Minimizing Deterministic Weighted Tree Automata

Andreas Maletti*

International Computer Science Institute
1947 Center Street, Suite 600
Berkeley, CA 94704, USA
maletti@icsi.berkeley.edu

Abstract. The problem of efficiently minimizing deterministic weighted tree automata (wta) is investigated. Such automata have found promising applications as language models in Natural Language Processing. A polynomial-time algorithm is presented that given a deterministic wta over a commutative semifield, of which all operations including the computation of the inverses are polynomial, constructs an equivalent minimal (with respect to the number of states) deterministic and total wta. If the semifield operations can be performed in constant time, then the algorithm runs in time $O(rmn^4)$ where r is the maximal rank of the input symbols, m is the number of transitions, and n is the number of states of the input wta.

1 Introduction

Weighted tree automata (wta) [1–4] 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 largely 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]) based on 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 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 recognizable. The deterministically recognizable tree series are exactly those recognizable tree series that can be computed by deterministic wta. Recognizable and deterministically recognizable tree series have been thoroughly investigated (see [3, 13] and references provided therein). In fact, [4] and [14] show which recognizable tree series are also deterministically recognizable.

In this contribution, we consider deterministically recognizable tree series. To the author's knowledge, we propose the first polynomial-time minimization algorithm for deterministic wta over semifields. A MYHILL-NERODE theorem for tree series recognized by such automata is

* Author on leave from *Technische Universität Dresden, Faculty of Computer Science, 01062 Dresden, Germany* with the help of financial support by a DAAD (German Academic Exchange Service) grant.

known [15]. However, it only asserts the existence of a unique, up to slight changes of representation, minimal (with respect to the number of states) deterministic wta recognizing a given tree series. The construction of such a wta, which is given in [15], is not effective, but with the help of the pumping lemma of [16] an exponential-time algorithm, which given a deterministic wta constructs an equivalent minimal deterministic and total wta, could easily be derived. For (not necessarily deterministic) wta over fields the situation is similar. In [1, 17] the existence of a unique, up to slight changes of representation, minimal wta is proved. Moreover, [17] 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 exponential.

ANGLUIN [18] learning algorithms exist for both general [19] and deterministic [20, 21] 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 [22, 16], a simple implementation would return counterexamples of exponential size, which would yield an exponential-time minimization algorithm. Clearly, this can be avoided by the method presented in this contribution.

Finally, let us mention the minimization procedures [23, 24] 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 deterministic wta recognizing the given series after the unweighted minimization.

Here we give a direct minimization construction, which uses partition refinement as in the unweighted case [25]. To this end, we first define the MYHILL-NERODE relation on states of the deterministic input wta. This definition, as well as the MYHILL-NERODE relation on tree series [15], will include a scaling factor and Algorithm 2 will determine those scaling factors. In the refinement process (see Definition 13) 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 $O(rmn^4)$ where r is the maximal rank of the input symbols, m is the number of transitions, and n is the number of states of the input wta.

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 $\equiv'$ be equivalence relations on Q. Then $\equiv$ is a refinement of $\equiv'$ if $\equiv \subseteq \equiv'$. The equivalence class of $q \in Q$ is $[q]_\equiv = \{q' \in Q \mid q' \equiv q\}$. 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 $P \neq P'$. A mapping $r\colon (Q/\equiv) \to Q$ is a representative mapping if $r(P) \in P$ for every $P \in (Q/\equiv)$. 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 $\Pi' \subseteq \Pi$.

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 $\sigma(t_1,\dots,t_n) \in T_\Sigma(Q)$ for every $\sigma \in \Sigma_n$ and $t_1,\dots,t_n \in T_\Sigma(Q)$. 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 $\mathrm{var}(\sigma(t_1,\dots,t_n)) = \bigcup_{i=1}^n \mathrm{var}(t_i)$ for every $\sigma \in \Sigma_n$ and $t_1,\dots,t_n \in T_\Sigma(Q)$. 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 $t \in T_\Sigma(Q)$. Finally, 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 $\mathrm{ht}(\sigma(t_1,\dots,t_n)) = 1 + \max\{\mathrm{ht}(t_i) \mid i \in [1,n]\}$ and $\mathrm{size}(\sigma(t_1,\dots,t_n)) = 1 + \sum_{i=1}^n \mathrm{size}(t_i)$ for every $\sigma \in \Sigma_n$ and $t_1,\dots,t_n \in T_\Sigma(Q)$. Note that $\max\emptyset = 0$.

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

A *weighted tree automaton* [1–4] (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. 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 inductively defined by

$$h_\mu(\sigma(t_1,\dots,t_n))_q = \sum_{q_1,\dots,q_n\in Q} \mu_n(\sigma)_{(q_1,\dots,q_n),q} \cdot \prod_{i=1}^n h_\mu(t_i)_{q_i}$$

for every $\sigma \in \Sigma_n$, $q \in Q$, and $t_1,\dots,t_n \in T_\Sigma$. 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 $\mu_n(\sigma)_{w,q} \neq 0$. Since we will exclusively deal with deterministic and total wta over semifields from now on, we will use the following representation: $M = (Q,\Sigma,\mathcal{A},\delta,c,\nu)$ where $\delta \subseteq \bigcup_{n\geq 0} Q^n \times \Sigma_n \times Q$ is finite and $c\colon \delta \to A\setminus\{0\}$. In particular, $(w,\sigma,q) \in \delta$ if and only if $\mu_n(\sigma)_{w,q} \neq 0$, and for every $\tau = (w,\sigma,q) \in \delta$ we have $c(\tau) = \mu_n(\sigma)_{w,q}$. The determinism and totality restriction ensures that δ can be represented as $(\delta_\sigma)_{\sigma\in\Sigma}$ with $\delta_\sigma\colon Q^n \to Q$. We extend δ to a mapping $\delta\colon T_\Sigma(Q) \to Q$ as follows: $\delta(q) = q$ for every $q \in Q$ and $\delta(\sigma(t_1,\dots,t_n)) = \delta_\sigma(\delta(t_1),\dots,\delta(t_n))$ for every $\sigma \in \Sigma_n$ and $t_1,\dots,t_n \in T_\Sigma(Q)$. A state $q \in Q$ is useful if there exists $t \in T_\Sigma$ such that $\delta(t) = q$. The deterministic and total wta M is said to have no useless states if all states of Q are useful.

Similarly, c can be represented as $(c_\sigma)_{\sigma\in\Sigma}$ with $c_\sigma\colon Q^n \to A\setminus\{0\}$. Due to the semifield restriction, this can be extended to a mapping $c\colon T_\Sigma(Q) \to A\setminus\{0\}$ 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.

Proposition 1 (cf. [15, 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 $i \in [1,n]$.*

Finally, let us recall the MYHILL-NERODE congruence relation [15] for tree series. To this end, we first 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. Note that (Q, δ) is a Σ-algebra. 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 $f_\sigma(s_1,\ldots,s_n) \equiv f_\sigma(t_1,\ldots,t_n)$.

Let $\varphi \in A\langle\langle T_\Sigma\rangle\rangle$. The MYHILL-NERODE [15] 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 $a \in A \setminus \{0\}$ such that $(\varphi, C[t]) = a \cdot (\varphi, C[u])$ for every $C \in C_\Sigma$. We note that $\equiv_\varphi$ is a congruence of (T_Σ, Σ) [15, Lemma 5].

3 Myhill-Nerode relation

In this section, we recall the theoretical foundations for the minimization procedure and introduce the MYHILL-NERODE relation on states of a deterministic and total wta. We keep it short because most of the material is only slightly adapted. Readers who are familiar with the MYHILL-NERODE congruence $\equiv_\varphi$ for a tree series φ may decide to read only Definition 2 and proceed to the next section. For the rest of the paper, let $M = (Q, \Sigma, \mathcal{A}, \delta, c, \nu)$ be a deterministic and total wta without useless states and $\mathcal{A} = (A, +, \cdot, 0, 1)$ a semifield, of which multiplication and calculation of inverses can be performed in constant time. Note that, depending on the actual semifield used, 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. Finally, let $\varphi = \varphi_M$.

Definition 2 (cf. [15, page 8]). *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 $a \in A \setminus \{0\}$ such that $(\varphi, C[p]) = a \cdot (\varphi, C[q])$ for every $C \in C_\Sigma$. We denote such a scaling factor a by $a_{p,q}$ for every $p, q \in Q$ such that $p \equiv q$.*

Note the similarity with the definition of the MYHILL-NERODE relation $\equiv_\varphi$ [15]. This similarity allows us to retain some of the useful properties of $\equiv_\varphi$. In the remainder of this section, we show some of those properties. We start with the fact that $\equiv$ is a congruence relation on (Q, δ).

Proposition 3 (cf. [15, Lemma 5]). *The relation $\equiv$ is a congruence relation on (Q, δ).*

The next lemma introduces a more restricted variant of $\equiv$ and relates it to $\equiv$. The more restricted variant will be very useful in the sequel and allows us to avoid an exponential blow-up.

Lemma 4. *The relation $\equiv' \subseteq Q \times Q$, which is given for every $p, q \in Q$ by $p \equiv' q$ if and only if there exists $a \in A \setminus \{0\}$ such that $(\varphi, C[p]) = a \cdot (\varphi, C[q])$ for every $C \in C_\Sigma(Q)$, coincides with $\equiv$.*

In fact, for every $p, q \in Q$ such that $p \equiv q$ the scaling factor $a_{p,q}$ also verifies $p \equiv' q$. This leads to the second main property (the first being the congruence property) that we will later use for refinement (see Proposition 12). The final lemma of this section establishes the relation of $\equiv$ to $\equiv_\varphi$. Note that $\mathrm{index}(\equiv_\varphi)$ coincides with the number of states of a minimal deterministic and total wta recognizing φ [15, Theorem 3].

Lemma 5. $\mathrm{index}(\equiv) = \mathrm{index}(\equiv_\varphi)$.

Proof. Let $t, u \in T_\Sigma$ such that $t \equiv_\varphi u$. There exists $a \in A \setminus \{0\}$ 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, $\equiv$ thus has at most as many equivalence classes as $\equiv_\varphi$. For the converse, let $p, q \in Q$ such that $p \equiv q$. Moreover, let $t, u \in T_\Sigma$ be such that $\delta(t) = p$ and $\delta(u) = q$. Then analogous to the above we can prove that $t \equiv_\varphi u$. Hence, $\mathrm{index}(\equiv)$ and $\mathrm{index}(\equiv_\varphi)$ coincide. □

4 Minimization algorithm

In this section, we will develop our minimization algorithm for deterministic wta. Throughout, let $F = \{q \in Q \mid \nu_q \neq 0\}$. Note that any deterministic wta M' can be converted in linear time (in the number of transitions) into an equivalent deterministic and total wta without useless states. In contrast to the classical minimization algorithm for deterministic unweighted tree automata, we need to determine the scaling factor $a_{p,q}$ (see Definition 2) for each pair (p, q) of equivalent states. We will use the concept of a *sign of life* to help us determine it.

Definition 6. *A state $q \in Q$ is* live *if $\delta(C[q]) \in F$ 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.

Roughly speaking, a state q is live if some final state can be reached from it. A sign of life of q shows one such path. Note that the sign-of-life context may contain states. Our first task is to determine signs of life for all states. This also identifies live and dead states. Let n be the number of states of M, m the number of transitions of M, and r the maximal rank of the symbols in Σ. To simplify the complexity statements, suppose that $r \geq 1$. Note that consequently $m \geq n$.

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

Require: deterministic and total wta $M = (Q, \Sigma, \mathcal{A}, \delta, c, \nu)$
$D \leftarrow Q \setminus F$ // unexplored states
2: $\mathrm{sol} \leftarrow \{(q, \square) \mid q \in F\}$ // final states have trivial sign of life
$T \leftarrow \{(w, \sigma, q) \in \delta \mid q \in F\}$ // add all transitions leading to a final state to FIFO queue
4: **while** $T \neq \emptyset$ **do**
let $\tau = ((q_1, \ldots, q_k), \sigma, q) \in T$ // get first element in FIFO queue T
6: $I \leftarrow \{i \in [1, k] \mid q_i \in D, \forall j \in [1, i-1] \colon q_j \neq q_i\}$ // select indexes of unexplored states
$\mathrm{sol} \leftarrow \mathrm{sol} \cup \{(q_i, \mathrm{sol}(q)[\sigma(q_1, \ldots, q_{i-1}, \square, q_{i+1}, \ldots, q_k)]) \mid i \in I\}$ // add signs of life
8: $P \leftarrow \{q_i \mid i \in I\}$ // new live states
$D \leftarrow D \setminus P$ // remove new live states from unexplored states
10: $T \leftarrow (T \setminus \{\tau\}) \cup \{(w, \gamma, p) \in \delta \mid p \in P\}$ // add all transitions leading to new live states
end while
12: $\Pi \leftarrow \{\{q \in Q \setminus D \mid \mathrm{ht}(\mathrm{sol}(q)) = i\} \mid i \geq 1\} \cup \{D\}$ // group states by height of sign of life
return (Π, sol, D)

Algorithm 1 returns an initial partition Π, signs of life sol, and the set D of dead states. Let us make two remarks. First, it is essential that T is handled as a FIFO queue with additions at the end and removal at the beginning. This guarantees that the height of the constructed signs of life is minimal. Second, $\emptyset$ might be an element of Π. To avoid complicated case distinctions, we permit this slightly nonstandard behavior, which does not affect the correctness of our algorithms.

Lemma 7. *Let (Π, sol, D) be the result of running Algorithm 1 on M. Then $\mathrm{sol}(q)$ is a sign of life of q of size at most rn for every state q of $Q \setminus D$, D is the set of all dead states, Π saturates F, and $\equiv$ is a refinement of $\equiv_\Pi$. Moreover, Algorithm 1 can be implemented to run in time $O(rm)$.*

Proof. Lines 1–3 run in time $O(m)$ because $m \geq n$. Clearly, each transition can be added at most once to T, so lines 4–11 can be executed at most m times. Lines 6–8 can be executed in time $O(r)$; note that this requires a list representation of the signs of life (i.e., a sign of life is a list of pairs consisting of a transition and an integer indicating the position of $\Box$) to avoid the creation and/or copying of transitions. If we suppose that access to the list of transitions leading to a certain state is constant (which can be achieved by a $O(m)$ preprocessing step sorting the transitions in n buckets), then line 10 can be executed in $O(r)$ time. Since $r \geq 1$, we obtain a running time of $O(rm)$. Finally, we note that the partition constructed in line 12 could have been constructed during the loop at no additional expense; we presented it this way for clarity.

Next, we prove that $\mathrm{sol}(q)$ is indeed a sign of life of q for every $q \in Q \setminus D$. The trivial contexts added for each final state in line 2 are obviously signs of life. It remains to show that C, the second component in the pair of line 7, is a sign of life of q_i. By induction hypothesis, we may assume that $\mathrm{sol}(q)$ is a sign of life of q; i.e., $\delta(\mathrm{sol}(q)[q]) \in F$. Since M is deterministic and $((q_1, \ldots, q_k), \sigma, q) \in \delta$, we obtain $\delta(C[q_i]) = \delta(\mathrm{sol}(q)[q]) \in F$ and thus C is a sign of life of q_i. It is obvious that Π saturates F. We leave the proof of the fact that D is indeed the set of all dead states to the reader.

Clearly, final states are assigned a sign of life of height 1 and size $1 \leq r$. Further signs of life are always constructed as $C = \mathrm{sol}(q)[\sigma(q_1, \ldots, q_{i-1}, \Box, q_{i+1}, \ldots, q_k)]$ (see line 7). Thus, the height (respectively, the size) of the new sign of life C is 1 (respectively, at most r) greater than that of the sign of life $\mathrm{sol}(q)$. Consequently, height and size of every sign of life $\mathrm{sol}(q)$ with $q \in Q \setminus D$ are at most n and rn, respectively. Finally, we have to show that $\equiv$ is a refinement of $\equiv_\Pi$. To this end, we have to show that $p \equiv q$ implies that $p \equiv_\Pi q$ for every $p, q \in Q$. Let $p, q \in Q$ be such that $p \equiv q$. Clearly, p and q share all signs of life (see Lemma 4). Consequently, if p is dead, then also q must be dead, and in that case, $p \equiv_\Pi q$. Otherwise, p and q are live. We already remarked that Algorithm 1 computes signs of life that are minimal in height; the proof of that statement is left as an exercise. The height-minimal sign of life $\mathrm{sol}(p)$ must be a sign of life of q as well, and consequently, $\mathrm{ht}(\mathrm{sol}(p)) = \mathrm{ht}(\mathrm{sol}(q))$, which yields $p \equiv_\Pi q$. □

We allow contexts of $C_\Sigma(Q)$ instead of only contexts of C_Σ as signs of life in order to obtain the linear size complexity given in Lemma 7. The more common approach to use contexts of C_Σ would yield signs of life, whose size might be exponential in n. Since we will run M on signs of life, this would have led to an exponential time complexity.

The principal approach of the minimization algorithm is partition refinement as, for example, in the classical minimization algorithm for minimizing unweighted deterministic tree automata [25]. We successively refine the initial partition returned by Algorithm 1 until $\equiv$ is reached. Before we turn to more detail, let us introduce the main data structure.

Definition 8. *Let Π be a partition of Q that saturates F, $L \subseteq Q$ be the set of live states, $\mathrm{sol}\colon L \to C_\Sigma(Q)$ be such that $\mathrm{sol}(q)$ is a sign of life of q for every $q \in L$, $f\colon L \to A \setminus \{0\}$, and $r\colon (\Pi \setminus \{\emptyset, Q \setminus L\}) \to Q$ a representative mapping. Then $(\Pi, \mathrm{sol}, f, r)$ is a* stage *if*

(i) $\equiv$ is a refinement of $\equiv_\Pi$;
(ii) $\mathrm{sol}(q) = \Box$ for every $q \in F$; and
(iii) for every $q \in L$ we have $(\varphi, \mathrm{sol}(p)[q]) = f(q) \cdot (\varphi, \mathrm{sol}(p)[p])$ where $p = r([q])$.

The stage is stable *if additionally*

(iv) $\equiv_\Pi$ is a congruence of (Q, δ); and

(v) for every $q \in L$, $\sigma \in \Sigma_n$, and $C \in C_n(Q)$ such that $\delta_\sigma(C[q]) \in L$

$$f(q)^{-1} \cdot c_\sigma(C[q]) \cdot f(\delta_\sigma(C[q])) = c_\sigma(C[p]) \cdot f(\delta_\sigma(C[p]))$$

where $p = r([q])$.

In a stage, we have a partition, signs of life, and two new components. The mapping r assigns to each nonempty block (apart from the block of dead states) of the partition a representative and the mapping f assigns to each live state the scaling factor to the representative of its block [see Condition (iii)]. A stable stage additionally requires $\equiv_\Pi$ to be a congruence of (Q, δ) and Condition (v), which is of paramount importance in the implementation (as a wta) of a stable stage. Let us show how to derive a deterministic and total wta recognizing φ from a stable stage.

Definition 9 (cf. [15, Definition 4]). *Let $S = (\Pi, \mathrm{sol}, f, r)$ be a stable stage and D the set of dead states. The wta $M_S = (\Pi \setminus \{\emptyset\}, \Sigma, \mathcal{A}, \delta', c', \nu')$ is constructed as follows: for every $\sigma \in \Sigma_k$ and $q_1, \ldots, q_k \in Q$ let*

- $\delta'_\sigma([q_1], \ldots, [q_k]) = [\delta_\sigma(q_1, \ldots, q_k)]$;
- $c'_\sigma([q_1], \ldots, [q_k]) = 1$ *if* $\delta_\sigma(q_1, \ldots, q_k) \in D$ *and otherwise*

$$c'_\sigma([q_1], \ldots, [q_k]) = \prod_{i=1}^{k} f(q_i)^{-1} \cdot c_\sigma(q_1, \ldots, q_k) \cdot f(\delta_\sigma(q_1, \ldots, q_k))$$

- $\nu'_B = \nu_{r(B)}$ *for every* $B \in \Pi \setminus \{\emptyset, D\}$ *and if* $D \in \Pi \setminus \{\emptyset\}$, *then* $\nu'_D = 0$.

The construction of M_S can be implemented to run in time $O(rm)$. However, some remarks are required here. First, δ' is well-defined because $\equiv_\Pi$ is a congruence on (Q, δ). Second, let us consider the definition of c'. Suppose that $p_1, \ldots, p_k, q_1, \ldots, q_k \in Q$ such that $p_i \equiv_\Pi q_i$ for every $i \in [1, k]$. By the congruence property and Condition (i) of Definition 8, the case distinction is well-defined. We show that

$$\begin{aligned}
&\prod_{i=1}^{k} f(p_i)^{-1} \cdot c_\sigma(p_1, \ldots, p_k) \cdot f(\delta_\sigma(p_1, \ldots, p_k)) \\
&= \prod_{i=2}^{k} f(p_i)^{-1} \cdot c_\sigma(r([p_1]), p_2, \ldots, p_k) \cdot f(\delta_\sigma(r([p_1]), p_2, \ldots, p_k)) \\
&= \ldots \\
&= c_\sigma(r([p_1]), \ldots, r([p_k])) \cdot f(\delta_\sigma(r([p_1]), \ldots, r([p_k]))) \\
&= c_\sigma(r([q_1]), \ldots, r([q_k])) \cdot f(\delta_\sigma(r([q_1]), \ldots, r([q_k]))) \\
&= \ldots \\
&= \prod_{i=2}^{k} f(q_i)^{-1} \cdot c_\sigma(r([q_1]), q_2, \ldots, q_k) \cdot f(\delta_\sigma(r([q_1]), q_2, \ldots, q_k)) \\
&= \prod_{i=1}^{k} f(q_i)^{-1} \cdot c_\sigma(q_1, \ldots, q_k) \cdot f(\delta_\sigma(q_1, \ldots, q_k))
\end{aligned}$$

by repeated application of Condition (v) of Definition 8, which proves the well-definedness of c'. It is obvious that M_S has index$(\equiv_\Pi)$ many states. We should finally show that M_S recognizes φ.

Theorem 10. *Let $S = (\Pi, \mathrm{sol}, f, r)$ be a stable stage and $M_S = (Q', \Sigma, \mathcal{A}, \delta', c', \nu')$. Then M_S is a minimal deterministic and total wta recognizing φ.*

Proof. Let us first show that M_S recognizes φ. From the classical minimization construction it should be clear that $\delta(t) \in \delta'(t)$ for every $t \in T_\Sigma$. We prove the statement $c(t) = c'(t) \cdot f(\delta(t))^{-1}$ by induction on t. Let $t = \sigma(t_1, \ldots, t_k)$ for some $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$. By definition

$$c(t) = c_\sigma(\delta(t_1), \ldots, \delta(t_k)) \cdot \prod_{i=1}^{k} c(t_i) = c_\sigma(\delta(t_1), \ldots, \delta(t_k)) \cdot \prod_{i=1}^{k} \Big(c'(t_i) \cdot f(\delta(t_i))^{-1} \Big)$$

where the last equality is by induction hypothesis. We finish the proof of the auxiliary statement using the definition of c' and $\delta(t_i) \in \delta'(t_i)$

$$c(t) = c'_\sigma(\delta'(t_1), \ldots, \delta'(t_k)) \cdot \prod_{i=1}^{k} c'(t_i) \cdot f(\delta_\sigma(\delta(t_1), \ldots, \delta(t_k)))^{-1} = c'(t) \cdot f(\delta(t))^{-1} \quad .$$

We now continue for every $t \in T_\Sigma$ with a case distinction. Let $q = \delta(t)$. If $q \notin F$, then $(\varphi, t) = 0$ and $(\varphi_{M_S}, t) = 0$ because $\nu'_{[q]} = 0$ (recall that Π saturates F). If $q \in F$, then

$$(\varphi_{M_S}, t) = c'(t) \cdot \nu'_{[q]} = c(t) \cdot f(q) \cdot \nu_{r([q])} = c(t) \cdot \nu_q = (\varphi, t)$$

by Conditions (ii) and (iii) of Definition 8 since $\mathrm{sol}(r([q])) = \square$. Thus, we proved that M_S recognizes φ. By Lemma 5 and [15, Theorem 3], $\mathrm{index}(\equiv)$ is the number of states of a minimal deterministic and total wta recognizing φ. The wta M_S has $\mathrm{index}(\equiv_\Pi)$ states and by Condition (i) of Definition 8, $\equiv$ is a refinement of $\equiv_\Pi$, hence $\mathrm{index}(\equiv_\Pi) \leq \mathrm{index}(\equiv)$. Consequently, $\equiv_\Pi$ coincides with $\equiv$ and M_S is a minimal deterministic and total wta recognizing φ. $\square$

Note that the above theorem also shows that $(\Pi, \mathrm{sol}, f, r)$ can only be a stable stage if $\equiv_\Pi$ coincides with $\equiv$. Since we already have a suitable initial partition that saturates F along with suitable signs of life, our next step is to determine the scaling factors (see Definition 2). To this end, we employ Algorithm 2, which is given a partition and computes the scaling factor for each element relative to a chosen representative of its block. If it cannot compute such a scaling factor (which only happens when the sign of life of the representative is not a sign of life of the considered state), then it splits the state from its current block. The algorithm completely ignores the block of dead states. This can be done because the block of dead states will never be split (since $p \equiv q$ whenever both p and q are dead). The idea of our minimization algorithm is to refine the initial partition returned by Algorithm 1 until we reach $\equiv$. Algorithm 2 completes a suitable partition to a stage.

Lemma 11. *Given signs of life of Algorithm 1 and a partition Π such that $\equiv$ is a refinement of $\equiv_\Pi$, Algorithm 2 can be implemented to run in time $O(rn^3)$ and returns a stage $(\Pi', \mathrm{sol}, f, r)$ such that $\equiv_{\Pi'}$ is a refinement of $\equiv_\Pi$.*

Proof. We defer correctness for the moment. The loop in lines 5–12 can be executed at most n times as each time at least one state is processed. Evaluating the sign of life takes at most $O(rn)$ since the size of any sign of life is at most rn. Thus, lines 7-8 execute in $O(rn^2)$ and the whole algorithm runs in time $O(rn^3)$. Now, let us consider correctness. It should be clear that Condition (ii) holds and Condition (iii) holds because it is enforced in line 8. It remains to check Condition (i). Clearly, $\equiv_{\Pi'}$ is a refinement of $\equiv_\Pi$ and by assumption $\equiv$ is a refinement of $\equiv_\Pi$. Let $p, q \in Q$ such that $p \equiv q$. Consequently, $p \equiv_\Pi q$. Let $p' \in Q$ be such that $p \equiv_\Pi p'$ (cf. the selection in line 6). Then $\delta(\mathrm{sol}(p')[p]) \in F$ if and only if $\delta(\mathrm{sol}(p')[q]) \in F$ because $(\varphi, \mathrm{sol}(p')[p]) \neq 0$ if and only if $(\varphi, \mathrm{sol}(p')[q]) \neq 0$ by $p \equiv q$. This yields that independently of the selection of the representative in line 6, p and q cannot be split in line 7, and hence $p \equiv_{\Pi'} q$. $\square$

Algorithm 2 COMPLETE$(M, \Pi, \mathrm{sol}, D)$: Compute scaling factors.

Require: deterministic and total wta $M = (Q, \Sigma, \mathcal{A}, \delta, c, \nu)$, set D of dead states, sign of life $\mathrm{sol}(q)$ for every $q \in Q \setminus D$, and a partition Π such that $\equiv$ is a refinement of $\equiv_\Pi$

$f \leftarrow \emptyset$ // map of scaling factors
$r \leftarrow \emptyset$ // map of representatives
$\Pi \leftarrow \Pi \setminus \{D\}$ // remove block of dead states
$\Pi' \leftarrow \{D\}$ // output partition containing block of dead states
while $\Pi \neq \emptyset$ and $\Pi \neq \{\emptyset\}$ **do**
  let $P \in \Pi$ and $p \in P$ // select new block and representative
  $P' \leftarrow \{q \in P \mid \delta(\mathrm{sol}(p)[q]) \in F\}$ // collect states that share sign of life $\mathrm{sol}(p)$
  $f \leftarrow f \cup \{(q, c(\mathrm{sol}(p)[q]) \cdot c(\mathrm{sol}(p)[p])^{-1}) \mid q \in P'\}$ // add scaling factors
  $r \leftarrow r \cup \{(P', p)\}$ // add representative for P'
  $\Pi' \leftarrow \Pi' \cup \{P'\}$ // block P' is processed
  $\Pi \leftarrow (\Pi \setminus \{P\}) \cup \{P \setminus P'\}$ // remove old block and add new block $P \setminus P'$
end while
return $(\Pi', \mathrm{sol}, f, r)$

Note that Π' saturates F whenever Π does because $\equiv_{\Pi'}$ is a refinement of $\equiv_\Pi$. As in the unweighted case [25], we use a partition refinement algorithm to compute the MYHILL-NERODE relation. To this end, we start with the initial partition computed by Algorithm 1 and complete it to a stage with the help of Algorithm 2. Then we refine according to Conditions (iv) and (v) of Definition 8. For this to work, variants of these conditions should also be fulfilled by $\equiv$. We already proved in Proposition 3 that $\equiv$ is a congruence of (Q, δ) as in the classical unweighted case. Second, the weights of transitions from equivalent states leading to live states must obey a certain compatibility requirement, which we show in the next proposition.

Proposition 12. *For every $\sigma \in \Sigma_k$, $C \in C_k(Q)$, and $p, q \in Q$ such that $p \equiv q$ and $\delta_\sigma(C[p])$ is live, we have $a_{p,q}^{-1} \cdot c_\sigma(C[p]) \cdot a_{p',q'} = c_\sigma(C[q])$ where $p' = \delta_\sigma(C[p])$ and $q' = \delta_\sigma(C[q])$.*

Proof. Since p' is live, there exists a context $C' \in C_\Sigma(Q)$ such that $\delta(C'[p']) \in F$. Consider the context $C'' = C'[\sigma(C)]$. Since $p \equiv q$, and thus $p \equiv' q$ by Lemma 4, it follows that $(\varphi, C''[p]) = a_{p,q} \cdot (\varphi, C''[q])$. In addition, $p' \equiv q'$ because $\equiv$ is a congruence. Now we compute as follows:

$$\begin{aligned} c_\sigma(C[p]) \cdot (\varphi, C'[p']) = (\varphi, C''[p]) = a_{p,q} \cdot (\varphi, C''[q]) \\ = a_{p,q} \cdot c_\sigma(C[q]) \cdot (\varphi, C'[q']) = a_{p,q} \cdot c_\sigma(C[q]) \cdot a_{p',q'}^{-1} \cdot (\varphi, C'[p']) \ . \end{aligned}$$

Since $(\varphi, C'[p']) \neq 0$ because $\delta(C'[p']) \in F$, we obtain the statement by cancelling $(\varphi, C'[p'])$. □

In the classical unweighted case, only the congruence property 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. Altogether, the previous proposition suggests the following refinement step.

Definition 13. *Let $(\Pi, \mathrm{sol}, f, r)$ be a stage and D be the set of dead states. Then the* refinement REFINE$(M, \Pi, \mathrm{sol}, f, r, D)$ *is defined to be the partition Π' such that for every $p, q \in Q$ we have $p \equiv_{\Pi'} q$ if and only if $p \equiv_\Pi q$ and for every $\sigma \in \Sigma_k$ and $C \in C_k(Q)$*

(i) $\delta_\sigma(C[p]) \equiv_\Pi \delta_\sigma(C[q])$; *and*
(ii) $f(p)^{-1} \cdot c_\sigma(C[p]) \cdot f(\delta_\sigma(C[p])) = f(q)^{-1} \cdot c_\sigma(C[q]) \cdot f(\delta_\sigma(C[q]))$, *if $\delta_\sigma(C[p])$ is live.*

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 $\equiv_{\Pi'}$. Thus, we only refine to the level of $\equiv$ and never beyond. This simple property follows in a straightforward manner from Definition 13 and Proposition 12.

Lemma 14. *Let $(\Pi, \mathrm{sol}, f, r)$ be a stage and D the set of dead states.* $\textsc{Refine}(M, \Pi, \mathrm{sol}, f, r, D)$ *can be implemented to run in time $O(rmn^2)$. Moreover, the resulting partition Π' is such that $\equiv_{\Pi'}$ is a refinement of $\equiv_\Pi$, and if $\equiv$ is a refinement of $\equiv_\Pi$, then $\equiv$ is a refinement of $\equiv_{\Pi'}$.*

Again note that whenever Π saturates F, then also Π' saturates F simply because $\equiv_{\Pi'}$ is a refinement of $\equiv_\Pi$. We are now ready to state the main minimization algorithm.

Algorithm 3 Minimization of deterministic wta

Require: deterministic and total wta $M = (Q, \Sigma, \mathcal{A}, \delta, c, \nu)$ without useless states
$(\Pi', \mathrm{sol}, D) \leftarrow \textsc{ComputeSoL}(M)$ // see Algorithm 1; complexity: $O(rm)$
2: **repeat**
$(\Pi, \mathrm{sol}, f, r) \leftarrow \textsc{Complete}(M, \Pi', \mathrm{sol}, D)$ // see Algorithm 2; complexity: $O(rn^3)$
4: $\Pi' \leftarrow \textsc{Refine}(M, \Pi, \mathrm{sol}, f, r, D)$ // see Definition 13; complexity: $O(rmn^2)$
until $\Pi' = \Pi$
6: **return** $M_{(\Pi, \mathrm{sol}, f, r)}$ // see Definition 9; complexity $O(rm)$

Theorem 15. *A minimal deterministic and total wta M' recognizing φ can be obtained in time $O(rmn^4)$.*

Proof. The loop in lines 2–5 in Algorithm 3 can be entered at most n times, which immediately yields the required time bound using Lemmata 7, 11, and 14. The initial partition Π' in line 1 saturates F and so does every subsequent partition. Moreover, by Lemmata 11 and 14, $\equiv$ is a refinement of every subsequent $\equiv_\Pi$ and $\equiv_{\Pi'}$ because $\equiv$ is a refinement of $\equiv_{\Pi'}$ in line 1 by Lemma 7. Consequently, every Π is a stage by Lemma 11, and if $\Pi = \Pi'$ in line 5, then $(\Pi, \mathrm{sol}, f, r)$ is a stable stage. By Theorem 10, the wta returned in line 6 is a minimal deterministic and total wta recognizing φ. □

5 A small example

Let us discuss the example of [21] (with one minor modification), which presents a simplistic wta for simple English sentences. It penalizes long sentences by decreasing their score. The score will be a real number and we will use $(\mathbb{R}, +, \cdot, 0, 1)$ as the underlying field. Our ranked alphabet is

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

of which σ is binary and all other symbols have rank 0. We abbreviate the multi-letter symbols by their first letter (e.g., Alice by just A). As states we have $Q = \{\mathrm{NN}, \mathrm{VB}, \mathrm{ADJ}, \mathrm{VP}, \mathrm{NP}, \mathrm{S}, \bot\}$ of which only S is final (with $\nu_S = 1$). Transitions and transition weights are given as follows:

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

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

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 completely specified our deterministic and total input wta 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

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

Next, we call $\textsc{Complete}(M, \Pi', \mathrm{sol}, \{\bot\})$, which may return the stage $(\Pi, \mathrm{sol}, f, r)$ where

- $\Pi = \{\{\mathrm{S}\}, \{\mathrm{VP}\}, \{\mathrm{NP}, \mathrm{NN}\}, \{\mathrm{ADJ}\}, \{\mathrm{VB}\}, \{\bot\}\}$;
- $f(x) = 1$ for all live states x; and
- $r(\{\mathrm{NP}, \mathrm{NN}\}) = \mathrm{NP}$ and $r(\{x\}) = x$ for all other live states x.

Finally, we refine this partition, but NP and NN will not be split. Thus, we construct the deterministic and total wta $M_{(\Pi, \mathrm{sol}, f, r)} = (\Pi, \Sigma, \mathbb{R}, \delta', c', \nu')$ with the final state $\{\mathrm{S}\}$ (with $\nu'_{\{S\}} = 1$). Transitions and transition weights are given as follows (we drop the parentheses from the singleton sets):

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

and

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

For all remaining combinations (x, y) we have $\delta'_\sigma(x, y) = \{\bot\}$ and $c'_\sigma(x, y) = 1$. Note that a different minimal deterministic wta was obtained in [21]; note that this different wta cannot be obtained by our algorithm (since all transitions not involving NP and NN are essentially kept).

Conclusion and open problems

We presented the first polynomial-time minimization algorithm for deterministic weighted tree automata over semifields. If we suppose that the semifield operations can be performed in constant time, then our algorithm runs in time $O(rmn^4)$. In fact, our algorithm works equally well for wta with final states (i.e., $\nu_q \in \{0, 1\}$ for every $q \in Q$) because it then returns a minimal equivalent wta with final states. This contrasts the situation encountered with the pushing strategy of [23, 24], which needs final weights in general.

Finally, let us mention some open problems. Can a Hopcroft-like strategy [26] improve the presented algorithm? A more detailed complexity analysis should be conducted to obtain a tighter bound on the time complexity of the algorithm. Can minimization be performed in a similar manner as presented in [23, 24] for deterministic weighted string automata? This might lead to an algorithm that outperforms our algorithm. Finally, the theoretical foundations for minimization of (even nondeterministic) weighted tree automata over fields have been laid in [1, 17], but to the author's knowledge a polynomial-time minimization algorithm is still missing.

References

1. Bozapalidis, S., Louscou-Bozapalidou, O.: The rank of a formal tree power series. Theoret. Comput. Sci. **27**(1–2) (1983) 211–215
2. Bozapalidis, S.: Equational elements in additive algebras. Theory Comput. Systems **32**(1) (1999) 1–33
3. Kuich, W.: Formal power series over trees. In: Proc. 3rd Int. Conf. Developments in Language Theory, Aristotle University of Thessaloniki (1998) 61–101
4. Borchardt, B., Vogler, H.: Determinization of finite state weighted tree automata. J. Autom. Lang. Combin. **8**(3) (2003) 417–463
5. Kuich, W., Salomaa, A.: Semirings, Automata, Languages. Volume 5 of Monographs in Theoretical Computer Science. An EATCS Series. Springer (1986)
6. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984)
7. Gécseg, F., Steinby, M.: Tree languages. In: Handbook of Formal Languages. Volume 3. Springer (1997) 1–68
8. Graehl, J.: Carmel finite-state toolkit. ISI/USC, `http://www.isi.edu/licensed-sw/carmel`. (1997)
9. Frishert, M., Cleophas, L.G., Watson, B.W.: Fire station: An environment for manipulating finite automata and regular expression views. In: Proc. 9th Int. Conf. Implementation and Application of Automata. Volume 3317 of LNCS., Springer (2004) 125–133
10. Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., Mohri, M.: OpenFst — a general and efficient weighted finite-state transducer library. In: Proc. 12th Int. Conf. Implementation and Application of Automata. Volume 4783 of LNCS., Springer (2007) 11–23
11. Knight, K., Graehl, J.: An overview of probabilistic tree transducers for natural language processing. In: Proc. 6th Int. Conf. Computer Linguistics and Intelligent Text Processing. Volume 3406 of LNCS., Springer (2005) 1–24
12. May, J., Knight, K.: Tiburon: A weighted tree automata toolkit. In: Proc. 11th Int. Conf. Implementation and Application of Automata. Volume 4094 of LNCS., Springer (2006) 102–113
13. Borchardt, B.: The Theory of Recognizable Tree Series. PhD thesis, Technische Universität Dresden (2005)
14. May, J., Knight, K.: A better n-best list: Practical determinization of weighted finite tree automata. In: Proc. North American Chapter of the Association for Computational Linguistics. (2006) 351–358
15. Borchardt, B.: The Myhill-Nerode theorem for recognizable tree series. In: Proc. 7th Int. Conf. Developments in Language Theory. Volume 2710 of LNCS., Springer (2003) 146–158
16. Borchardt, B.: A pumping lemma and decidability problems for recognizable tree series. Acta Cybernet. **16**(4) (2004) 509–544
17. Bozapalidis, S.: Effective construction of the syntactic algebra of a recognizable series on trees. Acta Inform. **28**(4) (1991) 351–363
18. Angluin, D.: Learning regular sets from queries and counterexamples. Inform. and Comput. **75**(2) (1987) 87–106
19. Habrard, A., Oncina, J.: Learning multiplicity tree automata. In: Proc. 8th Int. Colloquium Grammatical Inference. Volume 4201 of LNAI., Springer (2006) 268–280
20. Drewes, F., Vogler, H.: Learning deterministically recognizable tree series. J. Autom. Lang. Combin. (2007) to appear.
21. Maletti, A.: Learning deterministically recognizable tree series — revisited. In: Proc. 2nd Int. Conf. Algebraic Informatics. Volume 4728 of LNCS., Springer (2007) 218–235
22. Seidl, H.: Deciding equivalence of finite tree automata. SIAM J. Comput. **19**(3) (1990) 424–437
23. Mohri, M.: Minimization algorithms for sequential transducers. Theoret. Comput. Sci. **234**(1–2) (2000) 177–201
24. Eisner, J.: 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) 64–71
25. Comon-Lundh, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata—techniques and applications. see: `http://tata.gforge.inria.fr/` (2007)
26. Hopcroft, J.E.: An $n \log n$ algorithm for minimizing states in a finite automaton. In: Theory of Machines and Computations. Academic Press (1971) 189–196