Pushing for weighted tree automata

Andreas Maletti* and Daniel Quernheim*

Universität Stuttgart, Institut für Maschinelle Sprachverarbeitung
Azenbergstraße 12, 70174 Stuttgart, Germany
{Andreas.Maletti,Daniel.Quernheim}@ims.uni-stuttgart.de

Abstract. Explicit pushing for weighted tree automata over semifields is introduced. A careful selection of the pushing weights allows a normalization of bottom-up deterministic weighted tree automata. Automata in the obtained normal form can be minimized by a simple transformation into an unweighted automaton followed by unweighted minimization. This generalizes results of Mohri and Eisner for deterministic weighted string automata to the tree case. Moreover, the new strategy can also be used to test equivalence of two bottom-up deterministic weighted tree automata M_1 and M_2 in time $O(|M| \log |Q|)$, where $|M| = |M_1| + |M_2|$ and $|Q|$ is the sum of the number of states of M_1 and M_2. This improves the previously best running time $O(|M_1| \cdot |M_2|)$.

1 Introduction

Automata theory is a main branch of theoretical computer science with successful applications in many diverse fields such as natural language processing, system verification, and pattern recognition. Recently, renewed interest in tree automata was sparked by applications in natural language processing and XML processing. These applications require efficient algorithms for basic manipulations of tree automata such as determinization [7], inference [20], and minimization [17, 16].

In natural language processing, weighted devices are often used to model probabilities, cost functions, or other features. In this contribution, we consider pushing [21, 10] for weighted tree automata [1, 11] over commutative semifields [15, 14]. Roughly speaking, pushing moves transition weights along a path. If the weights are properly selected, then pushing can be used to canonicalize a (bottom-up) deterministic weighted tree automaton [3]. The obtained canonical representation has the benefit that it can be minimized using unweighted minimization, in which the weight is treated as a transition label. This strategy has successfully been employed in [21, 10] for deterministic weighted (finite-state) string automata, and we adapt it here for tree automata. In particular, we improve the currently best minimization algorithm [19] for deterministic weighted tree automata from $O(|M| \cdot |Q|)$ to $O(|M| \log |Q|)$, which coincides with the complexity of minimization in the unweighted case [17].

* Both authors were supported by the German Research Foundation (DFG) grant MA/4959/1-1.

The improvement is achieved by a careful selection of signs of life [19]. Intuitively, a sign of life for a state q is a context which takes q into a final state. In particular, equivalent states will receive the same sign of life, which ensures that their pushing weights are determined using the same evaluation context. This property sets our algorithm apart from the similar algorithm in [19, Algorithm 1] and allows a proper canonicalization. After the pushing weights are determined we perform pushing, which we define for general (potentially nondeterministic) weighted tree automata. We prove that the semantics is preserved and that equivalent states have equally weighted corresponding transitions after pushing, which allows us to reduce minimization to the unweighted case [17].

Secondly, we apply pushing to equivalence testing. The currently fastest algorithm [9] for checking equivalence of two deterministic weighted tree automata M_1 and M_2 runs in time $O(|M_1| \cdot |M_2|)$. Our algorithm that computes signs of life can also handle states in different automata with the help of a particular sum construction. The pushing weight (and the evaluation context) is determined carefully, so that equivalent states in different automata receive the same sign of life. This allows us to minimize both input automata and then only test the corresponding unweighted automata for isomorphism. This approach reduces the run-time complexity to $O(|M| \log|Q|)$, where $|M| = |M_1| + |M_2|$ and $|Q| = |Q_1| + |Q_2|$ is the number of total states.

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 $k \in \mathbb{N}$ and Q a set. We write Q^k for the k-fold Cartesian product of Q, and the empty tuple $() \in Q^0$ is displayed as ε. An alphabet is a finite, nonempty set of symbols. A ranked alphabet (Σ, rk) consists of 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 Σ_k of k-ary symbols of Σ is $\Sigma_k = \{\sigma \in \Sigma \mid \mathrm{rk}(\sigma) = k\}$. We let $\Sigma(Q) = \{\sigma(w) \mid \sigma \in \Sigma_k, w \in Q^k\}$. To improve the readability, we often write $\sigma(q_1, \ldots, q_k)$ instead of $\sigma(q_1 \cdots q_k)$, where $\sigma \in \Sigma_k$ and $q_1, \ldots, q_k \in Q$. 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 $\Sigma(T_\Sigma(Q)) \subseteq T_\Sigma(Q)$. We write T_Σ for $T_\Sigma(\emptyset)$. The size $|t|$ of a tree t is inductively defined by $|q| = 1$ for every $q \in Q$ and $|\sigma(t_1, \ldots, t_k)| = 1 + \sum_{i=1}^{k} |t_i|$ for every $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(Q)$.

We reserve the use of a special symbol $\Box \notin Q$ that is not an element in any considered alphabet. 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 $\sigma(t_1, \ldots, t_{i-1}, C, t_{i+1}, \ldots, t_k) \in C_\Sigma(Q)$ for every $\sigma \in \Sigma_k$ with $k \geq 1$, index $i \in [1, k]$, $t_1, \ldots, t_k \in T_\Sigma(Q)$, and $C \in C_\Sigma(Q)$. We write C_Σ for $C_\Sigma(\emptyset)$. Note that $C_\Sigma(Q) \subseteq T_\Sigma(Q \cup \{\Box\})$. Let $C \in C_\Sigma(Q)$ and $t \in T_\Sigma(Q)$. Then $C[t]$ is the tree obtained from C by replacing $\Box$ by t.

A (commutative) semifield [15, 14] is a tuple $(A, +, \cdot, 0, 1)$ such that $(A, +, 0)$ and $(A, \cdot, 1)$ are commutative monoids, of which $(A \setminus \{0\}, \cdot, 1)$ is a group; $a \cdot 0 = 0$ for every $a \in A$; and $\cdot$ distributes over $+$. The multiplicative inverse of $a \in A \setminus \{0\}$ is denoted by a^{-1}; i.e., $a \cdot a^{-1} = 1$ for every $a \in A \setminus \{0\}$.

Let $(A, +, \cdot, 0, 1)$ be an arbitrary commutative semifield.

A weighted (finite-state) tree automaton [6, 5, 18, 4, 3] (for short: wta) is a tuple $M = (Q, \Sigma, \mu, F)$ such that (i) Q is an alphabet of states; (ii) Σ is a ranked alphabet; (iii) $\mu\colon \Sigma(Q) \times Q \to A$ is a transition weight mapping; and (iv) $F \subseteq Q$ is a set of final states. Note that the restriction to final states instead of final weights does not restrict the expressive power [3, Lemma 6.1.4]. We often write $t \to q$ for (t, q). The size $|M|$ of M is $|M| = \sum_{(t\to q)\in\mu^{-1}(A\setminus\{0\})}(|t| + 1)$. We extend μ to $h_\mu\colon T_\Sigma(Q) \times Q \to A$ by $h_\mu(q \to q) = 1$, $h_\mu(p \to q) = 0$, and

$$h_\mu(\sigma(t_1, \ldots, t_k) \to q) = \sum_{q_1,\ldots,q_k\in Q} \mu(\sigma(q_1, \ldots, q_k) \to q) \cdot \prod_{i=1}^{k} h_\mu(t_i \to q_i)$$

for all $p, q \in Q$ with $p \neq q$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma(Q)$. The wta M recognizes the weighted language $M\colon T_\Sigma \to A$ such that $M(t) = \sum_{q\in F} h_\mu(t \to q)$ for every $t \in T_\Sigma$. Two wta M and M' are equivalent if $M = M'$.

The wta $M = (Q, \Sigma, \mu, F)$ is *(bottom-up total) deterministic* (or a dwta) if for every $t \in \Sigma(Q)$ there exists exactly one $q \in Q$ such that $\mu(t \to q) \neq 0$. For dwta we prefer the presentation $(Q, \Sigma, \delta, c, F)$ with $\delta\colon \Sigma(Q) \to Q$ and $c\colon \Sigma(Q) \to A \setminus \{0\}$, which are defined such that $\delta(t) = q$ and $c(t) = \mu(t \to q)$ for every $t \in \Sigma(Q)$, where q is the unique state such that $\mu(t \to q) \neq 0$. The mappings δ and c are extended to $\underline{\delta}\colon T_\Sigma(Q) \to Q$ and $\underline{c}\colon T_\Sigma(Q) \to A \setminus \{0\}$ by

$$\underline{\delta}(q) = q \qquad \underline{\delta}(\sigma(t_1, \ldots, t_k)) = \delta(\sigma(\underline{\delta}(t_1), \ldots, \underline{\delta}(t_k)))$$

$$\underline{c}(q) = 1 \qquad \underline{c}(\sigma(t_1, \ldots, t_k)) = c(\sigma(\underline{\delta}(t_1), \ldots, \underline{\delta}(t_k))) \cdot \prod_{i=1}^{k} \underline{c}(t_i)$$

for every $q \in Q$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma(Q)$. For every $t \in T_\Sigma$ we can observe that $M(t) = \underline{c}(t)$ if $\underline{\delta}(t) \in F$ and $M(t) = 0$ otherwise.

An equivalence relation $\equiv$ on Q is a reflexive, symmetric, and transitive subset $\equiv \subseteq Q^2$. The equivalence class (or block) of $q \in Q$ is $[q]_\equiv = \{p \in Q \mid p \equiv q\}$, and $(P/\equiv) = \{[p]_\equiv \mid p \in P\}$ for every $P \subseteq Q$. Whenever $\equiv$ is obvious from the context, we simply omit it. The equivalence $\equiv$ respects finality if $[q] \subseteq F$ or $[q] \subseteq Q \setminus F$ for all $q \in Q$ (i.e., all states of a block are either final or nonfinal).

Suppose that M is deterministic. Let $\equiv_M \subseteq Q^2$ be the MYHILL-NERODE equivalence relation [2]

$$\equiv_M = \{(q, p) \in Q^2 \mid \exists a \in A \setminus \{0\}, \forall C \in C_\Sigma(Q)\colon \underline{c}(C[q]) = a \cdot \underline{c}(C[p])\} \ .$$

If M is clear from the context, then we just write $\equiv$ instead of $\equiv_M$. The deterministic (unweighted) tree automaton (dta) [12, 13, 8] for M is $M' = (Q, \Sigma, \delta, F)$. It recognizes the tree language $L(M') \subseteq T_\Sigma$, which is $\{t \in T_\Sigma \mid \underline{\delta}(t) \in F\}$.

Let $M_1 = (Q_1, \Sigma, \delta_1, c_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, c_2, F_2)$ be two arbitrary dwta. A *congruential relation* $\sim \subseteq Q_1 \times Q_2$ (between M_1 and M_2) is such that $\delta_1(\sigma(q_1, \ldots, q_k)) \sim \delta_2(\sigma(p_1, \ldots, p_k))$ for every $\sigma \in \Sigma_k$, $q_1, \ldots, q_k \in Q_1$, and $p_1, \ldots, p_k \in Q_2$ such that $q_i \sim p_i$ for every $i \in [1, k]$. Note that this definition

of a congruential relation completely disregards the weights. It coincides with the classical notion of *congruence* if $M_1 = M_2$. The equivalence $\equiv_M$ is a congruence [2]. If $Q_1 \cap Q_2 = \emptyset$, then a congruential relation $\sim$ extends to a congruential equivalence $\cong \subseteq (Q_1 \cup Q_2)^2$ such that $\cong = (\sim \cup \sim^{-1})^*$.

Finally, let us introduce a particular sum of 2 dwta, which preserves determinism. Without loss of generality, let $Q_1 \cap Q_2 = \emptyset$. The dwta $M_1 \uplus M_2$ is $(Q_1 \cup Q_2, \Sigma, \delta, c, F_1 \cup F_2)$, where $\delta(t_1) = \delta_1(t_1)$, $\delta(t_2) = \delta_2(t_2)$, $c(t_1) = c_1(t_1)$, and $c(t_2) = c_2(t_2)$ for every $t_1 \in \Sigma'(Q_1)$ and $t_2 \in \Sigma'(Q_2)$ with $\Sigma' = \Sigma \setminus \Sigma_0$. As usual, we assume that unspecified transitions go to a nonfinal sink state. Clearly, $M_1 \uplus M_2$ does not compute the sum of M_1 and M_2 because it is missing all transitions for nullary symbols. Outside of nullary symbols (the initial steps) it behaves just like the standard sum [3] of M_1 and M_2.

3 Efficient computation of signs of life

In this section, we show how to efficiently compute signs of life (see Def. 1), which are evidence that a final state can be reached, and weights for pushing (see Sect. 4). Our algorithm (see Alg. 1) is similar to [19, Alg. 1], but we guarantee that equivalent states receive the same sign of life. This last property will prove to be essential in Sects. 5 and 6. Since we also want to use the computed signs of life for equivalence testing (see Sect. 6), we potentially work with a sum $M_1 \uplus M_2$ of two dwta M_1 and M_2 here, where the transitions for nullary symbols have been removed in order to preserve determinism.

> *Let $M = (Q, \Sigma, \delta, c, F)$ be a dwta with $Q = Q_1 \cup Q_2$ and $Q_1 \cap Q_2 = \emptyset$, and let $g \colon Q_1 \to Q_2$.*

We extend g to a mapping $\underline{g} \colon T_\Sigma(Q_1) \to T_\Sigma(Q_2)$ such that $\underline{g}(q_1) = g(q_1)$ for every $q_1 \in Q_1$ and $\underline{g}(\sigma(t_1, \ldots, t_k)) = \sigma(\underline{g}(t_1), \ldots, \underline{g}(t_k))$ for every $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(Q_1)$. Let us first recall the definition of a sign of life from [19], which we adjust here to handle the potential sum.

Definition 1 (see [19, Sect. 2]). *Let $C \in C_\Sigma(Q_1)$, $q_1 \in Q_1$, and $q_2 \in Q_2$. Then C is a* sign of life *for q_1 if $\underline{\delta}(C[q_1]) \in F$. It is a* sign of life *for q_2 if $\underline{\delta}(\underline{g}(C)[q_2]) \in F$. Any state that has a sign of life is* live*, and any state without a sign of life is* dead*.*

Next, we explain Alg. 1 briefly. Let M_1 be the dwta obtained by restricting M to Q_1. First we realize that every final state $q \in F$ is trivially live. We set the sign of life for its block $[q]$ to the trivial context $\Box$ (line 3) and its weight to 1 (line 4). Since the congruence $\cong$ respects finality, the block $[q]$ contains only final states. Overall, this initialization takes time $O(|Q|)$. Next, we add all transitions leading to a final state of $F \cap Q_1$ to the FIFO queue T, which takes time $O(|M_1|)$. Clearly each transition using Q_1 can be added at most once to T, so the 'while-loop' executes at most once per transition. In the loop, we inspect the transition $\tau = \sigma(q_1, \ldots, q_k)$ that we took from T. We check each

Alg. 1 ComputeSoL: Compute signs of life and their weight.

Require: dwta $M = (Q, \Sigma, \delta, c, F)$, $g\colon Q_1 \to Q_2$ with $Q = Q_1 \cup Q_2$ and $Q_1 \cap Q_2 = \emptyset$, and congruential equivalence $\cong$ such that $g \subseteq \cong$ and $\cong$ respects finality

Ensure: live states $L \subseteq Q$, $\mathrm{sol}\colon (L/\cong) \to C_\Sigma(Q_1)$, and $\lambda\colon L \to A \setminus \{0\}$ such that $\lambda(q_1) = \underline{c}(\mathrm{sol}(B)[q_1])$ for all $q_1 \in L \cap Q_1$ and $\lambda(q_2) = \underline{c}(\underline{g}(\mathrm{sol}(B))[q_2])$ for all $q_2 \in L \cap Q_2$, where $B = [q_1]_\cong$

```
    L ← F                                                    // all final states are live
 2: for all q ∈ F do
      sol([q]_≅) ← □                  // sign of life is the trivial context for final states...
 4:   λ(q) ← 1                                                // ... with trivial weight
    T ← new FifoQueue
 6: Append(T, {τ ∈ Σ(Q_1) | δ(τ) ∈ F ∩ Q_1}) // add all transitions leading to F ∩ Q_1
    while T is not empty do
 8:   τ ← RemoveHead(T)                                       // get first transition
      let τ = σ(q_1, ..., q_k) with σ ∈ Σ_k and q_1, ..., q_k ∈ Q_1
10:   for all i ∈ [1, k] such that q_i ∉ L do
        let C = σ(q_1, ..., q_{i-1}, □, q_{i+1}, ..., q_k)          // prepare context
12:     L ← L ∪ [q_i]_≅                    // all equivalent states are live; add to L
        sol([q_i]_≅) ← sol([δ(τ)]_≅)[C]   // add transition to sign of life of target state
14:     for all q ∈ [q_i]_≅ do
          if q ∈ Q_1 then
16:         λ(q) ← λ(δ(C[q])) · c(C[q])                          // set new weight
            Append(T, {τ' ∈ Σ(Q_1) | δ(τ') = q})   // add transitions leading to q
18:       else
            λ(q) ← λ(δ(g(C)[q])) · c(g(C)[q])                    // set new weight
20: return (L, sol, λ)
```

source state $q_i \in Q_1$ (with $i \in [1, k]$) whether it has been explored before. If not, then its whole block $[q_i]$ is unexplored, since we explore the states by blocks. Overall, we thus perform at most $|M_1|$ checks. Once we discover an unexplored state q_i (i.e., a state not yet marked as live), we mark its whole block $[q_i]$ as explored (and live). In addition, we set the block's sign of life to the sign of life of the target state's block $[\delta(\tau)]$ extended by the context C created from the current transition τ (line 13). Finally, for each state q in the current block $[q_i]$ we compute the weight of the sign of life by plugging q into the current transition instead of q_i. If $q \in Q_2$, then we adjust the transition using $\underline{g}$. In this way, we obtain a transition weight a and a target state p. Since the weight of the sign of life for p has already been computed (because otherwise the transition τ would not have been in the queue T), we simply set $\lambda(q) = \lambda(p) \cdot a$ (line 16 and 19). Clearly, this is done at most once for each state, so we obtain a total complexity (counted over all loops) of $O(|Q|)$ for this part. We complete the iteration by adding all transitions to newly explored states of Q_1 to T. Overall, we obtain the complexity $O(|M_1| + |Q|)$.

Theorem 2. *Algorithm 1 is correct and runs in time $O(|M_1| + |Q|)$.*

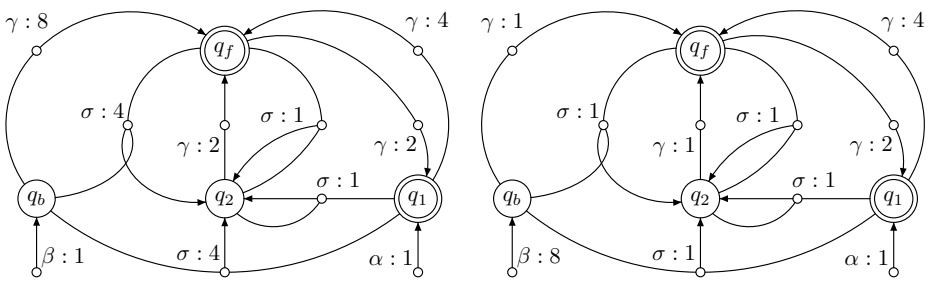

Fig. 1. Dwta over the Real numbers before (left) and after (right) pushing.

Proof. We already argued the run-time complexity, so let us prove the post-condition. For $q \in F$, we have $\lambda(q) = 1 = \underline{c}(q) = \underline{c}(\Box[q])$ by lines 3–4, which proves the post-condition because $\mathrm{sol}([q]_{\cong}) = \Box$. Let $C' = C$ if $q \in Q_1$, and otherwise let $C' = \underline{g}(C)$. In the main loop, we set $\lambda(q) = \lambda(\delta(C'[q])) \cdot c(C'[q])$ in line 16 or 19. Since $q' = \delta(C'[q])$ has already been explored in a previous iteration, we have $\lambda(q') = \underline{c}(C''[q'])$ by the induction hypothesis, where $C'' = \mathrm{sol}([q']_{\cong})$ if $q' \in Q_1$ and $C'' = \underline{g}(\mathrm{sol}([q']_{\cong}))$. Consequently,

$$\lambda(q) = \underline{c}(C''[q']) \cdot c(C'[q]) = \underline{c}(C''[C'[q]]) = \underline{c}((C''[C'])[q]) \ ,$$

which proves the post-condition because $\mathrm{sol}([q]_{\cong}) = \mathrm{sol}([\delta(C[q])]_{\cong})[C]$ by line 13. Since $\lambda(q) \neq 0$, it also proves that $\mathrm{sol}([q]_{\cong})$ is a sign of life for q and that q is live. The proof that all states $q \notin L$ are indeed dead is simple and omitted here. □

Example 3. An example dwta is depicted in Fig. 1 (left). For any transition (small circles), the arrow leads to the target state and the source states have been arranged in a counter-clockwise fashion (starting from the target arrow). As usual, final states are indicated by double-circles. Let us note that the coarsest congruence $\cong$ that respects finality is $\{\{q_1, q_f\}, \{q_2, q_b\}\}$. We use this partition together with $g = \emptyset$ in Alg. 1. First, we mark all final states $\{q_1, q_f\}$ as live. Their block is assigned the trivial context $\Box$ and each final state is assigned the trivial weight 1. We then initialize the FIFO queue with the transitions $\{\gamma(q_b), \gamma(q_2), \gamma(q_1), \gamma(q_f)\}$ leading to q_1 or q_f. Let us pick the first transition $\gamma(q_b)$ from the queue. Since q_b has not yet been marked as live, we consider all transitions $\gamma(\Box)[q] = \gamma(q)$ where $q \in [q_b]_{\cong} = \{q_b, q_2\}$. The sign of life for $[q_b]_{\cong}$ is $\gamma(\Box)$, and the corresponding weights $\lambda(q_b)$ and $\lambda(q_2)$ are $\lambda(q_b) = \lambda(q_f) \cdot 2 = 2$ and $\lambda(q_2) = \lambda(q_f) \cdot 8 = 8$, respectively. For all remaining transitions, all source states are already live. Consequently, we have computed all signs of life and the pushing weights $\lambda(q_1) = \lambda(q_f) = 1$, $\lambda(q_2) = 2$, and $\lambda(q_b) = 8$.

4 Pushing

Recall that the Myhill-Nerode congruence states that there is a unique scaling factor for every pair (p, q) of equivalent states. Thus, any fixed sign of life

can be used to determine the scaling factor between p and q. In the previous section, we computed a sign of life $\mathrm{sol}(q)$ for each live state $q \in L$ as well as the weight $\lambda(q)$ of $\mathrm{sol}(q)$. Now, we will use these weights to normalize the wta by *pushing* [21, 10, 22]. Intuitively, pushing cancels the scaling factor for equivalent states. In weighted (finite-state) string automata, pushing is performed from the final states towards the initial states. Since we work with bottom-up wta [3] (i.e., our notion of determinism is bottom-up), this works analogously here by moving weights from the root towards the leaves.

In this section we work with an arbitrary wta $M = (Q, \Sigma, \mu, F)$ and an arbitrary mapping $\lambda\colon Q \to A \setminus \{0\}$ such that $\lambda(q) = 1$ for every $q \in F$.[1]

Definition 4 (see [21, p. 296]). *The* pushed *wta* $\mathrm{push}_\lambda(M)$ *is* (Q, Σ, μ', F) *such that* $\mu'(\sigma(q_1, \ldots, q_k) \to q) = \lambda(q) \cdot \mu(\sigma(q_1, \ldots, q_k) \to q) \cdot \prod_{i=1}^{k} \lambda(q_i)^{-1}$ *for every* $\sigma \in \Sigma_k$ *and* $q, q_1, \ldots, q_k \in Q$.

The mapping λ records the pushed weights. In plain words, every transition leading to a state $q \in Q$ charges the additional weight $\lambda(q)$, and every transition leaving the state q compensates this by charging the weight $\lambda(q)^{-1}$. Next, let us show that M and $\mathrm{push}_\lambda(M)$ are equivalent.

Proposition 5 (see [21, Lm. 4]). *The wta M and $\mathrm{push}_\lambda(M)$ are equivalent. Moreover, if M is deterministic, then so is $\mathrm{push}_\lambda(M)$.*

Proof. Let $\mathrm{push}_\lambda(M) = (Q, \Sigma, \mu', F)$. The preservation of determinism is obvious. We prove that $h_{\mu'}(t \to q) = \lambda(q) \cdot h_\mu(t \to q)$ for every $t \in T_\Sigma$ and $q \in Q$ by induction. Let $t = \sigma(t_1, \ldots, t_k)$ for some $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$. We have $h_{\mu'}(t_i \to q_i) = \lambda(q_i) \cdot h_\mu(t_i \to q_i)$ for every $i \in [1, k]$ and $q_i \in Q$ by the induction hypothesis. Consequently,

$$
\begin{aligned}
& h_{\mu'}(t \to q) \\
&= \sum_{q_1, \ldots, q_k \in Q} \mu'(\sigma(q_1, \ldots, q_k) \to q) \cdot \prod_{i=1}^{k} h_{\mu'}(t_i \to q_i) \\
&= \sum_{q_1, \ldots, q_k \in Q} \lambda(q) \cdot \mu(\sigma(q_1, \ldots, q_k) \to q) \cdot \prod_{i=1}^{k} \lambda(q_i)^{-1} \cdot \prod_{i=1}^{k} \Big(\lambda(q_i) \cdot h_\mu(t_i \to q_i)\Big) \\
&= \lambda(q) \cdot h_\mu(t \to q) \ .
\end{aligned}
$$

We complete the proof as follows.

$$\mathrm{push}_\lambda(M)(t) = \sum_{q \in F} h_{\mu'}(t \to q) = \sum_{q \in F} \lambda(q) \cdot h_\mu(t \to q) = \sum_{q \in F} h_\mu(t \to q) = M(t)$$

because $\lambda(q) = 1$ for every $q \in F$. □

[1] This additional requirement is necessary because our wta have final states [3, Section 4.1.3]. A model with final weights [3, Section 4.1.3] would be equally powerful, but could lift this restriction.

Alg. 2 Overall structure of our minimization algorithm.

Require: a dwta M
Ensure: return a minimal, equivalent dwta

```
   ≅ ← ComputeCoarsestCongruence(M)        // complexity: O(|M| log|Q|)
2: (L, sol, λ) ← ComputeSoL(M, ≅, ∅)       // complexity: O(|M|)
   M' ← push_λ(M)                          // complexity: O(|M|)
4: N ← Minimize(alph(M'), ≅)               // complexity: O(|M| log|Q|)
   return alph^{-1}(N)
```

Example 6 (cont'd). Let us return to our example dwta M in Fig. 1 (left) and perform pushing. The pushing weights λ are given in Ex. 3. For the transition $\tau = \sigma(q_b, q_f)$ we have $\delta(\tau) = q_2$ and $c(\tau) = 4$. In $\mathrm{push}_\lambda(M)$ we have the new weight $c'(\tau) = \lambda(q_2) \cdot c(\tau) \cdot \lambda(q_b)^{-1} \cdot \lambda(q_f)^{-1} = 2 \cdot 4 \cdot 8^{-1} \cdot 1^{-1} = 1$. The dwta $\mathrm{push}_\lambda(M)$ is presented in Fig. 1 (right). It is evident in $\mathrm{push}_\lambda(M)$ that q_2 and q_b are equivalent, whereas q_1 and q_f are not.

5 Minimization

We will now turn to the main application of weight pushing for dwta: efficient minimization. The overall structure is presented in Alg. 2. Note that the coarsest congruence for a dwta $M = (Q, \Sigma, \delta, c, F)$ that respects finality can be obtained by minimization [17] of the underlying unweighted automaton (Q, Σ, δ, F).

> *Let $M = (Q, \Sigma, \delta, c, F)$ be a dwta, $\lambda\colon Q \to A$ be the pushing weights computed in Alg. 1, and* $\mathrm{push}_\lambda(M) = (Q, \Sigma, \delta, c', F)$.

The dwta $\mathrm{push}_\lambda(M)$ has the property that $c'(\sigma(q_1, \ldots, q_k)) = c'(\sigma(p_1, \ldots, p_k))$ for all $\sigma \in \Sigma_k$ and states $q_1, \ldots, q_k, p_1, \ldots, p_k \in Q$ such that $q_i \equiv p_i$ for every $i \in [1, k]$. In analogy to the string case [10], this property allows us to treat the transition weight as part of the input symbol. In this way we obtain a dta, which we can minimize using, for example, the algorithm of [17]. After the minimization, we can expand the input symbol again into a symbol from Σ and the transition weight.

Definition 7. *Let $W = \{c(\tau) \mid \tau \in \Sigma(Q)\}$ be the set of occurring weights. The* syntactic dta *for M is* $\mathrm{alph}(M) = (Q, \Sigma \times W, \delta', F)$*, where*

- $(\Sigma \times W)_k = \Sigma_k \times W$ *for every $k \in \mathbb{N}$, and*
- $\delta'(\langle \sigma, w \rangle(q_1, \ldots, q_k)) = q$ *if and only if*

$$\delta(\sigma(q_1, \ldots, q_k)) = q \quad \textit{and} \quad c(\sigma(q_1, \ldots, q_k)) = w$$

for every $\sigma \in \Sigma_k$, $w \in W$, and $q_1, \ldots, q_k \in Q$. If no such $q \in Q$ exists, then $\delta'(\langle \sigma, w \rangle(q_1, \ldots, q_k))$ is undefined.

Note that a dta with the above structure can be turned back into a dwta. We will write alph^{-1} for this operation. Clearly, these constructions can be performed in time $O(|M|)$. To prove that the change to the unweighted setting is correct, we still have to prove that the involved congruences coincide. Let $\cong$ be the classical (state) equivalence for $\mathrm{alph}(\mathrm{push}(M))$, and let $\equiv$ be the (state) equivalence for M. To prove that they coincide, we show both inclusions.

Lemma 8. *The equivalence $\equiv$ is a congruence of* $\mathrm{alph}(\mathrm{push}(M))$ *that respects finality.*

Proof. Let $\mathrm{alph}(\mathrm{push}(M)) = (Q, \Sigma \times W, \delta', F)$ and $\mathrm{push}(M) = (Q, \Sigma, \delta, c', F)$. Since M and $\mathrm{alph}(\mathrm{push}(M))$ have the same final states, $\equiv$ respects finality. For the congruence property, let $\sigma \in \Sigma_k$ and $q_1, \ldots, q_k, p_1, \ldots, p_k \in Q$ be such that $q_i \equiv p_i$ for every $i \in [1, k]$. Then $\delta(\sigma(q_1, \ldots, q_k)) \equiv \delta(\sigma(p_1, \ldots, p_k))$ because $\equiv$ is a congruence for M. If $c'(\sigma(q_1, \ldots, q_k)) = w = c'(\sigma(p_1, \ldots, p_k))$, then

$$\begin{aligned}\delta'(\langle\sigma, w\rangle(q_1, \ldots, q_k)) &= \delta(\sigma(q_1, \ldots, q_k)) \\ &\equiv \delta(\sigma(p_1, \ldots, p_k)) = \delta'(\langle\sigma, w\rangle(p_1, \ldots, p_k)) \ .\end{aligned}$$

For the remaining combinations of $\langle\sigma, w'\rangle$ both transitions would be undefined (or go to the sink state), which would prove the congruence property. It remains to show that $c'(\sigma(q_1, \ldots, q_k)) = c'(\sigma(p_1, \ldots, p_k))$.

By Def. 4, we have

$$c'(\sigma(q_1, \ldots, q_k)) = \lambda(\delta(\sigma(q_1, \ldots, q_k))) \cdot c(\sigma(q_1, \ldots, q_k)) \cdot \prod_{i=1}^{k} \lambda(q_i)^{-1}$$

$$c'(\sigma(p_1, \ldots, p_k)) = \lambda(\delta(\sigma(p_1, \ldots, p_k))) \cdot c(\sigma(p_1, \ldots, p_k)) \cdot \prod_{i=1}^{k} \lambda(p_i)^{-1} \ .$$

Now we prove that

$$\begin{aligned}&\lambda(\delta(C_j[q_j])) \cdot c(C_j[q_j]) \cdot \prod_{i=1}^{j-1} \lambda(p_i)^{-1} \cdot \prod_{i=j}^{k} \lambda(q_i)^{-1} \\ &= \lambda(\delta(C_j[p_j])) \cdot c(C_j[p_j]) \cdot \prod_{i=1}^{j} \lambda(p_i)^{-1} \cdot \prod_{i=j+1}^{k} \lambda(q_i)^{-1}\end{aligned}$$

for every $j \in [1, k]$, where $C_j = \sigma(p_1, \ldots, p_{j-1}, \Box, q_{j+1}, \ldots, q_k)$. Let $q'_j = \delta(C_j[q_j])$ and $p'_j = \delta(C_j[p_j])$. Since $q_j \equiv p_j$, we obtain that $q'_j \equiv p'_j$ because $\equiv$ is a congruence. Consequently, $\mathrm{sol}(q'_j) = C = \mathrm{sol}(p'_j)$. Moreover, we have

$$\frac{\lambda(q_j)}{\lambda(p_j)} = \frac{c(C[C_j[q_j]])}{c(C[C_j[p_j]])} = \frac{c(C[q'_j]) \cdot c(C_j[q_j])}{c(C[p'_j]) \cdot c(C_j[p_j])} \quad \text{and} \quad \frac{\lambda(q'_j)}{\lambda(p'_j)} = \frac{c(C[q'_j])}{c(C[p'_j])} \ ,$$

where the former holds because $C[C_j]$ is a sign of life for both q_j and p_j and the latter holds by definition. With these equations, let us inspect the main equality.

$$\frac{\lambda(\delta(C_j[q_j])) \cdot c(C_j[q_j]) \cdot \prod_{i=1}^{j-1} \lambda(p_i)^{-1} \cdot \prod_{i=j}^{k} \lambda(q_i)^{-1}}{\lambda(\delta(C_j[p_j])) \cdot c(C_j[p_j]) \cdot \prod_{i=1}^{j} \lambda(p_i)^{-1} \cdot \prod_{i=j+1}^{k} \lambda(q_i)^{-1}}$$

$$= \frac{\lambda(q'_j) \cdot c(C_j[q_j]) \cdot \lambda(q_j)^{-1}}{\lambda(p'_j) \cdot c(C_j[p_j]) \cdot \lambda(p_j)^{-1}} = \frac{c(C[q'_j]) \cdot c(C_j[q_j]) \cdot \lambda(p_j)}{c(C[p'_j]) \cdot c(C_j[p_j]) \cdot \lambda(q_j)}$$
$$= \frac{c(C[q'_j]) \cdot c(C_j[q_j]) \cdot c(C[p'_j]) \cdot c(C_j[p_j])}{c(C[p'_j]) \cdot c(C_j[p_j]) \cdot c(C[q'_j]) \cdot c(C_j[q_j])} = 1$$

Repeated application (from $i = 1$ to k) yields the desired statement. □

Theorem 9. *We have* $\equiv = \cong$.

Proof. Lemma 8 shows that $\equiv$ is a congruence of alph(push(M)) that respects finality. Since $\cong$ is the coarsest congruence of alph(push(M)) that respects finality by [17], we obtain that $\equiv \subseteq \cong$. The converse is trivial to prove. □

The currently fastest dwta minimization algorithm is presented in [19]. It runs in time $O(|M| \cdot |Q|)$. With the help of pushing, we achieve the run-time of the fastest minimization algorithm in the unweighted case.

Corollary 10. *For every dwta* $M = (Q, \Sigma, \delta, c, F)$*, we can compute a minimal, equivalent dwta in time* $O(|M| \log |Q|)$.

6 Testing equivalence

In this section, we want to decide whether two given dwta are equivalent. To this end, let $M_1 = (Q_1, \Sigma, \delta_1, c_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, c_2, F_2)$ be dwta. The overall approach is presented in Alg. 3. First, we need to compute a correspondence between states. For every $q_1 \in Q_1$, we compute a tree $t \in \underline{\delta}_1^{-1}(q_1)$. If $\underline{\delta}_1^{-1}(q_1) = \emptyset$, then q_1 is not reachable and can be deleted. To avoid these details, let us assume that all states of Q_1 are reachable. In this case, we can compute an access tree $h(q_1) \in T_\Sigma$ for every state $q_1 \in Q_1$ in time $|M_1|$ using standard breadth-first search, where we unfold each state (i.e., explore all transitions leading to it) at most once. To keep the representation efficient, we store the access trees in the format $\Sigma(Q_1)$, where the states refer to their respective access trees. To obtain the correspondence g, we compute the corresponding state of Q_2 that is reached when processing the access trees. Formally, $g(q_1) = \underline{\delta}_2(h(q_1))$ for every $q_1 \in Q_1$. Consequently, we have that $h(q_1) \in \underline{\delta}_1^{-1}(q_1) \cap \underline{\delta}_2^{-1}(g(q_1))$ for every $q_1 \in Q_1$.

Next, we compute the coarsest congruence for the (reduced) sum $M_1 \uplus M_2$, where we intend to compute a congruential equivalence. This can be achieved by a simple modification of the standard minimization algorithms (for example [17]), in which we replace states $q_1 \in Q_1$ by their corresponding state $g(q_1)$, whenever we test a state of Q_2. For example, when splitting a block using the context $C = \sigma(q, \Box, q')$ with $q, q' \in Q_1$, we use this context for all states of Q_1 and the context $\underline{g}(C) = \sigma(g(q), \Box, g(q'))$ for all states of Q_2. If we find corresponding states that are not related by the congruence, then the dwta are obviously not equivalent because for corresponding states there exists a common tree t that leads M_1 and M_2 into the respective state. Since the states are related by the

Alg. 3 Overall structure of our equivalence test.

Require: dwta $M_1 = (Q_1, \Sigma, \delta_1, c_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, c_2, F_2)$
Ensure: return yes if and only if M_1 and M_2 are equivalent

$g \leftarrow \textsc{ComputeCorrespondence}(M_1, M_2)$ // complexity: $O(|M_1|)$
$M \leftarrow M_1 \uplus M_2$ // (reduced) sum of M_1 and M_2
$\cong \leftarrow \textsc{ComputeCoarsestCongruence}'(M, g)$ // complexity: $O(|M| \log |Q|)$
if $g \not\subseteq \cong$ **then**
  return no // g is not compatible with the coarsest congruence
$(L, \mathrm{sol}, \lambda) \leftarrow \textsc{ComputeSoL}(M, \cong, g)$ // complexity: $O(|M|)$
$\lambda_1 = \lambda|_{Q_1}$; $\lambda_2 = \lambda|_{Q_2}$ // prepare pushing weights
$M_1 \leftarrow \mathrm{push}_{\lambda_1}(M_1)$; $M_2 \leftarrow \mathrm{push}_{\lambda_2}(M_2)$ // complexity: $O(|M_1| + |M_2|)$
$N_1 \leftarrow \textsc{Minimize}(\mathrm{alph}(M_1), \cong|_{Q_1 \times Q_1})$ // complexity: $O(|M_1| \log |Q_1|)$
$N_2 \leftarrow \textsc{Minimize}(\mathrm{alph}(M_2), \cong|_{Q_2 \times Q_2})$ // complexity: $O(|M_2| \log |Q_2|)$
return $\textsc{Isomorphic?}(N_1, N_2)$

congruence, there exists a context C that is accepted in only one of the states. This yields a difference of acceptance on $C[t]$.

Next, we compute signs of life and pushing weights. It is again important that equivalent states (in $M_1 \uplus M_2$) receive the same sign of life. We minimize M_1 and M_2 using the method of Section 5 (i.e., we perform pushing followed by unweighted minimization). Finally, we test the obtained unweighted dta for isomorphism. An easy adaptation of the statement (and proof) of Lemma 8 can be used to show that $q_1 \in Q_1$ and $q_2 \in Q_2$ are equivalent in $\mathrm{alph}(\mathrm{push}_{\lambda_1}(M_1))$ and $\mathrm{alph}(\mathrm{push}_{\lambda_2}(M_2))$, respectively (see Algorithm 3), if and only if $q_1 \equiv_M q_2$. This proves the correctness of Algorithm 3, whose run-time should be compared to the previously (asymptotically) fastest equivalence test for dwta of [9], which runs in time $O(|M_1| \cdot |M_2|)$.

Theorem 11. *We can test equivalence of M_1 and M_2 in time $O(|M| \log |Q|)$, where $|M| = |M_1| + |M_2|$ and $|Q| = |Q_1| + |Q_2|$.*

Acknowledgements

The authors gratefully acknowledge the insight and suggestions provided by the reviewers.

References

1. Berstel, J., Reutenauer, C.: Recognizable formal power series on trees. Theoret. Comput. Sci. 18(2), 115–148 (1982)
2. Borchardt, B.: The Myhill-Nerode theorem for recognizable tree series. In: Proc. 7th Int. Conf. Developments in Language Theory. LNCS, vol. 2710, pp. 146–158. Springer (2003)
3. Borchardt, B.: The Theory of Recognizable Tree Series. Ph.D. thesis, TU Dresden (2005)

4. Borchardt, B., Vogler, H.: Determinization of finite state weighted tree automata. Journal of Automata, Languages and Combinatorics 8(3), 417–463 (2003)
5. Bozapalidis, S.: Equational elements in additive algebras. Theory Comput. Syst. 32(1), 1–33 (1999)
6. Bozapalidis, S., Louscou-Bozapalidou, O.: The rank of a formal tree power series. Theoret. Comput. Sci. 27(1–2), 211–215 (1983)
7. Büchse, M., May, J., Vogler, H.: Determinization of weighted tree automata using factorizations. J. Autom. Lang. Comb. 15(3–4) (2010)
8. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Löding, C., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications. Available on: `http://www.grappa.univ-lille3.fr/tata` (2007)
9. Drewes, F., Högberg, J., Maletti, A.: MAT learners for tree series — an abstract data type and two realizations. Acta Inform. 48(3), 165–189 (2011)
10. Eisner, J.: Simpler and more general minimization for weighted finite-state automata. In: Proc. Joint Meeting of the Human Language Technology Conference and the North American Chapter of the ACL. pp. 64–71. ACL (2003)
11. Fülöp, Z., Vogler, H.: Weighted tree automata and tree transducers. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata, chap. 9, pp. 313–403. EATCS Monographs in Theoretical Computer Science, Springer (2009)
12. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984)
13. Gécseg, F., Steinby, M.: Tree languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, chap. 1, pp. 1–68. Springer (1997)
14. Golan, J.S.: Semirings and their Applications. Kluwer Academic Publishers, Dordrecht (1999)
15. Hebisch, U., Weinert, H.: Semirings – Algebraic Theory and Applications in Computer Science. World Scientific, Singapore (1998)
16. Högberg, J., Maletti, A., May, J.: Bisimulation minimisation for weighted tree automata. In: Proc. 11th Int. Conf. Developments in Language Theory. LNCS, vol. 4588, pp. 229–241. Springer (2007)
17. Högberg, J., Maletti, A., May, J.: Backward and forward bisimulation minimization of tree automata. Theoret. Comput. Sci. 410(37), 3539–3552 (2009)
18. Kuich, W.: Formal power series over trees. In: Proc. 3rd Int. Conf. Developments in Language Theory. pp. 61–101. Aristotle University of Thessaloniki (1998)
19. Maletti, A.: Minimizing deterministic weighted tree automata. Inform. Comput. 207(11), 1284–1299 (2009)
20. May, J., Knight, K., Vogler, H.: Efficient inference through cascades of weighted tree transducers. In: Proc. 48th Annual Meeting of the ACL. pp. 1058–1066. ACL (2010)
21. Mohri, M.: Finite-state transducers in language and speech processing. Comput. Linguist. 23(2), 269–311 (1997)
22. Post, M., Gildea, D.: Weight pushing and binarization for fixed-grammar parsing. In: Proc. 11th Int. Workshop on Parsing Technologies. pp. 89–98. ACL (2009)