, then $childno(u) = j$ and $parent(u) = u'$.

We will freely use the concepts of a *regular tree language* and a *(finite) tree automaton*. The unfamiliar reader can consult the works [19, 20], and [9] for these concepts. Moreover, we will need the following known closure properties for regular tree languages, see, e.g., Theorem 4.2 of [19].

Proposition 1. *Regular tree languages are closed under Boolean operations.*

2.3 Semirings and tree series

A *semiring* is an algebraic structure $(K, +, \cdot, 0, 1)$ with binary operations addition $+$, multiplication $\cdot$, and constants 0 and 1 (with $0 \neq 1$) such that $(K, +, 0)$ is a commutative monoid, $(K, \cdot, 1)$ is a monoid, multiplication distributes over addition (both from left and right), and $a \cdot 0 = 0 \cdot a = 0$ for every $a \in K$. Frequently we

will write just K for $(K, +, \cdot, 0, 1)$. We say that K is *commutative* if $a \cdot b = b \cdot a$ for each $a, b \in K$. The semiring K is *proper* if it is not a ring, i.e., there is no additive inverse of 1.

Examples of commutative semirings are the *Boolean semiring* $\mathbb{B} = (\{0,1\}, \vee, \wedge, 0, 1)$ and the *arctic semiring* $\mathsf{Arct} = (\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$, where the operations max and + are extended to $\mathbb{N} \cup \{-\infty\}$ in the obvious way. Note that both $\mathbb{B}$ and Arct are proper.

A *tree series (over Σ and K)* is a mapping $S : T_\Sigma \to K$ where (S, s) is usually written rather than $S(s)$ for $s \in T_\Sigma$. We denote by $K\langle\langle T_\Sigma \rangle\rangle$ the set of tree series over Σ and K. Now we give an example of a tree series over Arct.

Example 2. Let Σ be such that $\Sigma^{(0)} = \{\alpha, \beta\}$. The tree series $\mathit{height}_\alpha \in \mathsf{Arct}\langle\langle T_\Sigma \rangle\rangle$ delivers, for $s \in T_\Sigma$, the length of the longest path from the root of s to an α-node. More exactly,

(i) $(\mathit{height}_\alpha, s) = \begin{cases} 0 & \text{if } s = \alpha \\ -\infty & \text{if } s = \beta, \end{cases}$

(ii) $(\mathit{height}_\alpha, s) = 1 + \max\{(\mathit{height}_\alpha, s_i) \mid 1 \le i \le k\}$ if $s = \sigma(s_1, \ldots, s_k)$ for some $k \ge 1$, $\sigma \in \Sigma^{(k)}$, and $s_1, \ldots, s_k \in T_\Sigma$.

In an analogous way, we can define $\mathit{height}_\beta \in \mathsf{Arct}\langle\langle T_\Sigma \rangle\rangle$. Now by an α-β-path in s we mean a path from an α-node to a β-node of s along the edges of s such that the α-node precedes the β-node in the usual lexicographical order of the nodes and that every edge is taken at most once in the path. Finally, we define the tree series $\mathit{width}_{\alpha\beta} \in \mathsf{Arct}\langle\langle T_\Sigma \rangle\rangle$ which delivers, for $s \in T_\Sigma$, the length of the longest α-β-path in s. More exactly,

(i) $(\mathit{width}_{\alpha\beta}, s) = -\infty$ if $s = \alpha$ or $s = \beta$,

(ii) $(\mathit{width}_{\alpha\beta}, s) = \max\{\ \max\{(\mathit{height}_\alpha, s_i) + (\mathit{height}_\beta, s_j) + 2 \mid 1 \le i < j \le k\},$
$\max\{(\mathit{width}_{\alpha\beta}, s_i) \mid 1 \le i \le k\}\}$

if $s = \sigma(s_1, \ldots, s_k)$ for some $k \ge 1$, $\sigma \in \Sigma^{(k)}$, and $s_1, \ldots, s_k \in T_\Sigma$.

In particular, let $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$. Then, for the tree $s = \sigma(\sigma(\sigma(\sigma(\beta, \alpha), \beta), \gamma(\sigma(\beta, \alpha))), \beta)$, we have $\mathit{height}_\alpha(s) = 4$, $\mathit{height}_\beta(s) = 4$, and $\mathit{width}_{\alpha\beta}(s) = 6$. The longest α-β-path in s is visualized in Fig. 1.

Weighted tree automata were defined in several works in several ways, see e.g. [3], [2], [8], [5], [16], [11], [22], and the survey paper [18]. Here we give a definition which is equivalent with the standard one and, at the same time, is easy to handle.

A *weighted tree automaton (wta) over K* is a system $M = (Q, \Sigma, R, \nu)$, where Q is a finite set of *states*, Σ is the *input ranked alphabet*, ν is a mapping of type $Q \to K$, and R is a finite set of *(weighted) rules* of the form $\sigma(q_1, \ldots, q_k) \xrightarrow{a} q$, where $k \ge 0$, $\sigma \in \Sigma^{(k)}$, $q_1, \ldots, q_k, q \in Q$, and $a \in K$. In case $k = 0$ we write $\sigma \xrightarrow{a} q$ rather than $\sigma() \xrightarrow{a} q$.

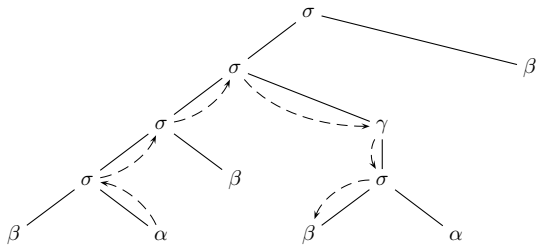

Figure 1: The longest α-β-path in the tree $s = \sigma(\sigma(\sigma(\sigma(\beta, \alpha), \beta), \gamma(\sigma(\beta, \alpha))), \beta)$.

The tree series $S_{M,q} \in K\langle\langle T_\Sigma \rangle\rangle$ recognized by M in a state $q \in Q$ is defined as follows. For every input tree $s = \sigma(s_1, \ldots, s_k) \in T_\Sigma$ with $k \geq 0$, $\sigma \in \Sigma^{(k)}$, and $s_1, \ldots, s_k \in T_\Sigma$, we have

$$(S_{M,q}, s) = \sum_{\substack{q_1,\ldots,q_k \in Q,\, a \in K \\ \sigma(q_1,\ldots,q_k) \xrightarrow{a} q \in R}} (S_{M,q_1}, s_1) \cdot \ldots \cdot (S_{M,q_k}, s_k) \cdot a.$$

Moreover, the tree series *recognized by* M is defined by

$$(S_M, s) = \sum_{q \in Q} (S_{M,q}, s) \cdot \nu(q)$$

for every input tree $s \in T_\Sigma$.

A tree series is *regular* if it can be recognized by a weighted tree automaton. We denote the class of regular tree series over Σ and K by $\mathrm{REG}(\Sigma, K)$.

Next we define the concept of a characteristic tree series and some operations on tree series which we need in the sequel. For a tree language $L \subseteq T_\Sigma$ the *characteristic tree series of* L is $\mathcal{X}_L \in K\langle\langle T_\Sigma \rangle\rangle$, defined by $(\mathcal{X}_L, s) = 1$ if $s \in L$ and $(\mathcal{X}_L, s) = 0$ otherwise for every $s \in T_\Sigma$.

Let $S_1, S_2 \in K\langle\langle T_\Sigma \rangle\rangle$ be tree series and $a \in K$. We define the tree series $aS_1 \in K\langle\langle T_\Sigma \rangle\rangle$ by $(aS_1, s) = a \cdot (S_1, s)$ for every $s \in T_\Sigma$. The *sum* and the *Hadamard product of* S_1 *and* S_2 are denoted by $S_1 + S_2$ and $S_1 \odot S_2$ in $K\langle\langle T_\Sigma \rangle\rangle$, respectively, and are defined as $(S_1 + S_2, s) = (S_1, s) + (S_2, s)$ and $(S_1 \odot S_2, s) = (S_1, s) \cdot (S_2, s)$ for each $s \in T_\Sigma$.

Regular tree series have the following closure properties.

Proposition 3. *Let K be commutative.*

(a) If $L \subseteq T_\Sigma$ is a regular tree language, then $\mathcal{X}_L \in \mathrm{REG}(\Sigma, K)$.

(b) If $S_1, S_2 \in \mathrm{REG}(\Sigma, K)$ and $a \in K$, then aS_1, $S_1 + S_2$, and $S_1 \odot S_2$ are also in $\mathrm{REG}(\Sigma, K)$.

For the proof of (a) we refer the reader to Lemma 3.3 of [12]. The closure under the multiplication with a, the sum, and the Hadamard product were proved in Lemmata 6.3 and 6.4 of [11] (cf. also Lemma 3.3 of [12]), and in Corollary 3.9

of [6]. Let us note that the proof of (a) and of the closure under sum do not need commutativity of K.

Now we define the concept of relabeling. A (deterministic) *relabeling* is a mapping $\tau : \Sigma \to \Delta$ such that for each $k \geq 0$ and $\sigma \in \Sigma^{(k)}$ we have $\tau(\sigma) \in \Delta^{(k)}$. Then τ extends to the mapping $\tau' : T_\Sigma \to T_\Delta$, where $\tau'(s) = \tau(\sigma)(\tau'(s_1), \ldots, \tau'(s_k))$ for every $s = \sigma(s_1, \ldots, s_k)$ with $k \geq 0$, $\sigma \in \Sigma^{(k)}$, and $s_1, \ldots, s_k \in T_\Sigma$. For each tree $t \in T_\Delta$, the set $\{s \in T_\Sigma \mid t = \tau'(s)\}$ is finite. Finally, τ' (and hence τ) extends to the mapping $\hat{\tau} : K\langle\langle T_\Sigma \rangle\rangle \to K\langle\langle T_\Delta \rangle\rangle$, where

$$(\hat{\tau}(S), t) = \sum_{s \in T_\Sigma,\, \tau'(s) = t} (S, s).$$

for each $S \in K\langle\langle T_\Sigma \rangle\rangle$ and $t \in T_\Delta$.

We will need the following result which was proved in Lemma 3.4 of [12].

Proposition 4. *Let K be commutative. If $S \in \mathrm{REG}(\Sigma, K)$ and $\tau : \Sigma \to \Delta$ is a relabeling, then $\hat{\tau}(S) \in \mathrm{REG}(\Delta, K)$.*

Now let K' be another semiring and $h : K \to K'$ a semiring homomorphism. Then h extends to the mapping $h : K\langle\langle T_\Sigma \rangle\rangle \to K'\langle\langle T_\Sigma \rangle\rangle$ by defining $h(S) = h \circ S$ for every $S \in K\langle\langle T_\Sigma \rangle\rangle$. We will need the following result, cf. Lemma 3 of [7] and Theorem 3.9 of [18].

Proposition 5. *(a) For every $S \in \mathrm{REG}(\Sigma, K)$ and semiring homomorphism $h : K \to K'$, we have $h(S) \in \mathrm{REG}(\Sigma, K')$, i.e., $h(\mathrm{REG}(\Sigma, K)) \subseteq \mathrm{REG}(\Sigma, K')$. (b) If, in addition, h is surjective, then $h(\mathrm{REG}(\Sigma, K)) = \mathrm{REG}(\Sigma, K')$.*

3 Weighted tree-walking automata

In this section we define weighted tree-walking automata. For the definition of the classical (unweighted) case, see [1] and [13].

Informally, a weighted tree-walking automaton A over a semiring K works as follows on input tree s. It is equipped with a pointer that walks on the edges of s, obeying its state behaviour. In a given moment of the computation, the current node is the node of s pointed by the pointer. Each computation step is determined by the state of the given moment, the label, and the child number of the current node. Besides, each computation step has a weight over K. An accepting run of A on s is a walk starting at the root of s in the initial state and finishing at an arbitrary node of s in the (only) accepting state. The weight of an accepting run is the product of the weights of the corresponding computation steps. Finally, the weight of s, computed by A, is the sum of the weights of the accepting runs of A on s. Now we give the exact definition.

Syntax

For each $\sigma \in \Sigma$, the set of *instructions determined by* σ is the set

$$I_{\sigma,j} = \begin{cases} \{stay, up, down_i \mid 1 \leq i \leq rank(\sigma)\} & \text{if } j > 0, \\ \{stay, down_i \mid 1 \leq i \leq rank(\sigma)\} & \text{if } j = 0. \end{cases}$$

Let K be a semiring. A *weighted tree-walking automaton* (shortly: *wtwa*) *over* K is a system $A = (Q, \Sigma, q_0, q_a, P)$, where

- Q is a finite set, the *set of states*,
- Σ is the *input ranked alphabet*,
- $q_0, q_a \in Q$ are the *initial* and *accepting state*, respectively, such that $q_0 \neq q_a$, and
- P is a finite *set of rules* of the form $\langle q, \sigma, j\rangle \xrightarrow{a} \langle q', \varphi\rangle$, where $q \in Q - \{q_a\}$, $\sigma \in \Sigma$, $j \in \{0, \ldots, maxrank(\Sigma)\}$, $a \in K$ such that $a \neq 0$, $q' \in Q - \{q_0\}$ and $\varphi \in I_{\sigma,j}$.

A rule $\pi = \langle q, \sigma, j\rangle \xrightarrow{a} \langle q', \varphi\rangle$ is called both a q-rule and a σ-rule. The left-state, right-state, and the weight of π are defined by $lstate(\pi) = q$, $rstate(\pi) = q'$, and $wt(\pi) = a$, respectively. Moreover, we let $test(\pi) = (\sigma, j)$. Rules with left-state q_0 (right-state q_a) are called *initial rules* (*accepting rules*).

Computation relation

Let $s \in T_\Sigma$ be an input tree. For a node $u \in pos(s)$, we define $test_s(u) = (\sigma, j)$, where $\sigma = s(u)$ and $j = childno(u)$. If s is clear from the context, then we will write $test(u)$ for $test_s(u)$. Now assume that $test(u) = (\sigma, j)$ and let $\varphi \in I_{\sigma,j}$ be an instruction. Then we define the effect of φ on u by

$$\varphi(u) = \begin{cases} u & \text{if } \varphi = stay, \\ parent(u) & \text{if } \varphi = up, \\ ui & \text{if } \varphi = down_i. \end{cases}$$

A *configuration* (of A over s) is a tuple $\langle q, u\rangle$, where $q \in Q$ and $u \in pos(s)$. We denote the set of configurations of A over s by $C_{A,s}$ and write just C_s if A is clear from the context. In particular, $\langle q_0, \varepsilon\rangle$ is the *initial configuration* and $\langle q_a, u\rangle$ is an *accepting configuration* for every $u \in pos(s)$.

Let $\pi \in P$ be a rule. The *π-transition relation (of A with respect to s)* is the binary relation $\vdash_{A,s,\pi}$ over C_s defined as follows. For arbitrary configurations $\langle q, u\rangle$, $\langle q', u'\rangle \in C_s$ we have $\langle q, u\rangle \vdash_{A,s,\pi} \langle q', u'\rangle$ if and only if

- π has the form $\langle q, \sigma, j\rangle \xrightarrow{a} \langle q', \varphi\rangle$,
- $test(u) = (\sigma, j)$, and $u' = \varphi(u)$.

If A is clear from the context, then we write $\vdash_{s,\pi}$ for $\vdash_{A,s,\pi}$. Finally, we define the *transition relation $\vdash_s$ (of A with respect to s)* as $\vdash_s = \bigcup_{\pi \in P} \vdash_{s,\pi}$.

Looping property

We define A to be *looping* if there is an input tree $s \in T_\Sigma$ and a configuration $\langle q, u\rangle \in C_s$ such that $\langle q_0, \varepsilon\rangle \vdash_s^* \langle q, u\rangle \vdash_s^+ \langle q, u\rangle$. Otherwise, A is *non-looping*. We call $\langle q, u\rangle$ a *looping configuration*. The looping problem of wtwa is decidable. In fact, the decidability of the more general *circularity problem* is proved in a more general setting for pebble macro tree transducers, cf. Section 4. of [17].

Accepting runs

Let $s \in T_\Sigma$ be an input tree. An *accepting run (of A on s)* is a string

$$r = \langle q_0, u_0\rangle\pi_0\langle q_1, u_1\rangle\pi_1 \ldots \pi_k\langle q_{k+1}, u_{k+1}\rangle$$

over $C_s \cup P$, where $k \geq 0$, $\langle q_0, u_0\rangle, \ldots, \langle q_{k+1}, u_{k+1}\rangle \in C_s$ with $u_0 = \varepsilon$ and $q_{k+1} = q_a$, $\pi_0, \ldots, \pi_k \in P$, and $\langle q_i, u_i\rangle \vdash_{s,\pi_i} \langle q_{i+1}, u_{i+1}\rangle$ for each $0 \leq i \leq k$. The *weight of r* is $wt(r) = wt(\pi_0) \cdot \ldots \cdot wt(\pi_k)$. We denote the set of accepting runs of A on s by $Accrun_s$.

Tree series recognized by non-looping wtwa

Let A be a non-looping wtwa. The tree series $S_A \in K\langle\langle T_\Sigma\rangle\rangle$ *recognized by A* is defined as

$$(S_A, s) = \sum_{r \in Accrun_s} wt(r).$$

for each $s \in T_\Sigma$. Note that we need non-looping property to guarantee that the set $Accrun_s$ is finite and hereby the above sum has finitely many members. We denote by $\mathrm{TWA}(\Sigma, K)$ the class of tree series that are recognizable by non-looping wtwa over Σ and K.

Example

Next we give an example of wtwa which recognizes the tree series $width_{\alpha\beta}$ defined in Example 2.

Example 6. Let $A = (\{q_0, q_a, q_1, q_2, q_{up}^{(l)}, q_{up}^{(r)}\}, \Sigma, q_0, q_a, P)$ be a wtwa over Arct, where Σ is the particular alphabet in Example 2 and P is the set of the following rules.
Initial rules:

$$\pi_1 : \langle q_0, \sigma, 0\rangle \xrightarrow{0} \langle q_1, down_1\rangle \quad \pi_2 : \langle q_0, \sigma, 0\rangle \xrightarrow{0} \langle q_1, down_2\rangle$$
$$\pi_3 : \langle q_0, \gamma, 0\rangle \xrightarrow{0} \langle q_1, down_1\rangle$$

Intermediate rules:

$$\begin{aligned}
&\pi_4 : \langle q_1, \sigma, 1\rangle \xrightarrow{0} \langle q_1, down_1\rangle && \pi_{15} : \langle q_{up}^{(l)}, \sigma, 1\rangle \xrightarrow{1} \langle q_{up}^{(l)}, up\rangle\\
&\pi_5 : \langle q_1, \sigma, 1\rangle \xrightarrow{0} \langle q_1, down_2\rangle && \pi_{16} : \langle q_{up}^{(l)}, \sigma, 2\rangle \xrightarrow{1} \langle q_{up}^{(r)}, up\rangle\\
&\pi_6 : \langle q_1, \sigma, 2\rangle \xrightarrow{0} \langle q_1, down_1\rangle && \pi_{17} : \langle q_{up}^{(r)}, \sigma, 1\rangle \xrightarrow{1} \langle q_{up}^{(l)}, up\rangle\\
&\pi_7 : \langle q_1, \sigma, 2\rangle \xrightarrow{0} \langle q_1, down_2\rangle && \pi_{18} : \langle q_{up}^{(r)}, \sigma, 2\rangle \xrightarrow{1} \langle q_{up}^{(r)}, up\rangle\\
&\pi_8 : \langle q_1, \gamma, 1\rangle \xrightarrow{0} \langle q_1, down_1\rangle && \pi_{19} : \langle q_{up}^{(l)}, \gamma, 1\rangle \xrightarrow{1} \langle q_{up}^{(l)}, up\rangle\\
&\pi_9 : \langle q_1, \gamma, 2\rangle \xrightarrow{0} \langle q_1, down_1\rangle && \pi_{20} : \langle q_{up}^{(l)}, \gamma, 2\rangle \xrightarrow{1} \langle q_{up}^{(r)}, up\rangle\\
&\pi_{10} : \langle q_1, \alpha, 1\rangle \xrightarrow{1} \langle q_{up}^{(l)}, up\rangle && \pi_{21} : \langle q_2, \sigma, 1\rangle \xrightarrow{1} \langle q_2, down_1\rangle\\
&\pi_{11} : \langle q_1, \alpha, 2\rangle \xrightarrow{1} \langle q_{up}^{(r)}, up\rangle && \pi_{22} : \langle q_2, \sigma, 1\rangle \xrightarrow{1} \langle q_2, down_2\rangle\\
&\pi_{12} : \langle q_{up}^{(l)}, \sigma, 0\rangle \xrightarrow{1} \langle q_2, down_2\rangle && \pi_{23} : \langle q_2, \sigma, 2\rangle \xrightarrow{1} \langle q_2, down_1\rangle\\
&\pi_{13} : \langle q_{up}^{(l)}, \sigma, 1\rangle \xrightarrow{1} \langle q_2, down_2\rangle && \pi_{24} : \langle q_2, \sigma, 2\rangle \xrightarrow{1} \langle q_2, down_2\rangle\\
&\pi_{14} : \langle q_{up}^{(l)}, \sigma, 2\rangle \xrightarrow{1} \langle q_2, down_2\rangle && \pi_{25} : \langle q_2, \gamma, 1\rangle \xrightarrow{1} \langle q_2, down_1\rangle\\
& && \pi_{26} : \langle q_2, \gamma, 2\rangle \xrightarrow{1} \langle q_2, down_1\rangle
\end{aligned}$$

Accepting rules:

$$\pi_{27} : \langle q_2, \beta, 1\rangle \xrightarrow{0} \langle q_a, stay\rangle \quad \pi_{28} : \langle q_2, \beta, 2\rangle \xrightarrow{0} \langle q_a, stay\rangle$$

Note that A is non-looping. Moreover, A works on an input tree s as follows.

1) In the first phase, A moves the pointer nondeterministically to an α-node of s with its rules π_1-π_9. The weights of these steps are 0.

2) Then A moves its pointer upwards (using π_{10} and π_{11}), such that states $q_{up}^{(l)}$ and $q_{up}^{(r)}$ store whether the previous step was made from the left child or the right child, respectively. If A is in state $q_{up}^{(l)}$ and the pointed node is labeled by σ, then A nondeterministically decides whether to continue moving up (using $\pi_{15} - \pi_{20}$) or to move down to the second child in state q_2 (using $\pi_{12} - \pi_{14}$). Each of these steps has weight 1. (We do not need γ-rules with left-state $q_{up}^{(r)}$ because going up to a node labelled by γ leads to state $q_{up}^{(l)}$.)

3) In the third phase, A searches nondeterministically for a β-node of s by descending with its rules π_{21}-π_{26}. Each of these steps has weight 1. If it finds a β-node, then the run terminates in the accepting state q_a with the 0-weighted rules π_{27} or π_{28}.

Now it is easy to see that an accepting run r of A on s contains an α-β path in s and that the weight of r is the length of that α-β path. Hence we conclude that, for each input tree $s \in T_\Sigma$, the wtwa A computes $(width_{\alpha\beta}, s)$ and thus it recognizes the tree series $width_{\alpha\beta}$.

An accepting run of A on the input tree s of Example 2 is

$$\begin{aligned}
r = {} & \langle q_0, \varepsilon\rangle \pi_1 \langle q_1, 1\rangle \pi_4 \langle q_1, 11\rangle \pi_4 \langle q_1, 111\rangle \pi_5 \langle q_1, 1112\rangle \pi_{11} \langle q_{up}^{(r)}, 111\rangle \pi_{17}\\
& \langle q_{up}^{(l)}, 11\rangle \pi_{15} \langle q_{up}^{(l)}, 1\rangle \pi_{13} \langle q_2, 12\rangle \pi_{26} \langle q_2, 121\rangle \pi_{21} \langle q_2, 1211\rangle \pi_{27} \langle q_a, 1211\rangle. \quad (3.1)
\end{aligned}$$

In fact, r contains the longest α-β-path in s, which can be seen in Fig. 1.

Tree-walking automata as the Boolean case of wtwa

Later in the paper we will consider wtwa over $\mathbb{B}$. For such a wtwa A, the weight of each rule and hence of each accepting run is 1. Moreover, since 1+1=1 in $\mathbb{B}$, we do not need the restriction that A is non-looping to define the semantics of A. Hence, we define S_A for an (arbitrary) wtwa A over $\mathbb{B}$ by the formula used for a non-loping one. It is easily seen that $(S_A, s) = 1$, i.e., A *accepts* s, if and only if there is an accepting run of A on the input tree s (even in the case that A has infinitely many accepting runs on s, i.e., that $Accrun_s$ is infinite). In this way A can also be considered as a *tree recognizer* and we obtain the *tree-walking automata (twa)* of [1], cf. also [13]. In fact, we call a wtwa A over $\mathbb{B}$ a twa, we drop the weight 1 from the specification of its rules, and we denote by $L(A)$ the set of trees accepted by A and call it the tree language *recognized by* A.

Twa with one pebble

We will also need a more general tree recognizer, the so called *one pebble tree-walking automata (*1*-ptwa)*, see e.g. [13].

A 1-ptwa A works as follows on input tree s. Similarly to a twa, A is equipped with a pointer that walks on the edges of s. Moreover, A has one pebble that is able to mark a node of s, i.e., that can be dropped at and lifted from the current node. After marking a node by the pebble, A can walk away from the marked node. When accessing a node, A is able to test whether the current node is marked by the pebble or not and the further computation of A may depend on the result of this test. This gives an extra computation power for A. Like a twa, A will accept s if and only if it has at least one accepting run on s, and the set of trees accepted by A is denoted by $L(A)$. Again, we call $L(A)$ the tree series *recognized by* A. For a formal definition of a 1-ptwa, the reader is advised to consult [13] or [14].

The tree languages recognized by 1-ptwa (and hence by twa) are effectively regular, which is proved in [13] for ptwa and also in [24] for the more general *pebble alternating tree-walking automata*. This yields the following proposition.

Proposition 7. *The tree languages recognized by* 1*-ptwa (and hence by twa) are effectively regular.*

4 The recognizing power of non-looping wtwa

We can prove our main results for non-looping wtwa over commutative semirings.

> *Therefore in the rest of this paper K denotes a commutative semiring and $A = (Q, \Sigma, q_0, q_a, P)$ denotes a non-looping wtwa over K.*

We will prove that the tree series recognized by A is effectively regular. For this, we will consider annotated input trees. These are trees the σ-nodes of which are annotated by sets of σ-rules in P, where $\sigma \in \Sigma$. The annotation of a node by a set of rules intuitively means that those rules may be applied at that node. Since A is

non-looping, it will be sufficient to consider consistent annotations. More exactly, let $P(\sigma)$ be the set of all σ-rules in P and define a pair $\langle \sigma, J\rangle \in \Sigma \times P(\sigma)$ with $J = \{\pi_1, \ldots, \pi_m\}$ to be *consistent* if the following conditions hold:

a) $\mathit{test}(\pi_1) = \ldots = \mathit{test}(\pi_m)$ and

b) the states $\mathit{lstate}(\pi_1), \ldots, \mathit{lstate}(\pi_m)$ are pairwise different.

Then, we introduce the ranked alphabet $\Sigma(P) \subseteq \Sigma \times P(\sigma)$ such that

$$\Sigma(P)^{(k)} = \{\langle \sigma, J\rangle \in \Sigma^{(k)} \times P(\sigma) \mid \langle \sigma, J\rangle \text{ is consistent}\}$$

for every $k \geq 0$. Finally we define $\mathit{rules}(\langle \sigma, J\rangle) = J$.

It is useful to introduce the tree series $\mathit{weight} \in K\langle\langle T_{\Sigma(P)}\rangle\rangle$ which associates an annotated tree $t \in T_{\Sigma(P)}$ with the product of the weights of the rules appearing in t. More exactly, for every $t \in T_{\Sigma(P)}$, we have

$$(\mathit{weight}, t) = \prod_{\substack{u \in \mathit{pos}(t) \\ \pi \in \mathit{rules}(t(u))}} \mathit{wt}(\pi),$$

where the empty product yields weight 1. We will need the following result.

Lemma 8. *The tree series weight is regular.*

Proof. Let $M = (\{q\}, \Sigma(P), R, \nu)$ be a wta such that $\nu(q) = 1$ and R the smallest set containing all rules $\langle \sigma, J\rangle(q, \ldots, q) \xrightarrow{a} q$ with $\langle \sigma, J\rangle \in \Sigma(P)$ such that $a = \prod_{\pi \in J} \mathit{wt}(\pi)$. It is easy to see that $S_M = \mathit{weight}$. □

In particular, we will be interested in those annotated trees which encode an accepting run. We call such trees run trees and define them in the following way.

Let $s \in T_\Sigma$ be an input tree and

$$r = \langle q_0, u_0\rangle \pi_0 \langle q_1, u_1\rangle \pi_1 \ldots \langle q_k, u_k\rangle \pi_k \langle q_a, u_{k+1}\rangle$$

an accepting run on s. The *run tree of s and r* is the tree $\mathit{rtree}(s, r) \in T_{\Sigma \times 2^P}$ defined by the following conditions:

- $\mathit{pos}(\mathit{rtree}(s, r)) = \mathit{pos}(s)$ and
- for each $u \in \mathit{pos}(\mathit{rtree}(s, r))$ we have

$$\mathit{rtree}(s, r)(u) = \langle s(u), \{\pi_i \in P \mid 0 \leq i \leq k,\ u_i = u\}\rangle.$$

We say that $\mathit{rtree}(s, r)$ *encodes* r. Moreover, the *set of run trees of s* is $\mathit{Rtree}_s = \{\mathit{rtree}(s, r) \mid r \in \mathit{Accrun}_s\}$. In Fig. 2 we visualize the run tree $\mathit{rtree}(s, r)$, where s is the input tree of Example 2 and r is the run of A on s appearing in Example 6.

We can prove easily that run trees are in fact trees over $\Sigma(P)$.

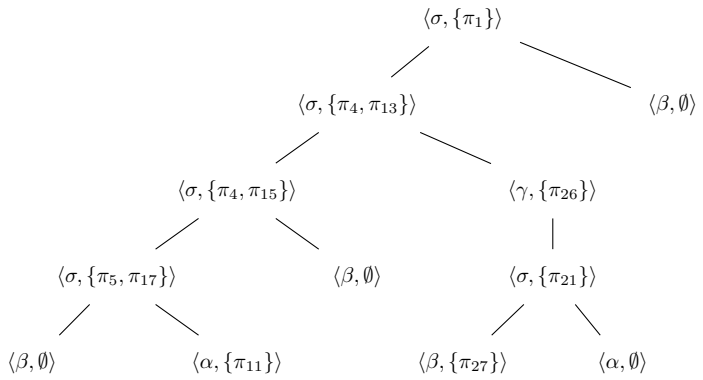

Figure 2: A run tree.

Lemma 9. *For every $s \in T_\Sigma$, we have $Rtree_s \subseteq T_{\Sigma(P)}$.*

Proof. Let $t \in Rtree_s$ be a run tree and $u \in pos(t)$ $(= pos(s))$ a node. Assume that $t(u) = \langle \sigma, \{\pi_1, \ldots, \pi_m\}\rangle$, where $\pi_1, \ldots, \pi_m \in P$. We show that both a) and b) of the definition of consistency hold. Since $t \in Rtree_s$, there is an accepting run $r \in Accrun_s$ such that $t = rtree(s, r)$.

Condition a) obviously holds because a rule π can be applied at a node u of s only if $test(\pi) = test(u)$.

We prove b) by contradiction. Assume that there are $q \in Q$, μ, and ν such that $1 \leq \mu \neq \nu \leq m$ and $q = lstate(\pi_\mu) = lstate(\pi_\nu)$. Then it directly follows that configuration $\langle q, u\rangle$ occurs twice in the accepting run r, i.e., $\langle q_0, \varepsilon\rangle \vdash^*_{A,s} \langle q, u\rangle \vdash^+_{A,s} \langle q, u\rangle$. This contradicts the fact that A is non-looping. □

We will also need the following straightforward result.

Lemma 10. *Let $s \in T_\Sigma$ be an input tree. For each accepting run $r \in Accrun_s$ we have $wt(r) = (weight, rtree(s, r))$.*

Next we show that $Accrun_s$ and $Rtree_s$ have the same number of elements. For this, we define the mapping $\theta_s : Accrun_s \to Rtree_s$ such that $\theta_s(r) = rtree(s, r)$ for each $r \in Accrun_s$ and show that it is a bijection.

Lemma 11. *The mapping θ_s is a bijection.*

Proof. It should be clear that θ_s is surjective. To show that it is injective, we consider $r_1, r_2 \in Accrun_s$ with $rtree(s, r_1) = rtree(s, r_2)$ and show that $r_1 = r_2$. Let

$$r_1 = \langle q_0, u_0\rangle\pi_0\langle q_1, u_1\rangle\pi_1 \ldots \langle q_k, u_k\rangle\pi_k\langle q_a, u_{k+1}\rangle$$

and

$$r_2 = \langle q_0, u'_0\rangle\pi'_0\langle q'_1, u'_1\rangle\pi'_1 \ldots \langle q'_l, u'_l\rangle\pi'_l\langle q_a, u'_{l+1}\rangle,$$

where $u_0 = u'_0 = \varepsilon$.

First we prove by contradiction that $\pi_i = \pi'_i$ for every $0 \leq i \leq \min\{k, l\}$. Assume that there is an i such that $\pi_i \neq \pi'_i$ and $\pi_j = \pi'_j$ for every $0 \leq j < i$. The latter implies $q_j = q'_j$ and $u_j = u'_j$ for every $0 \leq j \leq i$. Since $\pi_i \in \mathit{rules}(\mathit{rtree}(s, r_2)(u_i))$ and, in particular, $\mathit{rtree}(s, r_1)(u_i) = \mathit{rtree}(s, r_2)(u_i)$, there is an $i < m \leq l$ such that $u'_m = u_i$ and $\pi'_m = \pi_i$. If $i = 0$, this is a contradiction because the left-state of π'_m cannot be q_0. If $0 < i$, then we get

$$\langle q_i, u_i \rangle = \langle q'_i, u'_i \rangle \vdash_s^+ \langle q'_m, u'_m \rangle = \langle q_i, u_i \rangle,$$

which is a contradiction again because A is non-looping. Hence the statement follows.

This statement and the fact that there are no rules with left-state q_a imply that $k = l$. From the latter $u_{k+1} = u'_{k+1}$ follows, hence $r_1 = r_2$. □

Thus we obtain the following, which we need later.

Corollary 12. *For each input tree $s \in T_\Sigma$ we have $(S_A, s) = \sum_{t \in \mathit{Rtree}_s} (\mathit{weight}, t)$.*

Proof.

$$\begin{aligned}
(S_A, s) &= \sum_{r \in \mathit{Accrun}_s} \mathit{wt}(r) \\
&= \sum_{r \in \mathit{Accrun}_s} (\mathit{weight}, \mathit{rtree}(s, r)) && \text{(by Lemma 10)} \\
&= \sum_{t \in \mathit{Rtree}_s} (\mathit{weight}, t) && \text{(by Lemma 11).}
\end{aligned}$$

□

The *set of run trees of A* is $\mathit{Rtree}_A = \bigcup_{s \in T_\Sigma} \mathit{Rtree}_s$.

We will show that Rtree_A is a regular tree language. In fact, we will construct a twa A' and a 1-ptwa A'' and then show that Rtree_A is the Boolean combination $L(A') \cap \overline{L(A'')}$ of the tree languages recognized by them.

The twa A' works on trees over $\Sigma(P)$ and accepts all the run trees of A, i.e., $\mathit{Rtree}_A \subseteq L(A')$. Let $A' = (Q, \Sigma(P), q_0, q_a, P')$, where, for every $\langle \sigma, J \rangle \in \Sigma(P)$, the set P' contains the rule

$$\langle q, \langle \sigma, J \rangle, j \rangle \to \langle q', \varphi \rangle$$

if and only if the rule $\langle q, \sigma, j \rangle \to \langle q', \varphi \rangle$ is in J. Note that A' is deterministic in the sense that, due to condition b) of the definition of consistency, it has no different rules with the same left-hand side.

Let us consider an input tree $t \in T_{\Sigma(P)}$ to A' and let $s \in T_\Sigma$ be the tree obtained from t by dropping the rule sets from its labels. It should be clear that A' simulates on t the steps of A on s which apply the rules in the nodes of t in a state-to-state and node-to-node manner. Hence A' accepts t if and only if A accepts s applying the rules in the nodes of t. In particular, for every $s \in T_\Sigma$, the twa A' accepts each $t \in \mathit{Rtree}_s$ because such a t encodes an accepting run of s. (Here we use Lemma 9.)

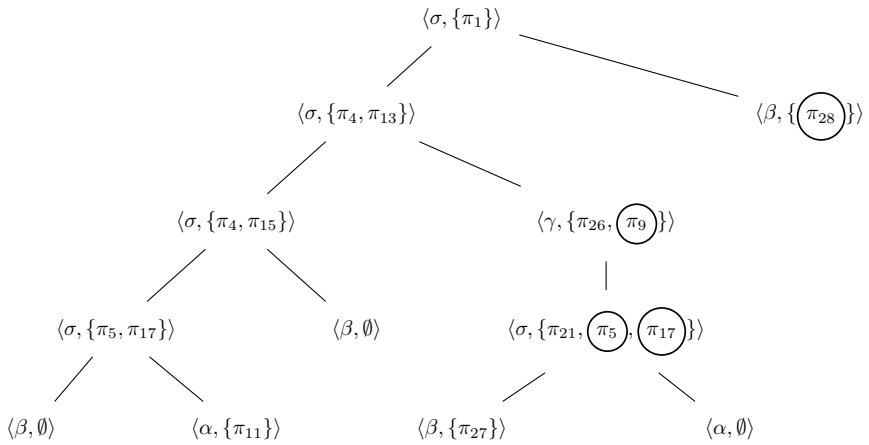

Figure 3: A tree accepted by A'. The circled rules are superfluous.

Hence $Rtree_A \subseteq L(A')$. Note that A' may also accept trees which do not encode accepting runs because they contain some "superfluous rules" in their labels.

In Fig. 3 we show a tree which is accepted by A' but is not a run tree of A, where A is now the wtwa of Example 6. At the same time, the tree contains the accepting run shown in Fig. 2.

In order to clarify the relation between $Rtree_A$ and $L(A')$, we define the concept of the superfluous rule in an exact way.

Let $t \in L(A')$ be a tree and $u \in pos(t)$. A rule $\pi \in rules(t(u))$ *is superfluous at node* u *(in* t*)* if the tree t' is also in $L(A')$, where t' is obtained from t by dropping π from the set $rules(t(u))$. If this is the case, then we say that t contains a superfluous rule.

Lemma 13. $Rtree_A = \{t \in L(A') \mid t$ *contains no superfluous rules*$\}$.

Proof. If $t \in Rtree_A$, then $t \in L(A')$. We show that t contains no superfluous rules by contradiction. For this, let $s \in T_\Sigma$ and $r \in Rtree_s$ be such that $t = rtree(s, r)$. Consider an accepting run

$$r' = \langle q_0, u_0\rangle \Pi_0 \langle q_1, u_1\rangle \Pi_1 \ldots \langle q_k, u_k\rangle \Pi_k \langle q_a, u_{k+1}\rangle$$

of A' on t and let π_i be the rule of A corresponding to Π_i, $1 \leq i \leq n$, see the definition of A' above. It should be clear that the sequence obtained from r' by replacing Π_i with π_i for every $1 \leq i \leq n$ is the accepting run r of A on s. Assume that t contains a superfluous rule. Since r contains all rules in the nodes of t there is an index i such that π_i is superfluous at u_i. Assume that i is minimal. Since π_0 is the only initial rule in r, we have $i > 0$. Let t' be the tree obtained from t by dropping π_i (whose left-state is q_i) from $rules(t(u_i))$. Note that, by definition, A' accepts t' with an accepting run $\overline{r}$. Since i is minimal and the run r' simulates r, the first $i-1$ rules applied in $\overline{r}$ are $\Pi_1, \ldots, \Pi_{i-1}$. Then, the left-state of the ith rule

of A' in $\overline{r}$ is also q_i. This means, by the definition of A', that there is a rule of A in $rules(t'(u_i))$ with left state q_i, i.e, there are more than one rules in $rules(t(u_i))$ with left-state q_i. This contradicts the definition of $\Sigma(P)$, hence t does not contain any superfluous rule.

To prove the other inclusion, assume that $t \in L(A')$ contains no superfluous rules. Let $s \in T_\Sigma$ be the tree obtained from t by dropping the rule sets from its labels. By the above discussion concerning A', we get that A accepts the tree s applying the rules in the nodes of t. Let r be the so obtained accepting run on s. Since all rules in the nodes of t are applied, t encodes r, hence $t \in Rtree_s \subseteq Rtree_A$. □

Next we informally introduce a 1-pebble twa A'' that accepts those trees in $L(A')$ which contain superfluous rules. Intuitively, the 1-ptwa A'' works as follows on an input tree $t \in T_{\Sigma(P)}$.

Phase 1: A'' nondeterministically chooses a node u of t and places the pebble at u. Assume that $t(u) = \langle \sigma, J \rangle$. If $J = \emptyset$ then, there is no next step and the computation terminates without acceptance.

Phase 2: If $J \neq \emptyset$, then A'' nondeterministically picks a rule $\pi = \langle q, \sigma, j \rangle \to \langle q', \varphi \rangle \in J$. Let $\Pi = \langle q, \langle \sigma, J \rangle, j \rangle \to \langle q', \varphi \rangle$ be the rule of A' corresponding to π. Our A'' stores Π in its state.

Phase 3: In the rest, A'' computes deterministically. First A'' moves back to the root node and then it simulates A' on t. However, during the simulation of A', our A'' is not allowed to use the rule Π (stored in its memory) at node u (being marked by the pebble).

It is clear that A'' has an accepting computation on t if and only if A' has an accepting computation on t which does not apply at least one rule in a label of t. Hence, we obtain the following result.

Lemma 14. $L(A'') = \{t \in T_{\Sigma(P)} \mid t \in L(A')$ *and* t *contains a superfluous rule*$\}$.

Now we can prove the following statement easily.

Corollary 15. *The tree language* $Rtree_A$ *is effectively regular.*

Proof. It follows from Lemmata 13 and 14 that $Rtree_A = L(A') \cap \overline{L(A'')}$. Finally, by Propositions 1 and 7, we obtain that $Rtree_A$ is effectively regular. □

Now we are ready to prove our main result. For this we will need the relabeling $\tau : \Sigma(P) \to \Sigma$ which drops the rule component from each symbol, i.e., which is defined by $\tau(\langle \sigma, J \rangle) = \sigma$ for every $\langle \sigma, J \rangle \in \Sigma(P)$. Note, for later use, that $Rtree_s = \{t \in Rtree_A \mid \tau'(t) = s\}$.

Theorem 16. *The tree series S_A is effectively regular.*

Proof. By Corollary 15 and Proposition 3(a), the characteristic tree series $\mathcal{X}_{Rtree_A} : T_{\Sigma(P)} \to K$ is effectively regular. Moreover, since the tree series *weight* is effectively regular (see Lemma 8), we obtain by Proposition 3(b) that the Hadamard product $S = \mathcal{X}_{Rtree_A} \odot weight$ is an effectively regular tree series.

Let us note that for each tree $t \in T_{\Sigma(P)}$ we have

$$(S,t) = \begin{cases} (weight, t) & \text{if } t \in Rtree_A, \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, $\hat{\tau}(S)$ is a tree series in $K\langle\langle T_\Sigma \rangle\rangle$, were $\tau : \Sigma(P) \to \Sigma$ is the relabeling introduced above, and it follows from Proposition 4 and the fact that S is regular that $\hat{\tau}(S)$ is effectively regular. Finally, for every tree $s \in T_\Sigma$,

$$\begin{aligned} (\hat{\tau}(S), s) &= \sum_{t \in T_{\Sigma(P)},\, \tau'(t)=s} (S,t) &&= \sum_{t \in T_{\Sigma(P)},\, \tau'(t)=s} (\mathcal{X}_{Rtree_A} \odot weight, t) \\ &= \sum_{t \in Rtree_A,\, \tau'(t)=s} (weight, t) &&= \sum_{t \in Rtree_s} (weight, t) \\ &= (S_A, s), \end{aligned}$$

where the last equality is justified by Corollary 12. Hence $S_A = \hat{\tau}(S)$, which proves that S_A is also effectively regular. □

Since A is an arbitrary non-looping wtwa over K, we also proved the following result.

Theorem 17. $\mathrm{TWA}(\Sigma, K) \subseteq \mathrm{REG}(\Sigma, K)$.

In the remainder of the paper we show that the inclusion is strict provided K is proper. For this, let K' be another commutative semiring and $h : K \to K'$ a semiring homomorphism. Then we can prove easily the analogy of Proposition 5 for wtwa, i.e., that h preserves recognizability by wtwa.

Proposition 18. *(a) For every $S \in \mathrm{TWA}(\Sigma, K)$ and semiring homomorphism $h : K \to K'$, we have $h(S) \in \mathrm{TWA}(\Sigma, K')$, i.e., $h(\mathrm{TWA}(\Sigma, K)) \subseteq \mathrm{TWA}(\Sigma, K')$. (b) If, in addition, h is surjective, then $h(\mathrm{TWA}(\Sigma, K)) = \mathrm{TWA}(\Sigma, K')$.*

Proof. Let $A = (Q, \Sigma, q_0, q_a, P)$ be a wtwa over K. Construct the wtwa $A = (Q, \Sigma, q_0, q_a, P')$ such that $P' = \{\langle q, \sigma, j\rangle \overset{h(a)}{\to} \langle q', \varphi\rangle \mid \langle q, \sigma, j\rangle \overset{a}{\to} \langle q', \varphi\rangle \in P\}$. It is easy to see that $S_{A'} = h(S_A)$. Moreover, if h is surjective, then every wtwa over K' appears as the "image" of a wtwa over K. □

Now we recall an important result from [4], namely, that there is a regular tree language which cannot be recognized by any twa. It is well known that regular tree languages and regular tree series over $\mathbb{B}$ can be identified, cf. e.g. [18], and it is easy to see that the same holds for tree languages recognizable by twa as well as

for tree series recognizable by wtwa over $\mathbb{B}$. Thus, the above mentioned result of [4] can be written in the form $\mathrm{REG}(\Sigma, \mathbb{B}) - \mathrm{TWA}(\Sigma, \mathbb{B}) \neq \emptyset$. Then we can prove the following result and thus generalize the main result of [4] for tree series recognizable by wtwa over proper commutative semirings.

Theorem 19. *If K is proper, then* $\mathrm{REG}(\Sigma, K) - \mathrm{TWA}(\Sigma, K) \neq \emptyset$.

Proof. Assume, on the contrary, that $\mathrm{REG}(\Sigma, K) \subseteq \mathrm{TWA}(\Sigma, K)$. Since K is proper, by Theorem 2.1 of [27], there is a surjective homomorphism $h : K \to \mathbb{B}$. Then obviously we have $h(\mathrm{REG}(\Sigma, K)) \subseteq h(\mathrm{TWA}(\Sigma, K))$. On the other hand, by Propositions 5 and 18, we have $h(\mathrm{REG}(\Sigma, K)) = \mathrm{REG}(\Sigma, \mathbb{B})$ and $h(\mathrm{TWA}(\Sigma, K)) = \mathrm{TWA}(\Sigma, \mathbb{B})$, which is a contradiction. □

5 Conclusion and an open problem

We generalized tree-walking automata of [1] by equipping each transition rule with a weight taken from a semiring K. For two reasons, we considered the non-looping model, i.e., which cannot fall into infinite computation from the initial configuration. The first reason is that the weight of an input tree s is defined as the sum of the weights of the accepting runs on s. The second one is that the proofs of Corollaries 12 and 15, and hence of Theorem 16 work only for non-looping wtwa.

If a wtwa is looping, then there may be infinitely many accepting runs on s, hence computing the weight of s leads to an infinite sum in the semiring. This problem can be handled by considering underlying semirings which are *complete*, i.e., in which the sum of infinitely many elements exists [21, 22]. Therefore, it is an open problem whether our main result can be generalized to arbitrary (including looping) wtwa over complete semirings.

Acknowledgment

We are grateful to the anonymous referees for their effort and valuable comments.

References

[1] A. V. Aho and J. D. Ullman. Translations on a context–free grammar. *Inform. Control*, 19:439–475, 1971.

[2] A. Alexandrakis and S. Bozapalidis. Weighted grammars and Kleene's theorem. *Information Processing Letters*, 24(1):1–4, January 1987.

[3] J. Berstel and C. Reutenauer. Recognizable power series on trees. *Theoret. Comput. Sci.*, 18:115–148, 1982.

[4] M. Bojańczyk and T. Colcombet. Tree-walking automata do not recognize all regular languages. *SIAM J. Comput.*, 38(3):658–701, 2008.

[5] B. Borchardt. The Myhill-Nerode Theorem for Recognizable Tree Series. In *7th International Conference on Developments in Language Theory, DLT'03, Szeged, Hungary, July 7-11, 2003, Proceedings*, volume 2710 of *LNCS*, pages 146–158. Springer-Verlag, July 2003.

[6] B. Borchardt. A Pumping Lemma and Decidability Problems for Recognizable Tree Series. *Acta Cybernet.*, 16(4):509–544, 2004.

[7] B. Borchardt, A. Maletti, B. Šešelja, A. Tepavčevic, and H. Vogler. Cut sets as recognizable tree languages. *Fuzzy Sets and Systems*, 157:1560–1571, 2006.

[8] B. Borchardt and H. Vogler. Determinization of Finite State Weighted Tree Automata. *Journal of Automata, Languages and Combinatorics*, 8(3):417–463, 2003.

[9] H. Comon, M. Dauchet, R. Gilleron, F. Jacquema, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 1997.

[10] J. Doner. Tree acceptors and some of their applications. *J. Comput. System Sci.*, 4:406–451, 1970.

[11] M. Droste, C. Pech, and H. Vogler. A Kleene theorem for weighted tree automata. *Theory of Computing Systems*, 38:1–38, 2005.

[12] M. Droste and H. Vogler. Weighted tree automata and weighted logics. *Theoret. Comput. Sci.*, 366:228–247, 2006.

[13] J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 72–83, London, UK, 1999. Springer-Verlag.

[14] J. Engelfriet and H. J. Hoogeboom. Automata with nested pebbles capture first-order logic with transitive closure. *Logical Methods in Computer Science 3*(2007), Issue 2, Paper 3.

[15] J. Engelfriet, H. J. Hoogeboom, and B. Samwel. XML transformation by tree-walking transducers with invisible pebbles. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '07)*, pages 63–72, New York, NY, USA, 2007. ACM Press.

[16] Z. Ésik and W. Kuich. Formal Tree Series. *Journal of Automata, Languages and Combinatorics*, 8:219–285, 2003.

[17] Z. Fülöp and L. Muzamel. Circularity and Decomposition Results for Pebble Macro Tree Transducers. *Journal of Automata, Languages and Combinatorics*, 13(1):3–44, 2008.

[18] Z. Fülöp and H. Vogler. Weighted tree automata and tree transducers. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*, Chapter 9. Springer-Verlag, 2009.

[19] F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[20] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 1–68. Springer-Verlag, 1997.

[21] U. Hebisch and H. J. Weinert. *Semirings - Algebraic Theory and Applications in Computer Science*. World Scientific, Singapore, 1998.

[22] W. Kuich. Formal power series over trees. In S. Bozapalidis, editor, *3rd International Conference on Developments in Language Th eory, DLT 1997, Thessaloniki, Greece, Proceedings*, pages 61–101. Aristotle University of Thessaloniki, 1998.

[23] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. *J. of Comput. Syst. Sci.*, 66:66–97, 2003.

[24] L. Muzamel. Pebble Alternating Tree-Walking Automata and Their Recognizing Power. *Acta Cybernetica*, 18(3):427–450, 2008.

[25] F. Neven. Automata theory for XML researchers. *SIGMOD Rec.*, 31(3):39–46, 2002.

[26] T. Schwentick. Automata for XML – A survey. *J. Comput. Syst. Sci.*, 73(3):289–315, 2007.

[27] H. Wang. On characters of semirings. *Houston J. Math.*, 23:391–405, 1997.

Received 11th July 2008