An implementation of deterministic tree automata minimization

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

[1]Universidad de Alicante

[2]Gdańsk University of Technology

Prague, July 16, 2007

Abstract

DTAs are highly sparse (most transitions are undefined), equivalence of states depends on multiple inputs, and care must be taken in order to minimize them efficiently.
We fully describe a simple implementation of the standard minimization algorithm that needs a time in $\mathcal{O}(|A|^2)$.

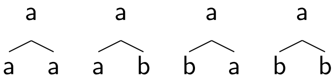
DTA as compact data structure/1

Minimal DTA can store (unranked ordered) tree data efficiently:

1. Each subtree which is common to several trees is assigned a single state.
2. A single state is assigned to groups of subtrees that may appear interchangeably in the collection.

DTA as compact data structure/2

- Sample:



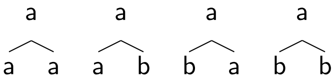
- States: $\{1, 2, \perp\}$
- Alphabet of labels: $\{a, b\}$
- Accepting states: $\{2\}$
- Transitions $\{(a, 1), (b, 1), (a, 1, 1, 2)\}$.

Bottom-up computations:

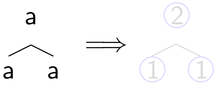
$\delta_0(a) = 1$
$\delta_2(a, 1, 1) = 2$

$\delta_0(a) = 1$
$\delta_1(a, 1) = \perp$

DTA as compact data structure/2

- Sample: 

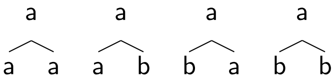
- States: $\{1, 2, \perp\}$
- Alphabet of labels: $\{a, b\}$
- Accepting states: $\{2\}$
- Transitions $\{(a, 1), (b, 1), (a, 1, 1, 2)\}$.

Bottom-up computations: 

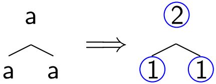
$\delta_0(a) = 1$
$\delta_2(a, 1, 1) = 2$

$\delta_0(a) = 1$
$\delta_1(a, 1) = \perp$

DTA as compact data structure/2

- Sample: 

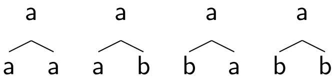
- States: $\{1, 2, \perp\}$
- Alphabet of labels: $\{a, b\}$
- Accepting states: $\{2\}$
- Transitions $\{(a, 1), (b, 1), (a, 1, 1, 2)\}$.

Bottom-up computations: 

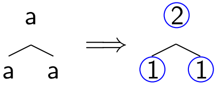
$$\delta_0(a) = 1$$
$$\delta_2(a, 1, 1) = 2$$

$$\delta_0(a) = 1$$
$$\delta_1(a, 1) = \perp$$

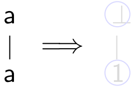
DTA as compact data structure/2

- Sample: 

- States: $\{1, 2, \perp\}$
- Alphabet of labels: $\{a, b\}$
- Accepting states: $\{2\}$
- Transitions $\{(a, 1), (b, 1), (a, 1, 1, 2)\}$.

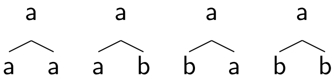
Bottom-up computations: 

$$\delta_0(a) = 1$$
$$\delta_2(a, 1, 1) = 2$$

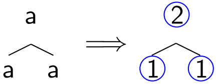


$$\delta_0(a) = 1$$
$$\delta_1(a, 1) = \perp$$

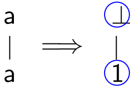
DTA as compact data structure/2

- Sample: 

- States: $\{1, 2, \perp\}$
- Alphabet of labels: $\{a, b\}$
- Accepting states: $\{2\}$
- Transitions $\{(a, 1), (b, 1), (a, 1, 1, 2)\}$.

Bottom-up computations: 

$$\delta_0(a) = 1$$
$$\delta_2(a, 1, 1) = 2$$



$$\delta_0(a) = 1$$
$$\delta_1(a, 1) = \perp$$

Congruences in DTA

In a minimal DTA $p \equiv q$ implies

$$p \in F \leftrightarrow q \in F$$

and for all $m > 0$, all $k \leq m$ and all $(\sigma, r_1, \ldots, r_m) \in \Sigma \times Q^m$

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

DTAs vs DFAs

Compared to DFAs, DTAs

- Lack initial states (transitions with $m = 0$ as $(a, 1)$ and $(b, 1)$ are used as seeds).
- Transitions depend on m states (all siblings).
- Are highly sparse (there are n^m possible inputs of size m, n is num. states).

DFA minimization/1

- DFAs can be minimized in time $\mathcal{O}(kn \log n)$ (k is alphabet size).
- Customary initialization is $\mathcal{O}(|A|^2 \log |A|)$ for sparse DFA.
- A suitable finer initialization leads to $\mathcal{O}(|A| \log |A|)$ cost.

DFA minimization/2

Standard DFA minimization builds the partition $P_0 = \{F, Q - F\}$ and a coarse transition function for all $I, J \in P$:

$$\Delta_{IaJ} = \{(i, a, j) \in \Delta : i \in I \wedge j \in J\}$$

Whenever $s = |\Delta_{Ia}| > 1$, I is split into s classes.

- Finding such (I, a) and updating Δ_{IaJ} is $\mathcal{O}(n)$.
- Number of iterations is $\mathcal{O}(n)$.
- Complexity $\mathcal{O}(kn \log n)$ requires that the largest I subset (that with largest Δ_{IaJ}) remains as I.

Signatures/1

Sparse DFA require:

- Identify useless states and collapse them to $\perp$.
- Initialize the partition P with subsets of states with identical signature and class (accepting or not).

The *signature* of q is

$$\text{sig}(q) = \{a \in \Sigma : \exists (q, a, p) \in \Delta\}$$

Then, only defined transitions are checked.

Signatures/2

In a DTA different definitions of signature are possible

$$
\begin{aligned}
\mathrm{sig}(q) &= \{\sigma \in \Sigma : \exists(\sigma, i_1, \ldots, i_m, j) \in \Delta : \exists k \le m : i_k = q\} \\
\mathrm{sig}(q) &= \{(\sigma, m) : \exists(\sigma, i_1, \ldots, i_m, j) \in \Delta : \exists k \le m : i_k = q\} \\
\mathrm{sig}(q) &= \{(\sigma, m, k) : \exists(\sigma, i_1, \ldots, i_m, j) \in \Delta : \exists k \le m : i_k = q\} \\
\mathrm{sig}(q) &= f(\{(\sigma, i_1, \ldots i_m, j)) \in \Delta : \exists k \le m : i_k = q\})
\end{aligned}
$$

Homomorphism f is:

$$
f(i_k) = \begin{cases} * & \text{if } i_k = q \\ 0 & \text{if } i_k \neq q \wedge i_k \notin F \\ 1 & \text{otherwise} \end{cases}
$$

Our implementation works will all definitions.

DTA minimization/1

DTA coarse transition function

$$\Delta_{\sigma I_1 \ldots I_m J} = \{(\sigma, i_1, \ldots i_m, j) \in \Delta : i_1 \in I_1, \ldots, i_m \in I_m, j \in J\}$$

If $s = |\Delta_{\sigma I_1 \ldots I_m}| > 1$ at least one I_k needs split. However:

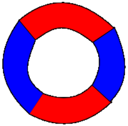
- It is possible that more than one I_k needs split.
- Different $I_{k'} = I_k$ may lead (partially) to same subclasses.
- Which is the largest subset in I_k has nothing to do with the number of transitions in $\Delta_{\sigma I_1 \ldots I_m J}$ (the other I's play).

DTA minimization/2

Useful properties:

- Equivalence is transitive: we define $\texttt{next}_n(q)$ to return next (or first) element in the equivalence class.
- If two states are not equivalent there exists a pair of distinguishing transitions and at least one leads to $q \neq \perp$.

Graphical interpretation: at least one red-to-blue transition.

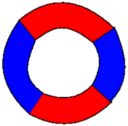


DTA minimization/2

Useful properties:

- Equivalence is transitive: we define $\texttt{next}_n(q)$ to return next (or first) element in the equivalence class.
- If two states are not equivalent there exists a pair of distinguishing transitions and at least one leads to $q \neq \perp$.

Graphical interpretation: at least one red-to-blue transition.



Accessible and coaccessible states/1

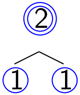
Some definitions:

- State q is *inaccessible* iff $L_A(q) = \emptyset$.
- Accessible state q is *coaccessible* iff there exists $t \in L(A)$ with a subtree s such that $q = A(s)$.
- States which are not coaccessible (and accessible) are *useless*.

For instance, the absorption state $\perp$ is accessible and useless.

Accessible and coaccessible states/2

Accessible states can be found with bottom-up procedure and useless states with a top-down one.
For instance, if $F = \{2\}$ with the computation



- 1 makes 2 accessible,
- 2 makes 1 coaccessible.

Description: algorithm findInaccessible

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

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
   - $B_n \leftarrow m$ [Num. of inaccessible pos. in arg(τ_n)].
   - For $k = 1, ..., m$ append n to R_{i_k} [Occurs 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 $\mathrm{output}(\tau_n) \in I$ then move $\mathrm{output}(\tau_n)$ from I to K. [Whole argument accessible]
5. Return $I - \{\perp\}$.

Description: algorithm findUseless

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

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 (kind of Δ^{-1})].
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.

Description: algorithm minimizeDTA

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

1. Initialize partition P and queue K.
2. Main loop (refine P).
3. Output $A^{\min}$.

Notation:

- P_n is the partition at iteration n.
- $[q]_n$ is the equivalence class of q in P_n.
- $p \sim_n q \leftrightarrow [p]_n = [q]_n$.

Description: Initialization

- 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 $\mathrm{sig}(i_k)$ for $k = 1, ..., m$.
- For all $q \in F$ add $(\#, 1, 1)$ to $\mathrm{sig}(q)$. [include acceptance in signature]
- Create an empty set B_{sig} for every different signature sig and for all $q \in Q$ add q to set $B_{\mathrm{sig}(q)}$.
- 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.

Description: Main loop

While K is not empty

1. Remove the first state q in K.
2. For all $(\sigma, i_1, ..., i_m, j) \in \Delta$ such that $j \sim_n q$ and for all $k \leq m$ such that $\delta_m(\sigma, i_1, ..., \mathrm{next}_n(i_k), ..., i_m) \not\sim_n j$
   1. Create P_{n+1} from P_n by splitting $[i_k]_n$ into so many subsets as different classes $[\delta_m(\sigma, i_1, ., i'_k, .., i_m)]_n$ are found for all $i'_k \in [i_k]_n$.
   2. Add to K the first element from every new subset.New splits induced]
   3. Set $n \leftarrow n + 1$.

Description: Output

- Output $(Q^{\min}, \Sigma, \Delta^{\min}, F^{\min})$ with
  - $Q^{\min} = \{[q]_n : q \in Q\}$;
  - $F^{\min} = \{[q]_n : q \in F\}$;
  - $\Delta^{\min} = \{(\sigma, [i_1]_n, \ldots, [i_m]_n, [j]_n) : (\sigma, i_1, \ldots, i_m, j) \in \Delta \wedge [j]_n \neq [\perp]_n\}$.

Analysis/1

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

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

One can assume $j \neq \perp$ (otherwise, one can exchange p and q)

Analysis/2

Define $p^{[1]} = p$ and, for $s > 0$, $p^{[s+1]} = \mathrm{next}(p^{[s]})$. Then, there is $s > 0$ such that

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

and

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

Analysis/3

The check 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 $(\sigma, i_1, ..., \mathrm{next}(i_k), ...i_m, j')$

Complexity/1

- While K is not empty
  1. Remove the first state q in K.
  2. For all $(\sigma, i_1, ..., i_m, j) \in \Delta$ such that $j \sim_n q$ and for all $k \leq m$ such that $\delta_m(\sigma, i_1, ..., \mathrm{next}_n(i_k), ..., i_m) \not\sim_n j$
     1. Create P_{n+1} from P_n by splitting $[i_k]_n$ into so many subsets as different classes $[\delta_m(\sigma, i_1, .., i'_k, .., i_m)]_n$ are found for all $i'_k \in [i_k]_n$.
     2. Add to K the first element from every new subset.
     3. Set $n \leftarrow n + 1$.

- A state enters K for every finer class created.
- The refinement process cannot create more than $2|Q| - 1$ different classes (size of a binary tree with $|Q|$ leaves)
- The main loop always removes a state from K; then it performs at most $2|Q| - 1$ iterations.

Complexity/2

- While K is not empty
  1. Remove the first state q in K.
  2. For all $(\sigma, i_1, ..., i_m, j) \in \Delta$ such that $j \sim_n q$ and for all $k \leq m$ such that $\delta_m(\sigma, i_1, ..., \mathrm{next}_n(i_k), ..., i_m) \not\sim_n j$
     1. Create P_{n+1} from P_n by splitting $[i_k]_n$ into so many subsets as different classes $[\delta_m(\sigma, i_1, ., i'_k, .., i_m)]_n$ are found for all $i'_k \in [i_k]_n$.
     2. Add to K the first element from every new subset.
     3. Set $n \leftarrow n + 1$.

At every iteration, the internal loop over arguments involves at most $|A|$ iterations.

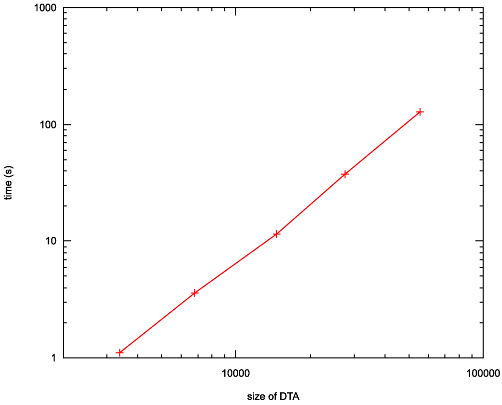
Complexity/3

- While K is not empty
  1. Remove the first state q in K.
  2. For all $(\sigma, i_1, ..., i_m, j) \in \Delta$ such that $j \sim_n q$ and for all $k \leq m$ such that $\delta_m(\sigma, i_1, ..., \mathrm{next}_n(i_k), ..., i_m) \not\sim_n j$
     1. Create P_{n+1} from P_n by splitting $[i_k]_n$ into so many subsets as different classes $[\delta_m(\sigma, i_1, ., i'_k, .., i_m)]_n$ are found for all $i'_k \in [i_k]_n$.
     2. Add to K the first element from every new subset.
     3. Set $n \leftarrow n + 1$.

- If class $[i_k]_n$ is split, its states are classified according to the transition output in less than $|Q|$ steps;
- Updating K adds at most $|Q|$ states.
- Number of splits $< |Q|$; then the conditional block involves at most $|Q|^2$ steps.

Results

Time to minimize acyclic DTA accepting parse trees (up to 2000 trees and 60 labels) from a tree bank.



Results

The time needed to minimize the DTA grows less than quadratically with the size of the automaton (the best fit for this example is $|A|^{1.7}$).

Conclusions and future work

- Simple and efficient minimization of DTA is possible: the search for inconsistent classes can be efficiently performed and undefined transitions and the absorption state can be properly handled.
- A better asymptotic behavior may be still possible.
- We are studyng incremental minimization of DTAs (minimization of a partially minimized automaton).
- Incremental construction (construction of a minimal DTA by adding new trees to the language accepted by an existing one) has also been addressed.