An implementation of deterministic tree automata minimization

Rafael C. Carrasco[1], Jan Daciuk[2], and Mikel L. Forcada[3]

[1] Dep. de Lenguajes y Sistemas Informáticos, Universidad de Alicante, E-03071 Alicante, Spain (carrasco@dlsi.ua.es)
[2] Knowledge Engineering Department, Gdańsk University of Technology, Ul. G. Narutowicza 11/12, 80-952 Gdańsk, Poland (jandac@eti.pg.gda.pl)
[3] Dep. de Llenguatges i Sistemes informàtics, Universitat d'Alacant, E-03071 Alacant, Spain (mlf@dlsi.ua.es).

Abstract. A frontier-to-root deterministic finite-state tree automaton (DTA) can be used as a compact data structure to store collections of unranked ordered trees. DTAs are usually sparser than string automata, as most transitions are undefined and therefore, special care must be taken in order to minimize them efficiently. However, it is difficult to find simple and detailed descriptions of the minimization procedure in the published literature. Here, we fully describe a simple implementation of the standard minimization algorithm that needs a time in $\mathcal{O}(|A|^2)$, with $|A|$ being the size of the DTA.

1 Introduction

A data structure that stores unranked ordered tree data efficiently is a minimal frontier-to-root deterministic tree automaton (DTA) where each subtree which is common to several trees in the collection is assigned a single state. Furthermore, the number of such states is minimized by assigning a single state to groups of subtrees that may appear interchangeably in the collection. The general procedure to obtain a minimal DTA is well known [1–3]. However, it is difficult to find detailed descriptions of the minimization algorithm. Here, we describe a simple and efficient implementation of the algorithm to minimize DTAs.

Given an *alphabet*, that is, a finite set of symbols $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$, we define T_Σ as the set of unranked ordered trees with labels in Σ: every symbol $\sigma \in \Sigma$ belongs to T_Σ and every $\sigma(t_1 \cdots t_m)$ with $\sigma \in \Sigma$, $m > 0$ and $t_1, \ldots, t_m \in T_\Sigma$ is also a tree in T_Σ. The trees so defined are ordered and unranked, that is, the order of descendents $t_1, ..., t_m$ is relevant but symbols in Σ are not assigned a fixed *valence* m. Any subset of T_Σ will be called a *tree language*. In particular, the language of subtrees $\mathrm{sub}(t)$ of $t = \sigma(t_1, ..., t_m)$ is the union of $\{t\}$ and $\bigcup_{k=1}^{m} \mathrm{sub}(t_k)$.

A *finite-state frontier-to-root tree automaton* is defined as $A = (Q, \Sigma, \Delta, F)$, where $Q = \{q_1, \ldots, q_{|Q|}\}$ is a finite set of *states*, $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ is the *alphabet*, $F \subseteq Q$ is the subset of *accepting states*, and $\Delta = \{\tau_1, \ldots, \tau_{|\Delta|}\} \subset \cup_{m=0}^{\infty} \Sigma \times Q^{m+1}$ is a finite set of *transitions*.

A tree automaton A is a *deterministic finite-state frontier-to-root tree automaton*, or deterministic tree automaton (DTA) for short, if for every argument $(\sigma, i_1, ..., i_m)$ there is at most one possible output, that is, for all $\sigma \in \Sigma$, for all $m \geq 0$ and for all $(i_1, ..., i_m) \in Q^m$, there is at most one $j \in Q$ such that $(\sigma, i_1, ..., i_m, j) \in \Delta$. In a DTA, the transition output for argument $(\sigma, i_1, ..., i_m)$ is

$$\delta_m(\sigma, i_1, ..., i_m) = \begin{cases} j & \text{if } (\sigma, i_1, ..., i_m, j) \in \Delta \\ \perp & \text{if no such } j \text{ exists} \end{cases} \tag{1}$$

where the symbol $\perp$ will be interpreted as a special *absorption state* such that $\perp \in Q - F$ but cannot appear in Δ. With this convention, Δ remains finite but the output for all possible transition arguments is a state in Q. The *size* of the DTA is defined as the size of its transition function (which, in contrast with string automata, cannot be directly obtained from its number of transitions, that is,

$$|A| = \sum_{n=1}^{|\Delta|} |\tau_n|, \tag{2}$$

with $|(\sigma, i_1, ..., i_m, j)| = m + 1$.

The output $A(t)$ when DTA A operates on $t \in T_\Sigma$ is the state in Q recursively computed as

$$A(t) = \begin{cases} \delta_0(\sigma) & \text{if } t = \sigma \in \Sigma \\ \delta_m(\sigma, A(t_1), \ldots, A(t_m)) & \text{if } t = \sigma(t_1 \cdots t_m) \in T_\Sigma - \Sigma \end{cases} \tag{3}$$

The tree *language* $L_A(q)$ *accepted* at state $q \in Q$ is the subset of T_Σ with output q

$$L_A(q) = \{t \in T_\Sigma : A(t) = q\} \tag{4}$$

and the tree language $L(A)$ accepted by A is the subset of trees in T_Σ accepted at the states in F

$$L(A) = \bigcup_{q \in F} L_A(q) = \{t \in T_\Sigma : A(t) \in F\}. \tag{5}$$

In a DTA A, a state q is *inaccessible* if $L_A(q) = \emptyset$, that is, if there is no tree t in T_Σ such that $A(t) = q$. Therefore, inaccessible states and the transitions using them are useless and can be safely removed from Q and Δ respectively without affecting $L(A)$. It is worth to note that in DTAs the absorption state $\perp$ is always accessible because Δ is a finite subset of $\bigcup_{m=0}^{\infty} \Sigma \times Q^{m+1}$ and there is an infinite number of arguments leading to $\perp$. A DTA with no inaccessible state is said to be *reduced*.

An accessible state $q \in Q$ is said to be *coaccessible* in A if there is at least one tree $t \in L(A)$ containing a subtree s such that $q = A(s)$. States which are not coaccessible (and accessible) are *useless*. In particular, as no transition in Δ contains $\perp$, the absorption state is useless. As will be shown later, the identification of inaccessible and useless states can be done in time $\mathcal{O}(|A|)$.

2 Minimal deterministic tree automata

The standard procedure to minimize [1–3] a deterministic tree automaton A removes its inaccessible states and then merges all its equivalent states.

On the one hand, the subset $I \subseteq Q$ of inaccessible states can be easily identified by means of an iterative procedure: start with $n \leftarrow 0$ and $I_0 \leftarrow Q$ and while there is a transition $(\sigma, i_1, ..., i_m, j) \in \Delta$ such that $j \in I_n$ and $(i_1, ..., i_m) \in (Q - I_n)^m$, make $I_{n+1} \leftarrow I_n - \{j\}$ and $n \leftarrow n + 1$. A detailed implementation of this procedure, which runs in time $\mathcal{O}(|A|)$, is shown in Figure 1.

`Algorithm findInaccessible`
Input: A DTA $A = (Q, \Sigma, \Delta, F)$
Output: The subset of inaccessible states in A.
Method:

1. For all q in Q create an empty list R_q.
2. For all $\tau_n = (\sigma, i_1, ..., i_m, j)$ in Δ do
   - $B_n \leftarrow m$ (* Store num. of inaccessible positions in argument of τ_n *).
   - For $k = 1, ..., m$ append n to R_{i_k} (* Store all occurrences in $i_1, ...i_m$ *).
3. $K \leftarrow \{\delta_0(\sigma) : \sigma \in \Sigma\}$; $I \leftarrow Q - K$
4. While $K \neq \emptyset$ and $I \neq \emptyset$ remove a state q from K and for all n in R_q do
   - $B_n \leftarrow B_n - 1$
   - If $B_n = 0$ and output$(\tau_n) \in I$ then move output(τ_n) from I to K.
5. Return $I - \{\perp\}$.

Fig. 1: Algorithm for the identification of inaccessible states in a DTA.

On the other hand, equivalent states can be found, as shown in figure 2, by creating a partition $P_0 = (Q)$ and iteratively refining this partition until it becomes a congruence. A *congruence* $\simeq$ on A is an equivalence relation such that $p \simeq q$ implies:

1. $p \in F$ if and only if $q \in F$.
2. If $m > 0$, $k \leq m$ and $(\sigma, r_1, ..., r_m) \in \Sigma \times Q^m$ then

$$\delta_m(\sigma, r_1, \ldots, r_{k-1}, p, r_{k+1} \ldots, r_m) \simeq \delta_m(\sigma, r_1, \ldots, r_{k-1}, q, r_{k+1} \ldots, r_m) \quad (6)$$

In other words, the equivalence relation is closed under context and, thus, equivalent states are interchangeable as the output of automaton A on any tree or subtree without any effect on $L(A)$.

As the standard algorithms [4–6] for the minimization of deterministic finite-state automata do, the minimization of DTAs partitions the set of states Q into equivalence classes by iterative refinement. In the following, P_n will denote the partition at iteration n, $\Phi_n[p]$ will denote the class in P_n that contains p and we

will write $p \sim_n q$ if and only if $\Phi_n[p] = \Phi_n[q]$. We will say that P_n is *inconsistent* if there exist $m > 0$, $k \leq m$ and $(\sigma, r_1, ..., r_m) \in \Sigma \times Q^m$ such that

$$\delta_m(\sigma, r_1, \ldots, r_{k-1}, p, r_{k+1} \ldots, r_m) \not\sim_n \delta_m(\sigma, r_1, \ldots, r_{k-1}, q, r_{k+1} \ldots, r_m) \quad (7)$$

Then, the standard algorithm refines the partition until it becomes consistent, as shown in Figure 2.

`Algorithm minimizeDTA`
Input: a reduced DTA $A = (Q, \Sigma, \Delta, F)$ with $F \neq \emptyset$.
Output: a minimal DTA $A^{\min} = (Q^{\min}, \Sigma, \Delta^{\min}, F^{\min})$ equivalent to A.
Method:

1. Create the initial partition $P_0 \leftarrow (Q)$, $P_1 \leftarrow (F, Q - F)$ and set $n \leftarrow 1$.
2. While $P_n \neq P_{n-1}$ create P_{n+1} by refining P_n so that $p \sim_{n+1} q$ if and only if for all $m > 0$, for all $k \leq m$ and for all $(\sigma, r_1, ..., r_m) \in \Sigma \times Q^m$

$$\delta_m(\sigma, r_1, \ldots, r_{k-1}, p, r_{k+1} \ldots, r_m) \sim_n \delta_m(\sigma, r_1, \ldots, r_{k-1}, q, r_{k+1} \ldots, r_m)$$

3. Output $(Q^{\min}, \Sigma, \Delta^{\min}, F^{\min})$ with
   - $Q^{\min} = \{\Phi_n[q] : q \in Q\}$;
   - $F^{\min} = \{\Phi_n[q] : q \in F\}$;
   - $\Delta^{\min} = \{(\sigma, \Phi_n[i_1], ..., \Phi_n[i_m], \Phi_n[j]) : (\sigma, i_1, ..., i_m, j) \in \Delta \wedge j \not\sim_n \perp\}$.

Fig. 2: Standard algorithm for DTA minimization. Function $\Phi_n(q)$ returns the identifier of the class in P_n that contains q.

However, the efficient implementation of this procedure requires a fast method to search for arguments $(\sigma, r_1, ..., r_m) \in \Sigma \times Q^m$ where replacing r_k with p and q leads to non-equivalent outputs and a correct treatment of undefined transitions. The implementation of the algorithm shown in Figure 4 meets these requirements and realizes it as follows.

- Initialization: All useless states in the automaton are replaced by a single one (the absorption state $\perp$) and then, the partition P is initialized with classes where all states have identical signature, the *signature* of a state being a set defined as

$$\text{sig}(q) = \begin{cases} \{(\sigma, m, k) : \exists(\sigma, i_1, ..., i_m, j) \in \Delta : i_k = q\} \cup \{(\#, 1, 1)\} & \text{if } q \in F \\ \{(\sigma, m, k) : \exists(\sigma, i_1, ..., i_m, j) \in \Delta : i_k = q\} & \text{otherwise} \end{cases}$$

  where $\#$ is a symbol not in Σ used to distinguish accepting and non-accepting states. Useless states can be identified in time $\mathcal{O}(|A|)$ by means of the procedure shown in Figure 3.

– The main loop refines the partition P_n at every iteration and keeps a queue K containing representatives of the new classes in the partition. It makes use of a function $\text{next}_n(i)$ that returns the state following i in class $\Phi_n[i]$ or the first state in $\Phi_n[i]$ if i is the last state in that class (a fixed, arbitrary order of states is assumed).

Merging all useless states with $\perp$ is supported by the fact that, once inaccessible states are removed, q is coaccessible if and only if $q \not\approx \perp$. On the other hand, it is clear that after removing useless states, $\text{sig}(p) \neq \text{sig}(q) \Rightarrow p \not\approx q$ and π can be safely initialized with the classes of states with identical signature.

`Algorithm findUseless`
Input: A reduced DTA $A = (Q, \Sigma, \Delta, F)$ with $F \neq \emptyset$.
Output: The subset of useless states in A.
Method:

1. For all q in Q create an empty list L_q.
2. For all $\tau_n = (\sigma, i_1, ..., i_m, j)$ in Δ add n to L_j (* Store n such that j is the output of τ_n *).
3. $K \leftarrow F$; $U \leftarrow Q - F$
4. While $K \neq \emptyset$ and $U \neq \emptyset$ remove a state q from K and for all n in L_q and for all i_k in $\{i_1, ..., i_m\}$ do
   – If $i_k \in U$ then then move i_k from U to K.
5. Return U.

Fig. 3: Algorithm for the identification of useless states in a DTA.

The correctness of the main loop requires that all inequivalent pairs are eventually found through the search at step 2. Indeed, according to eq. (6), if $p \not\sim_{n+1} q$ there exist $m > 0$, $k \leq m$ and $(\sigma, r_1, ..., r_m, j) \in \Sigma \times Q^{m+1}$ with $r_k = p$ such that

$$\delta_m(\sigma, r_1, \ldots, r_{k-1}, q, r_{k+1} \ldots, r_m) \not\sim_n j.$$

Let us assume that $j \not\approx \perp$ (otherwise, one can exchange p and q) and write $p^{[1]} = p$ and, for $s > 0$, $p^{[s+1]} = \text{next}(p^{[s]})$. Then, there is a value of $s > 0$ such that

$$\delta_m(\sigma, r_1, \ldots, r_{k-1}, p^{[s]}, r_{k+1} \ldots, r_m) \sim_n j$$

and

$$\delta_m(\sigma, r_1, \ldots, r_{k-1}, p^{[s+1]}, r_{k+1} \ldots, r_m) \not\sim_n j.$$

Therefore, the check performed at step 2 of the minimization algorithm over all $m > 0$, all $k \leq m$ and all transitions in $\Sigma \times Q^m$ can be limited to those transitions in Δ and every $(\sigma, i_1, ..., i_m, j) \in \Delta$ needs only to be compared with m transitions of the type $(\sigma, i_1, ..., \text{next}(i_k), ...i_m, j')$

`Algorithm minimizeDTA`
Input: a DTA $A = (Q, \Sigma, \Delta, F)$ without inaccessible states.
Output: a minimal DTA $A^{\min} = (Q^{\min}, \Sigma, \Delta^{\min}, F^{\min})$.
Method:

1. (* Initialize π and K *)
   - Remove useless states from Q and transitions using them from Δ and set $Q \leftarrow Q \cup \{\perp\}$ and $n \leftarrow 1$.
   - For all $(\sigma, i_1, ..., i_m) \in \Delta$ add (σ, m, k) to $\text{sig}(i_k)$ for $k = 1, ..., m$.
   - For all $q \in F$ add $(\#, 1, 1)$ to $\text{sig}(q)$.
   - Create an empty set B_{sig} for every different signature sig and for all $q \in Q$ add q to set $B_{\text{sig}(q)}$.
   - Set $P_0 \leftarrow (Q)$ and $P_1 \leftarrow \{B_s : B_s \neq \emptyset\}$.
   - Enqueue in K the first element from every class in P_1.
2. While K is not empty
   (a) Remove the first state q in K.
   (b) For all $(\sigma, i_1, ..., i_m, j) \in \Delta$ such that $j \sim_n q$ and for all $k \leq m$
      i. If $\delta_m(\sigma, i_1, ..., \text{next}_n(i_k), ..., i_m) \not\sim_n j$ then
         A. Create P_{n+1} from P_n by splitting $\Phi_n[i_k]$ into so many subsets as different classes $\Phi_n[\delta_m(\sigma, i_1, ., i'_k, .., i_m)]$ are found for all $i'_k \in \Phi_n[i_k]$.
         B. Add to K the first element from every subset created at the previous step.
         C. Set $n \leftarrow n + 1$.
3. Output $(Q^{\min}, \Sigma, \Delta^{\min}, F^{\min})$ with
   - $Q^{\min} = \{\Phi_n[q] : q \in Q\}$;
   - $F^{\min} = \{\Phi_n[q] : q \in F\}$;
   - $\Delta^{\min} = \{(\sigma, \Phi_n[i_1], ..., \Phi_n[i_m], \Phi_n[j]) : (\sigma, i_1, ..., i_m, j) \in \Delta \wedge \Phi_n[j] \neq \Phi_n[\perp]\}$.

Fig. 4: Modified algorithm `minimizeDTA`.

Finally, this minimization algorithm runs in time $\mathcal{O}(|A|^2)$, as can be easily checked if we take into account that, in a DTA without inaccessible states, $|Q| \leq |A|$ and:

- A state may enter K every time a finer class is created in the partition. As the refinement process cannot create more than $2|Q| - 1$ different classes (the number of nodes in a binary tree with $|Q|$ leaves), the main loop, which always removes a state from K, performs at most $2|Q| - 1$ iterations.
- At every iteration, a loop over some transitions in Δ and their arguments is performed: obviously, this internal loop involves at most $|A|$ iterations.
- If $\delta_m(\sigma, i_1, ..., \text{next}_n(i_k), ..., i_m) \not\sim_n j$ then class $\Phi_n[i_k]$ becomes split and its states are classified according to the transition output using less than $|Q|$ steps; also updating K adds at most $|Q|$ states. As the maximum number of splits is $|Q| - 1$, the conditional block involves at most $|Q|^2$ steps.

This theoretical bound has been tested by appliyng the algorithm to compress an acyclic DTA accepting parse trees (upto 20000 trees and 60 labels) obtained from a tree bank [7]. The results, depicted in figure 5, show that the time needed to minimize the DTA grows less than quadratically with the size of the automaton (the best fit for this example is $|A|^{1.47}$).

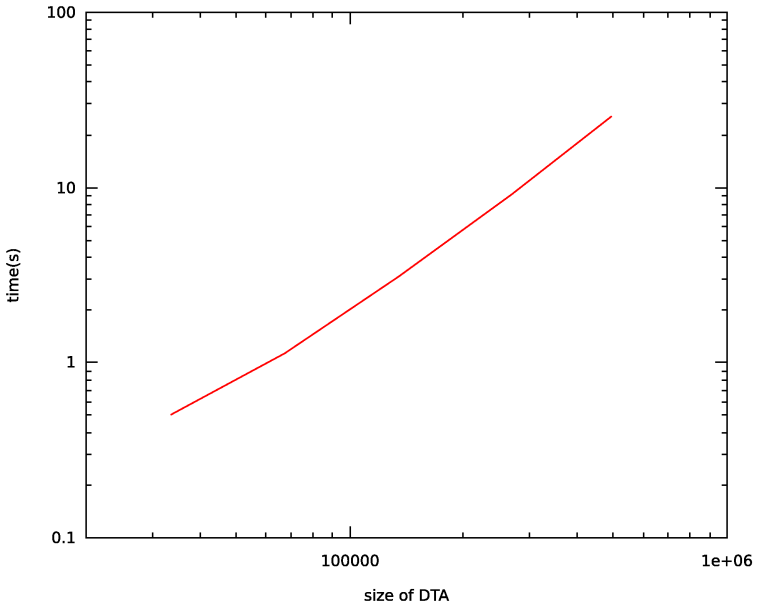

Fig. 5: Time needed to minimize a DTA as a function of the size of the DTA.

3 Conclusion

We presented a simple implementation of the standard algorithm for the minimization of deterministic frontier-to-root tree automata which runs in time $\mathcal{O}(|A|^2)$ by showing that the search for inconsistent classes can be efficiently performed and that undefined transitions and the absorption state can be properly handled. As the partition may be initialized with more than two classes and also subsequent refinements beyond binary splitting are possible the convergence is usually reached with fewer iterations. The question if a modification exists with better asymptotic behavior, such as those applicable for sparse string automata [8], remains open. Incremental minimization of DTAs, that is, the construction of a minimal DTA by adding new trees to the language accepted by an existing one, will be addressed elsewhere [9].

Acknowledgments. Work supported by the Spanish CICyT through grant TIN2006-15071-C03-01.

References

1. Brainerd, W.S.: The minimalization of tree automata. Information and Control **13**(5) (1968) 484–491
2. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984)
3. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata` (1997) release October, 1rst 2002.
4. Hopcroft, J., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison–Wesley, Reading, MA (1979)
5. Blum, N.: An O(n log n) implementation of the standard method for minimizing n-state finite automata. Information Processing Letters **57**(2) (1996) 65–69
6. Watson, B.W.: A taxonomy of finite automata minimization algorithmes. Computing Science Note 93/44, Eindhoven University of Technology, The Netherlands (1993)
7. Marcus, M.P., Santorini, B., Marcinkiewicz, M.: Building a large annotated corpus of english: the penn treebank. Computational Linguistics **19** (1993) 313–330
8. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Computing **16**(6) (1987) 973–989
9. Carrasco, R.C., Daciuk, J., Forcada, M.L.: Incremental construction of minimal tree automata. Submitted (2007)