Minimizing Deterministic Weighted Tree Automata[☆,☆☆]

Andreas Maletti[*]

International Computer Science Institute, Berkeley, CA, USA

Abstract

Deterministic weighted tree automata (dwta) have found promising applications as language models in Natural Language Processing. It is known that dwta over commutative semifields can be effectively minimized. An efficient algorithm for minimizing them is presented. It is polynomial-time given that all operations of the semifield including the computation of the inverses are polynomial. More precisely, if the operations can be performed in constant time, then the algorithm constructs an equivalent minimal (with respect to the number of states) dwta in time $O(lmn)$ where l is the maximal rank of the input symbols, m is the number of (useful) transitions, and n is the number of states of the input dwta.

Key words: weighted tree automaton, minimization, tree series, determinism, partition refinement

1. Introduction

Weighted tree automata (wta) [1, 2, 3, 4] are a joint generalization of weighted string automata [5] and tree automata [6, 7]. Weighted string automata have successfully been applied as language models in Natural Language Processing due to their ability to easily incorporate n-gram models. Several toolkits (e.g., CARMEL [8], FIRE STATION [9], and OPENFST [10]) enable language engineers to rapidly prototype and develop language models because of the standardized implementation model and the consolidated algorithms made available by the toolkits.

In recent years, the trend toward more syntactical approaches in Natural Language Processing [11] sparked renewed interest in tree-based devices. The weighted tree automaton is the natural tree-based analogue of the weighted string automaton. First experiments with toolkits (e.g., TIBURON [12]) implementing tree-based devices show that the situation is not as consolidated here. In particular, many basic algorithms are missing in the weighted setting.

In general, a wta processes a given input tree stepwise using a locally specified transition behavior. During this process transition weights are combined using the operations (addition and

[☆]This is the full version of: *Minimizing Deterministic Weighted Tree Automata.* In Proc. 2nd Int. Conf. Language and Automata Theory and Applications. LNCS 5196, pages 357–372, Springer-Verlag, 2008.

[☆☆]This work was supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).

[*]Address: Universitat Rovira i Virgili, Departament de Filologies Romàniques, Av. Catalunya 35, 43002 Tarragona, Spain

Email address: andreas.maletti@urv.cat (Andreas Maletti)

URL: http://wwwtcs.inf.tu-dresden.de/~maletti (Andreas Maletti)

multiplication) of a semiring to form the weight associated with the input tree. Altogether, the wta thus recognizes (or computes) a mapping $\varphi\colon T_\Sigma \to A$ where T_Σ is the set of all input trees and A is the carrier set of the semiring. Such a mapping is also called a tree series, and if it can be computed by a wta, then it is called recognizable. The deterministically recognizable tree series are exactly those recognizable tree series that can be computed by deterministic wta (dwta). Recognizable and deterministically recognizable tree series have been thoroughly investigated (see [3, 13] and references provided therein). In fact, [4] and [14] show some recognizable tree series that are also deterministically recognizable.

In this contribution, we consider dwta, in which the additive operation of the semiring is irrelevant. To the author's knowledge, we propose the first polynomial-time minimization algorithm (not counting [15]) for dwta over semifields. A Myhill-Nerode theorem for tree series recognized by such automata is known [16]. However, it only asserts the existence of a unique, up to slight changes of representation, minimal (with respect to the number of states) dwta recognizing a given tree series. The construction of such a dwta, which is given in [16], is not effective, but with the help of the pumping lemma of [17] a simple exponential-time algorithm, which given a dwta constructs an equivalent minimal dwta, could easily be conceived. For (not necessarily deterministic) wta over fields the situation is similar. In [1, 18] the existence of a unique, up to slight changes of representation, minimal wta is proved. Moreover, [18] shows that minimization is effective by providing the analogue to the pumping argument already mentioned above in this more general setting. However, the trivially obtained algorithm is again exponential.

Angluin [19] learning algorithms exist for both general [20] and deterministic [21, 22] wta. In principle, those polynomial-time learning algorithms could also be used for minimization since they produce minimal wta recognizing the taught tree series. However, this also requires us to implement the oracle, which answers coefficient and equivalence queries. Although equivalence is decidable in polynomial time in both cases [23, 17], a simple implementation would return counterexamples of exponential size, which would again yield an exponential-time minimization algorithm. Clearly, this can be avoided for dwta by the method presented in this contribution.

Finally, let us mention the minimization procedures [24, 25] for deterministic weighted string automata. They rely on a weight normal-form obtained by a procedure called *pushing*. After this normal form is obtained, the weight of a transition is treated as an input symbol and the automaton is minimized as if it were unweighted. We do not follow this elegant approach here because we might have to explore several distributions of the weight to the input states of a transition (in a tree automaton a transition can have any number of input states whereas in a string automaton it has exactly one) during pushing. It remains open whether there is an efficient heuristic that prescribes how to distribute the weight such that we obtain a minimal dwta recognizing the given series after the unweighted minimization. In fact, our minimization procedure works by first running unweighted minimization (disregarding the weights) and then refining further where transition weights do not match. Roughly speaking, we follow the program of [24, 25] in reversed order.

Our minimization construction uses partition refinement as in the unweighted case [26]. We first define the Myhill-Nerode relation on states of the input dwta. This definition, as well as the Myhill-Nerode relation on tree series [16], will include a scaling factor and Algorithm 3 will determine those scaling factors. In the refinement process (see Definition 14) we check for the congruence property (as in the unweighted case) and the consistency of the weight placement on the transitions. Overall, our algorithm runs in time $O(lmn)$ where l is the maximal rank of the input symbols, m is the number of transitions, and n is the number of states of the input dwta. We thus improve the algorithm proposed in [15], which is reported to run in time $O(lmn^4)$.

Apart from this Introduction, the paper comprises 4 sections. In Sect. 2 we recall standard notions and notation, and the main computational model: the deterministic weighted tree automaton. We then present the theoretical foundations of minimization in Sect. 3. The minimization algorithm is presented in Sect. 4. The final section presents a short example and some experimental results.

2. Preliminaries

The set of nonnegative integers is $\mathbb{N}$. Given $l, u \in \mathbb{N}$ we denote $\{i \in \mathbb{N} \mid l \leq i \leq u\}$ simply by $[l, u]$. Let $n \in \mathbb{N}$ and Q a set. We write Q^n for the n-fold Cartesian product of Q. The empty tuple $() \in Q^0$ is sometimes displayed as ε. We reserve the use of a special symbol $\square \notin Q$. The set of n-ary contexts over Q, denoted by $C_n(Q)$, is $\bigcup_{i+j+1=n} Q^i \times \{\square\} \times Q^j$. Given $C \in C_n(Q)$ and $q \in Q$ we write $C[q]$ to denote the tuple of Q^n obtained from C by replacing $\square$ by q.

An equivalence relation $\equiv$ on Q is a reflexive, symmetric, and transitive subset of Q^2. Let $\equiv$ and $\cong$ be equivalence relations on Q. Then $\equiv$ refines $\cong$ if $\equiv \subseteq \cong$. The equivalence class (or block) of $q \in Q$ is $[q]_\equiv = \{p \in Q \mid p \equiv q\}$. Whenever $\equiv$ is obvious from the context, we simply omit it. The system $(Q/\equiv) = \{[q] \mid q \in Q\}$ actually forms a partition of Q; i.e., a system Π of subsets (also called blocks) of Q such that $\bigcup_{P \in \Pi} P = Q$ and $P \cap P' = \emptyset$ for every $P, P' \in \Pi$ with $P \neq P'$. The number of blocks of $(Q/\equiv)$ is denoted by $\mathrm{index}(\equiv)$. Let Π be any partition on Q and $F \subseteq Q$. The equivalence relation $\equiv_\Pi$ on Q is defined for every $p, q \in Q$ by $p \equiv_\Pi q$ if and only if $\{p, q\} \subseteq P$ for some block $P \in \Pi$. We say that Π saturates F if $\equiv_\Pi$ is a refinement of $\equiv_{\{F, Q \setminus F\}}$; i.e., $\bigcup_{P \in \Pi'} P = F$ for some $\Pi' \subseteq \Pi$. In the sequel, we do not distinguish between a partition Π and the equivalence relation $\equiv_\Pi$.

An alphabet is a finite and nonempty set of symbols. A ranked alphabet (Σ, rk) is an alphabet Σ and a mapping $\mathrm{rk} \colon \Sigma \to \mathbb{N}$. Whenever rk is clear from the context, we simply drop it. The subset of n-ary symbols of Σ is $\Sigma_n = \{\sigma \in \Sigma \mid \mathrm{rk}(\sigma) = n\}$. The set $T_\Sigma(Q)$ of Σ-trees indexed by Q is inductively defined to be the smallest set such that $Q \subseteq T_\Sigma(Q)$ and $\sigma(t_1, \ldots, t_n) \in T_\Sigma(Q)$ for every $\sigma \in \Sigma_n$ and $t_1, \ldots, t_n \in T_\Sigma(Q)$. We write T_Σ for $T_\Sigma(\emptyset)$. The mapping $\mathrm{var} \colon T_\Sigma(Q) \to \mathcal{P}(Q)$, where $\mathcal{P}(Q)$ is the power set of Q, is inductively defined by $\mathrm{var}(q) = \{q\}$ for every $q \in Q$ and $\mathrm{var}(\sigma(t_1, \ldots, t_n)) = \bigcup_{i=1}^n \mathrm{var}(t_i)$ for every $\sigma \in \Sigma_n$ and $t_1, \ldots, t_n \in T_\Sigma(Q)$. For every $P \subseteq Q$, we use $\mathrm{var}_P(t)$ as a shorthand for $\mathrm{var}(t) \cap P$. Moreover, we use $|t|_q$ to denote the number of occurrences of $q \in Q$ in $t \in T_\Sigma(Q)$. We define the height and size of a tree with the help of the mappings $\mathrm{ht}, \mathrm{size} \colon T_\Sigma(Q) \to \mathbb{N}$ inductively for every $q \in Q$ by $\mathrm{ht}(q) = \mathrm{size}(q) = 1$ and $\mathrm{ht}(\sigma(t_1, \ldots, t_n)) = 1 + \max\{\mathrm{ht}(t_i) \mid i \in [1, n]\}$ and $\mathrm{size}(\sigma(t_1, \ldots, t_n)) = 1 + \sum_{i=1}^n \mathrm{size}(t_i)$ for every $\sigma \in \Sigma_n$ and $t_1, \ldots, t_n \in T_\Sigma(Q)$. Note that $\max \emptyset = 0$. The set $\mathrm{pos}(t) \subseteq \mathbb{N}^*$ of positions in t is inductively defined by $\mathrm{pos}(t) = \{\varepsilon\}$ for every $t \in Q$ and

$$\mathrm{pos}(\sigma(t_1, \ldots, t_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{iw \mid w \in \mathrm{pos}(t_i)\}$$

for every $\sigma \in \Sigma_n$ and $t_1, \ldots, t_n \in T_\Sigma(Q)$. For every $w \in \mathrm{pos}(t)$ we denote the label of t at w by $t(w)$ and the subtree at w by $t|_w$. Finally, $\mathrm{pos}_\Gamma(t) = \{w \in \mathrm{pos}(t) \mid t(w) \in \Gamma\}$ for every $\Gamma \subseteq \Sigma \cup Q$.

The set $C_\Sigma(Q)$ of Σ-contexts indexed by Q is defined as the smallest set such that $\square \in C_\Sigma(Q)$ and $\sigma(t_1, \ldots, t_{i-1}, C, t_{i+1}, \ldots, t_n) \in C_\Sigma(Q)$ for every $\sigma \in \Sigma_n$ with $n \geq 1$, index $i \in [1, n]$, $t_1, \ldots, t_n \in T_\Sigma(Q)$, and $C \in C_\Sigma(Q)$. We write C_Σ for $C_\Sigma(\emptyset)$. Note that $C_\Sigma(Q) \subseteq T_\Sigma(Q \cup \{\square\})$. Next we recall substitution. Let V be an alphabet (possibly containing $\square$), $v_1, \ldots, v_n \in V$ be pairwise distinct, and $t_1, \ldots, t_n \in T_\Sigma(V)$. Then we denote by $t[v_i \leftarrow t_i \mid 1 \leq i \leq n]$ the tree obtained

from t by replacing every occurrence of v_i by t_i for every $i \in [1, n]$. We abbreviate $C[\square \leftarrow t]$ simply by $C[t]$ for every $C \in C_\Sigma(Q)$ and $t \in T_\Sigma(V)$.

A (commutative) semiring is a tuple $\mathcal{A} = (A, +, \cdot, 0, 1)$ such that $(A, +, 0)$ and $(A, \cdot, 1)$ are commutative monoids; $a \cdot 0 = 0 = 0 \cdot a$ for every $a \in A$; and $\cdot$ distributes over $+$ from both sides. The semiring $\mathcal{A}$ is a semifield if for every $a \in A \setminus \{0\}$ there exists $a^{-1} \in A$ such that $a \cdot a^{-1} = 1$. A tree series is a mapping $\varphi \colon T \to A$ where $T \subseteq T_\Sigma(Q)$. The set of all such tree series is denoted by $A\langle\langle T \rangle\rangle$. For every $\varphi \in A\langle\langle T \rangle\rangle$ and $t \in T$, the coefficient $\varphi(t)$ is usually denoted by (φ, t).

Next, let us recall Σ-algebras and congruences. A Σ-algebra (S, f) consists of a carrier set S and $f = (f_\sigma)_{\sigma \in \Sigma}$ such that $f_\sigma \colon S^n \to S$ for every $\sigma \in \Sigma_n$. The term Σ-algebra is given by $(T_\Sigma, \overline{\Sigma})$ where $\overline{\Sigma} = (\overline{\sigma})_{\sigma \in \Sigma}$ with $\overline{\sigma}(t_1, \ldots, t_n) = \sigma(t_1, \ldots, t_n)$ for every $\sigma \in \Sigma_n$ and $t_1, \ldots, t_n \in T_\Sigma$. In the sequel, we will drop the overlining. Let $\equiv$ be an equivalence relation on S. Then $\equiv$ is a congruence of (S, f) if for every $\sigma \in \Sigma_n$ and $s_1, \ldots, s_n, t_1, \ldots, t_n \in S$ such that $s_i \equiv t_i$ for every $i \in [1, n]$ we also have $f_\sigma(s_1, \ldots, s_n) \equiv f_\sigma(t_1, \ldots, t_n)$.

A weighted tree automaton [1, 2, 3, 4] (for short: wta) is a tuple $M = (Q, \Sigma, \mathcal{A}, \mu, \nu)$ such that (i) Q is an alphabet of states; (ii) Σ is a ranked alphabet; (iii) $\mathcal{A} = (A, +, \cdot, 0, 1)$ is a (commutative) semiring; (iv) $\mu = (\mu_n)_{n \geq 0}$ with $\mu_n \colon \Sigma_n \to A^{Q^n \times Q}$; and (v) $\nu \in A^Q$ is a final weight vector. Then $(A^Q, (\mu_\sigma)_{\sigma \in \Sigma})$ becomes a Σ-algebra where

$$\mu_\sigma(v_1, \ldots, v_k)_q = \sum_{q_1, \ldots, q_n \in Q} \mu_n(\sigma)_{(q_1, \ldots, q_n), q} \cdot \prod_{i=1}^{n} (v_i)_{q_i}$$

for every $\sigma \in \Sigma_n$, $q \in Q$, and $v_1, \ldots, v_k \in A^Q$. The semantics of M is the tree series $\varphi_M \in A\langle\langle T_\Sigma \rangle\rangle$ given by $(\varphi_M, t) = \sum_{q \in Q} h_\mu(t)_q \cdot \nu_q$ (or simply the scalar product $h_\mu(t) \cdot \nu$) where $h_\mu \colon T_\Sigma \to A^Q$ is the initial homomorphism [from (T_Σ, Σ) to $(A^Q, (\mu_\sigma)_{\sigma \in \Sigma})$]. The wta M is said to recognize φ_M and two wta are equivalent if they recognize the same tree series.

The wta M is deterministic and total [4] if for every $\sigma \in \Sigma_n$ and $w \in Q^n$ there exists exactly one $q \in Q$ such that $\mu_n(\sigma)_{w,q} \neq 0$. In a deterministic wta the additive operation of the semiring is irrelevant [4]. We will exclusively deal with deterministic and total wta (dwta) over semifields from now on. We could have equivalently defined dwta over a (commutative) group, but we chose to follow the presentation usually found in the literature. To simplify the notation, we will use the following representation: $M = (Q, \Sigma, \mathcal{A}, \delta, c, \nu)$ with $\delta = (\delta_\sigma)_{\sigma \in \Sigma}$ and $c = (c_\sigma)_{\sigma \in \Sigma}$ such that (Q, δ) is a Σ-algebra and $c_\sigma \colon Q^n \to A \setminus \{0\}$ for every $\sigma \in \Sigma_n$. In particular, $\delta_\sigma(w) = q$ and $c_\sigma(w) = \mu_n(\sigma)_{w,q}$ if and only if $\mu_n(\sigma)_{w,q} \neq 0$ for every $\sigma \in \Sigma_n$, $q \in Q$, and $w \in Q^n$. The initial homomorphism from $(T_\Sigma(Q), \Sigma)$ to (Q, δ) that extends the identity on Q is also denoted by δ. A state $q \in Q$ is useful if there exists $t \in T_\Sigma$ such that $\delta(t) = q$. A state that is not useful is called useless. The dwta M is said to have no useless states if all states of Q are useful. Note that any dwta can be converted in linear time (in the number of transitions) into an equivalent dwta without useless states. Finally, a state $q \in Q$ is live if $\nu_{\delta(C[q])} \neq 0$ for some context $C \in C_\Sigma(Q)$. Such a context is called a sign of life of q. If no sign of life of q exists, then q is dead.

Due to the semifield restriction, c can similarly be extended to $c \colon T_\Sigma(Q) \to A \setminus \{0\}$ by $c(q) = 1$ for every $q \in Q$ and $c(\sigma(t_1, \ldots, t_n)) = c_\sigma(\delta(t_1), \ldots, \delta(t_n)) \cdot \prod_{i=1}^{n} c(t_i)$ for every $\sigma \in \Sigma_n$ and $t_1, \ldots, t_n \in T_\Sigma(Q)$. It is then easy to show that $(\varphi_M, t) = c(t) \cdot \nu_{\delta(t)}$ for every $t \in T_\Sigma$. In fact, we extend φ_M to a tree series of $A\langle\langle T_\Sigma(Q) \rangle\rangle$ by defining $(\varphi_M, t) = c(t) \cdot \nu_{\delta(t)}$ for every $t \in T_\Sigma(Q)$. The following property, which will be used without explicit mention in the sequel, follows immediately.

Notation	Explanation
$\mathcal{A} = (A, +, \cdot, 0, 1)$	Commutative semifield
$M = (Q, \Sigma, \mathcal{A}, \delta, c, \nu)$	Input dwta without useless states
$F \subseteq Q$	Final states of M
$D \subseteq Q$	Dead states of M
$\varphi = \varphi_M$	Tree series recognized by M
$\equiv_\varphi \subseteq T_\Sigma \times T_\Sigma$	Myhill-Nerode relation on trees
$\equiv \subseteq Q \times Q$	Myhill-Nerode relation on states
$\cong \subseteq Q \times Q$	Coarsest congruence on (Q, δ) that saturates F
$a_{p,q} \in A \setminus \{0\}$	Scaling factor relating the states p and q
$r\colon \mathcal{P}(Q) \to Q$	Representative mapping
$\mathrm{sol}(q)$	Sign of life of state $q \in Q \setminus D$
l	Maximal rank of an input symbol of Σ
m	Number of transitions of M
n	Number of states of M

Table 1: Notation employed.

Lemma 1 (cf. [16, Theorem 1]). *We have $(\varphi_M, t) = 0$ if and only if $\nu_{\delta(t)} = 0$ for every $t \in T_\Sigma(Q)$. Moreover, $c(t[q_i \leftarrow t_i \mid 1 \le i \le n]) = c(t) \cdot \prod_{i=1}^{n} c(t_i)^{|t|_{q_i}}$ for all pairwise distinct $q_1, \ldots, q_n \in Q$ and $t_1, \ldots, t_n \in T_\Sigma(Q)$ such that $\delta(t_i) = q_i$ for every $i \in [1, n]$.*

3. The Myhill-Nerode relation

In this section, we recall the theoretical foundations for the minimization of dwta. It is heavily inspired by [18, 16] and can be skipped on first reading. We present the definition of the Myhill-Nerode relation on states of a dwta, the principal results showing that this relation allows the construction of a minimal equivalent dwta, and simple supportive algorithms to compute, for example, the set of dead states.

We first recall the theoretical foundations and prove the main properties. Second, we present the foundational algorithms and analyze their complexity. Table 1 shows the notation we use throughout the rest of the paper. Note that some notions mentioned in Table 1 will be introduced (or exactly defined) only later in the paper.

3.1. Theoretical foundation

Let us first recall the Myhill-Nerode congruence relation [16, page 153] for a tree series. This section mostly follows the development of [16]. From now on, let $\mathcal{A} = (A, +, \cdot, 0, 1)$ be a commutative semifield and $M = (Q, \Sigma, \mathcal{A}, \delta, c, \nu)$ be a dwta without useless states. Finally, let $F = \{q \in Q \mid \nu_q \neq 0\}$ be the set of final states, D be the set of dead states, and $\varphi = \varphi_M$.

The index of the Myhill-Nerode congruence relation for a tree series determines whether it is deterministically recognizable or not [16, Theorem 2]. More precisely, the index is finite if and only if the tree series is deterministically recognizable. Moreover, it permits the construction of a minimal dwta that recognizes the tree series. For a thorough introduction of this relation, we refer the reader to [16, 13].

Definition 2 (see [16, page 153]). The Myhill-Nerode relation $\equiv_\varphi \subseteq T_\Sigma \times T_\Sigma$ is defined for every $t, u \in T_\Sigma$ by $t \equiv_\varphi u$ if and only if there exists an element $a \in A \setminus \{0\}$ such that $(\varphi, C[t]) = a \cdot (\varphi, C[u])$ for every $C \in C_\Sigma$.

Clearly, φ is deterministically recognizable because we are given a dwta M that recognizes φ. In order to use that additional knowledge, we define the Myhill-Nerode relation on Q and then show that it shares many of the properties of the Myhill-Nerode relation $\equiv_\varphi$.

Definition 3 (Myhill-Nerode relation). The Myhill-Nerode relation $\equiv \subseteq Q \times Q$ is defined for every $p, q \in Q$ by $p \equiv q$ if and only if there exists an element $a \in A \setminus \{0\}$ such that $(\varphi, C[p]) = a \cdot (\varphi, C[q])$ for every $C \in C_\Sigma(Q)$. We denote such an element by $a_{p,q}$.

In comparison to Definition 2, we demand the property also for contexts of $C_\Sigma(Q)$ instead of just C_Σ. It is shown in [15] that the relations defined by these two versions of Definition 3 actually coincide. The main benefits of the version in Definition 3 are that (i) it avoids an exponential blow-up (the one also mentioned in [27]) and (ii) it is consistent with our definition of signs of life.

Let us investigate the Myhill-Nerode relation in more detail. We immediately note that (i) all dead states are equivalent, (ii) no dead state is equivalent to a live one, and (iii) the scaling factor $a_{p,q}$ is uniquely determined whenever p is live. Let us call the last statement (†) for later reference. Next we show that $\equiv$ is also a congruence.

Lemma 4 (cf. [16, Lemma 5]). *The relation $\equiv$ is a congruence on (Q, δ).*

Proof. Let us prove that $\equiv$ is an equivalence relation. Reflexivity and symmetry are trivial. For transitivity, let $p, q, r \in Q$ be such that $p \equiv q$ and $q \equiv r$. Then $a_{p,r} = a_{p,q} \cdot a_{q,r} \neq 0$ provides evidence that $p \equiv r$. Note that for live p the above equality provides a way to compute the unique scaling factor. We use this fact in Sect. 4.

Now, let us proceed with the congruence property. Let $\sigma \in \Sigma_k$ and $p_1, \ldots, p_k, q_1, \ldots, q_k \in Q$ be such that $p_i \equiv q_i$ for every $i \in [1, k]$. For every $C \in C_\Sigma(Q)$

$$(\varphi, C[\delta_\sigma(p_1, \ldots, p_k)]) \cdot c_\sigma(p_1, \ldots, p_k) = \prod_{i=1}^{k} a_{p_i,q_i} \cdot (\varphi, C[\delta_\sigma(q_1, \ldots, q_k)]) \cdot c_\sigma(q_1, \ldots, q_k) \quad , \quad (1)$$

which is shown as follows (recall that we use Lemma 1 without explicit mention):

$$\begin{aligned}
&(\varphi, C[\delta_\sigma(p_1, \ldots, p_k)]) \cdot c_\sigma(p_1, \ldots, p_k) = (\varphi, C[\sigma(\Box, p_2, \ldots, p_k)]\,[p_1]) \\
&= a_{p_1,q_1} \cdot (\varphi, C[\sigma(\Box, p_2, \ldots, p_k)]\,[q_1]) = a_{p_1,q_1} \cdot (\varphi, C[\sigma(q_1, \Box, p_3, \ldots, p_k)]\,[p_2]) \\
&= \cdots \\
&= \prod_{i=1}^{k} a_{p_i,q_i} \cdot (\varphi, C[\sigma(q_1, \ldots, q_k)]) = \prod_{i=1}^{k} a_{p_i,q_i} \cdot (\varphi, C[\delta_\sigma(q_1, \ldots, q_k)]) \cdot c_\sigma(q_1, \ldots, q_k) \quad .
\end{aligned}$$

Hence $\delta_\sigma(p_1, \ldots, p_k) \equiv \delta_\sigma(q_1, \ldots, q_k)$. Consequently, $\equiv$ is a congruence on (Q, δ). □

Now let us show that we can use $\equiv$ to minimize M. We first present how to construct the quotient dwta $M/\equiv$ and then prove that the dwta obtained in this way still recognizes φ. Finally, we verify that $M/\equiv$ is minimal. Let us begin with the construction of $M/\equiv$. To avoid well-definedness issues, we fix a representative mapping $r\colon \mathcal{P}(Q) \to Q$, i.e., a mapping such that

$r(P) \in P$ for every nonempty $P \subseteq Q$. For example, if $Q \subseteq \mathbb{N}$ then $r(P) = \min P$ is a representative mapping. Formally, our construction depends not only on M and $\equiv$, but also on r.

Roughly speaking, we collapse equivalent states in M to just the representative of their block. Then we compute solely on those representatives. Note that this approach is slightly different from the one of [15]. We chose this style of presentation because it is immediately clear that $M/\equiv$ is well-defined (cf. [16]) in contrast to the unweighted case [6]. On the other hand, our construction is dependent of the selection of r, but this allows us to improve the efficiency of the minimization algorithm.

Definition 5 (cf. [16, Definition 4]). Let $M/\equiv$ be the dwta $(Q/\equiv, \Sigma, \mathcal{A}, \delta', c', \nu')$ such that for every $\sigma \in \Sigma_k$, $B_1, \ldots, B_k \in (Q/\equiv)$, and $q = \delta_\sigma(r(B_1), \ldots, r(B_k))$

- $\delta'_\sigma(B_1, \ldots, B_k) = [q]$,
- $c'_\sigma(B_1, \ldots, B_k) = c_\sigma(r(B_1), \ldots, r(B_k)) \cdot a_{q,r([q])}$, and
- $\nu'_B = \nu_{r(B)}$ for every $B \in (Q/\equiv)$.

Note that the state q is not necessarily the representative of its block, so we multiply with the correct scaling factor in the second item. Moreover, it is obvious that $M/\equiv$ has index($\equiv$) states. Let us proceed by showing that $M/\equiv$ recognizes φ.

Lemma 6 (cf. [16, Lemma 12]). *The dwta $M/\equiv$ recognizes φ.*

Proof. Let $(M/\equiv) = (Q/\equiv, \Sigma, \mathcal{A}, \delta', c', \nu')$. First we prove that $\delta'(t) = [\delta(t)]$ for every $t \in T_\Sigma$. Let us call this statement (‡). Let $t = \sigma(t_1, \ldots, t_k)$ for some $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$. Moreover, let $q_i = \delta(t_i)$ for every $i \in [1, k]$. Then with the help of the induction hypothesis we conclude

$$\delta'(t) = \delta'_\sigma(\delta'(t_1), \ldots, \delta'(t_k)) \overset{\text{I.H.}}{=} \delta'_\sigma([q_1], \ldots, [q_k]) = [\delta_\sigma(r([q_1]), \ldots, r([q_k]))]$$

and $[\delta_\sigma(q_1, \ldots, q_k)] = \delta'(t)$ since $\equiv$ is a congruence on (Q, δ) by Lemma 4. Thus, $\delta'(t) = [\delta(t)]$.

Next we need to relate $c_\sigma(q_1, \ldots, q_k)$ to $c_\sigma(r([q_1]), \ldots, r([q_k]))$ if $q = \delta_\sigma(q_1, \ldots, q_k)$ is live. To this end, let $p = \delta_\sigma(r([q_1]), \ldots, r([q_k]))$. Reconsidering (1) [see proof of Lemma 4] and remark (†) on page 6, which shows that $a_{q,p}$ is uniquely determined whenever q is live, we obtain

$$a_{q,p} \cdot c_\sigma(q_1, \ldots, q_k) = \prod_{i=1}^{k} a_{q_i,r([q_i])} \cdot c_\sigma(r([q_1]), \ldots, r([q_k])) \quad . \tag{2}$$

Now, we are ready to prove (§) $c'(t) = c(t) \cdot a_{q,r([q])}$ if q is live. By the induction hypothesis, which is applicable because $q_1, \ldots, q_k$ are clearly live if q is live, we obtain

$$\begin{aligned} c'(t) &\overset{(\ddagger)}{=} c'_\sigma([q_1], \ldots, [q_k]) \cdot \prod_{i=1}^{k} c'(t_i) \overset{\text{I.H.}}{=} c'_\sigma([q_1], \ldots, [q_k]) \cdot \prod_{i=1}^{k} \Big(c(t_i) \cdot a_{q_i,r([q_i])}\Big) \\ &= \prod_{i=1}^{k} a_{q_i,r([q_i])} \cdot c_\sigma(r([q_1]), \ldots, r([q_k])) \cdot a_{p,r([p])} \cdot \prod_{i=1}^{k} c(t_i) \\ &\overset{(2)}{=} a_{q,p} \cdot c_\sigma(q_1, \ldots, q_k) \cdot a_{p,r([p])} \cdot \prod_{i=1}^{k} c(t_i) = c(t) \cdot a_{q,r([q])} \end{aligned}$$

because $[p] = [q]$ by Lemma 4, which completes the induction.

It remains to prove that $(\varphi_{(M/\equiv)}, t) = (\varphi, t)$. We distinguish two cases. First, suppose that q is not final. Then clearly, $(\varphi, t) = \nu_q = 0$. By (‡) we have $\delta'(t) = [q]$. In addition, $\nu'_{[q]} = \nu_{r([q])}$ by Definition 5. Since $q \equiv r([q])$ we have $\nu_{r([q])} = 0$ and hence $(\varphi_{(M/\equiv)}, t) = \nu'_{[q]} = 0$. Second, suppose that q is final and thus live. Then

$$(\varphi_{(M/\equiv)}, t) = c'(t) \cdot \nu'_{[q]} \overset{(\S)}{=} c(t) \cdot a_{q,r([q])} \cdot \nu_{r([q])} = c(t) \cdot \nu_q = (\varphi, t)$$

by Definition 3 (for $q \equiv r([q])$ and $C = \square$). Thus, $M/\equiv$ recognizes φ. □

We have seen that $\equiv$ allows us to construct a dwta recognizing φ. We still need to show that the dwta obtained in this way is minimal. To this end, we prove that the indices of $\equiv$ and $\equiv_\varphi$ coincide. The latter coincides with the number of states of a minimal dwta recognizing φ by [16, Theorem 2]. Since $M/\equiv$ has index($\equiv$) states, this proves the minimality.

Lemma 7 (Minimality). index($\equiv$) = index($\equiv_\varphi$).

Proof. Let $t, u \in T_\Sigma$ be such that $t \equiv_\varphi u$. Then there exists an element $a \in A \setminus \{0\}$ such that $(\varphi, C[t]) = a \cdot (\varphi, C[u])$ for every $C \in C_\Sigma$. We reason as follows:

$$c(t) \cdot (\varphi, C[\delta(t)]) = (\varphi, C[t]) = a \cdot (\varphi, C[u]) = a \cdot c(u) \cdot (\varphi, C[\delta(u)]) \ .$$

Since $a \cdot c(t)^{-1} \cdot c(u)$ does not depend on C, we obtain $\delta(t) \equiv \delta(u)$. Since M has no useless states, $\delta\colon T_\Sigma \to Q$ is surjective and thus index($\equiv$) $\leq$ index($\equiv_\varphi$). We already proved that $M/\equiv$ recognizes φ in Lemma 6. By [16, Theorem 2] every dwta that recognizes φ has at least index($\equiv_\varphi$) states. Consequently, index($\equiv$) $\geq$ index($\equiv_\varphi$), which proves the statement. □

Theorem 8. *The dwta $M/\equiv$ is a minimal dwta recognizing φ.*

Proof. The theorem follows from Lemma 6, Lemma 7, and [16, Theorem 2], which shows that index($\equiv_\varphi$) coincides with the number of states of a minimal dwta recognizing φ. □

Finally, we recall another important congruence on (Q, δ), which is used to minimize in the unweighted case. It is the coarsest congruence $\cong$ on (Q, δ) that saturates F. Its use for minimization of unweighted tree automata is shown in [6, Theorem II.6.10] (roughly speaking, it plays the role of our $\equiv$ in the unweighted setting). Moreover, it is known [28, Theorem 27] that it can be computed in $O(lm \log n)$ where l is the maximal rank of an input symbol, m is the number of transitions, and n is the number of states of the input automaton. The following lemma shows that equivalent states share their signs of life in any equivalence relation that refines $\cong$.

Lemma 9. *Let $\sim$ be an equivalence on Q such that $\sim$ refines $\cong$. Then every sign of life of p is also a sign of life of q for every $p \sim q$.*

Proof. Let $p \sim q$ and $C \in C_\Sigma(Q)$ be a sign of life of p. Clearly, $p \cong q$ since $\sim$ refines $\cong$. Since $\cong$ is a congruence on (Q, δ) we conclude $\delta(C[p]) \cong \delta(C[q])$. We observe that $\delta(C[p]) \in F$ because C is a sign of life of p. This yields $\delta(C[q]) \in F$ since $\cong$ saturates F. Consequently, C is a sign of life of q. □

3.2. Algorithm

Let us cover some fundamental algorithms here. First, we show a standard reachability algorithm that computes the set of live states (and thus also the dead states). Along the way we also compute a sign of life for every live state. We present complexity results for this algorithm and for the construction of Definition 5, which defines the quotient dwta. For the complexity analysis, we make the following assumptions:

- All operations of the semifield (we only need multiplication and calculation of inverses) can be performed in constant time. This might be an unrealistic assumption, but it simplifies the complexity analysis and is typically true for the fixed-precision arithmetic implemented on stock hardware.
- We assume that the input is encoded properly (e.g., constant space is needed to store a state, an input symbol, etc.).

Let us also present exact definitions for the maximal rank l of an input symbol, the number m of transitions of M that lead into a live state, and the number n of states of M.

$$l = \max\{k \mid \Sigma_k \neq \emptyset\} \qquad m = \operatorname{card}\{(\sigma, q_1, \ldots, q_k) \mid \delta_\sigma(q_1, \ldots, q_k) \notin D\} \qquad n = \operatorname{card}(Q)$$

Note that $m \geq n$ since each state is useful. We present the algorithms detailed enough to easily verify the time complexity claims. For versions of the algorithms without implementation details we refer the reader to [15]. Simple set operations and maps are assumed to be implemented in an efficient manner.

Let us begin with the algorithm that computes the signs of life (see Algorithm 1). It implements a simple reachability algorithm. We start with the final states, for which we store the sign of life $\Box$. We then explore co-reachable states by checking which states could lead to a currently explored state. For each state that we explore, we store a sign of life as evidence that the state is live. Moreover, we also group the explored states by the height of their stored signs of life. Our implementation is such that the signs of life are of minimal height, so that this grouping can be used to derive an initial partition for our minimization problem. Strictly speaking, this part would not be necessary as the simple partition $\{F, Q \setminus F\}$ would be sufficient for our purposes (also for the complexity results), but practical experiments have shown that it can lead to sufficiently large improvements in the run-time of the minimization algorithm, so we chose to present it.

Algorithm 1 returns an initial partition Π, a mapping sol that assigns signs of life to each live state, and the set D of dead states. Some remarks are necessary. First, it is essential that T (in Algorithm 1) is handled as a FIFO queue, where we append at the end and remove from the beginning. This guarantees that the height of the constructed signs of life is minimal. Second, the signs of life are encoded using a linked list. Nil denotes the empty list and AddFront adds its first parameter in front of all elements of the list in its second parameter (and returns a pointer to the resulting list). The elements in the linked list are triples (w, σ, i) where $w \in Q^k$, $\sigma \in \Sigma_k$, and $i \in [1, k]$. Roughly speaking, an element is a transition (w, σ) and a subtree marker i. The sign of life is stored as a list of transitions in reversed order. Since dwta work bottom-up this representation is both space-efficient and easy to process by the dwta. Let us show how to recover the sign of life. The empty list translates into the trivial context $\Box$. Suppose that the representation consists of (w, σ, i) and a rest list, which yields the context C. Then the context represented by the full list is obtained as $C[\sigma(w_1, \ldots, w_{i-1}, \Box, w_{i+1}, \ldots, w_k)]$.

For the run-time complexity consider the graph consisting of the states as nodes and for every transition $q = \delta_\sigma(q_1, \ldots, q_k)$ there is a edge (q, q_i) for every $i \in [1, k]$. Then a breadth-first search

Algorithm 1 ComputeSoL: Compute signs of life and initial partition.

```
    for all q ∈ F do
2:      sol(q) ← Nil                                   // final states have sign of life □
    D ← Q \ F                          // set of potentially dead states; final states are not dead
4:  P ← F                                              // current block; initially the final states
    h ← 0                                  // current height of signs of life; initially 0
6:  Π ← ∅                                              // initial partition; initially empty
    T ← new FifoQueue                                  // initialize empty FIFO queue T of transitions
8:  Append(T, {(w, σ, δ_σ(w), 1) | δ_σ(w) ∈ F})        // append to T the transitions leading to F
    while T is not empty do
10:     (w, σ, q, j) ← RemoveHead(T)                   // get first element in T
        if h ≠ j then
12:         Π ← Π ∪ {P}                                // add current block to initial partition
            P ← ∅                                      // reset current block
14:         h ← j                                      // increase current height of signs of life
        for all 1 ≤ i ≤ |w| do
16:         if w_i ∈ D then
                sol(w_i) ← AddFront((w, σ, i), sol(q))  // we found a sign of life of w_i
18:             P ← P ∪ {w_i}                          // add w_i to current block
                D ← D \ {w_i}                          // w_i is explored and not dead
20:             Append(T, {(u, γ, w_i, j + 1) | δ_γ(u) = w_i})   // append transitions leading to w_i
    if D ≠ ∅ then
22:     Π ← Π ∪ {D}                                    // finally add block of dead states
    return (Π, sol, D)
```

algorithm finds the states reachable from the final states. Since $m \leq n$, its time complexity is $O(lm)$ because there are at most lm edges. Fortunately, the computation of the signs of life on the side does not increase the asymptotical time complexity.

Lemma 10. *Algorithm 1 runs in time $O(lm)$.*

Proof. Clearly, lines 1–7 run in time $O(m)$. Supposing an efficient representation of δ, line 8 also runs in time $O(m)$. The while-loop in lines 9–20 can be executed at most m times. Lines 10–14 execute in $O(1)$ time. The loop in lines 15–20 executes at most l times (for each iteration of the surrounding while-loop) and each such iteration except for line 20 takes at $O(1)$ time. Finally, let us consider line 20 globally. Obviously, each transition can be added at most once to T, hence line 20 runs in overall time $O(m)$. Putting the pieces together, we obtain that Algorithm 1 runs in time $O(lm)$. □

Theorem 11. *Let (Π, sol, D) be the result of running Algorithm 1. Then*

(i) *$\equiv$ refines Π and $F \in \Pi$,*
(ii) *D is the set of all dead states.*
(iii) *$\mathrm{sol}(q) = \square$ for every $q \in F$, and*
(iv) *$\mathrm{sol}(q)$ represents a sign of life of q comprising at most n transitions for every $q \in Q \setminus D$.*

Proof. The facts that $F \in \Pi$ and $\mathrm{sol}(q) = \square$ for every $q \in F$ are obvious. Note that in any element (w, σ, q, j) of T, the integer j is one larger than the length of $\mathrm{sol}(q)$. Thus, it is clear that all elements of P have equally long lists representing their signs of life. Moreover, all states with signs of life represented by lists of a certain length will eventually be collected in P (in Algorithm 1). From this and the minimality of the signs of life, we can conclude that Π groups states by their minimal number of transitions in a sign of life. Finally, suppose that $p \equiv q$ and let $C \in C_\Sigma(Q)$ be a sign of life of p. Then C is a sign of life of q by Definition 3, and thus $\equiv$ refines Π.

Concerning (iii), final states trivially have sign of life $\square$ (see line 2 of Algorithm 1). If $\mathrm{sol}(q)$ is a sign of life of q and $\delta_\sigma(w) = q$, then $\mathrm{sol}(q)[\sigma(w_1, \ldots, w_{i-1}, \square, w_{i+1}, \ldots, w_k)]$ is a sign of life of w_i for every $i \in [1, k]$. This proves (iv). Finally, the algorithm trivially explores all co-reachable states and thus D is the set of dead states. $\square$

From now on, we fix sol and D to those returned by Algorithm 1; i.e., D is the set of dead states and sol assigns to each live state a sign of life with the additional properties mentioned in Theorem 11. We end this section with a complexity analysis of the quotient dwta construction of Definition 5. We assume that the representative mapping r is implemented such that $r(P)$ can be computed in constant time for every nonempty $P \subseteq Q$. Moreover, we assume in the next statement that an efficient representation of $\equiv$ is available.

Lemma 12. *Supposing that the scaling factors $a_{q,r([q])}$ are available for all $q \in Q$, the construction of $M/{\equiv}$ can be implemented to run in time $O(lm)$.*

Proof. The proof is straightforward and omitted. $\square$

4. Minimization

4.1. Theoretical foundation

In this section, we develop the theoretical underpinnings for the refinement procedure that we use in the next section for our minimization algorithm. We show that it eventually yields the Myhill-Nerode relation $\equiv$ and thus, by Theorem 8, allows us to construct a minimal dwta that is equivalent to M.

Our approach basically follows the development of [15], but there are some major changes. Let us highlight them: (i) We avoid pairwise comparison in the refinement step and instead compare states only to the representative of their block. This saves roughly a factor n. (ii) We avoid refining the current partition in two places (in [15] it could be refined in the main refinement step and in the computation of the scaling factors). This avoids a stark overestimation of the number of refinement steps, which essentially saves another factor n in the complexity analysis. (iii) To avoid refinements in the computation of the scaling factors, we need to make sure that our partition refines $\cong$. Fortunately, this can easily be achieved with the known algorithm for the unweighted case. Finally, (iv) we slightly relax the conditions placed on the scaling factors, which makes them easier to compute after the main refinement step. This also yields roughly an improvement by a factor n. Overall, we managed to improve the run-time complexity from $O(lmn^4)$ as reported in [15] to just $O(lmn)$. In addition, we do not use the main data structure (called stage) of [15] because two components of it never changed and avoiding it allows us to easily separate the theoretical development from the implementation (at the cost of some additional proof obligations).

Let us start with the data structure used to compute the scaling factors. A scaling map holds the scaling factor for each state to its representative with respect to a partition. Once we refined to $\equiv$, the scaling map stores exactly the scaling factors required for Lemma 12.

Definition 13 (Scaling map). Let Π be a partition of Q and $f\colon Q \to A$ a mapping. We say that f is a *scaling map for* Π if

(i) $\nu_q = f(q) \cdot \nu_p$ where $p = r([q]_{\equiv_\Pi})$ for every $q \in F$,
(ii) $f(q) = 1$ for every $q \in D$, and
(iii) for every $B \in \Pi$ there exists $C \in C_\Sigma(Q)$ such that $(\varphi, C[q]) = f(q) \cdot (\varphi, C[r(B)])$ and C is a sign of life of q for every live $q \in B$.

Note that $f\colon Q \to A \setminus \{0\}$ whenever f is a scaling map. In contrast to [15] we only demand the existence of a sign of life in condition (iii) of Definition 13, whereas in [15] this sign of life was fixed to $\mathrm{sol}(r(B))$.

The principal approach of the minimization algorithm in the next section is partition refinement as, for example, in the classical minimization algorithm for minimizing unweighted deterministic tree automata [26]. We successively refine an initial partition until $\equiv$ is reached. Let us proceed with the definition of a single refinement step.

Definition 14 (Refinement). Let Π be a partition of Q and $f\colon Q \to A$ be a scaling map for Π. The *refinement* $\textsc{Refine}(\Pi, f)$ is the partition

$$\Pi' = \Big(\bigcup_{B \in \Pi} \{\mathrm{block}(B), B \setminus \mathrm{block}(B)\}\Big) \setminus \{\emptyset\}$$

where $\mathrm{block}(B)$ contains all $q \in B$ for which, for every $\sigma \in \Sigma_k$ and $C \in C_k(Q)$ such that $\delta_\sigma(C[q])$ is live, the following two conditions hold.

(i) $\delta_\sigma(C[q]) \equiv_\Pi \delta_\sigma(C[r(B)])$
(ii) $f(q)^{-1} \cdot c_\sigma(C[q]) \cdot f(\delta_\sigma(C[q])) = c_\sigma(C[r(B)]) \cdot f(\delta_\sigma(C[r(B)]))$

In the classical unweighted case, only the congruence property (condition (i) in Definition 14) is used to refine. The additional constraint basically restricts the weights on the transitions whereas the congruence property only restricts the presence/absence of transitions. Compared to [15] we replaced the pairwise tests by conditions that relate a state and the representative of its block. The following lemma shows that Refine refines in the desired manner. In particular, whenever $\equiv$ is a refinement of $\equiv_\Pi$, then $\equiv$ is also a refinement of $\equiv_{\Pi'}$ where $\Pi' = \textsc{Refine}(\Pi, f)$. Thus, if we start with a suitable partition, then we only refine to the level of $\equiv$ and never beyond.

Lemma 15 (Lower bound of refinement). *Let Π be a partition of Q such that $\equiv$ refines Π, which in turn refines $\cong$. Moreover, let $f\colon Q \to A$ be a scaling map for Π and $\Pi' = \textsc{Refine}(\Pi, f)$. Then $\equiv$ refines Π', which in turn refines Π.*

Proof. It is obvious from Definition 14 that Π' refines Π. Let $q_1, q_2 \in Q$ be such that $q_1 \equiv q_2$, let $B \in \Pi$ be such that $q_1, q_2 \in B$, and let $p = r(B)$. Moreover, let $\sigma \in \Sigma_k$ and $C \in C_k(Q)$ be such that $\delta_\sigma(C[q_1])$ is live. Since $\equiv$ refines Π and $\delta_\sigma(C[q_1]) \equiv \delta_\sigma(C[q_2])$ by Lemma 4, we conclude $\delta_\sigma(C[q_1]) \equiv_\Pi \delta_\sigma(C[q_2])$. Consequently, condition (i) in Definition 14 is either true for both q_1 and q_2 or false for both.

Let $q_1' = \delta_\sigma(C[q_1])$ and $q_2' = \delta_\sigma(C[q_2])$. Moreover, let $p' = r([q_1']_{\equiv_\Pi})$ be the representative of the block of q_1' in Π, which is the same as the one of q_2' since $q_1' \equiv_\Pi q_2'$. A standard calculation using (2) on page 7 yields $c_\sigma(C[q_1]) \cdot a_{q_1',q_2'} = c_\sigma(C[q_2]) \cdot a_{q_1,q_2}$. By remark (†) on page 6 every sign of life of q_1 (resp., q_1') determines a_{q_1,q_2} (resp., $a_{q_1',q_2'}$). Since $q_1 \equiv_\Pi p \equiv_\Pi q_2$ and Π refines $\cong$, all Π-equivalent states share their signs of life by Lemma 9. Since f is a scaling map for Π, there exist $C', C'' \in C_\Sigma(Q)$ such that

(i) C' is a sign of life of q_1 and q_2,
(ii) C'' is a sign of life of q_1' and q_2', and
(iii) the following equations hold:

$$(\varphi, C'[q_1]) = f(q_1) \cdot (\varphi, C'[p]) \qquad (\varphi, C''[q_1']) = f(q_1') \cdot (\varphi, C''[p'])$$
$$(\varphi, C'[q_2]) = f(q_2) \cdot (\varphi, C'[p]) \qquad (\varphi, C''[q_2']) = f(q_2') \cdot (\varphi, C''[p']) \ .$$

Hence, C' determines a_{q_1,q_2} and C'' determines $a_{q_1',q_2'}$. In addition, C' determines $a_{q_2,p}$ and C'' determines $a_{q_2',p'}$. We obtain

$$c_\sigma(C[q_1]) \cdot \frac{(\varphi, C''[q_1'])}{(\varphi, C''[q_2'])} = c_\sigma(C[q_2]) \cdot \frac{(\varphi, C'[q_1])}{(\varphi, C'[q_2])} \ .$$

Multiplying both sides by $a = (\varphi, C'[p]) \cdot (\varphi, C''[p'])^{-1}$ and exchanging some terms, we obtain

$$\frac{(\varphi, C'[p])}{(\varphi, C'[q_1])} \cdot c_\sigma(C[q_1]) \cdot \frac{(\varphi, C''[q_1'])}{(\varphi, C''[p'])} = \frac{(\varphi, C'[p])}{(\varphi, C'[q_2])} \cdot c_\sigma(C[q_2]) \cdot \frac{(\varphi, C''[q_2'])}{(\varphi, C''[p'])} \ .$$

Now we can apply the equations of (iii) to obtain

$$f(q_1)^{-1} \cdot c_\sigma(C[q_1]) \cdot f(q_1') = f(q_2)^{-1} \cdot c_\sigma(C[q_2]) \cdot f(q_2') \ ,$$

which shows that also condition (ii) in Definition 14 is either true for both q_1 and q_2 or false for both. Consequently, $q_1 \equiv_{\Pi'} q_2$. □

In the minimization algorithm we iteratively refine a given partition until no further refinement is possible (i.e., $\textsc{Refine}(\Pi, f) = \Pi$). Let us investigate the properties of such a partition. We show that, given a suitable partition with which to start the refinement process, the fixpoint of the refinement procedure (we assume that suitable scaling maps are supplied) refines the Myhill-Nerode relation, which together with Lemma 15 proves that the fixpoint is $\equiv$. In addition, we show that the scaling map then supplies the required scaling factors (see Lemma 12).

Lemma 16 (Upper bound of refinement). *Let Π be a partition of Q that saturates F, and let $f\colon Q \to A$ be a scaling map for Π. If $\Pi = \textsc{Refine}(\Pi, f)$, then*

(i) *Π refines $\equiv$, and*
(ii) *for every live $q \in Q$ we have $f(q) = a_{q,p}$ where $p = r([q]_{\equiv_\Pi})$.*

Proof. If $\Pi = \textsc{Refine}(\Pi, f)$, then $\mathrm{block}(B) = B$ for every $B \in \Pi$ since $\mathrm{block}(B) \neq \emptyset$ for every nonempty $B \subseteq Q$. This immediately yields that Π is a congruence of (Q, δ) that saturates F. Next, we prove that $(\varphi, C[q]) = f(q) \cdot (\varphi, C[p])$ for every live $q \in Q$, $C \in C_\Sigma(Q)$, and $p = r([q]_{\equiv_\Pi})$. Let $t = C[q]$, $u = C[p]$, $q' = \delta(t)$, and $p' = \delta(u)$. Finally, let $v \in \mathrm{pos}_\Box(C)$. Note that the statement

is trivially true if $q' \notin F$ because Π is a congruence that saturates F and hence $p' \notin F$. Let $\mathrm{Pref}(v) = \{w \mid \exists w' \neq \varepsilon\colon v = ww'\}$ be the set of strict prefixes of v. Then we claim that

$$\nu_{q'} \cdot \prod_{w \in \mathrm{Pref}(v)} c_{t(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk})) = \nu_{p'} \cdot f(q) \cdot \prod_{w \in \mathrm{Pref}(v)} c_{u(w)}(\delta(u|_{w1}), \ldots, \delta(u|_{wk})) \quad . \tag{3}$$

Now we will use the run-semantics of dwta (see [13, Definition 4.1.12] for a detailed exposition). In [13, Lemma 4.1.13] it is proved that $(\varphi, C[q]) = \nu_{q'} \cdot \prod_{w \in \mathrm{pos}_\Sigma(t)} c_{t(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk}))$. Thus, we compute

$$\begin{aligned}
&(\varphi, C[q]) = \nu_{q'} \cdot \prod_{w \in \mathrm{pos}_\Sigma(t)} c_{t(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk})) \\
&= \nu_{q'} \cdot \prod_{w \in \mathrm{pos}_\Sigma(t) \setminus \mathrm{Pref}(v)} c_{t(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk})) \cdot \prod_{w \in \mathrm{Pref}(v)} c_{t(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk})) \quad .
\end{aligned}$$

Note that $t(w) = u(w)$ and $t|_{wi} = u|_{wi}$ for every $i \in [1, k]$, $t(w) \in \Sigma_k$, and position $w \in \mathrm{pos}_\Sigma(t)$ such that $w \notin \mathrm{Pref}(v)$. Finally, $\mathrm{pos}_\Sigma(t) = \mathrm{pos}_\Sigma(u)$. Using (3) we thus obtain

$$\begin{aligned}
&\nu_{q'} \cdot \prod_{w \in \mathrm{pos}_\Sigma(t) \setminus \mathrm{Pref}(v)} c_{t(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk})) \cdot \prod_{w \in \mathrm{Pref}(v)} c_{t(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk})) \\
&= \nu_{p'} \cdot f(q) \cdot \prod_{w \in \mathrm{pos}_\Sigma(u) \setminus \mathrm{Pref}(v)} c_{u(w)}(\delta(u|_{w1}), \ldots, \delta(u|_{wk})) \cdot \prod_{w \in \mathrm{Pref}(v)} c_{u(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk})) \\
&= f(q) \cdot (\varphi, C[p]) \quad .
\end{aligned}$$

It remains to prove (3). For every $w \in \mathrm{Pref}(v)$, let $i_w \in \mathbb{N}$ be such that $wi_w \in \mathrm{Pref}(v)$ or $wi_w = v$. Moreover, let $q'' = r([q']_{\equiv_\Pi})$. Since $q' \in F$, we have $\nu_{q'} = f(q') \cdot \nu_{q''}$ and thus

$$\begin{aligned}
&\nu_{q'} \cdot f(q)^{-1} \cdot \prod_{w \in \mathrm{Pref}(v)} c_{t(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk})) \\
&= \nu_{q''} \cdot f(q)^{-1} \cdot \prod_{w \in \mathrm{Pref}(v)} c_{t(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk})) \cdot f(q') \\
&= \nu_{q''} \cdot \prod_{w \in \mathrm{Pref}(v)} f(\delta(t|_{wi_w}))^{-1} \cdot \prod_{w \in \mathrm{Pref}(v)} c_{t(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk})) \cdot \prod_{w \in \mathrm{Pref}(v)} f(\delta(t|_w)) \\
&= \nu_{q''} \cdot \prod_{w \in \mathrm{Pref}(v)} \left(f(\delta(t|_{wi_w}))^{-1} \cdot c_{t(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk})) \cdot f(\delta(t|_w)) \right) \quad .
\end{aligned}$$

Since $q' \equiv_\Pi p'$ and $\delta(t|_{wi_w}) \equiv_\Pi \delta(u|_{wi_w})$ we can use condition (ii) in Definition 14. We continue with

$$\begin{aligned}
&\nu_{q''} \cdot \prod_{w \in \mathrm{Pref}(v)} \left(f(\delta(t|_{wi_w}))^{-1} \cdot c_{t(w)}(\delta(t|_{w1}), \ldots, \delta(t|_{wk})) \cdot f(\delta(t|_w)) \right) \\
&= \nu_{q''} \cdot \prod_{w \in \mathrm{Pref}(v)} \left(f(\delta(u|_{wi_w}))^{-1} \cdot c_{u(w)}(\delta(u|_{w1}), \ldots, \delta(u|_{wk})) \cdot f(\delta(u|_w)) \right) \\
&= \nu_{p'} \cdot f(p)^{-1} \cdot \prod_{w \in \mathrm{Pref}(v)} c_{u(w)}(\delta(u|_{w1}), \ldots, \delta(u|_{wk})) \quad .
\end{aligned}$$

Note that p is a representative and hence $f(p) = 1$. This finally yields

$$\begin{aligned} &\nu_{p'} \cdot f(p)^{-1} \cdot \prod_{w \in \mathrm{Pref}(v)} c_{u(w)}(\delta(u|_{w1}), \ldots, \delta(u|_{wk})) \\ = \; &\nu_{p'} \cdot \prod_{w \in \mathrm{Pref}(v)} c_{u(w)}(\delta(u|_{w1}), \ldots, \delta(u|_{wk})) \quad . \end{aligned}$$

By remark (†) on page 6, the scaling factor $a_{q,p}$ is unique for all live q, and hence $a_{q,p} = f(q)$, which proves (ii). Moreover, $(\varphi, C[p]) = f(p) \cdot f(q)^{-1} \cdot (\varphi, C[q])$ for every $C \in C_\Sigma(Q)$ and $p, q \in Q$ such that $p \equiv_\Pi q$. Thus, $\equiv_\Pi$ is a refinement of $\equiv$, which proves (i). □

This means that, given an initial partition that fulfills the restrictions of Lemmata 15 and 16, the fixpoint is exactly the Myhill-Nerode relation because we proved it to be a lower and upper bound in Lemmata 15 and 16, respectively.

Theorem 17. *Let Π_0 be the partition that induces $\cong$. For every $i \in \mathbb{N}$, let $f_i \colon Q \to A$ be a scaling map for Π_i and $\Pi_{i+1} = \textsc{Refine}(\Pi_i, f_i)$. Then Π_j is the* Myhill-Nerode *relation for every $j \in \mathbb{N}$ such that $\Pi_{j+1} = \Pi_j$.*

Proof. We proved in Lemma 4 that $\equiv$ is a congruence on (Q, δ). Moreover, $\equiv$ clearly saturates F. Hence $\equiv$ refines $\cong$. Now we can apply Lemma 15 to prove that $\equiv$ refines Π_j for every $j \in \mathbb{N}$. Moreover, Π_j refines $\equiv$ for every $j \in \mathbb{N}$ such that $\Pi_{j+1} = \Pi_j$ by Lemma 16. □

So we showed that iteration of the refinement indeed yields the Myhill-Nerode relation and the required scaling factors (see Lemmata 12 and 16). However, we need a scaling map for every partition produced during the refinement steps. An initial scaling map can easily be computed (as in [15]) using the signs of life sol that we already computed, but for efficiency reasons the later scaling maps are computed in a simpler way. In the setup of [15] this was not possible, but our definition of scaling maps (see Definition 13) allows us to use any sign of life. We exploit this in the next lemma, where we show how to update the scaling map after a refinement step.

Lemma 18 (Scaling map update). *Let Π be a partition of Q that saturates F. Moreover, let $f \colon Q \to A$ be a scaling map for Π and $\Pi' = \textsc{Refine}(\Pi, f)$. Then the mapping $g \colon Q \to A$ with $g(q) = f(q) \cdot f(r([q]_{\equiv_{\Pi'}}))^{-1}$ for every $q \in Q$ is a scaling map for Π'.*

Proof. By Definition 13 we need to prove that

(i) $\nu_q = f(q) \cdot f(p)^{-1} \cdot \nu_p$ where $p = r([q]_{\equiv_{\Pi'}})$ for every $q \in F$,
(ii) $g(q) = 1$ for every $q \in D$.
(iii) for every $B \in \Pi'$ there exists $C \in C_\Sigma(Q)$ such that $(\varphi, C[q]) = f(q) \cdot f(r(B))^{-1} \cdot (\varphi, C[r(B)])$ and C is a sign of life of q for every live $q \in B$.

Let us start with condition (i). Let $q \in F$, $p = r([q]_{\equiv_\Pi})$, and $p' = r([q]_{\equiv_{\Pi'}})$. We observe that

$$f(q) \cdot f(p')^{-1} \cdot \nu_{p'} = \nu_q \cdot \nu_p^{-1} \cdot \nu_{p'}^{-1} \cdot \nu_p \cdot \nu_{p'} = \nu_q$$

since $q \equiv_\Pi p \equiv_\Pi p'$ and thus $\{q, p, p'\} \subseteq F$ (because $\equiv_\Pi$ saturates F). Condition (ii) trivially holds because $r([q]_{\equiv_{\Pi'}})$ is dead whenever q is dead.

Algorithm 2 Minimization of dwta M.

```
    (Π, sol, D) ← ComputeSoL                    // see Algorithm 1; complexity: O(lm)
 2: Π ← RefineCong(Π)                            // refine to coarsest congruence; complexity: O(lm log n)
    f ← ComputeSM(Π)                            // see Algorithm 3; complexity: O(n²)
 4: repeat
      Π′ ← Π                                    // store old partition
 6:   Π ← Refine(Π, f)                          // see Algorithm 4; complexity: O(lm)
      f ← UpdateSM(Π, f)                        // see Algorithm 5; complexity: O(n)
 8: until Π′ = Π
    return M/≡_Π                                // see Definition 5; complexity O(lm)
```

Finally, let us consider condition (iii). Let $B \in \Pi'$ and let $S \in \Pi$ be such that $B \subseteq S$. By Definition 13 let $C \in C_\Sigma(Q)$ be such that $(\varphi, C[q]) = f(q) \cdot (\varphi, C[r(S)])$ and C is a sign of life of q for every live $q \in S$. Since $r(B) \in S$

$$(\varphi, C[q]) = f(q) \cdot (\varphi, C[r(S)]) = f(q) \cdot f(r(B))^{-1} \cdot (\varphi, C[r(B)])$$

for every live $q \in B$, which proves the statement. □

4.2. Algorithm

In this section, we finally present the minimization algorithm and its auxiliary procedures. In addition, we analyze their complexity. Let us start with the principal structure of the minimization algorithm. We already remarked that we iteratively refine an initial partition until we reach a fixpoint. The general approach is shown in Algorithm 2. Note that M, sol, and D are treated as global variables in our algorithms to simplify the presentation. The variables sol and D are set once and then retain their value without further changes. The procedure RefineCong(Π) returns the coarsest congruence on (Q, δ) that refines Π. It is known [28] that it can be implemented to run in time $O(lm \log n)$.

Our approach is quite the opposite of the approach of [24, 25]. Their approach first handles the transition weights and then uses RefineCong to derive the final partition. Here we apply RefineCong directly to the initial partition computed by Algorithm 1. This guarantees that all equivalent states share signs of life by Lemma 9. Then we compute the initial scaling map using the already computed signs of life. After that we iteratively refine and update the scaling map until a fixpoint is reached. By Theorem 17 we know that this fixpoint is $\equiv$.

Let us proceed in order of occurrence in Algorithm 2. Next we show in Algorithm 3 how to compute the initial scaling map. In this algorithm we use the computed signs of life. Note that sol and D refer, as usual, to the signs of life and the set of dead states, respectively, as computed by Algorithm 1.

Lemma 19. *For every partition Π of Q such that $\equiv$ refines Π and Π refines $\cong$, Algorithm 3 can be implemented to run in time $O(n^2)$ and returns a scaling map f for Π.*

Proof. Since $\equiv$ refines Π and Π refines $\cong$, the set D of dead states is a block in Π. Thus, each state is handled exactly once (either in line 4 or 6 of Algorithm 3). Recall that we assume that $r(P)$ can be computed in constant time for every $P \subseteq Q$. Thus, only the computations $(\varphi, \mathrm{sol}(q)[q])$ and $(\varphi, \mathrm{sol}(q)[p])$ in lines 2 and 4, respectively, remain. By Theorem 11(iv),

Algorithm 3 ComputeSM(Π): Compute scaling map.

Require: partition Π of Q
for all $P \in \Pi$ such that $P \cap D = \emptyset$ **do**
2: let $q = r(P)$ and $a = (\varphi, \mathrm{sol}(q)[q])^{-1}$ // compute representative and weight
for all $p \in P$ **do**
4: $f(p) \leftarrow a \cdot (\varphi, \mathrm{sol}(q)[p])$ // compute scaling factor
for all $q \in D$ **do**
6: $f(q) \leftarrow 1$ // dead states have scaling factor 1
return f

$\mathrm{sol}(q)$ consists of at most n transitions and since M is a dwta we can compute $(\varphi, \mathrm{sol}(q)[q])$ and $(\varphi, \mathrm{sol}(q)[p])$ in time $O(n)$. This yields that Algorithm 3 runs in time $O(n^2)$.

For the correctness we should establish conditions (i)–(iii) of Definition 13. Condition (ii) is trivially fulfilled by lines 5–6. Since no block of Π apart from D itself intersects with D, for every $P \in \Pi$ and live $p \in P$, line 4 enforces

$$(\varphi, \mathrm{sol}(r(P))[p]) = f(p) \cdot (\varphi, \mathrm{sol}(r(P))[r(P)]) \ .$$

Thus, condition (iii) is fulfilled if we show that $\mathrm{sol}(r(P))$ is a sign of life of q for every $q \in P$. By Theorem 11(iv), $\mathrm{sol}(r(P))$ is a sign of life of $r(P)$. Consequently, $\mathrm{sol}(r(P))$ is a sign of life of q by Lemma 9 because $r(P) \equiv_\Pi q$ for every $q \in P$. Moreover, $\mathrm{sol}(q) = \Box$ for every $q \in F$ by Theorem 11(iii), which shows condition (i) in a straightforward manner. □

Definition 14 already gives a concise description of the refinement process. Let us nevertheless present pseudo-code for the procedure to make the complexity analysis easier. In Algorithm 4 we present an implementation of the procedure outlined in Definition 14. Note that essentially all transitions are checked in Definition 14 and in the algorithm we make this fact apparent.

Lemma 20. *Algorithm 4 implements* Refine(Π, f) *and runs in time* $O(lm)$.

Proof. We leave correctness as an exercise to the reader. For the time complexity, we observe that the loop in lines 2–11 is executed at most m times. In addition, the inner loop in lines 5–11 runs at most l times per iteration of the outer loop. Using suitable representations, the remaining operations in lines 1–11 can be made to run in constant time, so that we need $O(lm)$ for lines 1–12. The loop in lines 13–16 runs at most n times and since $n \leq m$, we obtain the overall time complexity $O(lm)$. □

Finally, we present the easy algorithm for the update of the scaling map. Lemma 18 already reveals how to compute the update instead of recomputing the scaling map using ComputeSF, which was used in [15]. The reuse of the old scaling map to compute the new one yields a performance gain of a factor n (cf. Lemma 19).

Lemma 21. *Algorithm 5 runs in time* $O(n)$.

Proof. The proof is straightforward and omitted. □

Theorem 22. *Algorithm 2 returns a minimal dwta recognizing* φ *and runs in time* $O(lmn)$.

Algorithm 4 REFINE(Π, f): Refine partition.

Require: partition Π of Q and scaling map f for Π

$S \leftarrow \emptyset$ // collect all split states in S

2: **for all** $\sigma \in \Sigma_k$ and $q_1, \ldots, q_k \in Q$ such that $\delta_\sigma(q_1, \ldots, q_k) \notin D$ **do**

  $q \leftarrow \delta_\sigma(q_1, \ldots, q_k)$ // get target state

4: $a \leftarrow c_\sigma(q_1, \ldots, q_k)$ // get transition weight

  for all $i \in [1, k]$ **do**

6: $p \leftarrow \delta_\sigma(q_1, \ldots, q_{i-1}, r([q_i]), q_{i+1}, \ldots, q_k)$ // get target state using representative

  if $q \not\equiv_\Pi p$ **then**

8: $S \leftarrow S \cup \{q_i\}$ // violation of condition (i) of Definition 14; split state

  $b \leftarrow c_\sigma(q_1, \ldots, q_{i-1}, r([q_i]), q_{i+1}, \ldots, q_k)$ // get transition weight using representative

10: **if** $f(q_i)^{-1} \cdot a \cdot f(q) \neq b \cdot f(p)$ **then**

  $S \leftarrow S \cup \{q_i\}$ // violation of condition (ii) of Definition 14; split state

12: $\Pi' \leftarrow \emptyset$ // output partition

for all $P \in \Pi$ **do**

14: $\Pi' \leftarrow \Pi \cup \{P \setminus S\}$ // take out split states from current block

  if $P \setminus S \neq P$ **then**

16: $\Pi' \leftarrow \Pi' \cup \{P \cap S\}$ // if states were split, add split states as new block

return Π'

PROOF. By Theorem 11, the partition Π in line 1 is such that $\equiv$ refines Π, which saturates F. Since $\equiv$ is a congruence on (Q, δ) by Lemma 4, the partition Π in line 2 still has the properties that $\equiv$ refines it and that it saturates F. Moreover, it clearly refines $\cong$ because $\cong$ is the coarsest congruence on (Q, δ). Thus, we meet the requirements for Lemma 19 and f in line 3 is a scaling map for Π.

By repeated applications of Lemmata 15 and 18 we can easily show that every Π in lines 2–9 is a partition such that $\equiv$ refines Π and Π refines $\cong$. Moreover, every Π in lines 5–9 is a refinement of Π' by Lemma 15. This yields that the loop in lines 4–8 can be executed at most n times. Finally, by Lemma 16 we have that Π coincides with $\equiv$ in line 9 and thus by Theorem 8 the algorithm returns a minimal dwta recognizing φ.

The comments in Algorithm 2 list the complexity of the subalgorithms. Those complexities are known or were proved in Theorem 11 and Lemmata 19, 20, 21, and 12. We already remarked that the loop in lines 4–8 can be entered at most n times. The contents of the loop runs in time $O(lm)$ because $m \geq n$. This immediately yields that the overall algorithm runs in time $O(lmn)$. □

5. A small example

Let us rediscuss the example of [15] using our new minimization algorithm. We use the field $(\mathbb{R}, +, \cdot, 0, 1)$ and the ranked alphabet

$$\Sigma = \{\sigma, \text{Alice}, \text{Bob}, \text{loves}, \text{hates}, \text{ugly}, \text{nice}, \text{mean}\} \ ,$$

of which σ is binary and all other symbols are nullary. We abbreviate the multi-letter symbols by their first letter (e.g., Alice by just A). As states we use

$$Q = \{\text{NP}, \text{VB}, \text{ADJ}, \text{VP}, \text{NN}, \text{S}, \bot\}$$

Algorithm 5 UpdateSM(Π, f): Update scaling map.

Require: partition Π of Q and map $f\colon Q \to A \setminus \{0\}$
  for all $P \in \Pi$ **do**
    $a \leftarrow f(r(P))^{-1}$ // get old scaling factor of representative of block P
    for all $p \in P$ **do**
      $f'(p) \leftarrow f(p) \cdot a$ // compute new scaling factor
  return f'

of which only S is final (with $\nu_S = 1$). We order the states in the same sequence as given in the previous display and we use the representative mapping r, which for every nonempty $P \subseteq Q$, returns the smallest state in P with respect to that order. Transitions and transition weights are given as follows:

$$\begin{array}{llll} \delta_\sigma(\mathrm{NN},\mathrm{VP}) = \mathrm{S} & \delta_\sigma(\mathrm{NP},\mathrm{VP}) = \mathrm{S} & \delta_\sigma(\mathrm{VB},\mathrm{NN}) = \mathrm{VP} & \delta_\sigma(\mathrm{VB},\mathrm{NP}) = \mathrm{VP} \\ c_\sigma(\mathrm{NN},\mathrm{VP}) = 0.5 & c_\sigma(\mathrm{NP},\mathrm{VP}) = 0.5 & c_\sigma(\mathrm{VB},\mathrm{NN}) = 0.5 & c_\sigma(\mathrm{VB},\mathrm{NP}) = 0.5 \\ \delta_\sigma(\mathrm{ADJ},\mathrm{NN}) = \mathrm{NP} & \delta_\sigma(\mathrm{ADJ},\mathrm{NP}) = \mathrm{NP} & & \\ c_\sigma(\mathrm{ADJ},\mathrm{NN}) = 0.5 & c_\sigma(\mathrm{ADJ},\mathrm{NP}) = 0.5 & & \end{array}$$

and

$$\begin{array}{lllllll} \delta_{\mathrm{A}}() = \mathrm{NN} & \delta_{\mathrm{B}}() = \mathrm{NN} & \delta_{\mathrm{l}}() = \mathrm{VB} & \delta_{\mathrm{h}}() = \mathrm{VB} & \delta_{\mathrm{u}}() = \mathrm{ADJ} & \delta_{\mathrm{n}}() = \mathrm{ADJ} & \delta_{\mathrm{m}}() = \mathrm{ADJ} \\ c_{\mathrm{A}}() = 0.5 & c_{\mathrm{B}}() = 0.5 & c_{\mathrm{l}}() = 0.5 & c_{\mathrm{h}}() = 0.5 & c_{\mathrm{u}}() = 0.33 & c_{\mathrm{n}}() = 0.33 & c_{\mathrm{m}}() = 0.33 \ . \end{array}$$

For all remaining combinations (x, y) we set $\delta_\sigma(x, y) = \bot$ and $c_\sigma(x, y) = 1$. Now, we have completely specified our input dwta M, which has no useless states.

Next, we compute signs of life according to Algorithm 1. It may return $(\Pi, \mathrm{sol}, \{\bot\})$ where $\Pi = \{\{S\}, \{\mathrm{VP}, \mathrm{NP}, \mathrm{NN}\}, \{\mathrm{ADJ}, \mathrm{VB}\}, \{\bot\}\}$ and the signs of life are

$$\begin{array}{lll} \mathrm{sol}(\mathrm{S}) = \square & \mathrm{sol}(\mathrm{NP}) = \sigma(\square, \mathrm{VP}) & \mathrm{sol}(\mathrm{ADJ}) = \sigma(\sigma(\square, \mathrm{NN}), \mathrm{VP}) \\ \mathrm{sol}(\mathrm{VP}) = \sigma(\mathrm{NN}, \square) & \mathrm{sol}(\mathrm{NN}) = \sigma(\square, \mathrm{VP}) & \mathrm{sol}(\mathrm{VB}) = \sigma(\mathrm{NN}, \sigma(\square, \mathrm{NN})) \ . \end{array}$$

Then we call RefineCong(Π), which returns

$$\Pi' = \{\{\mathrm{S}\}, \{\mathrm{VP}\}, \{\mathrm{NP}, \mathrm{NN}\}, \{\mathrm{ADJ}\}, \{\mathrm{VB}\}, \{\bot\}\}$$

because the sign of life of NN is not a sign of life of VP and analogously for ADJ and VB. The subsequent call ComputeSM(Π') returns the scaling map f with $f(q) = 1$ for all live states q.

Finally, we refine this partition, but NP and NN are not split. Thus, we construct the dwta $(M/{\equiv_{\Pi'}}) = (Q', \Sigma, \mathbb{R}, \delta', c', \nu')$ with

$$Q' = \{\mathrm{NP}, \mathrm{VB}, \mathrm{ADJ}, \mathrm{VP}, \mathrm{S}, \bot\} \ ,$$

of which only S is final with $\nu'_{\mathrm{S}} = 1$. Transitions and transition weights are given as follows:

$$\begin{array}{lll} \delta'_\sigma(\mathrm{NP},\mathrm{VP}) = \mathrm{S} & \delta'_\sigma(\mathrm{VB},\mathrm{NP}) = \mathrm{VP} & \delta'_\sigma(\mathrm{ADJ},\mathrm{NP}) = \mathrm{NP} \\ c'_\sigma(\mathrm{NP},\mathrm{VP}) = 0.5 & c'_\sigma(\mathrm{VB},\mathrm{NP}) = 0.5 & c'_\sigma(\mathrm{ADJ},\mathrm{NP}) = 0.5 \end{array}$$

and

$$\begin{array}{llllll} \delta'_{\mathrm{A}}() = \mathrm{NP} & \delta_{\mathrm{B}}() = \mathrm{NP} & \delta'_{\mathrm{l}}() = \mathrm{VB} & \delta_{\mathrm{h}}() = \mathrm{VB} & \delta'_{\mathrm{u}}() = \mathrm{ADJ} & \delta'_{\mathrm{n}}() = \mathrm{ADJ} \quad \delta'_{\mathrm{m}}() = \mathrm{ADJ} \\ c'_{\mathrm{A}}() = 0.5 & c'_{\mathrm{B}}() = 0.5 & c'_{\mathrm{l}}() = 0.5 & c'_{\mathrm{h}}() = 0.5 & c'_{\mathrm{u}}() = 0.33 & c'_{\mathrm{n}}() = 0.33 \quad c'_{\mathrm{m}}() = 0.33 \ . \end{array}$$

For all remaining combinations (x, y) we have $\delta'_\sigma(x, y) = \bot$ and $c'_\sigma(x, y) = 1$.

Conclusion and open problems

We presented the first efficient minimization algorithm for deterministic weighted tree automata over semifields. If we suppose that the semifield operations can be performed in constant time, then the algorithm runs in time $O(lmn)$. In fact, our algorithm works equally well for dwta with final states (i.e., $\nu_q \in \{0, 1\}$ for every $q \in Q$) because it then returns a minimal equivalent dwta with final states. This contrasts the situation encountered with the pushing strategy of [24, 25], which, in general, needs final weights.

Finally, let us mention some open problems. Can a Hopcroft-like strategy [29] improve the presented algorithm to run in time $O(lm \log n)$? The author doubts that the presented approach yields to this method, however the approach of [24, 25] for deterministic weighted string automata might. This could lead to an algorithm that outperforms our algorithm. Finally, the theoretical foundation for minimization of (even nondeterministic) wta over fields has been laid in [1, 18], but an efficient algorithm and a detailed complexity analysis are still missing.

Acknowledgements

The author gratefully acknowledges the financial support of the German Academic Exchange Service (DAAD). In addition, I would like to thank Johanna Högberg and Jonathan May for inspiring discussions. Finally, I would like to thank the reviewers for pointing out imperfections in the presentation.

References

[1] S. Bozapalidis, O. Louscou-Bozapalidou, The rank of a formal tree power series, Theoret. Comput. Sci. 27 (1–2) (1983) 211–215.
[2] S. Bozapalidis, Equational elements in additive algebras, Theory Comput. Systems 32 (1) (1999) 1–33.
[3] W. Kuich, Formal power series over trees, in: S. Bozapalidis (Ed.), Proc. 3rd Int. Conf. Developments in Language Theory, Aristotle University of Thessaloniki, 1998, pp. 61–101.
[4] B. Borchardt, H. Vogler, Determinization of finite state weighted tree automata, J. Autom. Lang. Combin. 8 (3) (2003) 417–463.
[5] W. Kuich, A. Salomaa, Semirings, Automata, Languages, Vol. 5 of Monographs in Theoretical Computer Science. An EATCS Series, Springer, 1986.
[6] F. Gécseg, M. Steinby, Tree Automata, Akadémiai Kiadó, Budapest, 1984.
[7] F. Gécseg, M. Steinby, Tree languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 3, Springer, 1997, Ch. 1, pp. 1–68.
[8] J. Graehl, Carmel finite-state toolkit, ISI/USC, `http://www.isi.edu/licensed-sw/carmel` (1997).
[9] M. Frishert, L. G. Cleophas, B. W. Watson, Fire Station: An environment for manipulating finite automata and regular expression views, in: M. Domaratzki, A. Okhotin, K. Salomaa, S. Yu (Eds.), Proc. 9th Int. Conf. Implementation and Application of Automata, Vol. 3317 of LNCS, Springer, 2004, pp. 125–133.
[10] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, M. Mohri, OpenFst — a general and efficient weighted finite-state transducer library, in: J. Holub, J. Žďárek (Eds.), Proc. 12th Int. Conf. Implementation and Application of Automata, Vol. 4783 of LNCS, Springer, 2007, pp. 11–23.

[11] K. Knight, J. Graehl, An overview of probabilistic tree transducers for natural language processing, in: A. F. Gelbukh (Ed.), Proc. 6th Int. Conf. Computer Linguistics and Intelligent Text Processing, Vol. 3406 of LNCS, Springer, 2005, pp. 1–24.
[12] J. May, K. Knight, Tiburon: A weighted tree automata toolkit, in: O. H. Ibarra, H.-C. Yen (Eds.), Proc. 11th Int. Conf. Implementation and Application of Automata, Vol. 4094 of LNCS, Springer, 2006, pp. 102–113.
[13] B. Borchardt, The theory of recognizable tree series, Ph.D. thesis, Technische Universität Dresden (2005).
[14] J. May, K. Knight, A better n-best list: Practical determinization of weighted finite tree automata, in: R. C. Moore, J. A. Bilmes, J. Chu-Carroll, M. Sanderson (Eds.), Proc. North American Chapter of the Association for Computational Linguistics, 2006, pp. 351–358.
[15] A. Maletti, Minimizing deterministic weighted tree automata, in: C. Martín Vide, F. Otto, H. Fernau (Eds.), Proc. 2nd Int. Conf. Language and Automata: Theory and Applications, Vol. 5196 of LNCS, Springer, 2008, pp. 357–372.
[16] B. Borchardt, The Myhill-Nerode theorem for recognizable tree series, in: Z. Ésik, Z. Fülöp (Eds.), Proc. 7th Int. Conf. Developments in Language Theory, Vol. 2710 of LNCS, Springer, 2003, pp. 146–158.
[17] B. Borchardt, A pumping lemma and decidability problems for recognizable tree series, Acta Cybernet. 16 (4) (2004) 509–544.
[18] S. Bozapalidis, Effective construction of the syntactic algebra of a recognizable series on trees, Acta Inform. 28 (4) (1991) 351–363.
[19] D. Angluin, Learning regular sets from queries and counterexamples, Inform. and Comput. 75 (2) (1987) 87–106.
[20] A. Habrard, J. Oncina, Learning multiplicity tree automata, in: Y. Sakakibara, S. Kobayashi, K. Sato, T. Nishino, E. Tomita (Eds.), Proc. 8th Int. Colloquium Grammatical Inference, Vol. 4201 of LNAI, Springer, 2006, pp. 268–280.
[21] F. Drewes, H. Vogler, Learning deterministically recognizable tree series, J. Autom. Lang. Combin. 12 (3) (2007) 332–354.
[22] A. Maletti, Learning deterministically recognizable tree series — revisited, in: S. Bozapalidis, G. Rahonis (Eds.), Proc. 2nd Int. Conf. Algebraic Informatics, Vol. 4728 of LNCS, Springer, 2007, pp. 218–235.
[23] H. Seidl, Deciding equivalence of finite tree automata, SIAM J. Comput. 19 (3) (1990) 424–437.
[24] M. Mohri, Minimization algorithms for sequential transducers, Theoret. Comput. Sci. 234 (1–2) (2000) 177–201.
[25] J. Eisner, Simpler and more general minimization for weighted finite-state automata, in: Human Language Technology Conf. of the North American Chapter of the Association for Computational Linguistics, 2003, pp. 64–71.
[26] H. Comon-Lundh, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree automata—techniques and applications, see: `http://tata.gforge.inria.fr/` (2007).
[27] J. Besombes, J.-Y. Marion, Learning tree languages from positive examples and membership queries, Theoret. Comput. Sci. 382 (3) (2007) 183–197.
[28] J. Högberg, A. Maletti, J. May, Backward and forward bisimulation minimisation of tree automata, in: J. Holub, J. Žďárek (Eds.), Proc. 12th Int. Conf. Implementation and Application of Automata, Vol. 4783 of LNCS, Springer, 2007, pp. 109–121.
[29] J. E. Hopcroft, An $n \log n$ algorithm for minimizing states in a finite automaton, in: Z. Kohavi (Ed.), Theory of Machines and Computations, Academic Press, 1971, pp. 189–196.