Journal of Universal Computer Science, vol. 8, no. 2 (2002), 363-367
submitted: 15/9/01, accepted: 27/1/02, appeared: 28/2/02

A Fast and Simple Algorithm for Constructing Minimal Acyclic Deterministic Finite Automata[1]

Bruce W. Watson
(Software Construction Research Group, Department of Computer Science,
Technological University of Eindhoven, the Netherlands
Email: watson@win.tue.nl)
(Department of Computer Science, University of Pretoria, South Africa
Email: watson@cs.up.ac.za)
(FST Labs & Ribbit Software Systems Inc.
Email: watson@fst-labs.com)

Abstract: In this paper, we present a fast and simple algorithm for constructing a minimal acyclic deterministic finite automaton from a finite set of words. Such automata are useful in a wide variety of applications, including computer virus detection, computational linguistics and computational genetics. There are several known algorithms that solve the same problem, though most of the alternative algorithms are considerably more difficult to present, understand and implement than the one given here. Preliminary benchmarking indicates that the algorithm presented here is competitive with the other known algorithms.
Key Words: minimal acyclic deterministic finite automata, automata construction algorithms
Category: F.1.1

1 Introduction

In this paper, we present a fast and simple algorithm for constructing a minimal acyclic deterministic finite automaton from a finite set of words. Such automata (or variants of them) are useful in a wide variety of applications, such as: computer virus detection, computational linguistics (spell checking/correction, morphological analysis, etc.) and computational genetics. There are several known algorithms that solve the same problem [see Watson 2002], however, most of the alternative algorithms are considerably more difficult to present, understand and implement than the one given here. The presentation of the new algorithm is particularly elegant, relying on a minimum of formal language principles.

1.1 Preliminaries

We assume that readers are familiar with the fundamentals of formal languages and automata (including determinism and minimality) [see Hopcroft and Ullman 1979]. Here, we briefly present the additional required mathematical preliminaries.

Throughout, we assume a fixed alphabet Σ. Given a language L, define $\mathbf{pref}(L)$ (resp. $\mathbf{suff}(L)$) to be the set of all (not necessarily proper) prefixes (resp. suffixes) of words in L; note that $\varepsilon \in \mathbf{pref}(L), \mathbf{suff}(L)$ when $L \neq \emptyset$.

[1] C. S. Calude, K. Salomaa, S. Yu (eds.). *Advances and Trends in Automata and Formal Languages. A Collection of Papers in Honour of the 60th Birthday of Helmut Jürgensen.*

1.2 Problem statement

Given a finite set of words W, construct a minimal acyclic deterministic finite automaton accepting W. Such an automaton is known to be unique (up to isomorphism of states).

2 Automata with structured states

In typical presentations of automata algorithms, a state has no internal structure (perhaps they are simply integers, as would be the case in a typical implementation). Instead, our states carry additional information useful in showing the correctness of the algorithm.

Definition 1 (Structured states) *A state* (L, R) *is a pair of languages. We define two 'language extraction' operators such that* $\overleftarrow{(L, R)} = L$ *and* $\overrightarrow{(L, R)} = R$.

The intuition behind this definition is that $\overleftarrow{p}$ (resp. $\overrightarrow{p}$) is the 'left (resp. right) language' of p. The left (resp. right) language of a state is the set of words appearing on paths from a start state to p (resp. from p to a final state). Any state with $\emptyset$ as left or right language is a *useless* state and can be eliminated from the automaton without changing the language accepted.

Not all possible state sets necessarily make a valid finite automaton, so we now define when a state set is 'valid'.

Definition 2 (Valid state set) *Given language* W*, a state set* Q *is 'valid' iff:*

- *(Containment) For all states* $p : \overleftarrow{p}\,\overrightarrow{p} \subseteq W$ *(only valid words are accepted); and*
- *(Coverage)* $W = \cup_{p \in Q} \overleftarrow{p}\,\overrightarrow{p}$ *(all words in* W *are accepted); and*
- *(Prefix closure)* $\mathbf{pref}(W) = \cup_{p \in Q} \overleftarrow{p}$ *(all prefixes of* W *lead to some state); and*
- *(Suffix closure)* $\mathbf{suff}(W) = \cup_{p \in Q} \overrightarrow{p}$ *(all suffixes of* W *lead from some state to a final state).*

Additionally, there must exist some 'transition relation' δ *(a ternary relation over* $Q \times \Sigma \times Q$*) such that for all states* q *we have* $\overrightarrow{q} = \cup_{a,r:(q,a,r) \in \delta} \{a\} \overrightarrow{r}$ *and* $\overleftarrow{q} = \cup_{p,a:(p,a,q) \in \delta} \overleftarrow{p} \{a\}$.

Throughout this paper, we assume some valid state set Q.

Remark 1 (Transitions and start and final states) *With the definition of valid state sets, we need not explicitly discuss the transition function (it follows easily from the definition of valid state sets). The start state(s) are those* p *such that* $\varepsilon \in \overleftarrow{p}$ *and the final state(s) are those* p *such that* $\varepsilon \in \overrightarrow{p}$.

Thanks to the structured states, we can define a number of additional properties of automata, including determinism and minimality.

Definition 3 (Determinism) *An automaton is 'deterministic' iff for all* $p, q \in Q : p \neq q$

$$\overleftarrow{p} \cap \overleftarrow{q} = \emptyset$$

Intuitively, in a deterministic automaton, a word w *cannot appear in the left language of two distinct states since that would imply two* w *paths from a start state. Determinism also implies no more than one start state.*

Definition 4 (Minimality) *A deterministic automaton is 'minimal' iff for all* $p, q \in Q : p \neq q$

$$\overrightarrow{p} \neq \overrightarrow{q}$$

In a minimal deterministic automaton, if two distinct states have the same right language, they could be merged (see below) into one, making a smaller deterministic automaton.

Towards presenting an automaton construction algorithm, we give two transformations on automata — both of which preserve the state validity and the language accepted by the automaton.

Definition 5 (State merging) *Two states* p, q *can be merged when* $\overrightarrow{p} = \overrightarrow{q}$*, giving a single state*

$$(\overleftarrow{p} \cup \overleftarrow{q}, \overrightarrow{p})$$

In practice, this is done by combining their in-transitions and using the out-transitions from just one of them. This transformation preserves determinism.

Definition 6 (State splitting) *Two states* p, q *can be split (also known as 'intersected') and replaced by three states:*

- *(Intersection)* $(\overleftarrow{p} \cap \overleftarrow{q}, \overrightarrow{p} \cup \overrightarrow{q})$.
- *(p residual)* $(\overleftarrow{p} - \overleftarrow{q}, \overrightarrow{p})$.
- *(q residual)* $(\overleftarrow{q} - \overleftarrow{p}, \overrightarrow{q})$.

Clearly, in cases where one of the resulting left languages is $\emptyset$*, the state need not be constructed at all.*

Both of these operations can be expanded to multi-ary operations.

3 Constructing a minimal acyclic deterministic automaton

In order to build a minimal acyclic deterministic automaton accepting W, we have at least two possible choices:

1. Build an acyclic deterministic automaton accepting W and then merge states which have equal right languages. This is the traditional approach taken by most of the algorithms [see Watson 2002].
2. Build an acyclic (not necessarily deterministic) automaton accepting W and split states which share words in their left languages — eventually yielding an automaton in which the states have pairwise disjoint left languages (it is deterministic) — while simultaneously ensuring that the right languages are pairwise unequal (it is minimal).

Since the first approach is well-explored, we pursue the second approach in this paper.

As input to the state splitting phase, we require some property on the automaton so that state splitting preserves inequality of right languages (and therefore leads to minimality once the automaton is deterministic).

Definition 7 (Condition for minimality under state splitting) *A sufficient condition is:*

– *For all states p, $\overrightarrow{p}$ is a singleton set; and*
– *For all states $p, q : p \neq q$*

$$\overrightarrow{p} \neq \overrightarrow{q}$$

From the definition of a valid state set and the above condition, we also see that the right languages are a partition (into singletons) of $\mathbf{suff}(W)$.

Proposition 1 *Assume that the above condition holds. We perform multi-ary state splitting, always choosing a largest set of states Q' (to split simultaneously) such that $\cap_{p \in Q'} \overleftarrow{p} \neq \emptyset$. Such state splitting (starting with the above condition holding) can never yield two states $p, q : \overrightarrow{p} = \overrightarrow{q}$. This follows from*

– *$\overrightarrow{p} = \overrightarrow{q}$ does not hold initially (by the condition).*
– *If $\overrightarrow{p} = \overrightarrow{q}$, then p and q must both be the result of state splitting. (It is not possible for one of them to be an 'original' state and the other to result from state splitting, since state splitting in this case yields non-singleton right languages.) However, since the original states' right languages are a* partition *of* $\mathbf{suff}(W)$*, if $\overrightarrow{p} = \overrightarrow{q}$, then p and q must be the result of splitting the same states, in which case $p = q$.*

When no further splitting is possible, we have the minimal acyclic deterministic finite automaton for W.

4 Implementation

Two issues remain:

1. How to obtain an automaton satisfying our condition. This is rather easily satisfied, by observing that such an automaton is the reverse of the 'reverse trie' corresponding to W. Such an automaton can be built in time $\mathcal{O}(\sum_{w \in W} |w|)$ using an algorithm given in most automata text books or in [Watson 2002].
2. Implementing state splitting. We can perform the state splitting in a topological traversal (of the transition function), starting with the set of states $p : \varepsilon \in \overleftarrow{p}$. Interestingly, the algorithm corresponds to a version of the 'subset construction' algorithm (sometimes known as the 'powerset construction' algorithm) for determinizing an automaton. For details, consult [Hopcroft and Ullman 1979].

Since they are presented in a number of text books, these two component algorithms are not explicitly given here.

5 Commentary

The algorithm presented here can also be derived from Brzozowski's minimization algorithm as was done in [Watson 1999] and [Watson 2002]. (See [Watson 2001] for presentations of Brzozowski's minimization algorithm.) The running time of the algorithm presented here is known to be $\mathcal{O}(\sum_{w \in W} |w|)$ [see Watson 2002].

Acknowledgements

Naturally, I would like to thank Helmut for his contributions to this field and congratulate him on this birthday milestone. I would also like to thank Nanette Saes and Kai Salomaa for proofreading this paper.

References

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

[Watson 1999] Watson, B.W.: "A Taxonomy of Algorithms for Constructing Minimal Acyclic Deterministic Automata"; Proc. Workshop on Implementing Automata 1999, Potsdam, Germany (1999)

[Watson 2001] Watson, B.W.: "A History of Brzozowski's DFA Minimization Algorithm"; Computer Science Technical Report, University of Pretoria, South Africa (2001)

[Watson 2002] Watson, B.W.: "Constructing Minimal Acyclic Deterministic Finite Automata"; to appear (2002)