Journal of Universal Computer Science, vol. 22, no. 2 (2016), 180-196
submitted: 23/9/15, accepted: 22/1/16, appeared: 1/2/16 © *J.UCS*

A Taxonomy of Minimisation Algorithms for Deterministic Tree Automata

Johanna Björklund
(Umeå University, SE-901 87 Umeå, Sweden
johanna@cs.umu.se)

Loek Cleophas
(Umeå University, SE-901 87 Umeå, Sweden
and
Stellenbosch University, ZA-7602 Matieland, South Africa
loek@fastar.org)

Abstract: We present a taxonomy of algorithms for minimising deterministic bottom-up tree automata (DTAs) over ranked and ordered trees. Automata of this type and its extensions are used in many application areas, including natural language processing (NLP) and code generation. In practice, DTAs can grow very large, but minimisation keeps things manageable. The proposed taxonomy serves as a unifying framework that makes algorithms accessible and comparable, and as a foundation for efficient implementation. Taxonomies of this type are also convenient for correctness and complexity analysis, as results can frequently be propagated through the hierarchy. The taxonomy described herein covers a broad spectrum of algorithms, ranging from novel to well-studied ones, with a focus on computational complexity.

Key Words: deterministic bottom-up tree automata, automata minimisation, algorithm taxonomies
Category: F.1.1, F.4.3

1 Introduction

Deterministic bottom-up tree automata (DTAs) and their generalisations have a major role in natural language processing (NLP). Like the corresponding string automata (DFAs), DTAs can grow quite large, so minimisation and reduction techniques are necessary for efficient processing. To promote the practical application of tree automata, we compile a taxonomy of DTA minimisation algorithms. Each algorithm has its own characteristics in terms of worst and average case complexities, memory usage, robustness, and so forth, so their performance depends on the input data and execution environment. It is therefore unlikely that a single algorithm will be versatile enough to cover all use cases; rather we want a reasonable set to choose from and a taxonomy helps us understand our options.

Algorithm taxonomies have several advantages. First and foremost, they make algorithms more accessible and easier to compare, by placing them in a uniform framework. Furthermore, as the presentation sets out from an abstract, high-level specification, they show how more concrete specifications can be obtained by stepwise refinement. This process makes algorithm commonalities as well as differences explicit. Taxonomies also support formal argumentation, e.g. correctness proofs: since the root algorithm trivially satisfies its specification, if each of the refinement steps is correct, then each algorithm so derived is also correct. Finally, taxonomies allow for efficient implementation and maintenance in terms of effort involved, and of code size and quality [Watson(1995)].

In this paper, we give a taxonomy of minimisation algorithms for DTAs. Most of the algorithms compute the Nerode congruence as an intermediate step. Two of the algorithms—a DTA version of Hopcroft & Ullman's DFA minimisation algorithm, and Brzozowski's minimisation algorithm in a version for top-down determinisable DTAs—have not been previously presented for trees.

1.1 Related work

The theory underlying tree automata and tree transducers has been developed since the 1960s [Thatcher and Wright(1965), Brainerd(1967)]; see for example [Engelfriet(1975), Gécseg and Steinby(1984), Gécseg and Steinby(1997), Comon et al.(2007)] for surveys. The theory builds on that of finite state automata and was initially used as an alternative representation for context-free languages, and to solve decision problems in mathematical logic [Doner(1970)].

[Kron(1975)] appears to be the first work focusing on practical algorithms; apart from his work, most work for e.g. term rewriting or code generation in compilers appeared from the early-to-mid-1980s onwards (see e.g. [Burghardt(1988), Aho et al.(1989), Hoffmann and O'Donnell(1982), Aho and Ganapathi(1985)]).

Tree automata are useful in NLP because they capture the derivation process of context-free rewriting systems. Weighted tree transducers later were use e.g. to improve machine-translation quality [Yamada and Knight(2001)] and target-language fluency [Galley et al.(2006)], and to support translation between languages with different predicate-argument structure [Maletti(2011)].

Bottom-up tree automata can always be determinised without losing descriptive power. This is not the case if we add weights [Borchardt(2005)], or change direction: while non-deterministic top-down TAs are as powerful as bottom-up ones, deterministic top-down TAs are more restricted. There is, for example, no deterministic top-down TA to recognise $\{f[a,b], f[b,a]\}$. A slightly more powerful device is the r-l-deterministic top-down TA proposed by [Nivat and Podelski(1997)], with a descriptive power strictly in-between deterministic top-down TAs (which they generalise) and TAs.

In this paper, we have limited our scope to deterministic ranked automata, and only considered standard forms of minimisation. Connecting minimisation of unranked and ranked tree automata via stepwise tree automata is discussed by [Martens and Niehren(2007)]. [Carrasco et al.(2007)] present an implementation of DTA minimisation over unranked trees. This work is continued by the same team of researchers with the incremental construction of minimal DTAs for unranked trees [Carrasco et al.(2008)].

Minimisation is provably harder for non-deterministic devices, just as it is in the case of string automata; it is EXPTIME-complete for non-deterministic TAs [Martens and Niehren(2007)]. Heuristic algorithms for non-deterministic TA minimisation based on the use of various bisimulation and simulation relations as a substitute for the Nerode congruence are investigated in [Abdulla et al.(2007), Högberg et al.(2009), Abdulla et al.(2009)]. Standard minimisation algorithms are language-preserving, but sometimes it is acceptable to allow a limited number of mistakes to obtain a compact representation. This idea is explored under the name hyper-minimisation, and has been treated for unweighted and weighted tree automata [Holzer and Maletti(2010), Maletti and Quernheim(2012)].

Algorithm taxonomies have been used for computational problems such as sorting [Darlington(1978), Broy(1983)] and attribute evaluation [Marcelis(1990)]. The Taxonomy-BAsed Software COnstruction (TABASCO) project compiled taxonomies for the explicit purposes of correctness-by-construction and simplifying implementation and benchmarking. Applications of TABASCO included the minimisation of deterministic string automata [Watson(1995)]. [Cleophas(2008)] applied TABASCO to tree automata construction and pattern matching algorithms, relating the previously mentioned algorithms originating from code generation, and presenting them in a unifying framework. While some of the algorithms included use techniques to reduce the size for the resulting tree automata, minimisation as such was not covered.

2 Preliminaries

Sets and numbers. We write $\mathbb{N}$ for the set of natural numbers including 0. For $n \in \mathbb{N}$, $[n] = \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$. Thus, in particular, $[0] = \emptyset$. The cardinality of a set S is written $|S|$, and the powerset of S is denoted by $pow(S)$. Given a subset S' of S, we write $\overline{S'}$ for the complement of S' with respect to S.

Relations. Let $\mathcal{E}$ and $\mathcal{F}$ be equivalence relations on S. We say that $\mathcal{F}$ is *coarser* than $\mathcal{E}$ (or equivalently: that $\mathcal{E}$ is a *refinement* of $\mathcal{F}$), if $\mathcal{E} \subseteq \mathcal{F}$. The *equivalence class* or *block* of an element s in S with respect to $\mathcal{E}$ is the set $[s]_\mathcal{E} = \{s' \mid (s, s') \in \mathcal{E}\}$. Whenever $\mathcal{E}$ is obvious from the context, we simply write $[s]$ instead of $[s]_\mathcal{E}$. It should be clear that $[s]$ and $[s']$ are equal if s and s' are in relation $\mathcal{E}$, and disjoint otherwise, so $\mathcal{E}$ induces a partition $(S/\mathcal{E}) = \{[s] \mid s \in S\}$ of S. We denote the identity relation $\{(s, s) \mid s \in S\}$ on S by $\mathcal{I}_S$.

Strings and trees. An alphabet is a finite non-empty set. The empty string is denoted by ε. For an alphabet Σ, a Σ-labelled *tree* is a partial function

$t: \mathbb{N}^* \to \Sigma$ such that the domain $dom(t)$ of t is a finite prefix-closed set, and for every node $v \in dom(t)$ there exists a $k \in \mathbb{N}$ such that $\{i \in \mathbb{N} \mid vi \in dom(t)\} = [k]$. Here, k is called the *rank of* v. The *subtree* of a tree t rooted at v is the tree t/v defined by $dom(t/v) = \{u \in \mathbb{N}^* \mid vu \in dom(t)\}$ and $t/v(u) = t(vu)$ for every $u \in \mathbb{N}^*$. If $t(\varepsilon) = f$ and $t/i = t_i$ for all $i \in [k]$, where k is the rank of ε in t, then we denote t by $f[t_1, \ldots, t_k]$. If $k = 0$, then $f[]$ is shortened to f.

A *ranked alphabet* is an alphabet $\Sigma = \bigcup_{k \in \mathbb{N}} \Sigma_{(k)}$, partitioned into pairwise disjoint subsets $\Sigma_{(k)}$. For every $k \in \mathbb{N}$ and $f \in \Sigma_{(k)}$, the *rank* of f is $rank(f) = k$. We use r for the maximum rank of a symbol in Σ. The set T_Σ of all trees over Σ consists of all Σ-labelled trees t such that the rank of every node $v \in dom(t)$ coincides with the rank of $t(v)$. Nodes labeled by symbols of rank 0 are called *leaves.* A tree language is a subset of T_Σ.

For a set Q (of e.g. states) we denote by $\Sigma(Q)$ the set of trees

$$\{f[q_1, \ldots, q_k] \mid k \in \mathbb{N}, f \in \Sigma_k, \text{ and } q_1, \ldots, q_k \in Q\} \ .$$

Contexts and substitution. Let Σ be a ranked alphabet and let $\square \notin \Sigma$ be a special symbol of rank 0. The set of *contexts over* Σ is the set

$$C_\Sigma = \{c \in T_{\Sigma \cup \{\square\}} \mid \text{ there is exactly one } v \in dom(c) \text{ with } c(v) = \square\} \ .$$

Consider a context $c \in C_\Sigma$ and let $v \in dom(c)$ be the unique node such that $c(v) = \square$. The *substitution* of a tree t into c, denoted $c[\![t]\!]$, is defined by $dom(c[\![t]\!]) = dom(c) \cup \{vu \mid u \in dom(t)\}$ and

$$c[\![t]\!](w) = \begin{cases} c(w) & \text{if } w \in dom(c) \setminus \{v\}, \text{ and} \\ t(u) & \text{if } w = vu \text{ for some } u \in dom(t) \ . \end{cases}$$

Tree automata. Formally, a *deterministic tree automaton* (DTA) is a tuple $M = (Q, \Sigma, \delta, Q_\mathrm{f})$ where Q is a finite set of *states*; Σ is a ranked alphabet of *input symbols*; $\delta: \Sigma(Q) \to Q$ is the partial *transition function*; and $Q_\mathrm{f} \subseteq Q$ is the set of *final states.* The size of M, written $|M|$, is $|\delta|$.

We define the behaviour of M on trees in $T_{\Sigma \cup Q}$, where states are considered to be symbols of rank 0. Let $\hat{\delta}: T_{\Sigma \cup Q} \to Q$ be defined by

$$\hat{\delta}(t) = \begin{cases} t(\varepsilon) & \text{if } t(\varepsilon) \in Q \\ \delta(t(\varepsilon)[\hat{\delta}(t_1), \ldots, \hat{\delta}(t_k)]) & \text{if } t(\varepsilon) \in \Sigma_{(k)} \end{cases}$$

The language recognised by M is $\mathcal{L}(M) = \{t \in T_\Sigma \mid \hat{\delta}(t) \in Q_\mathrm{f}\}$. From here on, we identify δ with $\hat{\delta}$.

In several of the algorithms, we iterate over the set of contexts representing left-hand sides of transition rules with a gap in them:

$$C_\delta = \{c \in C_{\Sigma \cup Q} \mid \delta(c[\![q]\!]) \text{ is defined for some } q \in Q\} \ ,$$

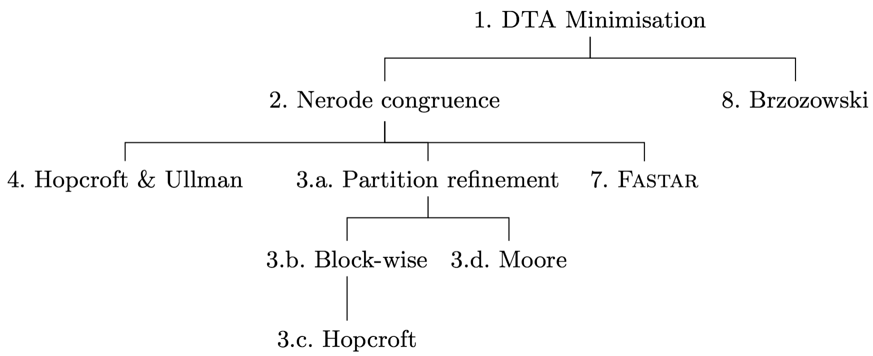

Figure 1: A taxonomy of minimisation algorithms for DTA. The numbering is with respect to the algorithm numbers in this paper.

Example 1. For the transition table

$$\delta = \{(a, p), (b, q), (f[p, q], p), (f[q, p], p), (f[p, p], p)\} \enspace ,$$

we have

$$C_\delta = \{f[p, \Box], f[q, \Box], f[\Box, p], f[\Box, q]\} \enspace .$$

Nerode congruence. The *upward language* of $q \in Q$, written $\mathcal{L}^\uparrow_M(q)$, is the set of contexts $\{c \in C_\Sigma \mid \delta(c[\![q]\!]) \in Q_f\}$. Similarly, the *downward language* of q is $\mathcal{L}^\downarrow_M(q) = \{t \in T_\Sigma \mid \delta(t) = q\}$. The Nerode congruence [Nerode(1958)] is the coarsest congruence relation $\mathcal{E}$ on Q with respect to δ. In other words, $\mathcal{E}(p, q)$ if and only if $\mathcal{L}^\uparrow(p) = \mathcal{L}^\uparrow(q)$ for all $p, q \in Q$.

3 Abstract DTA Minimisation

For the remainder of this paper, let $M = (Q, \Sigma, \delta, Q_\mathrm{f})$ be a DTA, and let $\mathcal{E}$ be the Nerode congruence on M. To avoid trivial corner cases, we assume that $|Q| > 1$ and that M is reduced in the sense that for all $q \in Q$, $\mathcal{L}^\downarrow_M(q) \neq \emptyset$ and $\mathcal{L}^\uparrow_M(q) \neq \emptyset$ (which also implies that $Q_\mathrm{f} \neq \emptyset$).

[Figure 1] shows a taxonomy of DTA minimisation algorithms. A pair of algorithms A and B is in an ancestor-descendant relationship in the taxonomy if B can be obtained by adding detail to the specification of A. At the top-most level, we have the prototypical [Algorithm 1]. It takes as input a DTA M, and uses an abstract statement S to compute M' satisfying the postcondition, i.e. to find the minimal language-equivalent DTA M'. [Algorithm 1] spans two families of algorithms, one that centers on the computation of the Nerode congruence $\mathcal{E}$, and one that uses repeated transition reversal and determinisation. The latter is something of a rare bird among minimisation algorithms and is treated separately in [Section 6].

Algorithm 1 Abstract DTA minimisation algorithm

Precondition: $M = (Q, \Sigma, \delta, Q_\mathrm{f})$ is a DTA

1: $M' : S$

Postcondition: $\mathcal{L}_M = \mathcal{L}_{M'}$ and M' is minimal

---

Continuing down the left taxonomy branch, we come to the slightly more concrete [Algorithm 2] as a refinement. It uses the fact that once the Nerode congruence $\mathcal{E}$ is known, the canonical automaton M' is easily computed.

Definition 1 cf. [Buchholz(2008), Definition 3.3]. The *aggregated* DTA *with respect to* M *and* $\mathcal{E}$, denoted by $(M/\mathcal{E})$, is the DTA $((Q/\mathcal{E}), \Sigma, \delta', Q'_f)$ given by $Q'_f = \{[q] \mid q \in Q_f\}$ and $\delta'(f[[q_1], \ldots, [q_k]]) = [\delta(f[q_1, \ldots, q_k])]$. The transition function δ' is well-defined because $\mathcal{E}$ is a congruence relation.

Lemma 2. *Let $M' = (M/\mathcal{E})$, then $\mathcal{L}(M) = \mathcal{L}(M')$ and M' is state minimal.*

Recall that we consider the size of an automaton to be the size (i.e. number of entries) of its transition table (i.e. $|\delta|$), rather than the size of its state set (i.e. $|Q|$). This makes it easier to understand how algorithms behave on partial automata (as opposed to total automata, which must necessarily be large when there are high-ranked symbols in the input alphabet). Since we restrict ourselves to deterministic and reduced automata, Lemma 2 (cf. [Högberg et al.(2009)]) is still applicable.

Lemma 3. *A reduced DTA is state minimal if and only if it is transition minimal.*

Proof. Let M be a state-minimal reduced DTA and let M' be a transition-minimal reduced DTA for $\mathcal{L}(M)$. We show that the two automata are isomorphic.

Since both M and M' are deterministic, for every state p in M' there is a state $q \in M$ such that $\mathcal{L}^{\downarrow}_M(p) \subseteq \mathcal{L}^{\downarrow}_{M'}(q)$. From this it follows that $\mathcal{L}^{\uparrow}_M(p) = \mathcal{L}^{\uparrow}_{M'}(q)$. This means that the language recognised by M' does not change if all pairs of states p and p' in M' are merged, for which there is a state q in M such that $\mathcal{L}^{\downarrow}_M(p) \subseteq \mathcal{L}^{\downarrow}_M(q)$ and $\mathcal{L}^{\downarrow}_M(p') \subseteq \mathcal{L}^{\downarrow}_M(q)$. Since any such merge would decrease the number of transitions of the already supposedly transition-minimal M' with at least 1, there can be no such states p and q. In other words, there is a one-to-one mapping φ between the states of M and M', such that $\mathcal{L}^{\uparrow}_M(q) = \mathcal{L}^{\uparrow}_{M'}(\varphi(q))$. Since both machines are reduced, a transition of the form $f[q_1, \ldots, q_k]$ in M implies that there is a transition $f[\varphi(q_q), \ldots, \varphi(q_k)]$ in M'. In other words, M' has no fewer transitions than M. $\square$

Algorithm 2 Abstract DTA minimisation algorithm based on $\mathcal{E}$

Precondition: $M = (Q, \Sigma, \delta, Q_\mathrm{f})$ is a DTA
1: $\mathcal{E} : S$
2: $M' \leftarrow (M/\mathcal{E})$
Postcondition: $\mathcal{L}_M = \mathcal{L}_{M'}$ and M' is minimal

[Algorithm 2] describes a family of algorithms, differing in how $\mathcal{E}$ is computed.

4 Algorithms based on partition refinement

In this section, we consider a family of algorithms that find $\mathcal{E}$ by partition refinement. They compute a series of gradually more refined hypothesis relations $\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}_2, \ldots$. Relation $\mathcal{E}_0$ is the coarsest equivalence relation that respects the separation of Q into final and non-final states. Relation $\mathcal{E}_{i+1}$ is obtained from $\mathcal{E}_i$ by selecting a subset of the blocks $B_1, \ldots, B_k$, and "splitting" the relation with respect to these. Intuitively, this is done by separating all pairs of states p, q such that there is some B_j, $j \in [k]$, and some context c such that exactly one of $\delta(c[\![p]\!])$ and $\delta(c[\![q]\!])$ is in B_j. To avoid repeated splitting against the same block, the algorithms also maintain a series of equivalence relations $\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2, \ldots$. For every $i \in \{0, 1, 2, \ldots\}$, it holds that $\mathcal{E}_i$ is a refinement of $\mathcal{F}_i$, and blocks are copied from $\mathcal{E}_i$ to $\mathcal{F}_i$ as they are used for splitting.

[Algorithm 3(a)] shows a prototype version of such a of partition-refinement algorithm. For the presentation, we use the contexts representing left-hand sides of transition rules with a gap in them (see Section 2) and a pair of auxiliary functions to manage equivalence relations.

Definition 4. Let $B \subseteq Q$.
- We write $cut(B)$ for the subset $\overline{B^2 \cup \overline{B}^2}$ of Q^2.
- We write $split(B)$ for the set of all pairs (q, q') in Q^2, for which there is a $c \in C_\delta$ such that exactly one of $\delta(c[\![q]\!])$ and $\delta(c[\![q']\!])$ is in B.

Correctness can be argued by observing that $\mathcal{E}$ must refine $\{\overline{Q_\mathrm{f}}, Q_\mathrm{f}\}$, and that for every $i \in \{0, 1, 2, \ldots\}$, $\mathcal{E}_i$ is a refinement of $\mathcal{E}$, since a pair of states are only separated if there is a witness to show that they are distinct under the Nerode congruence. When the refinement steps converge, the result is a congruence relation, and this must happen when all blocks are singletons, if not earlier. The final piece of the puzzle is that the union of two congruence relations is again a congruence relation, coarser than both of them. This means that the refinement process cannot arrive at two distinct coarsest possible refinements.

Different strategies exist for selecting the blocks that are used for splitting. By simply picking one block at a time at random, as in [Algorithm 3(b)], we have an

Table 1: The worst-case complexities of the algorithms in our taxonomy. Recall that m is the transition table size, n the number of states, and r the maximum rank of a symbol in the input alphabet. It can be shown that for each algorithm, when considering the case where $r = 1$ (i.e. trees representing strings), the complexity reduces to the known complexity for the respective string case variants.

Algorithm	**Complexity**
Hopcroft & Ullman's algorithm	$O\big(rmn^2\big)$
Moore's algorithm	$O(rmn)$
Hopcroft's algorithm	$O(rm \log n)$
The FASTAR algorithm	$O\big((rm)^{n-2}n^2\big)$
Brzozowski's algorithm	$O(2^{n^{n^r}})$

easily implemented algorithm that runs in time $O\big(rmn^2\big)$ [Högberg et al.(2009)], where m is the size of the transition table and n the number of states [see Table 1]. This can be improved with Hopcroft's strategy of always splitting against the smaller half. The idea is that if a block $B \in \mathcal{F}_i$ is the union of two blocks B' and B'' in $\mathcal{E}_i$, c is a context in C_δ, and we know

- the set of states $P = \{q \in Q \mid \delta(c[\![q]\!]) \in B\}$, and
- the set of states $P' = \{q \in P \mid \delta(c[\![q]\!]) \in B'\}$

then set $\{q \in P \mid \delta(c[\![q]\!]) \in B''\}$ is simply $P \setminus P'$, as M is deterministic.

Hopcroft's algorithm (here presented as [Algorithm 3(c)]) was originally defined for DFAs, and extended by [Paige and Tarjan(1987)] to non-deterministic string automata. Their addition is the observation that if the state p can move on a context c in n ways to a block B, and in m ways to the smaller block $B' \subseteq B$, where $m \leq n$, then p can move in $n - m$ ways to the block $B \setminus B'$. Paige and Tarjan's (and thus Hopcroft's) algorithms were generalised to (weighted and non-deterministic) tree automata by [Högberg et al.(2009)], whose algorithm runs in $O(rm \log n)$ time when the input is unweighted and deterministic.

An alternative efficiency gain is to work layer-wise, and simultaneously split against all blocks discovered in the previous iteration. This leads to Moore's algorithm [Moore(1956)], which was later generalised to DTAs by Brainerd (see [Algorithm 3(d)]). For trees, the algorithm first appeared in 1968 in [Brainerd(1968)]; Brainerd's earlier PhD thesis [Brainerd(1967)] leaves the algorithm implicit. The same layer-wise algorithm appears in [Comon et al.(2007)], and is covered implicitly in [Gécseg and Steinby(1984), pp. 93–94].

Algorithm 3 Four partition refinement algorithms

Precondition: $M = (Q, \Sigma, \delta, Q_\mathrm{f})$ is a DTA

1: $(\mathcal{E}_0,\ \mathcal{F}_0,\ i) \leftarrow (\overline{F}^2 \cup F^2,\ Q^2,\ 0)$
2: **while** $\mathcal{E}_i \neq \mathcal{F}_i$ **do**

3: ▷ (a) Prototypical partition refinement
4: Choose $\mathcal{B} \subseteq (Q/\mathcal{E}_i)$
5: $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \setminus \bigcup_{B_i \in \mathcal{B}} \mathit{cut}(B_i)$
6: $\mathcal{E}_{i+1} \leftarrow \mathcal{E}_i \setminus \bigcup_{B_i \in \mathcal{B}} \mathit{split}(B_i)$

▷ (b) Basic block-wise algorithm
4: Choose $B_i \in (Q/\mathcal{E}_i)$
5: $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \setminus \mathit{cut}(B_i)$
6: $\mathcal{E}_{i+1} \leftarrow \mathcal{E}_i \setminus \mathit{split}(B_i)$

▷ (c) Hopcroft's algorithm
4: Choose $S_i \in (Q/\mathcal{F}_i)$ and $B_i \in (Q/\mathcal{E}_i)$ s.t. $B_i \subset S_i$ and $|B_i| \leq |S_i|/2$
5: $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \setminus \mathit{cut}(B_i)$
6: $\mathcal{E}_{i+1} \leftarrow \mathcal{E}_i \setminus \mathit{split}(B_i)$

▷ (d) Moore's algorithm
4: ▷ All blocks in $(Q/\mathcal{E}_i)$ are implicitly chosen
5: $\mathcal{F}_{i+1} \leftarrow \mathcal{E}_i$
6: $\mathcal{E}_{i+1} \leftarrow \mathcal{E}_i \setminus \bigcup_{B \in (Q/\mathcal{F}_{i+1})} \mathit{split}(B)$

7: $i \leftarrow i + 1$
8: **end while**

Postcondition: $\mathcal{E}_i = \mathcal{E}$

In Moore's algorithm, the refinement steps can be implemented using the non-comparative sorting algorithm Radix sort. Radix sort is usually attributed to Herman Hollerith's work on tabulating machines in the late 19th century. The sorting algorithm relies on a *positional* form of representation, such as the arabic numerical system, and sorts keys one position at a time. When Radix sort is invoked in Moore's algorithm, the set of transitions associated with a state q is translated into a positional representation, encoded as an integer key, for q. These keys are then used to sort the states into equivalence classes. In practise, Line 6 is replaced by

6: $\mathcal{E}_{i+1} \leftarrow \textsc{RadixSort}(\{(q, [\delta(c_1[\![q]\!])]_{\mathcal{E}_i} \cdots [\delta(c_k[\![q]\!])]_{\mathcal{E}_i}) \mid q \in Q\})$

where $c_1, \ldots, c_k$ is an arbitrary enumeration of C_δ. The key here is thus a sequence of block labels, where the ith label is the block of $\mathcal{E}_i$ to which δ takes tree $c_i[\![q]\!]$. In the string case, this optimisation brings the worst-case complexity of $O(kn^3)$ down to $O(kn^2)$. In the tree case, it goes from $O(rmn^2)$ to $O(rmn)$, but as m can be up to n^r, the relative gain is smaller. For the string case,

---

Algorithm 4 Computing $\mathcal{E}$ from the complement side (Hopcroft & Ullman).

Precondition: $M = (Q, \Sigma, \delta, Q_\mathrm{f})$ is a DTA

$L(\rho) \leftarrow \emptyset$, for all $\rho \in Q^2$
$\mathcal{D} \leftarrow Q_\mathrm{f} \times \overline{Q_\mathrm{f}} \cup \overline{Q_\mathrm{f}} \times Q_\mathrm{f}$
for $(p, q) \notin (Q_\mathrm{f} \times \overline{Q_\mathrm{f}} \cup \overline{Q_\mathrm{f}} \times Q_\mathrm{f})$ **do**
    for $c \in C_\delta$ **do**
        $\rho \leftarrow (\delta(c[\![p]\!]), \delta(c[\![q]\!]))$
        if $\rho \in \mathcal{D}$ **then**
            separate$((p, q))$
        else
            $L(\rho) \leftarrow L(\rho) \cup \{(p, q)\}$
        end if
    end for
end for

Postcondition: $\overline{\mathcal{D}} = \mathcal{E}$

Algorithm 5 Separate pair ρ and all affected pairs of states

function *separate*(ρ)
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{\rho\}$
    for $\rho' \in L(\rho) \setminus \mathcal{D}$ **do**
        separate(ρ')
    end for
end function

---

the average-case time complexities of Moore's and Hopcroft's algorithms were recently shown to be $O(n \log \log n)$ [David(2012)] ("for the uniform distribution on complete deterministic automata"; see that paper for more details), but it is an open question how these results translate to the tree case.

The partition refinement can also be done through aggregation of the complement relation of $\mathcal{E}$, that is, state distinguishability relation $\mathcal{D}$. [Algorithm 4], due to [Hopcroft and Ullman(1979)], does precisely this. It iterates over all pairs of states (p, q) not yet distinguishable. For each such pair, it checks whether the pair can be distinguished based on what is currently known about $\mathcal{D}$, and then adds information about what additional information would cause p and q to be put into different equivalence classes. For this purpose, each pair of states (r, s) has a set $L((r, s))$ of pairs of states. If (p, q) is in $L((r, s))$, this means that if r and s turn out to be distinguishable, then so will p and q. The pair (p, q) is therefore placed in $L(\delta(c[\![p]\!]), \delta(c[\![q]\!]))$ for every $c \in C_\delta$. The algorithm uses the function *separate* (see [Algorithm 5]) to update $\mathcal{D}$ whenever it manages to distinguish a new pair.

Theorem 5. *Algorithm 4 is in* $O(rmn^2)$.

Proof. The initialisation of L and $\mathcal{D}$ is in $O(n^2)$. The two 'for' loops are executed at most $O(n^2)$ and $O(rm)$ times, respectively. The latter figure is simply the number of contexts that can be built from the transition table.

The function *separate* is invoked at most once for every $\rho \in Q \times Q$. Aside from adding ρ to $\mathcal{D}$, *separate* involves the computation of a set difference and a sequence of recursive calls. In an efficient implementation, the set difference would be replaced by removing ρ' from all $L(\rho)$ as soon as we learnt that $\rho' \in \mathcal{D}$. This comes at a total cost of $O(rmn^2)$ that is spread out over the entire computation. The recursive calls are "for free" since we have already counted the number of invocations of *separate*. The total amount of work done by *separate* is thus in $O(rmn^2)$.

Summing up, we see that the computational complexity of Algorithm 4 is in $O(n^2) + O(rmn^2) + O(n^2) + O(rmn^2) = O(rmn^2)$. □

5 An algorithm based on partition aggregation

The congruence relation $\mathcal{E}$ can also be found through partition aggregation, as suggested by the FASTAR research group in [Cleophas et al.(2009)]. This method starts with a singleton partition for each state of the initial DTA and approaches $\mathcal{E}$ by iteratively merging partitions found to be equivalent. When no more changes occur, we have found the solution.

This algorithm, presented as [Algorithm 7], starts out knowing that each state is equivalent to itself, and that each pair of final and non-final state is distinguishable. While there exist state pairs for which it is not known whether they are equivalent or distinguishable, function *equiv* in [Algorithm 6] is used to compute equivalence of such a pair of states, based on a recursive definition of $\mathcal{E}$: it is the greatest equivalence relation on Q such that

$$\mathcal{E}(p, q) \equiv (p \in Q_\mathrm{f} \equiv q \in Q_\mathrm{f}) \wedge \bigwedge_{c \in C_\delta} \mathcal{E}(\delta(c[\![p]\!]), \delta(c[\![q]\!])).$$

An additional variable, S, kept global for efficiency, is used during recursion to keep track of state pairs that are tentatively assumed equivalent. To ensure

Algorithm 6 Point-wise computation of $(p,q) \in \mathcal{E}$ for DTAs

Precondition: S is a globally accessible set variable, initialised to $\emptyset$.

function $equiv(p,q,k)$
    if $k = 0$ **then**
        $eq \leftarrow p \in Q_{\mathrm{f}} \equiv q \in Q_{\mathrm{f}}$
    else if $k \neq 0 \wedge (p,q) \in S$ **then**
        $eq \leftarrow true$
    else if $k \neq 0 \wedge (p,q) \notin S$ **then**
        $eq \leftarrow p \in Q_{\mathrm{f}} \equiv q \in Q_{\mathrm{f}}$
        $S \leftarrow S \cup \{(p,q),(q,p)\}$
        $eq \leftarrow eq \wedge \bigwedge_{c \in C_\delta} equiv(\delta(c[\![p]\!]), \delta(c[\![q]\!]), k-1)$
        $S \leftarrow S \backslash \{(p,q),(q,p)\}$
    end if
    return eq
end function

Postcondition: $equiv(p,q,k) \equiv (p,q) \in \mathcal{E}$

termination of the recursive computation, function *equiv* takes a third parameter, bounding the recursion depth. Depending on whether *equiv* determines a pair (p,q) to be equivalent or distinguishable, it is added to $\mathcal{E}_{i+1}$ or $\mathcal{F}_{i+1}$; in the former case, as equivalence is transitive, transitive closure is applied to $\mathcal{E}_{i+1}$.

Theorem 6. *Algorithm 7 is in* $O\big((rm)^{n-2}n^2\big)$.

Proof. In the computation of the function *equiv*, the recursion depth is $n-2$. Moreover, each invocation of *equiv* makes at most mr calls to itself; one for each context in C_δ. Since the main loop is executed at most n^2 times, this yields a total complexity of $O\big((rm)^{n-2}n^2\big)$.

While this algorithm is inferior to Hopcroft's algorithm in terms of worst-case performance [see Table 1], it also has an advantage: intermediate results are usable to reduce the original DTA, albeit not yet to a minimal one.

For the DFA case, [Watson and Daciuk(2003)] showed that the complexity of the function *equiv* could be brought down from $O(|\Sigma|^{n-2})$ to $O\big(n^2\alpha(n^2)\big)$ by combining memoisation with the classical union-find approach [Aho et al.(1974)]. This reduced the overall complexity from $O(|\Sigma|^{n-2}\, n^2)$ to $O\big(n^4\alpha(n^2)\big)\big)$, where α denotes the inverse of Ackermanns function which is such that $\alpha(n) \leq 5$ for all $n \leq 2^{2^{16}}$. The experiments conducted by the same set of authors suggest that the resulting algorithm also performs well in practice. The same approach is likely to be helpful also in the tree case: 'union' allows us to efficiently merge equivalence classes and 'find' helps to propagate evidence against state equivalence. The exact savings are however still an open question.

Algorithm 7 Incrementally compute $\mathcal{E}$ (FASTAR)

Precondition: $M = (Q, \Sigma, \delta, Q_\mathrm{f})$ is a DTA

$(S,\ \mathcal{E}_0,\ \mathcal{D}_0,\ i) \leftarrow (\emptyset,\ I_Q,\ (\overline{Q_\mathrm{f}} \times Q_\mathrm{f}) \cup (Q_\mathrm{f} \times \overline{Q_\mathrm{f}}),\ 0)$
$\triangleright$ Invariant: $\mathcal{E}_i \subseteq \mathcal{E}_{i+1} \subseteq \mathcal{E}$ and $\mathcal{D}_i \subseteq \mathcal{D}_{i+1}$
while $\exists(p,q) \in \overline{\mathcal{E}_i \cup \mathcal{D}_i}$ **do**
  if $\mathit{equiv}(p, q, |Q| - 2)$ **then**
    $\mathcal{E}_{i+1} \leftarrow (\mathcal{E}_i \cup \{p,q\}^2)^+$
    $\mathcal{D}_{i+1} \leftarrow \mathcal{D}_i$
  else
    $\mathcal{E}_{i+1} \leftarrow \mathcal{E}_i$
    $\mathcal{D}_{i+1} \leftarrow \mathcal{D}_i \cup \{p,q\}^2$
  end if
  $i \leftarrow i + 1$
end while

Postcondition: $\mathcal{E}_i = \mathcal{E}$

6 Brzozowski's algorithm

In this section, we give a DTA analog of Brzozowski's algorithm for minimising DFAs [Brzozowski(1962)], an algorithm that is perhaps more surprising than it is practical. Unlike the previously described algorithms, it does not explicitly compute the Nerode congruence, but rather depends on repeated determinisation and reversal. Due to the determinisation steps, the algorithm is exponential in the worst-case, though practical benchmarking suggest that it is sometimes competitive with the previously mentioned partition-refinement algorithms [Watson(1995)].

Brzozowski's DFA minimiser is the sequence of four DFA manipulations *reverse; determinise; reverse; determinise.* As the name suggests, *reverse* reverses all transitions in the DFA and makes final states start states and vice-versa, resulting in a (generally non-deterministic) automaton accepting the reverse of the words accepted by the original DFA. *Determinise* builds an equivalent DFA from a non-deterministic automaton. The algorithm relies on two important properties:

1. In a DFA, all distinct pair of states p and q have disjoint left-languages; if this were not the case, there would be a word w labeling paths from the start state to both p and to q, and hence the automaton would be non-deterministic.

2. *Determinise* takes an automaton as input and builds a new one whose states are *sets* of states taken from the input automaton. Each such new state's right-language is the union of its constituents' right-languages (in the input automaton). This is a property of all state-merging algorithms, such as determinisation, but also equivalence-based minimisation algorithms.

Thanks to the first property, the first three components of Brzozowski's algorithm yield an equivalent non-deterministic automaton whose *right* languages are pairwise disjoint. With that as input and the second property, the final determinization gives a DFA with pairwise inequivalent states—a minimal DFA.

[Algorithm 8] extends this to a DTA minimiser, where the comments capture the aforementioned arguments. The *reverse* operations of the DFA minimiser are of course embedded within the notions of top-down and bottom-up determinisation. Top-down determinisation of a DTA thus corresponds to reversing the DTA into a (generally non-deterministic) top-down tree automaton, followed by a subset construction—yielding a deterministic top-down TA whose states' up-languages are pairwise disjoint—say M'. Determinisation of non-deterministic top-down tree automata (NtdTAs) is a straightforward generalisation from the string case, and the reader is referred to e.g. [Cleophas(2008), Section 3.4.3] for a formal definition and treatment. It should be noted that such determinisation is not (losslessly) possible for any NtdTA, as there are languages for which no deterministic version exists: Consider e.g. the language consisting of trees $f[a, b]$ and $f[b, a]$. A deterministic top-down tree automaton from the start state, say q_s, has a transition on f to a single state, say q_1 and then requires transitions on both a and b from q_1 to exist in order to accept both trees, yet as a result will also accept e.g. $f[b, b]$. The precondition of [Algorithm 8] therefore mentions the important restriction that the algorithm is restricted to TAs that can be (losslessly) top-down determinised.

Following top-down determinisation, bottom-up determinisation corresponds to reversing M'—yielding a non-deterministic TA—and then determinising that automaton, resulting in a DTA whose states' downward languages are pairwise unique, making it minimal.

Theorem 7. *Brzozowski's algorithm for tree automata is in* $O(2^{n^{n^r}})$.

Proof. The top-down and bottom-up determinisation of M are both in $O(2^{n^r})$, which is also the maximal size of the output automata. When composed, the two operations have a combined complexity of $O(2^{n^{n^r}})$. □

7 Conclusion

On the practical side, the next step is to implement and benchmark the algorithms, so as to improve our understanding of how their performance depends on characteristics of the data and the input environment. The main challenge will be to find representative data sets for different NLP tasks. Once complete, the resulting toolkit will be shared with the community as open source.

Due to the hierarchical nature of the domain, algorithms on tree automata appear particularly suited for parallelisation, either on a multi-core CPU or

Algorithm 8 A Brzozowski-analog for DTA minimisation

Precondition: $M = (Q, \Sigma, \delta, Q_f)$ is a DTA and M can be top-down determinised

1: $M' \leftarrow$ *top-down determinise*(M)
2: $\triangleright$ M' is equivalent to M and up-languages of M' states are pairwise disjoint
3: $M'' \leftarrow$ *bottom-up determinise*(M')
4: $\triangleright$ M'' is equivalent to M and downward languages of M'' states are pairwise unique

Postcondition: $\mathcal{L}_M = \mathcal{L}_{M''}$ and M'' is minimal

GPU, or distributed across a network. A specification in Hoare's CSP is already available for [Algorithm 6] [Cleophas et al.(2009)]. It would be valuable to obtain similar ones for the other algorithms, and to implement and benchmark such parallelised versions.

On the theoretical side, it would be interesting to extend the taxonomy to cover also the non-deterministic and possibly weighted case, and to provide correctness proofs and a complexity analysis of [Algorithm 4] and [Algorithm 8].

Acknowledgments

The authors are indebted to Bruce W. Watson at Stellenbosch University for his helpful input, in particular related to the discussion of Brzozowski's algorithm.

References

[Abdulla et al.(2007)] Abdulla, P. A., Högberg, J., Kaati, L.: "Bisimulation minimization of tree automata."; International Journal of Foundations of Comp. Sci.; 18 (2007), 4, 699–713.

[Abdulla et al.(2009)] Abdulla, P. A., Holík, L., Kaati, L., Vojnar, T.: "A uniform (Bi-)simulation-based framework for reducing tree automata"; Electronic Notes in Theoretical Computer Science; 251 (2009), 0, 27 – 48; proceedings of the International Doctoral Workshop on Mathematical and Engineering Methods in Computer Science.

[Aho and Ganapathi(1985)] Aho, A. V., Ganapathi, M.: "Efficient tree pattern matching: an aid to code generation"; Proceedings of the 12th ACM Symposium on Principles of Programming Languages; 334–340; 1985.

[Aho et al.(1989)] Aho, A. V., Ganapathi, M., Tjiang, S. W. K.: "Code generation using tree matching and dynamic programming"; ACM Transactions on Programming Languages and Systems; 11 (1989), 4, 491–516.

[Aho et al.(1974)] Aho, A. V., Hopcroft, J. E., Ullman, J. D.: The design and analysis of computer algorithms; Addison-Wesley Series in Computer Science and Information Processing; Addison-Wesley, Reading, MA, 1974.

[Borchardt(2005)] Borchardt, B.: The theory of recognizable tree series; Akademische Abhandlungen zur Informatik; Verlag für Wissenschaft und Forschung, 2005.

[Brainerd(1967)] Brainerd, W. S.: Tree Generating Systems and Tree Automata; Ph.D. thesis; Purdue University (1967).

[Brainerd(1968)] Brainerd, W. S.: "The minimalization of tree automata"; Information and Control; 13 (1968), 5, 484–491.

[Broy(1983)] Broy, M.: "Program construction by transformations: a family tree of sorting programs"; A. W. Biermann, G. Guiho, eds., Computer Program Synthesis Methodologies; 1–49; Reidel, 1983.

[Brzozowski(1962)] Brzozowski, J. A.: "Canonical regular expressions and minimal state graphs for definite events"; Mathematical Theory of Automata; volume 12 of MRI Symposia Series; 529–561; Polytechnic Press, Polytechnic Institute of Brooklyn, 1962.

[Buchholz(2008)] Buchholz, P.: "Bisimulation relations for weighted automata"; Theoretical Computer Science; 393 (2008), 13, 109 – 123.

[Burghardt(1988)] Burghardt, J.: "A tree pattern matching algorithm with reasonable space requirements"; Proceedings of the 13th Colloquium on Trees in Algebra and Programming (CAAP); volume 299 of Lecture Notes in Computer Science; 1–15; 1988.

[Carrasco et al.(2007)] Carrasco, R. C., Daciuk, J., Forcada, M. L.: "An implementation of DTA minimization"; J. Holub, J. Žďárek, eds., Implementation and Application of Automata; volume 4783 of LNCS; 122–129; Springer Berlin Heidelberg, 2007.

[Carrasco et al.(2008)] Carrasco, R. C., Daciuk, J., Forcada, M. L.: "Incremental construction of minimal tree automata"; Algorithmica; (2008).

[Cleophas et al.(2009)] Cleophas, L., Kourie, D. G., Strauss, T., Watson, B. W.: "On minimizing deterministic tree automata"; J. Holub, J. Žďárek, eds., Prague Stringology Conference, Prague, Czech Republic, 2009; 173–182; 2009.

[Cleophas(2008)] Cleophas, L. G. W. A.: Tree Algorithms: Two Taxonomies and a Toolkit; Ph.D. thesis; Dept. of Mathematics and Computer Science, TU Eindhoven (2008).

[Comon et al.(2007)] Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: "Tree automata: Techniques and applications"; (2007).

[Darlington(1978)] Darlington, J.: "A synthesis of several sorting algorithms"; Acta Inf.; 11 (1978), 1–30.

[David(2012)] David, J.: "Average complexity of Moores and Hopcrofts algorithms"; Theoretical Computer Science; 417 (2012), 0, 50 – 65.

[Doner(1970)] Doner, J.: "Tree acceptors and some of their applications"; Journal of Computer and System Sciences; 4 (1970), 5, 406–451.

[Engelfriet(1975)] Engelfriet, J.: "Tree Automata and Tree Grammars"; Lecture Notes DAIMI FN-10; Aarhus University (1975).

[Galley et al.(2006)] Galley, M., Graehl, J., Knight, K., Marcu, D., DeNeefe, S., Wang, W., Thayer, I.: "Scalable inference and training of context-rich syntactic translation models"; Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics; 961–968; ACL, Stroudsburg, PA, USA, 2006.

[Gécseg and Steinby(1984)] Gécseg, F., Steinby, M.: Tree Automata; Akadémiai Kiadó, Budapest, 1984.

[Gécseg and Steinby(1997)] Gécseg, F., Steinby, M.: Tree Languages; volume 3 of Handbook of Formal Languages; 1–68; Springer, 1997.

[Hoffmann and O'Donnell(1982)] Hoffmann, C. M., O'Donnell, M. J.: "Pattern matching in trees"; Journal of the ACM; 29 (1982), 1, 68–95.

[Högberg et al.(2009)] Högberg, J., Maletti, A., May, J.: "Backward and forward bisimulation minimization of tree automata"; Theoretical Comp. Sci.; 410 (2009), 37, 3539–3552.

[Holzer and Maletti(2010)] Holzer, M., Maletti, A.: "An $n \log n$ algorithm for hyperminimizing a (minimized) deterministic automaton"; Theoretical Comp. Sci.; 411 (2010), 38–39, 3404–3413.

[Hopcroft and Ullman(1979)] Hopcroft, J. E., Ullman, J. D.: Introduction to Automata Theory, Languages, and Computation; Addison-Wesley, Reading, Massachusetts, USA, 1979.

[Kron(1975)] Kron, H.: Tree templates and subtree transformational grammars; Ph.D. thesis; University of California, Santa Cruz (1975).

[Maletti(2011)] Maletti, A.: "*Survey:* weighted extended top-down tree transducers — part I: Basics and expressive power"; Acta Cybernetica; 20 (2011), 2, 223–250.

[Maletti and Quernheim(2012)] Maletti, A., Quernheim, D.: "Unweighted and weighted hyper-minimization"; International Journal on the Foundations of Comp. Sci.; 23 (2012), 6, 1207–1225.

[Marcelis(1990)] Marcelis, A. J. J. M.: "On the classification of attribute evaluation algorithms"; Science of Computer Programming; 14 (1990), 1–24.

[Martens and Niehren(2007)] Martens, W., Niehren, J.: "On the minimization of XML schemas and tree automata for unranked trees"; J. Comput. Syst. Sci.; 73 (2007), 4, 550–583.

[Moore(1956)] Moore, E. F.: "Gedanken-experiments on sequential machines"; C. Shannon, J. McCarthy, eds., Automata Studies; 129–153; Princeton University Press, Princeton, NJ, 1956.

[Nerode(1958)] Nerode, A.: "Linear automaton transformations"; Proceedings of the American Mathematical Society; 9 (1958), 4, 541–544.

[Nivat and Podelski(1997)] Nivat, M., Podelski, A.: "Minimal ascending and descending tree automata"; SIAM Journal on Computing; 26 (1997), 39–58.

[Paige and Tarjan(1987)] Paige, R., Tarjan, R.: "Three partition refinement algorithms"; SIAM Journal on Computing; 16 (1987), 6, 973–989.

[Thatcher and Wright(1965)] Thatcher, J. W., Wright, J. B.: "Generalized finite automata"; Notices of the American Mathematical Society; 12 (1965), 820, 65T–469.

[Watson(1995)] Watson, B. W.: Taxonomies and Toolkits of Regular Language Algorithms; Ph.D. thesis; Dept. of Mathematics and Comp. Sci., TU Eindhoven (1995).

[Watson and Daciuk(2003)] Watson, B. W., Daciuk, J.: "An efficient incremental DFA minimization algorithm"; Natural Language Engineering; 9 (2003), 1, 49–64.

[Yamada and Knight(2001)] Yamada, K., Knight, K.: "A syntax-based statistical translation model"; Proceedings of the 39th Annual Meeting on Association for Computational Linguistics; 523–530; ACL, Stroudsburg, PA, USA, 2001.