Theory Comput Syst (2018) 62:1203–1222
https://doi.org/10.1007/s00224-017-9815-4

From Tree Automata to String Automata Minimization

Younes Guellouma[1] · Hadda Cherroun[1] ·
Djelloul Ziadi[2] · Bruce W. Watson[3]

Published online: 27 September 2017

Abstract In this paper, we propose a reduction of the minimization problem for a bottom-up deterministic tree automaton (DFTA), making the latter a minimization of a string deterministic finite automaton (DFA). To achieve this purpose, we proceed first by the transformation of the tree automaton into a particular string automaton, followed by minimizing this string automaton. In addition, we show that for our transformation, the minimization of the resulting string automaton coincides with the minimization of the original tree automaton. Finally, we discuss the complexity of our proposal for different types of tree automata, namely: standard, acyclic, incremental, and incrementally constructed tree automata.

Keywords Tree automata · Tree languages · Finite automata · Minimization

✉ Younes Guellouma
guellouma@mail.lagh-univ.dz

Hadda Cherroun
hadda_cherroun@mail.lagh-univ.dz

Djelloul Ziadi
Djelloul.Ziadi@univ-rouen.fr

Bruce W. Watson
bruce@fastar.org

[1] Laboratoire LIM, Université Amar Telidji, Laghouat, Algérie

[2] Laboratoire LITIS - EA 4108, Normandie Université, Rouen, France

[3] FASTAR Group, Department of Information Science, Stellenbosch University, Stellenbosch, South Africa

1 Introduction

Tree automata constitute a powerful theoretical tool used in many fields including XML Schema [1, 2], natural language processing [3], verification techniques, and program analysis. However, there are very few general tools and toolkits providing for real experiments of tree automata approaches. To overcome this lack, we developed a framework allowing both: tree automata manipulation, and large data amounts representation as trees.

Minimization is considered as useful technique to compact the size of automata. In the literature, almost all the minimization techniques [4, 5] were inspired by string automata minimization which was studied for the first time by Huffman and Moore [6]. In their proposal, indistinguishable states are merged, based on finding the distinguishable pairs of states. Later, Hopcroft [7] has defined a new algorithm proceeding by refining the coarsest partition until no more refinements are possible.

Following the same steps, minimization techniques for tree automata have emerged. In early 1967, Brainerd [4] proposed the first DFTA minimization method that we call the *standard method*. Since then, several algorithms and implementations have been proposed including [5], Gésceg and Steinby [9], and Comon et al [10], all of them following the same Brainerd's approach.

Later on, Watson [11] designed the first *incremental minimization* algorithm for string automata. It is based on a recursive function that decides if two states are equivalent. Unlike classical techniques, the process can be halted at any time and produces an automaton that recognizes the same language as the departure one. That algorithm was subsequently refined by Watson and Daciuk [12]. This incremental algorithm constitutes the basics of many other techniques [13–15].

Next, Carrasco et al. [16] introduce the *incremental construction* of tree automata acceptors from positive examples. Hence, the minimal tree automaton is constructed incrementally by adding trees to a given tree automaton language and then maintain minimality. Notice that this work is an extension of a previous work [17] on strings.

As mentioned above, in the wish to develop a toolkit manipulating tree automata, our observation from minimization techniques studies is that there exists a degree of analogy with the string automata minimization.[1] The question is therefore: does there exist some transformations from a tree automaton (DFTA — a deterministic finite tree automaton) to a string one (DFA — a deterministic finite string automaton) while its minimization exactly matches the minimization of the original one? The DFA can then be minimized using one of the well-known techniques. Thereafter, the result will be transformed back to a DFTA which will be the wanted minimal tree automaton.

In fact, this idea is not completely novel. First, Carrasco et al. [18] define a "signature" to each state which represents the "behaviour" of a state in the minimization process. Next, Abdulla et al. [19] extended this notion to compute an equivalence relation between states called "upward bisimulation" in order to minimize nondeterministic finite tree automata (NFTA). They transform the computation of the equivalence relation to the resolution of a transition system similar to string

[1] Similar approaches are being taken by several other tree automata researchers.

automata. Therefore, the complexity of this minimization is $\mathcal{O}(rm\log(n))$ where r is the maximum rank of the alphabet, m is the size of the automaton and n the number of its states. This complexity can be mapped to a deterministic finite tree automation (DFTA) using Högberg et al. [20].

In this paper, we continue in this direction and we construct a string automaton that can fully replace the tree one for minimization purposes. This paper focuses on proving that the tree automata minimization can be done through string automata minimization techniques which are well studied and their different implementations are available like [21–23]. After the definition of an associated string automaton to a given tree automaton, and the proof that Myhill-Nerode congruence coincides in both automata, we show that for the deterministic minimization, the complexity is improved. Next, we show that the associated string automaton minimization coincides also with the acyclic and the incremental minimizations. Finally, we discuss the complexity for all above minimization classes and we show that some results are new and improved. Thus, it will be shown that the associated string automaton can fully replace the initial tree automaton in any minimization technique and it can ensure lower complexity for almost all the cases, allowing us to use existing DFA toolkits like OpenFst [21] instead of redeveloping the whole algorithm for trees.

The rest of this paper is organized as follows: Section 2 recalls some preliminaries on trees and their automata. Next, the standard minimization algorithm is given with a complexity discussion. Then, in Section 4, we detail the basics of our approach, the algorithms, and its complexity discussion for the deterministic case. Section 5 discusses the method impact in acyclic, incremental and incremental construction minimization techniques and reports their complexities. Some experimental issues are discussed in Section 6. Finally, Section 7 provides some concluding remarks together with the future directions of our work.

2 Preliminaries

A ranked alphabet is a pair $(\Sigma,\ Arity)$ where Σ is a finite set of symbols, whereas $Arity$ is a mapping $Arity : \Sigma \rightarrow \mathbb{N}$ where $\mathbb{N}$ is the set of nonnegative integers. The arity of a symbol $f \in \Sigma$ is noted $Arity(f)$, the subset of p-ary symbols of Σ is $\Sigma_p = \{f \in \Sigma \mid Arity(f) = p\}$. We use the notation $f,\ f(\,),\ f(\,,\,),\ \ldots,\ f(\,,\ldots,\,)$ respectively for constant, unary, binary,…, p-ary symbols. For the sake of brevity, we use just Σ to represent a ranked alphabet $(\Sigma,\ Arity)$. The set of *trees* or *terms* $T(\Sigma)$ over a ranked alphabet Σ is the smallest set satisfying $\Sigma_0 \subseteq T(\Sigma)$ and if $p \geqslant 1$, $f \in \Sigma_p$ and $t_1, t_2, \ldots, t_p \in T(\Sigma)$ then $f(t_1, t_2, \ldots, t_p) \in T(\Sigma)$. A *tree language* L is a subset of $T(\Sigma)$. The set $St(t)$ of subtrees of a tree $t = f(s_1, \ldots, s_n)$ is defined by $St(t) = \{t\} \cup \bigcup_{k=1}^{n} St(s_k)$. The tree $t(r \leftarrow s)$ is the tree in which every occurrence of r is substituted with an occurrence of s. We define a new substitution as follows: $t(r \Leftarrow s)$ is the set of all trees resulting from the substitution of every occurrence of the subtree $r \in St(t)$ by the tree s once.

Example 1 Let $t = f(a, g(b, a))$ be a tree. $t(a \leftarrow g(b)) = f(g(b), g(b, g(b)))$ but $t(a \Leftarrow g(b)) = \{f(g(b), g(b, a)), f(a, g(b, g(b)))\}$.

A bottom up finite tree automaton (FTA) over a ranked alphabet Σ is a tuple $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ where Q is a finite set of *states*, $Q_f \subseteq Q$ is the set of final states and $\Delta \subset \bigcup_{n\geq 0} \Sigma_n \times Q^{n+1}$, $n \in \mathbb{N}$ is a finite set of transitions. The size of a transition $\varrho = (f, q_1, \ldots, q_n, q)$, $f \in \Sigma, q, q_1, \ldots, q_n \in Q$ is $|\varrho| = n + 1$. Then the size of the automaton $\mathcal{A}$ is defined as $|\mathcal{A}| = \sum_{\varrho\in\Delta} |\varrho|$. C_q denotes the number of transitions producing q. From now, we consider deterministic FTA (DFTA).

The transition function d (we use d since δ is usually used for string automata transition relation) for a DFTA is:

$$d : \bigcup_{n\geq 0} \Sigma_n \times Q^n \rightarrow Q \tag{1}$$

$$d(f, q_1, \ldots, q_n) = q, (f, q_1, \ldots, q_n, q) \in \Delta \tag{2}$$

$\Gamma(q) = \{(f, q_1, \ldots, q, \ldots, q_n) \mid \exists 1 \leq i \leq n, q' \in Q : q_i = q, (f, q_1, \ldots, q_n, q') \in \Delta\}$ denotes the set of arguments extracted from transitions in which the state q appears but not as an output.

$occ_q((f, q_1, \ldots, q_n)) = \{i \mid q_i = q\}$ denotes the set of positions of the state q in the argument $(f, q_1, \ldots, q_n)$.

Let $\rho = (f, q_1, \ldots, q_n)$ be an argument, then $\rho(q :_i p) = (f, q_1, \ldots, q_{i-1}, p, q_{i+1}, \ldots, q_n)$ such that $q_i = q$ denotes the argument computed by substituting q by p in a precise place i in ρ.

In fact, some authors add a special state denoted $\perp$ to complete a tree automaton, this completion is used to define equivalence between states. Therefore, we use Γ to avoid the completion of DFTA and then to define state equivalence using only the existing transitions. In contrast to complete DFTA, arguments that have no output are not defined. This may be useful because automata are usually incomplete in practice.

For $t \in T(\Sigma)$, the output $m_{\mathcal{A}}(t)$ when $\mathcal{A}$ operates in Q is the state in Q recursively computed as:

$$m_{\mathcal{A}}(t) = \begin{cases} d(t) & \text{If } t \in \Sigma_0 \\ d(f, m_{\mathcal{A}}(t_1), m_{\mathcal{A}}(t_2), \ldots, m_{\mathcal{A}}(t_n)) & \text{If } t = f(t_1, t_2, \ldots, t_n) \in T(\Sigma) - \Sigma_0 \end{cases} \tag{3}$$

A tree t is accepted by $\mathcal{A}$ if and only if $m_{\mathcal{A}}(t) \in Q_f$.

The language accepted by $\mathcal{A}$ is: $L(\mathcal{A}) = \{t \in T(\Sigma) \mid m_{\mathcal{A}}(t) \in Q_f\}$.

In the same way the accepted language (down language) by a state q is defined as follows: $L^{\downarrow}(q) = \{t \in T(\Sigma) \mid m_{\mathcal{A}}(t) = q\}$.

The residual (top) language of a state q is defined as follows:

$$L^{\uparrow}(q) = \bigcup_{\substack{t\in T(\Sigma)\\ s\in L^{\downarrow}(q)\\ m_{\mathcal{A}}(t)\in Q_f}} t(s \Leftarrow \#)$$

Then, a state q is *accessible* if $L^{\downarrow}(q) \neq \emptyset$. Furthermore, a state q is *co-accessible* if there exists $t \in T(\Sigma \cup \{q\})$ such as $q \in St(t)$ and $m_{\mathcal{A}}(t) \in Q_f$. A state is *useless* if it is neither accessible nor co-accessible. Useless states and transitions using them can be safely removed from Q and Δ respectively without affecting $L(\mathcal{A})$. We can remove all useless states in $\mathcal{O}(|A|)$ [18]. Thus, we suppose throughout this paper that every tree automaton is free of useless states.

3 Tree automata minimization

Since this work focusses on deterministic minimization, this section presents the standard deterministic approach and gives the most adopted algorithm [10]. In fact, this standard algorithm is a "reincarnation" of the first DFTA minimization due to Brainerd [4] from which the standard DFTA minimization algorithms are derived. We note that every deterministic tree automaton can be minimized by computing state equivalence classes and then merging equivalent states.

Let $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ be a DFTA. We define over Q the following equivalence relation: $p \equiv q$ if and only if

1. $p \in Q_f \Leftrightarrow q \in Q_f$ and,
2. for all $\rho \in \Gamma(p), i \in occ_p(\rho) : \rho(p :_i q) \in \Gamma(q)$ and$d(\rho) \equiv d(\rho(p :_i q))$

Minimization for DFTA was first discussed in late 1960s by Brainerd [4], and standardised in [18]. It computes the equivalence relation $\equiv$ through successive approximations $(\equiv_j)_{j \geq 0}$:

1. $p \equiv_0 q$ if and only if $(p \in Q_f \Leftrightarrow q \in Q_f)$
2. $p \equiv_{j+1} q$ if and only if $p \equiv_j q$ and for all $\rho \in \Gamma(p), i \in occ_p(\rho) : \rho(p :_i q) \in \Gamma(q)$ and$d(\rho) \equiv_j d(\rho(p :_i q))$

The computation of the family $(\equiv_j)_{j \geq 0}$ can then be done by successive approximations until reaching the fixed point, that is, some natural number k such that $\equiv_k = \equiv_{k+1}$.

Lemma 1 *For $k \geq |Q| - 2$, we have $\equiv_{k+1} = \equiv_k$*

Note that there exist some implementations of the standard algorithm such as Carrasco et al. [18] which is quadratic.

Furthermore, there exist other tree minimization techniques like acyclic, incremental and incrementally constructed ones. In the following, we start first by introducing our transformations, and then we discuss the above mentioned techniques.

4 From DFTA to DFA

The main idea of our reduction is to create an associated string automaton to the FTA we want to minimize. Instead of minimizing the targeted FTA, we proceed by minimizing its associated FA. In this section, we show how to construct this FA and we prove some efficient properties. The minimization of this FA is left to the next section.

In the following, we consider the DFTA $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$. $\hat{r}$ is the maximum rank of the alphabet Σ.

This idea is not completely novel. First, Carrasco et al. [18] designed a “signature” that stores for each state the places it occurs and the first symbol.

Definition 1 Let $q \in Q$ be a state then

$$sig(q) = \{(f, k) \mid \exists (f, q_1, \ldots, q_n) \in \Delta, q_k = q, 1 \leq k < n\} \cup \begin{cases} \{(\#, 1)\} & \text{if } q \in Q_f \\ \emptyset & otherwise \end{cases} \tag{4}$$

Later on, Abdulla et al. [19] used another way to identify states behaviour in order to provide a NFTA minimization. They compute a composed bisimulation relation including downward and upward bisimulation relations. The authors reduce the computation of the upward bisimulation to the resolution of word finite transition systems.

Here, we recall just the part that interests us in the Abdulla et al. approach.

Definition 2 An environment of a state q_i is a tuple $((q_1, \ldots, q_{i-1}, \bullet, q_{i+1}, \ldots, q_n), f, q)$ obtained by replacing the state q_i by a special symbol $\bullet \notin Q$. The set of all environments is denoted $Env(\mathcal{A})$.

The authors transform the NFTA to be minimized to a transition systems $TS = (Q^\bullet, \Delta^\bullet)$ in order to compute an equivalence relation as follows:

– $Q^\bullet$ contains a state $q^\bullet$ for each state $q \in Q$ and a state $e^\bullet$ for each environment $e \in Env(\mathcal{A})$.
– $\Delta^\bullet$ is the smallest set satisfying that if $(f, q_1 \ldots, q_n, q) \in \Delta$ then $\Delta^\bullet$ contains both $q_i^\bullet \rightarrow e_i^\bullet$ and $e_i^\bullet \rightarrow q^\bullet$ where $e_i^\bullet$ is in the form $((f, q_1, \ldots, q_{i-1}, \bullet, q_{i+1}, \ldots, q_n), f, q)$.

We note that this transition system can be viewed as a Labeled Transition System (LTS) if the environments are seen as labels between two states. Nevertheless, regarding the definition, environments are introduced as “states”. If environments are considered as labels, the transition system can be seen as a NFA without initial state. The authors proved that for this transformation, the computation of the upward bisimulation can be done through computing a similar equivalence function on the word TS. This can be done using the Tarjan-Paige algorithm [24]. For DFTA, this transformation works using results of Högberg et al. [20]. They report that the upward bisimulation can compute the minimal DFTA.

Based on the previous works, we continue in this direction and we propose a transformation using a similar reduction to that proposed by Abdulla et al. Our proposal creates a valid string DFA to prove then that it can replace the DFTA to be minimized in any of minimization techniques. Then, we prove that there is no need to reimplement those algorithms anew. We show also that in term of complexity, this transformation gives the same complexity as the direct techniques in some cases, and lower complexity in other ones.

Indeed, our approach proceeds on two steps. First, we construct the equivalence relation $\sim$ defined on the states of a DFTA $\mathcal{A}$ and then we regroup states which are possibly equivalent according to the equivalence relation $\equiv$. We show that ($p \equiv q \Rightarrow p \sim q$).

Second, using the relation $\sim$ we construct a string automaton $M_{\mathcal{A}}$ using the same states as $\mathcal{A}$. Then we prove that two states in $M_{\mathcal{A}}$ are equivalent by the Nerode equivalence relation $\cong$ if and only if they are equivalent by the equivalence relation $\equiv$.

In the following definition, we associate to the transitions set Δ a string language L_Δ called "horizontal language". For each transition $\varrho \in \Delta$, we deduce a set of strings L_ϱ. The union of all these languages L_ϱ constitutes L_Δ. An equivalence relation $\sim$ is defined using L_Δ in which we keep for each state a list of strings from L_Δ instead of keeping its signature.

Definition 3 Let $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ be a DFTA. The horizontal language of Δ noted L_Δ is defined as follows:

$$L_\Delta = \bigcup_{\varrho \in \Delta} L_\varrho \tag{5}$$

$$L_\varrho = \bigcup_{i=1}^{n} q_i f q_1 \ldots q_{i-1} \bullet q_{i+1} \ldots q_n \tag{6}$$

where $\varrho = (f, q_1, \ldots, q_n, q)$ and $\bullet \notin \Sigma_0 \cup Q$ is a special symbol.

Definition 4 Let $p, q \in Q$. We say that p and q are possibly equivalent (we note $p \sim q$), if and only if, $(p \in Q_f) \Leftrightarrow (q \in Q_f)$ and for all $f \in \Sigma, u, v \in Q^*$: $pfu \bullet v \in L_\Delta \Leftrightarrow qfu \bullet v \in L_\Delta$.

Proposition 1 *For $p, q \in Q$, we have $p \equiv q \Rightarrow p \sim q$*

Proof Let $p, q \in Q$ such that $p \not\sim q$. Then, there exists $pfq_1 \ldots q_{i-1} \bullet q_{i+1} \ldots q_n \in L_\Delta$ but $pfq_1 \ldots q_{i-1} \bullet q_{i+1} \ldots q_n \notin L_\Delta$. Using Definition 3 then we have $(f, q_1, \ldots, q_{i-1}, p, q_{i+1}, \ldots, q_n, p') \in \Delta$ but no transition with the form $(f, q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n, q') \in \Delta$. Here, we have $(f, q_1, \ldots, q_{i-1}, p, q_{i+1}, \ldots, q_n) \in \Gamma(p)$ but $(f, q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n)(p \stackrel{\cdot}{:}_i q) \notin \Gamma(q)$ although $i \in occ_p((f, q_1, \ldots, q_{i-1}, p, q_{i+1}, \ldots, q_n))$. By states equivalence $\equiv$ we have $p \not\equiv q$. □

Lemma 2 *The equivalence relation $\sim$ can be computed in $O(\hat{r}|\mathcal{A}|)$.*

Proof The size of L_Δ is $\hat{r}|\mathcal{A}|$. We can proceed by associating to each state an acyclic DFA that reads strings from L_Δ starting with the state in question. The assembly of all the constructed DFAs in one DFA requires $\hat{r}|\mathcal{A}|$ time and space. This DFA that we call A_{L_Δ} can be minimized in linear time (see [27]):

$$L_\Delta \xrightarrow{\hat{r}|\mathcal{A}|} A_{L_\Delta} \xrightarrow{\hat{r}|\mathcal{A}|} \min(A_{L_\Delta})$$

□

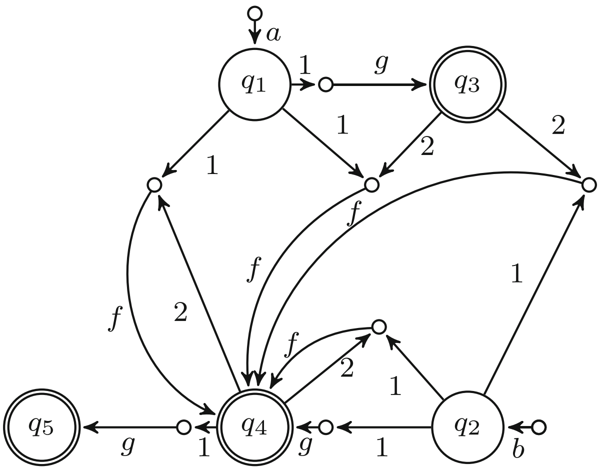

Fig. 1 FTA example

We present in Algorithm 1 an implementation of Lemma 2. First, a transition-empty acyclic DFAA_{L_Δ} is created. An elementary acyclic string DFA A_q is created for each state $q \in Q$ that reads string in the form $qfq_1 \ldots, \bullet, \ldots q_n \in L_\Delta$. This elementary DFA can be seen as a trie structure that reads strings from L_Δ and starting with the state in question. The function Add adds a state automaton A to A_{L_Δ} and creates a transition from 0 to the initial state of A. Finally, $Acyc\_Min$ minimizes acyclically the input automaton. It is clear that this Algorithm requires $\mathcal{O}(|Q| + \hat{r}|\mathcal{A}| + \hat{r}|\mathcal{A}|) = \mathcal{O}(\hat{r}|\mathcal{A}|)$

Algorithm 1 $\sim$ COMPUTATION

```
1: function $SIM\_COMPUTATION$(DFTA $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$)
2:     let $A_{L_\Delta} = (\{0\}, Q \cup \Sigma, \{0\}, \emptyset, \emptyset)$
3:     for all $q \in Q$ do
4:         let $A_q = (\emptyset, Q \cup \Sigma, \{0\}, \emptyset, \emptyset)$
5:     end for
6:     for all $(f, q_1, \ldots, q_n, q) \in \Delta$ do
7:         for all $q_i \in q_1, \ldots, q_n$ do
8:             $Add(A_{q_i}, fq_1 \ldots \bullet \ldots q_n)$
9:         end for
10:    end for
11:    for all $q \in Q$ do
12:        $Assembly(A_{L_\Delta}, A_q)$
13:    end for
14:    $Acyc\_Min(A_{L_\Delta})$
15:    $return(A_{L_\Delta})$
16: end function
```

Example 2 We consider the FTA $\mathcal{A}$ presented in Fig. 1.

Computing the horizontal language results in:

$$L_\Delta = \{a, b, q_1g\bullet, q_1f\bullet q_3, q_1f\bullet q_4, q_2g\bullet, q_2f\bullet q_4, q_2f\bullet q_3, q_3fq_2\bullet, q_3fq_1\bullet, q_4fq_2\bullet, q_4fq_1\bullet, q_4g\bullet, q_5\}$$

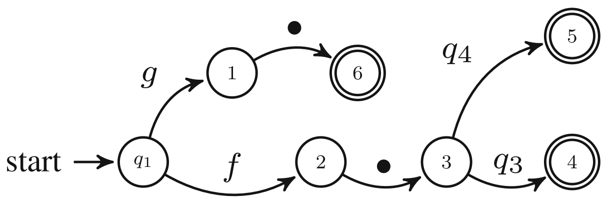

Fig. 2 Acyclic DFA for q_1

An elementary Acyclic DFA (trie structure) is created for each state. For example, the DFA corresponding to q_1 is presented in Fig. 2. This DFA recognizes the strings $g\bullet$, $f \bullet q_3$ and $f \bullet q_4$.

Thereafter, the minimal acyclic DFA recognizing L_Δ is presented in Fig 3. Obviously, $\overline{Q} = \{q_1, q_2\}$.

After the identification of the states that are possibly equivalent by $\equiv$, we associate for each state q in a transition $(f, q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots q_n, q')$ the "letter" $f_{q_1\cdots q_{i-1}, q_{i+1}\cdots q_n}$. Indeed, we transform the transition of the tree automaton to a transition of a string automaton.

Let $\overline{Q} = \{q \mid \exists p \neq q$ such that$q \sim p\}$. To each state, we associate an alphabet σ_q defined as follows: $\sigma_q = \{f_{u,v} \mid qfu \bullet v \in L_\Delta\}$.

Proposition 2 *Let* $\sigma = (\bigcup_{q\in\overline{Q}} \sigma_q)$ *be the alphabet associated to* $\overline{Q}$*, then* $|\sigma| \leq |\mathcal{A}|$

The automaton $M_\mathcal{A}$ will be defined on the alphabet σ. However, elements from σ still use strings in their representations. This could cause difficulties when comparing. Hence, a labelling system that associates each symbol from σ to a unique letter (number) can be done. The ideal structure is to use a trie that stores corresponding

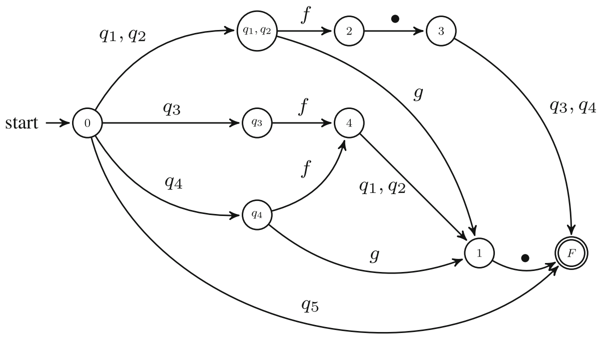

Fig. 3 Minimal acyclic DFA recognizing L_Δ

elements from L_Δ. Then, every symbol is labelled by its leaf number in the trie. The construction and the exploration of the trie structure requires $\mathcal{O}(|\mathcal{A}|)$ (see [12]).

Corollary 1 *σ can be computed and relabelled in $\mathcal{O}(|\mathcal{A}|)$.*

Example 3 Taking the same example as cited above, only paths leaving states q_1, q_2 are considered. Hence, we get $\sigma = \{g_{\epsilon,\epsilon}, f_{\epsilon,q_3}, f_{\epsilon,q_4}\}$. As q_1, q_2 are the only equivalent states, their acyclic DFA A_{q_1} or A_{q_2} (they share the same language) can be seen as a trie that represents σ. We can define the relabelling system as a mapping $\varphi : \sigma \to \mathbb{N}$. From Fig. 2, we get $\varphi(g_{\epsilon,\epsilon}) = 6$, $\varphi(f_{\epsilon,q_3}) = 4$, $\varphi(f_{\epsilon,q_4}) = 5$.

Below, we prove the equivalence between Nerode equivalence in the string and tree automata, and we also show that it is beneficial in other minimization algorithm especially in the incremental construction of tree automata.

Definition 5 Let $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ be a DFTA. The string automaton associated to $\mathcal{A}$ denoted $M_\mathcal{A}$ is the tuple $M_\mathcal{A} = (Q', \sigma, \{i_s\}, F, \delta)$ where $Q' = Q \cup \{i_s\}$ is the state set, $\sigma = \bigcup_{q \in \overline{Q}} \sigma_q \cup Q$ is the alphabet, $\{i_s\}$ is the initial state, $F = Q_f$ is the final states set and $\delta : Q' \times \sigma \to Q'$ is the transition function defined as follows.

- for all q in $\overline{Q}$: $\delta(q, a) = q'$ where $a = f_{q_1 \cdots q_{i-1}, q_{i+1} \cdots q_n}$, $d(f, q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n) = q'$.
- for all q in Q: $\delta(\underline{i_s}, q) = q$.
- for all q in $Q - \overline{Q}$: $\delta(q, q) = q$.

Example 4 W take the same example presented in Fig. 1. The resulting FTA is presented in Fig. 4. We use the labelling system φ from Example 3.

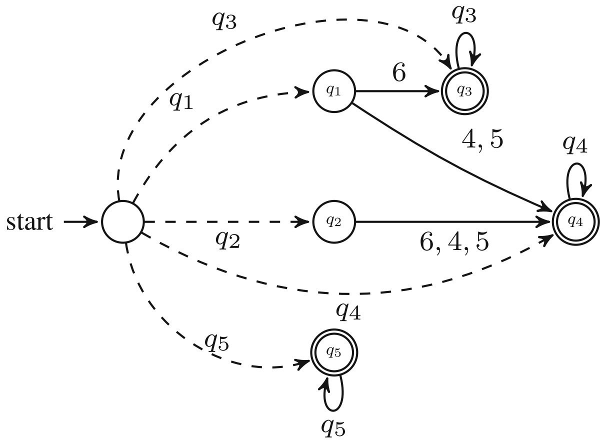

Fig. 4 The DFA $M_\mathcal{A}$ associated to $\mathcal{A}$

Notice that the state i_s, the alphabet symbols Q, and transitions leaving i_s have no importance in the minimization process. We use them just to construct the initial state which is customary in string automata. Moreover, we can see that states from $Q - \overline{Q}$ have no equivalent states because $\forall p \in Q, q \in Q - \overline{Q} : p \not\sim q$ then $p \not\equiv q$. Here, no transition is outgoing from that states.

We present in Algorithm 2 a filter that computes the associated string automaton of a given DFTA. The function $Compute\_Sigma(ADFA)$ extracts σ from $ADFA$ and stores the result in a trie structure. The function φ associates to each sequence $f_{u,v}$ from σ a unique letter corresponding to its leaf marking in the trie. The function φ^{-1} is the inverse function of φ. Finally, $export(DFA)$ returns the targeted DFA in an appropriate format for DFA minimization.

Algorithm 2 DFTA to DFA filter

```
 1: function $TTS\_Filter$(DFTA $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$)
 2:     $A_{L_\Delta} = SIM\_COMPUTATION(\mathcal{A})$
 3:     $\sigma \leftarrow Compute\_Sigma(A_{L_\Delta})$
 4:     $Q' \leftarrow Q \cup \{i_s\}$
 5:     $\delta \leftarrow \emptyset$
 6:     for $q \in Q$ do
 7:         $\delta \leftarrow \delta \cup \{(i_s, q) \rightarrow q\}$
 8:     end for
 9:     for all $a \in \sigma : \varphi^{-1}(a) = f_{q_1 \cdots q_{i-1}, q_{i+1} \cdots q_n}, d(f, q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n) = q'$ do
10:         $\delta \leftarrow \delta \cup \{(q, a) \leftarrow q'\}$
11:     end for
12:     $M_{\mathcal{A}} = (Q', \sigma, \{i_s\}, Q_f, \delta)$
13:     $export(M_{\mathcal{A}})$
14: end function
```

Lemma 3 *Algorithm 2 computes the associated DFA of $\mathcal{A}$ in $\mathcal{O}(\hat{r}|\mathcal{A}|)$.*

In the next section, we will prove that minimizing the associated string FA can fully replace DFTA minimization.

5 Minimization techniques using the associated FA

In this section, we show and discuss the use of the associated FA of a given FTA in different minimization techniques namely standard, acyclic, incremental and incrementally constructed minimization. We prove also that in some cases (acyclic and incremental techniques) the complexities are improved.

5.1 DFTA standard minimization

We show that the minimization of a given DFTA is no more than minimizing its associated string DFA. However, we start first by showing that the FA is deterministic.

Proposition 3 *The associated string automaton $M_{\mathcal{A}}$ of a DFTA $\mathcal{A}$ is deterministic.*

Proof By definition, we have for all $q \in Q' - \overline{Q}, a \in \sigma : |\delta(q, a)| \leq 1$. Let $q \in \overline{Q}$ and assuming that there exists a symbol $f_{u,v} \in \sigma$ such that

$\delta(q,a) = \{q', q''\}$ with $q' \neq q''$. Let $u = q_1 \dots q_{i-1}$ and $v = q_{i+1} \dots q_n$. Using Definition 3 we have then $(f, q_1, \dots, q_{i-1}, q, q_{i+1}, \dots, q_n, q') \in \Delta$ and $(f, q_1, \dots, q_{i-1}, q, q_{i+1}, \dots, q_n, q'') \in \Delta$. This leads to a contradiction because the tree automaton $\mathcal{A}$ is deterministic. □

We note that i_s and all transitions outgoing from it are not considered in the minimization process, then we use σ in what follows to denote $\sigma - Q$.

After the string automaton $M_{\mathcal{A}}$ is created, we show that the computation of the equivalence relation $\equiv$ defined on $\mathcal{A}$ can be done by computing the string Nerode equivalence $\cong$ relation. $\cong$ is defined as follows.

$$p \cong q \Leftrightarrow \begin{cases} p \in F \Leftrightarrow q \in F \\ \text{for all } a \in \sigma, \delta(p,a) \cong \delta(q,a) \end{cases} \tag{7}$$

Proposition 4 *For $p, q \in Q$, we have $(p \cong q) \Leftrightarrow (p \equiv q)$*

Proof In this proof, we use the facts that $\forall q \in (Q - \{i_s\}), i_s \not\cong q$ and $\forall p, q \in Q - \overline{Q}, p \neq q \Leftarrow p \not\cong q$. These two properties are implicit results of Lemma 1 and Definition 5.

It is well known that the Nerode equivalence $\cong$ can be computed by successive approximations $\cong_j$ defined as:

$$p \cong_0 q \text{ iff } (p \in F \Leftrightarrow q \in F) \tag{8}$$
$$p \cong_{j+1} q \text{ iff } p \cong_j q \text{ and for all } a \in \sigma, p, q \in Q' : \delta(p,a) \cong_j \delta(q,a). \tag{9}$$

Here, we remark that some states form $Q - \overline{Q}$ may be non co-accessible states and then may have an empty top language. Then, two states can be merged in the minimization process. We suppose then that the the string automaton is reduced.

To prove this proposition we show that for all $p, q \in Q, j \in \mathbb{N} : p \cong_j q \Leftrightarrow p \equiv_j q$. The proof is by induction on the definitions of $\equiv_j$ and $\cong_j$. For the base case $(j = 0)$, as $Q_f = F$, for $p, q \in Q$, we have $p \cong_0 q \Leftrightarrow p \equiv_0 q$. Assuming now that for all $p, q \in Q$, $(p \cong_k q) \Leftrightarrow (p \equiv_k q)$ for some $k \geq 0$.

First, we prove that $(p \cong_{k+1} q) \Rightarrow (p \equiv_{k+1} q)$:

Suppose that $(p \cong_{k+1} q)$. By the successive approximations of $\cong$ we have

$$(p \cong_{k+1} q) \Leftrightarrow p \cong_k q \text{ and for all } f \in \sigma, \delta(p,f) \cong_k \delta(q,f) \tag{10}$$

By applying the induction hypothesis, we get:

$$(p \cong_{k+1} q) \Leftrightarrow p \equiv_k q \text{ and for all} f \in \sigma, \delta(p,f) \equiv_k \delta(q,f) \tag{11}$$

Next, we prove that $(p \equiv_{k+1} q)$ by analyzing the states from Q and the different symbols from $\cup_{q \in \overline{Q}} \sigma_q \cup Q$.

Following the construction, let $q \in \{i_s\} \cup Q - \overline{Q}$. We can see that the $\forall p \neq q, p \not\equiv q$. It is because the right language of any state from $\{i_s\} \cup Q - \overline{Q}$ is unique (see Definition 5).

Therefore, we only check states from $\overline{Q}$. We can see that the outgoing transitions from these states are of the form $f_{u,v}$ since they belong to the constructed horizontal language.

Let $u = q_1 \cdots q_{i-1}$ and $v = q_{i+1} \cdots q_n$. Using the horizontal language definition we get $pfu \bullet v, qfu \bullet v \in L_\Delta$. So, there exist two states p' and q' in Q such that $(f, q_1, \ldots, q_{i-1}, p, q_{i+1}, \ldots, q_n, p')$ and $(f, q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n, q')$ are in Δ (see equation (6)). From Definition 5 we have $p' = \delta(p, f_{u,v}) = d(f, q_1, \ldots, q_{i-1}, p, q_{i+1}, \ldots, q_n)$ and $q' = \delta(q, f_{u,v}) = d(f, q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n)$. Finally, we get $(p \equiv_{k+1} q)$ by applying (11).

The next step of the proof is to show that $(p \equiv_{k+1} q) \Rightarrow (p \cong_{k+1} q)$. This proof can be done following the same steps as the first implication. Indeed, this second way is more simple given that elements from Δ are directly involved in L_Δ.

We recall the Nerode equivalence for the DFTA.

$$p \equiv_0 q \text{ if and only if } (p \in Q_f \Leftrightarrow q \in Q_f)$$

$$p \equiv_{j+1} q \text{ if and only if } p \equiv_j q \text{ and for all } \rho \in \Gamma(p), i \in occ_p(\rho) : \rho(p :_i q) \in \Gamma(q) \text{ and } d(\rho) \equiv_j d(\rho(p :_i q))$$

Then, suppose that $(p \equiv_{k+1} q)$. By the successive approximations of $\equiv$ we have By induction hypothesis, we get:

$$p \equiv_{j+1} q \Leftrightarrow p \cong_j q \text{ and } \forall \rho \in \Gamma(p), i \in occ_p(\rho) : \rho(p :_i q) \in \Gamma(q) \text{ and } d(\rho) \cong_j d(\rho(p :_i q)) \tag{12}$$

We put $\rho = (f, q_1, \ldots, q_n), d(\rho_{p:i}) = p'$ and $d(\rho_{q:i}) = q'$ where $p', q' \in Q$, then we have $(f, q_1, \ldots, q_{i-1}, \; p, q_{i+1}, \ldots, q_n, p')$, $(f, q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n, q') \in \Delta$. It is clear that $p \sim q$, so by applying Definition 5 and 1 , we get $p' = \delta(f_{q_1 \ldots q_{i-1}, q_{i+1} \ldots q_n}, p) = d(\rho_{p:i})$ and $q' = \delta(f_{q_1 \ldots q_{i-1}, q_{i+1} \ldots q_n}, q) = d(\rho_{q:i})$. This fact implies that $p \cong_{k+1} q$ (using (12)). □

We can also check this corollary:

Corollary 2 *Let $p \in Q' - \overline{Q}, q \in Q' (q \neq p)$ we have $p \ncong q$.*

As a consequence of this Corollary and as mentioned above, only states in $\overline{Q}$ are considered in the minimization algorithm. We can easily check that i_s is not equivalent to any state. Indeed, for each state $q \in Q' - \overline{Q}$, its equivalence class contains only q. As the incomplete case is discussed and using Corollary 1 and [8]:

Lemma 4 *Let $\mathcal{A}(Q, \Sigma, Q_f, \Delta)$ be a DFTA. Let $M_\mathcal{A}$ be its associated DFA. Then, $M_\mathcal{A}$ can be minimized in $\mathcal{O}(|Q| + |\mathcal{A}| \log |Q|) = \mathcal{O}(|\mathcal{A}| \log |Q|)$.*

Thus, using Lemma 2, Corollary 1 and Lemma 4 we get:

Theorem 1 *Let $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ be a DFTA, $\mathcal{A}$ can be minimized in $\mathcal{O}(\hat{r}|\mathcal{A}| + |\mathcal{A}| \log |Q|)$.*

5.2 Acyclic minimization

Acyclic automata are a beneficial data structure that represents and stores a finite set of objects. When objects are trees like in XML, it is useful to store finite XML files in a compact and acyclic structure [25]. Acyclic DFA (ADFA) minimization was widely discussed as in [26, 27] for instance, and almost of these techniques have linear asymptotic complexity. Here we show how the associated FA can be used to minimize an acyclic DFTA.

Although Proposition 4 is sufficient for proving the use of the associated DFA to minimize ADFTA, we acknowledge in what follows some useful definitions.

Definition 6 Let $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ be a DFTA. Then $\mathcal{A}$ is acyclic (ADFTA) if and only if for all $q \in Q$, if $t \in L^{\downarrow}(q)$ then $St(t) \cap L^{\downarrow}(q) = \{t\}$.

We can consider the following lemma.

Lemma 5 *The associated DFA of a ADFTA is acyclic.*

Using Proposition 4, we know that states from an ADFTA that are not in $\sim$ are distinguishable and cannot be merged during minimization process since ADFTA are trivial case of DFTA.

Thus, after computing the associated string ADFA $M_{\mathcal{A}}$ of a DFTA $\mathcal{A}$, the minimization is done in linear time [27, 31].

Theorem 2 *A ADFTA $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ can be minimized using its associated ADFA $M_{\mathcal{A}} = (Q', \sigma, \{i_s\}, F, \delta)$ in $\mathcal{O}(\hat{r}|\mathcal{A}| + |\mathcal{A}|) = \mathcal{O}(\hat{r}|\mathcal{A}|)$.*

5.3 Incremental minimization

Incremental minimization is a useful technique in practise. It is used when minimization process may be halted in any time producing a reduced automaton in terms of states number, it also produces a valid automaton which recognizes the same language as the departure automaton.

In the string case, Watson et al. [28] introduced the first incremental minimization for cyclic DFA, but with exponential complexity. Subsequently, Watson et al. [12] improved this algorithm and give an almost quadratic implementation. After that, Almeida et al. [14] presented the best known incremental implementation using the UNION-FIND algorithm. Recently, Garcia et al. [15] have proposed a new algorithm that out-performs that one of Almeida et al. in both asymptotic and average complexities.

However, in the tree case, Cleophas et al. [13] generalized the incremental approach of Watson et al. [28] to trees and gave a cubic implementation.

Here also, we show that the incremental minimization can be done using the associated DFA while the complexity of this minimization is better than previous work on trees.

Let $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ be a DFTA and $M_{\mathcal{A}} = (Q', \sigma, \{i_s\}, F, \delta)$ be its associated DFA. We extend the transition function δ by δ' as follows: $\delta'(q, a) = \delta(q, a)$ where $a \in \sigma$ and $\delta'(q, ax) = \delta'(\delta(q, a), x)$ where $ax \in \sigma^+$. We define the right language of a state $q \in Q$ by: $\overrightarrow{L}(q) = \{x \in \sigma^+, \delta'(q, x) \in F\}$.

Lemma 6 *Let $p, q \in Q$ then $L^{\uparrow}(p) = L^{\uparrow}(q) \Leftrightarrow \overrightarrow{L}(p) = \overrightarrow{L}(q)$.*

Using this lemma, we can compute $equiv(p, q)$ in $M_{\mathcal{A}}$ instead of computing it in $\mathcal{A}$.

The recursive computation of $equiv$ in both DFA(1) and DFTA(2) are given respectively using (1) and (2).

$$equiv(p, q) = \bigwedge_{a \in \sigma} (equiv(\delta(p, a)), d(\delta(q, a)) \wedge ((p \in F) \Leftrightarrow (q \in F)) \tag{13}$$

$$equiv(p, q) = \bigwedge_{\rho \in \Gamma(p), i \in Occ(p)} (equiv(d(\rho), d(\rho(p :_i q))) \wedge ((p \in Q_f) \Leftrightarrow (q \in Q_f)) \tag{14}$$

Thus, we can use the best-known complexity algorithm for DFA thanks to Garcia et al. [15] to minimize a DFTA by incrementally minimizing its associted DFA:

Theorem 3 *A DFTA $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ can be incrementally minimized using its associated DFA $M_{\mathcal{A}} = (Q', \sigma, \{i_s\}, F, \delta)$ in $\mathcal{O}(\hat{r}|\mathcal{A}| + |\mathcal{A}||Q|^2) = \mathcal{O}(|\mathcal{A}||Q|^2)$.*

5.4 Incremental construction of minimal tree automata

Incremental construction of automata is an important approach which is discussed in string and tree cases. It allows adding or deleting words (resp. trees) to/from an existing minimal automaton. In other words, if $\mathcal{A}$ is a DFA (resp. DFTA) and w is a word (resp. t is a tree) then the incremental construction consists of creating a new automaton that recognizes $L(\mathcal{A}) \cup \{w\}$ (resp. $L(\mathcal{A}) \cup \{t\}$) while maintaining minimality. First, incremental construction was presented by Daciuk et al. [29] for ADFA. Then later, Carrasco et al. [17] generalized this notion to cyclic DFA. Later on, the same authors redefined redefine the incremental construction for trees in [16].

Before presenting the incremental construction using the associated DFA, let us recall the basics of this construction.

The idea is to create an acyclic DFTA that recognizes the tree t to be added. Then, using the basic operation of the union (see [10]) we get a new automaton that recognizes the wanted language. Minimization is then used to get the new minimal DFTA. However, some computations are repeated every time a new tree is added. In fact, it is not necessary to re-process unchanged states when adding, executing union and minimizing — leading to the incremental construction idea.

Let $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ be a minimal DFTA, $t \notin L(\mathcal{A})$ a tree to be added. The first step is to try the recognition of t by $\mathcal{A}$. If $m_{\mathcal{A}}(t) \in Q_f$ then $\mathcal{A}$ is unchanged. Otherwise, we need to change the targeted DFTA. The incremental construction for DFTA, thanks to Carrasco et al. [17], is summarized in what follows:

1. for every tree s from $St(t)$:
   - If $m_{\mathcal{A}}(s)$ has no output then we can safely create a new state n such that $L^{\downarrow}(n) = s$.
   - If there exists a state $q \in Q$ such that $m_{\mathcal{A}}(s) = q$ then we face two cases:

(a) If C_q=1 then this subtree is recognized by the language and no states are added or deleted.
(b) Otherwise, $(C_q > 1)$ *cloned* state n is added to Q and Δ is modified to ensure that $n \equiv q$ by adding the necessary transitions.

2. Execute a local minimization which concerns only a subset of states computed or used when processing subtrees from t.

Now we discuss the use of the associated DFA to build the new minimal DFTA.

Let $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ be a minimal DFTA and $M_{\mathcal{A}} = (Q', \sigma, \{i_s\}, F, \delta)$ be its associated minimal DFA and $t \notin L(\mathcal{A})$ the tree to be added. Taking into account that $M_{\mathcal{A}}$ is not designed for tree acceptance, it cannot be used to accept trees from $T(\Sigma)$. This is due to the loss of some transitions when building the associated DFA. Then, states identification is done on $\mathcal{A}$. Note that $\sim$ must be maintained to complete the minimization because some transitions are not in δ but may appear after states addition.

Nevertheless, we adapt the algorithm for this reason as follows:

1. For every tree s from $St(t)$:
   - If $m_{\mathcal{A}}(s)$ has no output then we can safely create a new state n such that $L^{\downarrow}(n) = s$.
   - If there exists a state $q \in Q$ such that $m_{\mathcal{A}}(s) = q$ then we have two cases.

(a) if C_q=1 then this subtree is recognized by the language and no states are added or deleted.
(b) Otherwise, $(C_q > 1)$ add a *clone* state n in $M_{\mathcal{A}}$ as follows: Add to $nq_1 \ldots q_{i-1} \bullet q_{i+1} \ldots q_n$ to L_Δ such that $qq_1 \ldots q_{i-1} \bullet q_{i+1} \ldots q_n \in L_\Delta$. Here, $\sigma_n = \sigma_q$.

After that, and before updating $M_{\mathcal{A}}$, $\sim$ is updated and $\overline{Q}$ is recomputed. Finally, $M_{\mathcal{A}}$ is resolved as follows:

1. Remove from L_Δ all words of the form $pq_1 \ldots q_i \bullet q_{i+1} \ldots q_n$ where $d((f, q_1, \ldots, q_i, \underline{p}, q_{i+1}, \ldots, q_n)) = q$ for every state q which has clones.
2. Recompute both $\overline{Q}$ and $M_{\mathcal{A}}$.

At the end, the same local minimization proposed by Carrasco et al. [17] is applied on the associated DFA.

Concerning the complexity, the computation of clone states cannot be avoided for DFTA because unlike DFA, every symbol in trees has many successors, but in the word case, every symbol has only one successor. Therefore, the worst time complexity for clones computation is $\mathcal{O}(|\Delta|2^{\hat{r}})$ where $\hat{r}$ is the maximum rank of Σ. The re-computation of $\overline{Q}$ requires $\mathcal{O}(|\mathcal{A}|)$, whereas in the minimizing process, it requires $\mathcal{O}(|\mathcal{A}|)$ too because t is finite and cannot add cycles into the automaton.

Theorem 4 *Let $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ be a minimal DFTA and $M_{\mathcal{A}} = (Q', \sigma, \{i_s\}, F, \delta)$ be its associated minimal DFA then the minimal automaton that recognizes $L(\mathcal{A} \cup \{t\})$ where t is constructed in $\mathcal{O}(|\Delta|2^{\hat{r}} + \hat{r}|\mathcal{A}| + |\mathcal{A}|) = \mathcal{O}(|\Delta|2^{\hat{r}} + \hat{r}|\mathcal{A}|)$.*

Table 1 Variants of minimisation techniques: Comparative Complexities

Minimization techniques	Existing complexity	New complexity
Standard	$\mathcal{O}(\hat{r}\|\mathcal{A}\| \log(\|Q\|))$ Abdulla et al. [19] combined with Högberg et al. [20]	$O(\hat{r}\|\mathcal{A}\| + \|\mathcal{A}\| \log \|Q\|)$
Acyclic	–	$\mathcal{O}(\hat{r}\|\mathcal{A}\|)$
Incremental	$\mathcal{O}(\|\mathcal{A}\|^{\|Q\|-2}\|Q\|^2)$ Cleophas et al. [13, 32]	$\mathcal{O}(\hat{r}\|\mathcal{A}\| + \|\mathcal{A}\|\|Q\|^2)$
Incrementally constructed	$\mathcal{O}(\|\Delta\|2^{\hat{r}} + \|\mathcal{A}\|)$ Carrasco et al. [16]	$\mathcal{O}(\|\Delta\|2^{\hat{r}} + \hat{r}\|\mathcal{A}\|)$

Table 1 discussed minimization techniques' existing and obtained complexities.

6 Experimental issues

In this section, we present some experimental issues on the associated DFA's behaviour with regards to the DFTA minimization process.

First, let us mention that there is a lack of DFTA benchmarks and hard instances for this type of automata. Moreover, besides the existing random generators in literature (such as Hanneforth et al. [30]and Héam et al [33]), to our knowledge there is no implementation of a realistic generator.

To overcome this problem, we perform a straightforward generation of a set of arbitrary transitions that acts like L_Δ. This randomly generated set is assumed to be sufficient to understand the relation between FTA and its associated DFA's size. Algorithm 3 describes how the transition set is obtained. $Uniform$ uniformly generates a number in a given range. $Select\ uniformly$ randomly and uniformly selects an element from the alphabet or chooses a given number of states. Mind that every state have $1/2$ probability to be final. The generator uses four parameters, namely: the alphabet, the maximum rank, the wanted transitions number and the states set.

Algorithm 3 Samples random generation

```
1: function $GENERATE(Alphabet, Maximum\ rank, Transitions\ number, States)$
2:     for all $f \in Alphabet$ do
3:         $Rank(f) \leftarrow Uniform([0, Maximum\ Rank])$
4:     end for
5:     repeat
6:         $Select\ Uniformly(f, Alphabet)$
7:         $Select\ Uniformly(Rank(f), states)$
8:         $Generate\ transition$
9:     until $Transitions\ number$ is reached
10:    $SIM\_COMPUTATION((Alphabet, States, Final\ States, Transitions\ set))$
11:    if $\overline{Q}$is empty then
12:        the sample is rejected
13:    end if
14: end function
```

We obtained the benchmark following a “generation-rejection” strategy. Once generated, a sample is maintained only if $\overline{Q}$ is not empty. Therefore, the study focuses on samples containing states which are equivalent with respect to $\sim$. It is worth mentioning that Algorithm 2 indicates that a DFTA cannot be minimized if $\overline{Q}$ is empty. To avoid a high rate rejection, we can bound the maximum rank.

This random generator can be used to perform comparison between the classical minimization approaches — which have to be implemented — and the string minimization after applying the filter.

7 Conclusions and future work

In this paper, we have shown how the minimization problem of deterministic tree automata can be reduced to the minimization problem of deterministic string automata. Indeed, we use the environment (and the TS transformation) notion proposed by Abdulla et al. [19] to create a valid DFA and then minimize it. We clarified that the minimal DFA coincides with the minimization of the original tree automaton. Hence, we prove that there is actually no need to implement existing algorithms proposed for trees, since the latter enables the exploitation of the large range of minimization algorithms for strings to add minimization procedures in tree toolkits. Moreover, we prove that DFTA minimization can be done for the standard minimization in $\mathcal{O}(rm + m\log(n))$ where r is the maximum rank of the alphabet, m is the automaton size and n is the number of states (which is considered in terms of asymptotic complexity same as the best known one). We also showed that minimization using the associated DFA gives better complexities compared to the other existing minimization approaches, namely acyclic ($\mathcal{O}(rm)$) and incremental ($\mathcal{O}(rm + mn^2)$) minimization (which are clearly improved in our proposal).

In fact, the implemented algorithm consists of a filter transforming a given DFTA into its associated deterministic finite string automaton. It achieves this task by computing new string symbols based on a compact structure that stores the DFTA transition sequences. It would be interesting to extend this work by a comparative study of time and space average complexity. Unfortunately, this may be a large task, given the lack of implementations of the different minimization algorithms.

Furthermore, it is interesting to study the average size of the associated DFA. We have shown experimentation on transition sets with different sizes. In the context of randomly generated sequences, results show that the states-equivalence computation gives a smaller structure compared to the initial one. We plan also to consolidate this work with experimental tests and comparisons with other techniques after developing a consistent DFTA generator.

Acknowledgments This work is supported by a South Africa-Algeria Cooperation Project funded by the South African NRF under project 87462 and the Algerian MESRS-DGRSDT under project A/AS-2013-002. Any opinion, finding and conclusion or recommendation expressed in this material is that of the author(s) and the NRF/MESRS-DGRSDT do not accept any liability in this regard.

References

1. Zilio, S., Lugiez, D.: XML schema, tree logic and sheaves automata. In: Rewriting Techniques and Applications (R. Nieuwenhuis, ed.), vol. 2706 of Lecture Notes in Computer Science, Springer Berlin Heidelberg (2003)
2. Chidlovskii, B.: Using regular tree automata as XML schemas. In: Proceedings IEEE advances on digital libraries conference 2000, pp. 89–98 (1999)
3. Tommasi, M.: Structures arborescentes et apprentissage automatique, p. 3. PhD thesis, Université Charles de Gaulle - Lille (2006)
4. Brainerd, W.S.: The minimalization of tree automata. Inf. Control. **13**(5), 484–491 (1968)
5. Arbib, M.A., Give'on, Y.: Algebra automata I: Parallel programming as a prolegomena to the categorical approach. Inf. Control. **12**(4), 331–345 (1968)
6. Moore, E.F.: Gedanken Experiments on Sequential Machines. In: Automata Studies, pp. 129–153, Princeton U. (1956)
7. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Boston (1979)
8. Valmari, A.: Fast brief practical DFA minimization. Inf. Process. Lett. **112**, 213–217 (2012)
9. Gécseg, F., Steinby, M.: Minimal ascending tree automata. Acta Cybern. **4**, 37–44 (1980)
10. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications, 2007. release October 12th (2007)
11. Watson, B.W.: Taxonomies and Toolkits of Regular Language Algorithms. PhD thesis, Faculty of Mathematics and Computer Science, Eindhoven University of Technology (1995)
12. Watson, B.W., Daciuk, J.: An efficient incremental DFA minimization algorithm. Nat. Lang. Eng. **9**(1), 49–64 (2003)
13. Cleophas, L.G., Kourie, D.G., Strauss, T., Watson, B.W.: On minimizing deterministic tree automata. In: Stringology (J. Holub and J. Zdrek, eds.), pp. 173–182, Prague Stringology Club, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague (2009)
14. Almeida, M., Moreira, N., Reis, R.: Incremental DFA minimisation. In: CIAA (M. Domaratzki and K. Salomaa, eds.), vol. 6482 of Lecture Notes in Computer Science, pp. 39–48, Springer (2010)
15. García, P., Vázquez de Parga, M., Velasco, J.A., Lòpez, D.: A split-based incremental deterministic automata minimization algorithm. Theory Comput. Syst. **57**(2), 319–336 (2015)
16. Carrasco, R.C., Daciuk, J., Forcada, M.L.: Incremental construction of minimal tree automata. Algorithmica **55**(1), 95–110 (2009)
17. Carrasco, R.C., Forcada, M.L.: Incremental construction and maintenance of minimal finite-state automata. Comput. Linguist. **28**(2), 207–216 (2002)
18. Carrasco, R.C., Daciuk, J., Forcada, M.L.: An implementation of deterministic tree automata minimization. In: CIAA (J. Holub and J. Zdarek, eds.), vol. 4783 of Lecture Notes in Computer Science, pp. 122–129, Springer (2007)
19. Abdulla, P.A., Bouajjani, A., Holík, L., Kaati, L., Kaati, T.X.: Composed bisimulation for tree automata. Int. J. Found. Comput. Sci. **20**(4), 685–700 (2009)
20. Högberg, J., Maletti, A., May, J.: Backward and forward bisimulation minimization of tree automata. Theor. Comput. Sci. **410**(37), 3539–3552 (2009)
21. Riley, M., Allauzen, C., Jansche, M.: OpenFst: An open-source, weighted finite-state transducer library and its applications to speech and language. In: Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, May 31 - June 5, 2009, Boulder, Colorado, USA, Tutorial Abstracts, pp. 9–10 (2009)
22. Kanthak, S., Ney, H.: FSA: An Efficient and Flexible C++ Toolkit for Finite State Automata Using On-Demand Computation. In: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, Barcelona, Spain, pp. 510–517 (2004)
23. Watson, B.W.: Implementing and using finite automata toolkits . Nat. Lang. Eng. **2**, 295–302 (1996)
24. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Comput. **16**(6), 973–989 (1987)
25. Bousquet-Mèlou, M., Lohrey, M., Maneth, S., Nòth, E.: XML compression via directed acyclic graphs. Theory Comput. Syst. **57**(4), 1322–1371 (2015)

26. Watson, B.W.: A new algorithm for the construction of minimal acyclic DFAs. Sci. Comput. Program. **48**(2-3), 81–97 (2003)
27. Revuz, D.: Minimisation of acyclic deterministic automata in linear time. Theor. Comput. Sci. **92**(1), 181–189 (1992)
28. Watson, B.W.: An incremental DFA minimization algorithm. In: Finite State Methods in Natural Language Processing (2001)
29. Daciuk, J., Mihov, S., Watson, B.W., Watson, R.E.: Incremental construction of minimal acyclic finite-state automata, CoRR, vol. cs.CL/0007009 (2000)
30. Hanneforth, T., Maletti, A., Quernheim, D.: Random generation of nondeterministic finite-state tree automata. In: Proceedings Second International Workshop on Trends in Tree Automata and Tree Transducers, TTATT 2013, Hanoi, Vietnam, 19/10/2013, pp. 11–16 (2013)
31. Bubenzer, J.: Minimization of Acyclic DFAs. In: Proceedings of the Prague Stringology Conference 2011, Prague, Czech Republic, August 29-31 (2011)
32. Björklund, J., Cleophas, L.: A Taxonomy of Minimisation Algorithms for Deterministic Tree Automata. J. Universal Comput. Sci. **22**(2), 180–196 (2016)
33. Hėam, P.-C., Nicaud, C., Schmitz, S.: Parametric random generation of deterministic tree automata. Theor. Comput. Sci. **411**(38-39), 3469–3480 (2010)