From tree automata to string automata minimization

Younes Guellouma
Laboratoire dinformatique et de mathmatiques
University of Laghouat
y.guellouma@mail.lagh-univ.dz

Hadda Cherroune
Laboratoire dinformatique et de mathmatiques
University of Laghouat
h_cherroune@mail.lagh-univ.dz

Djelloul Ziadi
Laboratoire LITIS - EA 4108
Normandie Université, Rouen, France
Djelloul.Ziadi@univ-rouen.fr

Bruce W Watson
FASTAR Group, Department of Information Science
Stellenbosch University, Stellenbosch, South Africa
bruce@fastar.org

We propose a reduction of the minimization problem for a bottom-up deterministic tree automaton (DFTA) to the minimization problem for a string deterministic finite automaton (DFA). We proceed by a transformation of the tree automaton into a particular string automaton and then minimize the string automaton. We show that for our transformation, the minimization of the resulting string automaton coincides with minimization original tree automaton. We also discuss the complexity of this approach in different types of tree automata namely standard, acyclic, incremental, and incrementally constructed tree automata.

1 Introduction

Tree automata constitute a powerful theoretical tool used in many fields such as XML Schema [15], natural language processing, verification techniques, and program analysis, but relatively few general tools and toolkits exist for algorithm experimentation. In this context, we develop a framework [1] that allows manipulating tree automata and representing large amounts of data as trees.

Minimization is considered as a useful technique to compact the size of automata. In the literature, almost all the minimization techniques [4, 3] are inspired by string automata minimization which was studied for the first time by Huffman and Moore. Their algorithm is based on the definition of distinguishable pairs of states. At the end of the algorithm, all states judged undistinguishable are merged. Later, Hopcroft [11] defined a new algorithm which proceeds by refining the coarsest partition until no more refinements are possible.

Following the same steps, minimization techniques for tree automata have emerged. Early in 1967, Brainerd [4] proposed the first DFTA minimization method which we call *standard method*. Since then, several algorithms and implementations have been created, all following the same approach as Brainerd's algorithm, e.g. Arbib [3], Gésceg and Steinby, and Comon et al.

After that, Watson [12] designed the first *incremental minimization* algorithm. 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 a valid tree automaton that recognizes the same language as the departure one. That algorithm was subsequently refined by Watson and Daciuk [14]. This incremental algorithm constitutes the basics of many other techniques [2].

However, incrementality is an ambiguous term and has been used by Carrasco et al. [6] to interpret another manner of minimal automata building. In fact, this work is a generalization of a previous

[1]Project 8/U03/7015 supported by the MESRS - Algeria.

Submitted to:
TTATT 2015

work [7] on strings. The "incremental" notion is employed here to describe the construction instead of the construction instead of states verification. Hence the minimal tree automaton is constructed by adding trees to a minimal tree automaton while maintaining minimality.

As mentioned above, in the wish to develop a toolkit to manipulate tree automata, our observation from minimization techniques studies is that there exists a degree of analogy with the string automata minimization[2]. The question is therefore: does there exist some transformation from a tree automaton (DFTA — a deterministic finite tree automaton) to a string one (DFA — a deterministic finite string automaton) while its minimization coincides with the minimization of the original one. The DFA can then be minimized using one of the well-known techniques, then 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. [5] define a "signature" to each state which represents the "behaviour" of a state in the minimization process. Next, Abdulla et al. [1] extend this notion to compute an equivalance 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 which is similar with string automata. Therefore the complexity of this minimization is $\mathscr{O}(|\mathscr{A}||Q|\log(|Q|))$ where $|\mathscr{A}|$ is the size of the automaton and $|Q|$ the number of its states. This complexity can be mapped to deterministic finite tree automation (DFTA) by using Högberg et al. [8].

In this paper, we continue in this direction and we construct a string automaton which can fully replace the tree one for minimization purposes. Abdulla et al. defined an equivalent transition system that can be used to compute the Myhill-Nerode relation and discussed complexity for standard minimization. The focus of this present paper is on proving that tree automata minimization can be done through string automata minimization techniques which are well studied and the different implementation are available. 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 in the way that it is given in function of the string automaton instead of the tree automaton. Next, we show that the associated string automaton minimization coincides also with the acyclic, incremental minimizations. Finally we discuss the complexity in all of theses 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 reaches in almost all cases a better complexity.

The 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. After that, in Section 4, we detail the basics of our approach and the algorithm then we discuss its complexity in the deterministic case. Section 5 discusses the method impact in acyclic, incremental and incremental construction minimization techniques and reports their complexities. Finally, Section 6 presents some concluding remarks and suggestions for future work.

2 Preliminaries

A ranked alphabet is a pair (Σ, Arity) where Σ is a finite set of symbols and Arity is a mapping $\mathit{Arity}: \Sigma \to \mathbb{N}$ where $\mathbb{N}$ is the set of nonnegative integers. The arity of a symbol $f \in \Sigma$ is noted $\mathit{Arity}(f)$, the subset of p-ary symbols of Σ is $\Sigma_p = \{f \in \Sigma \mid \mathit{Arity}(f) = p\}$. We use the notation f, $f(\ )$, $f(\ ,\ )$, $\ldots$, $f(\ ,\ldots,\ )$ respectively for constant, unary, binary,..., p-ary symbols. For the sake of

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

simplicity, we use just Σ to represent a ranked alphabet (Σ, *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 set $t(r \leftarrow s)$ is the set all trees in which we substitute every occurrence of the subtree $r \in St(t)$ by the tree s once.

A bottom up finite tree automaton (FTA) over a ranked alphabet Σ is a tuple $\mathscr{A} = (Q,\Sigma,Q_f,\Delta)$ 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 $\rho = (f,q_1,\ldots,q_n,q), f \in \Sigma,\ q,q_1,\ldots,q_n \in Q$ is $|\rho| = n+1$. Then the size of the automaton $\mathscr{A}$ is defined as

$|\mathscr{A}| = \sum_{\rho\in\Delta} |\rho|$. From now, we consider the deterministic FTA (DFTA)
The transition function d for a DFTA is:

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

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

$\Gamma(q) = \{(f,q_1,\ldots,q,\ldots,q_n) \mid (f,q_1,\ldots,q,\ldots,q_n,q') \in \Delta\}$ 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 noted $\perp$ to complete a tree automaton, this completion is usually used to define equivalence between states. Here, we use Γ to avoid the completion of DFTA and then to define states equivalence using only the existing transitions.

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

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

A tree t is accepted by $\mathscr{A}$ if and only if $m_{\mathscr{A}}(t) \in Q_f$.
The language accepted by $\mathscr{A}$ is: $L(\mathscr{A}) = \{t \in T(\Sigma) \mid m_{\mathscr{A}}(t) \in Q_f\}$.
In the same way the accepted language (down language) by a state q is defined as follows: $L^{\downarrow}(q) = \{t \in T(\Sigma) \mid m_{\mathscr{A}}(t) = q\}$.
The residual (top) language of a state q is defined as follows: $L^{\uparrow}(q) = \{t(s \leftarrow \#) \mid t \in T(\Sigma), s \in L^{\downarrow}(q)$ and $m_{\mathscr{A}}(t) \in Q_f\}$.

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 $m_{\mathscr{A}}(t) \in Q_f$. A state is *useless* if it is neither accessible nor co-accessible. Useless states and the transitions using them can be safely removed from Q and Δ respectively without affecting $L(\mathscr{A})$. We can remove all useless states in $\mathscr{O}(|\mathscr{A}|)$. Thus, we suppose throughout this paper that every tree automaton is free from useless states.

3 Tree automata minimization

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

Let $\mathscr{A} = (Q, \Sigma, Q_f, \Delta)$ be a DFTA. We define over Q the following equivalence relation $\equiv$. $p \equiv q$ 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 the late 1960s by Brainerd [4], and standardised in [5]. It computes the equivalence relation $\equiv$ by 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 stable point, that is, some natural number k such that $\equiv_k = \equiv_{k+1}$.

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

Algorithm 1 describes in a general way the standard tree automata minimization. It iterates over a sequence of steps. First, the initial partition is set to $\{Q_f, Q - Q_f\}$. Next, at each iteration i, the current partition P_i is split by computing $\equiv_i$.

Let us recall that this standard algorithm is quadratic and needs $\mathscr{O}(|\mathscr{A}|^2)$. There exists several implementations of this standard algorithms like Carrasco et al. [5] which are quadratic too.

Furthermore, there exists other tree minimization techniques like acyclic, incremental and incrementally constructed ones. But before discussing them, let us introduce our transformation.

4 From DFTA to DFA

The main idea of our reduction is to create an associated string automaton to a FTA to be minimized. Instead of minimizing the wanted 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.

This idea is not completely novel. First, Carrasco et al. [5] designed a "signature" for each state. We can say the signature plays the role of states "behaviour" in the minimization process.

After that, Abdulla et al. [1] use another way to identify states behaviour in order to provide a NFTA minimization. They compute a composed bisimulation relation which composes downward and upward bisimulation relations. The authors reduce the computation of the upward bisimulation to the resolution of word finite transition systems.

The authors prove that for this transformation, the computation of the upward bisimulation can be done by computing an analog equivalence function on the word TS. This can be done using Tarjan-Paige algorithm [9].

For DFTA, this transformation holds using results of Högberg et al. [8]. 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. that creates a valid string DFA to prove then that it can replace the DFTA to be minimized in any of minimization techniques and then proving that there is no need to re-implement 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 better 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 $\mathscr{A}$ and then we regroup states which are possibly equivalent according to the equivalence relation $\equiv$. We show that $(p \equiv q \Rightarrow p \sim q)$.

Next, using the relation $\sim$ we construct a string automaton $M_{\mathscr{A}}$ using the same states as $\mathscr{A}$. Then we prove that two states in $M_{\mathscr{A}}$ 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 $\rho \in \Delta$, 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 1 *Let $\mathscr{A} = (Q, \Sigma, Q_f, \Delta)$ be DFTA. The horizontal language of Δ noted L_Δ is defined as follows:*

$$L_\Delta = \bigcup_{\rho \in \Delta} L_\rho \tag{4}$$

$$L_\rho = \bigcup_{i=1}^{n} q_i f q_1 \ldots q_{i-1} \bullet q_{i+1} \ldots q_n \tag{5}$$

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

Definition 2 *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 $p \equiv q \Rightarrow p \sim q$*

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

Now, 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.

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

Proposition 2 *We have* $|\bigcup_{q \in Q} \sigma_q| \leq |\mathscr{A}|$

The automaton $M_{\mathscr{A}}$ will be defined on the alphabet $\sigma = (\bigcup_{q \in Q} \sigma_q)$. It is clear that the size of the alphabet σ depends on the number of equivalence classes in $\sim$.

Indeed, the new alphabet coincides with the environment defined by Abdullah et al. [1] and we can construct an FA which represents the same transitions system. However, in deterministic case, there is no need to add transitions between states and environment because in this case the left and right sides are equal. Just transitions from environment to the output states are considered. It is possible to show that this construction minimizes the wanted DFTA using the fact that DFTA can be minimized using upward bisimulation (see Högberg et al. works [8]) and then using Paige-Tarjan algorithm to achieve it. But, we prove below the equivalence between Nerode equivalence in the string and tree automata and we show that it is beneficial in other minimization algorithm especially in the incremental construction of tree automata.

Definition 3 *Let $\mathscr{A} = (Q, \Sigma, Q_f, \Delta)$ be a DFTA. The string automaton associated to $\mathscr{A}$ noted $M_{\mathscr{A}}$ is the tuple $M_{\mathscr{A}} = (Q', \sigma, \{i_s\}, F, \delta)$ 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: $\delta(i_s, q) = q$.*

Let us 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 a habitual string automaton because it is usual to define string automata with an initial state. 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 those states.

In the next section, we will prove that the associated string FA can fully replace DFTA minimization the minimization processes namely standard, acyclic, incremental and incrementally constructed minimization and then avoid the re-implementation of those algorithms. We discuss also the complexity of this transformation and we show that it have no negative influence on the time and space process.

5 Minimization techniques using the associated FA

In what follows, we will show and discuss how that the associated FA can fully replace the FTA to be minimized in the specified deterministic minimization techniques namely standard, acyclic, incremental and incrementally constructed minimization. We prove also that in some cases (Acyclic and incremetal techniques) the complexities are improved.

5.1 DFTA standard minimization

We show in this part that the minimization of a given DFTA is no more than minimizing its associated string DFA. But first, let us show that the FA is deterministic.

Proposition 3 *The associated string automaton $M_{\mathscr{A}}$ of a DFTA $\mathscr{A}$ 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 $M_{\mathscr{A}}$ is defined, we show that the computation of the equivalence relation $\equiv$ defined on $\mathscr{A}$ can be done by computing the Nerode equivalence $\cong$ relation defined on $M_{\mathscr{A}}$. $\cong$ is defined as follows.

$$p \cong q \Leftrightarrow \begin{cases} p \in F \Leftrightarrow q \in F \\ \text{for all } a \in \sigma, \delta(p,a) \cong \delta(q,a) \end{cases} \tag{6}$$

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

We can also check this corollary:

Corollary 1 *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 is $[q]$. Using Hopcroft Algorithm [11], the automaton $M_{\mathscr{A}}$ can be minimized in $O(|\sigma||Q|\log|Q|)$.

Lemma 3 *The automaton $M_{\mathscr{A}}$ can be minimized in $O(|\sigma||Q|\log|Q|)$.*

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

Note that this complexity is the same as if Abdulla et al. [1] reduction is combined with results of Högberg et al. [8]. The only difference here is that the complexity is given in function of the string automaton which is the output of the transformation algorithm.

5.2 Acyclic minimization

Acyclic automata are a beneficial data structure that represent and store finite set of objects. When objects are trees like in Xml, it is useful to store a finite Xml files in a compact and small structure. Acyclic DFA (ADFA) minimization was largely discussed like in [13, 10] and almost of these techniques have linear asymptotic complexity. Here we show how the associated FA can positively minimize an acyclic DFTA.

Although Proposition 4 is sufficient for proving the use of the associated DFA to minimize ADFTA, but let us recall some useful definitions.

Definition 4 *Let $\mathscr{A} = (Q,\Sigma,Q_f,\Delta)$ be a DFTA. Then $\mathscr{A}$ 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 (the proof is trivial and then omitted).

Lemma 4 *The associated DFA of a ADFTA is acyclic.*

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

Thus, after computing the associated string ADFA $\mathscr{M}_{\mathscr{A}}$ of a DFTA $\mathscr{A}$ using one of the string acyclic minimizations like [10] in linear time:

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

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 and producing a valid one which recognizes the same language as the departure automaton.

In string case, Watson et al. introduce for the first time the incremental version for cyclic DFA, but the complexity as reported by authors is exponential. Next, Watson et al. [14] improve this algorithm and give an almost quadratic implementation. After that, Almeida et al. [2] present the best known incremental implementation using the UNION-FIND algorithm.

However, in tree case, Cleophas et al. generalize the incremental approach to trees and give a cubic implementation to this need.

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

Let $\mathscr{A} = (Q, \Sigma, Q_f, \Delta)$ be a DFTA and $\mathscr{M}_{\mathscr{A}} = (Q', \sigma, \{i_s\}, F, \delta)$ 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 5 *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 $\mathscr{M}_{\mathscr{A}}$ instead of computing it in $\mathscr{A}$.

Thus, we can use the best-known complexity algorithm for DFA due to Almeida et al. [2] to minimize a DFTA by incremntally minimizing its associted DFA:

Theorem 3 *A DFTA $\mathscr{A} = (Q, \Sigma, Q_f, \Delta)$ can be incrementally minimized using its associated DFA $\mathscr{M}_{\mathscr{A}} = (Q', \sigma, \{i_s\}, F, \delta)$ in $\mathscr{O}(|\sigma||Q|^2\alpha(|Q|))$ where α is the inverse Ackermann time function.*

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 to add or delete words (resp. trees) to an existing minimal automaton. In other words if $\mathscr{A}$ is a DFA (resp. DFTA) and w is a word (resp. t is a tree) then the incremental construction consists on creating a new automaton which recognizes $L(\mathscr{A}) \cup \{w\}$ (resp. $L(\mathscr{A}) \cup \{t\}$) while maintaining minimality. First, incremental construction was presented by Daciuk et al. [**?**] for ADFA. Next, Carrasco et al. [7] generalize this notion to cyclic DFA. Later, they redefine the incremental construction for trees in [6].

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

6 Conclusion

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 which is considered as well-studied since the 60s. Indeed, we use the environment (and the TS transformation) notion proposed by Abdulla et al. [1] to create a string alphabet which is read by an associated string automaton and then minimize it. Hence, we prove that there is actually no need to implement existing algortihms proposed for trees and exploit the large range of minimization algorithms for strings to add minimization procedures in tree toolkits. Moreover, We prove that DFTA minimization can be done in $\mathscr{O}(|\mathscr{A}| + |\sigma||Q|\log(|Q|))$, where σ is the alphabet of the DFA $M_{\mathscr{A}}$ associated to $\mathscr{A}$ for standard minimization (which is considered in term of asymptotic complexity as the same as the best known one) and we show that the minimization using associated DFA gives better complexities in other existing minimization approaches namely acyclic and incremental minimization (which are clearly improved in this present paper).

In fact, it is interesting to study the average size of σ. This leads us to consider the problem of random generation of deterministic tree automata. But instead of the existing random generators in literature, no real generator is developed. We hope that we consolidate this work with an experimental tests and comparisons with other techniques after developing a such generator.

References

[1] Parosh Aziz Abdulla, Ahmed Bouajjani, Lukás Holík, Lisa Kaati & Tomás Vojnar (2009): *Composed Bisimulation for Tree Automata*. Int. J. Found. Comput. Sci. 20(4), pp. 685–700, doi:10.1142/S0129054109006814.

[2] Marco Almeida, Nelma Moreira & Rogrio Reis (2010): *Incremental DFA Minimisation*. In Michael Domaratzki & Kai Salomaa, editors: *CIAA*, Lecture Notes in Computer Science 6482, Springer, pp. 39–48, doi:10.1007/978-3-642-18098-9_5.

[3] Michael A. Arbib & Yehoshafat Give'on (1968): *Algebra Automata I: Parallel Programming as a Prolegomena to the Categorical Approach*. Information and Control 12(4), pp. 331–345, doi:10.1016/S0019-9958(68)90374-4.

[4] Walter S. Brainerd (1968): *The Minimalization of Tree Automata*. Information and Control 13(5), pp. 484–491, doi:10.1016/S0019-9958(68)90917-0.

[5] Rafael C. Carrasco, Jan Daciuk & Mikel L. Forcada (2007): *An Implementation of Deterministic Tree Automata Minimization*. In Jan Holub & Jan Zdarek, editors: *CIAA*, Lecture Notes in Computer Science 4783, Springer, pp. 122–129, doi:10.1007/978-3-540-76336-9_13.

[6] Rafael C. Carrasco, Jan Daciuk & Mikel L. Forcada (2009): *Incremental Construction of Minimal Tree Automata*. Algorithmica 55(1), pp. 95–110, doi:10.1007/s00453-008-9172-4.

[7] Rafael C. Carrasco & Mikel L. Forcada (2002): *Incremental Construction and Maintenance of Minimal Finite-State Automata*. Computational Linguistics 28(2), pp. 207–216, doi:10.1162/089120102760173652.

[8] Johanna Högberg, Andreas Maletti & Jonathan May (2009): *Backward and forward bisimulation minimization of tree automata*. Theor. Comput. Sci. 410(37), pp. 3539–3552, doi:10.1016/j.tcs.2009.03.022.

[9] Robert Paige & Robert Endre Tarjan (1987): *Three Partition Refinement Algorithms*. SIAM J. Comput. 16(6), pp. 973–989, doi:10.1137/0216062.

[10] Dominique Revuz (1992): *Minimisation of Acyclic Deterministic Automata in Linear Time*. Theor. Comput. Sci. 92(1), pp. 181–189, doi:10.1016/0304-3975(92)90142-3.

[11] Antti Valmari (2012): *Fast Brief Practical DFA Minimization*. Inf. Process. Lett. 112(6), pp. 213–217, doi:10.1016/j.ipl.2011.12.004.

[12] Bruce W. Watson (1995): *Taxonomies and Toolkits of Regular Language Algorithms*. Ph.D. thesis, Faculty of Mathematics and Computer Science, Eindhoven University of Technology, doi:10.6100/IR444299.

[13] Bruce W. Watson (2003): *A new algorithm for the construction of minimal acyclic DFAs*. Sci. Comput. Program. 48(2-3), pp. 81–97, doi:10.1016/S0167-6423(03)00012-1.

[14] Bruce W. Watson & Jan Daciuk (2003): *An efficient incremental DFA minimization algorithm*. Natural Language Engineering 9(1), pp. 49–64, doi:10.1017/S1351324903003127.

[15] Silvano Zilio & Denis Lugiez (2003): *XML Schema, Tree Logic and Sheaves Automata*. In Robert Nieuwenhuis, editor: Rewriting Techniques and Applications, Lecture Notes in Computer Science 2706, Springer Berlin Heidelberg, doi:10.1007/s00200-006-0016-7.

A Tree automata minimization algorithm

B Proofs of section 4

B.1 Proof of Proposition 1

Proof 1 *Let $p,q \in Q$ such that $p \not\sim q$. Then, there exists $pfq_1 \dots q_{i-1} \bullet q_{i+1} \dots q_n \in L_\Delta$ but $pfq_1 \dots q_{i-1} \bullet q_{i+1} \dots q_n \notin L_\Delta$. Using Definition 1 then we have $(f, q_1, \dots, q_{i-1}, p, q_{i+1}, \dots, q_n, p') \in \Delta$ but no transition with the form $(f, q_1, \dots, q_{i-1}, q, q_{i+1}, \dots, q_n, q') \in \Delta$. Here, we have $(f, q_1, \dots, q_{i-1}, p, q_{i+1}, \dots, q_n) \in$*

Algorithm 1 Tree automata minimization

```
1:  function MINIMIZATION(DFTA $\mathscr{A} = (Q, \Sigma, Q_f, \Delta)$)
2:      $P_0 \leftarrow Q$
3:      $P_1 \leftarrow \{Q_f, Q - Q_f\}$
4:      $i \leftarrow 1$
5:      repeat
6:          create $P_{i+1}$ by refining $P_i$ so that $p \equiv_{i+1} q$ iff for all $\rho \in \Gamma(p), i \in occ_p(\rho) : \rho(p :_i q) \in \Gamma(q)$
7:          $d(\rho) \equiv d(\rho(p :_i q))$
8:          $i \leftarrow i + 1$
9:      until $(P_i = P_{i-1})$
10:     $Q_{min} \leftarrow \bigcup_{q \in Q} [q]$
11:     $\Delta_{min} \leftarrow \{(f, [q_1], \ldots, [q_n]) \mid (f, q_1, \ldots, q_n) \in \Delta\}$
12:     $Q_{min\ f} \leftarrow \{[q] \mid q \in Q_f\}$
13:     return $\mathscr{A}_{min} = (Q_{min}, \Sigma, Q_{min\ f}, \Delta_{min})$
14: end function
```

$\Gamma(p)$ but $(f, q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n)(p :_i q) \notin \Gamma(q)$ 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 $p \not\equiv q$.

B.2 Proof of Lemma 2

Proof 2 *As the language L_Δ is finite, equivalence relations $\sim$ can be computed in linear time on the size of $|\mathscr{A}|$. This can be done by minimizing an acyclic automaton which reads L_Δ.*

C Proofs of section 5

C.1 Proof of Proposition 3

Proof 3 *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 assume that there exists a symbol $f_{u,v} \in \sigma$ such that $\delta(q,a) = \{q', q''\}$ with $q' \neq q''$. Let $u = q_1 \ldots q_{i-1}$ and $v = q_{i+1} \ldots q_n$. Using Definition 1 we have then $(f, q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n, q') \in \Delta$ and $(f, q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n, q'') \in \Delta$. This leads to a contradiction because the tree automaton $\mathscr{A}$ is deterministic.*

C.2 Proof of Proposition 4

Proof 4 *It is well known that the Nerode equivalence $\cong$ can be computed by successive approximations $\cong_j$ defined as:*

$$p \cong_0 q \quad \text{iff} \quad (p \in F \Leftrightarrow q \in F) \tag{7}$$

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

Here, and as mentioned below, useless states which appear in the DFA are equivalent and distort the correctness of the approach. The solution is to initialize $\cong_0$ with what follows. $\cong_0 = \overline{Q}^2 \cup \{(q,q) \mid q \in Q - \overline{Q}\}$.

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 done by induction on the definitions of $\equiv_j$ and $\cong_j$. For the basic case $(j = 0)$, as $Q_f = F$, for $p,q \in Q$, we have $p \cong_0 q \Leftrightarrow p \equiv_0 q$. Assume 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

$$(p \cong_{k+1} q) \quad \Leftrightarrow \quad p \cong_k q \textit{ and for all } f_{u,v} \in \sigma, \delta(p, f_{u,v}) \cong_k \delta(q, f_{u,v}) \tag{9}$$

By applying induction hypothesis, we get that

$$(p \cong_{k+1} q) \quad \Leftrightarrow \quad p \equiv_k q \textit{ and for all } f_{u,v} \in \sigma, \delta(p, f_{u,v}) \equiv_k \delta(q, f_{u,v}) \tag{10}$$

Let $u = q_1 \cdots q_{i-1}$ and $v = q_{i+1} \cdots q_n$. Using the horizontal language definition we get that $pfu \bullet v, qfu \bullet v \in L_\Delta$. So, there exists 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 3 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 then $(p \equiv_{k+1} q)$ by applying (13).
Now, 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 (this part is omitted).