A taxonomy of algorithms for constructing minimal acyclic deterministic finite automata

A Taxonomy of Algorithms for Constructing Minimal Acyclic Deterministic Finite Automata

BW Watson

Department of Computer Science, University of Pretoria, Pretoria 0002, South Africa, watson@OpenFIRE.org
Ribbit Software Systems Inc. & FST Labs, watson@fst-labs.com

Abstract

In this paper, we present a taxonomy of algorithms for constructing minimal acyclic deterministic finite automata (MADFAs). Such automata represent finite languages and are therefore useful in applications such as storing words for spell-checking, computer and biological virus searching, text indexing and XML tag lookup. In such applications, the automata can grow extremely large (with more than 10^6 states) and are difficult to store without compression or minimization. The taxonomization method arrives at all of the known algorithms, and some which are likely new ones (though proper attribution is not always possible, since the algorithms are usually of commercial value and some secrecy frequently surrounds the identities of the original authors).

Keywords: *minimal automata, acyclic deterministic automata, dictionaries, algorithmics*
Computing Review Categories: *D.1.4, E.1, F.2.2, G.2.2*

1 Introduction

In this paper, we present a taxonomy of algorithms for constructing minimal acyclic deterministic finite automata (MADFAs). MADFAs represent finite languages and are therefore useful in applications such as storing words for spell-checking, computer and biological virus searching, text indexing and XML tag lookup. In such applications, the automata can grow extremely large and are difficult to store without compression or minimization. Whereas compression is considered in various other papers (and is usually specific to data-structure choices), here we focus on minimization.

We apply the following technique for taxonomizing the algorithms:

1. At the root of the taxonomy is a simple, if inefficient, algorithm whose correctness is either easy to prove or is simply postulated.

2. New algorithms are derived by adding an algorithm detail — a correctness-preserving transformation of the algorithm or elaboration of program statements. This yields an algorithm which is still correct.

3. By carefully choosing the details, all of the well-known algorithms appear in the taxonomy. Creative invention of new details also yields new algorithms.

This technique was applied on a large scale in the my Ph.D dissertation [12]. The dissertation also contains taxonomies of algorithms for constructing finite automata from regular expressions and for minimizing deterministic finite automata. In this paper, we only sketch the algorithms and relate them to one another. Full presentations and correctness arguments are given in the papers appearing in the reference list. Here, we assume some familiarity with the common algorithms for automata construction and minimization — see for example [12].

This paper is structured as follows:

- §2 begins with a naïve first algorithm, whose correctness is easily shown.

- §3, §4 and §5 elaborate on a family of closely-related algorithms all derived from a common skeleton.

- §6 discusses incremental algorithms which maintain minimality throughout the construction process.

- §7 gives algorithms for building an automaton (not necessarily minimal) for a set of words. These algorithms form the building blocks of some other algorithms.

- §8 gives the conclusions of this paper.

1.1 Related work

The work presented here is significantly different from the taxonomies presented in the dissertation, since specializing for MADFAs can yield particularly efficient algorithms.

Some of the algorithms included in this taxonomy were previously presented, for example, in [11, 6, 13, 16, 9, 5, 17]. Other algorithms for the MADFA construction problem have typically been kept as trade secrets (due to their commercial success in applications such as spell-checking). As such, many of them have likely been known for some number of years, but tracing the original authors will be difficult and proper attributions are not attempted

— though I would welcome hearing from researchers who performed some of the original work.

1.2 Preliminaries

We make the following definitions:

- FA is the set of all *finite automata*.
- DFA is the set of all *deterministic* FAs.
- ADFA is the set of all *acyclic* DFAs.
- MADFA is the set of all *minimal* ADFAs.

More precise definitions are not required here. In this paper, we are primarily interested in algorithms which build MADFAs. The algorithms are readily extended to work with acyclic deterministic *transducers*, though such an extension is not considered.

For any $M \in$ FA, $|M|$ is the number of states in M and $\mathcal{L}(M)$ is the language (set of words) accepted by M. The primary definition of minimality of an $M \in$ DFA is: $(\forall M' \in \text{DFA} : \mathcal{L}(M') = \mathcal{L}(M) : |M| \leq |M'|)$.

Predicate $Min(M)$ holds when $M \in$ DFA and the above definition of minimality both hold. A useful DFA property is: $\mathcal{L}(M)$ is finite $\wedge\ Min(M) \equiv M \in$ MADFA.

We assume the existence of three functions (which are easily implemented as procedures):

1. reverse reverses an automaton, yielding one accepting the reverse of the language accepted by the argument automaton.
2. determinize determinizes an automaton, yielding a DFA.
3. negate negates its argument automaton, yielding one accepting the negation of the language accepted by the argument.

We further assume that these functions do not introduce useless states.

We also take superscript operator R as the reverse operator on strings and languages.

All of the algorithms presented here are in the guarded command language — see [7].

2 A naïve first algorithm

In this section, we present our first algorithm and outline some ways in which to proceed. The problem is as follows: given alphabet Γ and some finite set of words $W \subset \Gamma^*$ (the containment is proper, since Γ^* is infinite), compute some $M \in$ ADFA such that $\mathcal{L}(M) = W \wedge Min(M)$. In the algorithms that follow, we give M the type FA, which is the most general type in the containment MADFA $\subset$ ADFA $\subset$ DFA $\subset$ FA. At certain points in the program, the variable M may actually contain a MADFA.

Given this, our first algorithm (where S is a program statement still to be derived) is:

Algorithm 2.1:

$\{ W \subseteq \Gamma^* \wedge W \text{ is finite} \}$
S
$\{ \mathcal{L}(M) = W \wedge Min(M) \}$

□

In order to make some progress, we consider a split of statement S to accomplish the postcondition in two steps:

Algorithm 2.2:

$\{ W \subseteq \Gamma^* \wedge W \text{ is finite} \}$
S_0;
$\{ \mathcal{L}(M) = f(W) \wedge X(M) \}$
S_1
$\{ \mathcal{L}(M) = W \wedge Min(M) \}$

□

There are, of course, infinitely many choices for function f and predicate X, some of which are not interesting. For example, if we define $f(W) = \emptyset$, then after S_0, we will have accomplished virtually nothing (since the automaton will accept the empty language), regardless of how we define X. For this reason, we restrict ourselves to the following three possibilities for f:

1. $f(W) = W$ (the identity function).
2. $f(W) = W^R$ (the reversal of the W).
3. $f(W) = \neg W$ (the complement of W: $\neg W = \Gamma^* - W$).

Other choices are possible. These were chosen because:

- in some sense, statement S_0 will accomplish a reasonable amount of work,
- it is easy to convert an $f(W)$-accepting DFA to a W-accepting one, and
- with these choices, we can arrive at many of the known algorithms.

We consider each choice of f in the following sections.

3 $f(W) = W$

We can now turn to choices for predicate X. Clearly, any predicate can be chosen, but we restrict our choices to (at least) arrive at the known algorithms. As a first option, consider *strengthenings* of Min, that is: $X(M) \Rightarrow Min(M)$. In that case, we choose S_1 to be the **skip** statement (which does nothing, since $Min(M)$ already holds), and we are left with a statement S_0 which is as difficult to derive as our first algorithm (Algorithm 2.1). For this reason, we abandon strengthenings of Min (including the possiblity $X(M) \equiv Min(M)$).

Instead, we turn our attention to weakenings of Min, and one predicate which is not related by implication to Min.

3.1 $X(M) \equiv M \in \mathsf{DFA}$

This yields:

Algorithm 3.1:

$\{ W \subseteq \Gamma^* \wedge W \text{ is finite } \}$
S_0;
$\{ \mathcal{L}(M) = W \wedge M \in \mathsf{DFA} \}$
S_1
$\{ \mathcal{L}(M) = W \wedge Min(M) \}$

□

For S_0, we can use any algorithm which yields an automaton M such that $\mathcal{L}(M) = W \wedge M \in \mathsf{DFA}$. In §7, we separately consider algorithms for doing this — though the algorithm for building a *trie* is easiest to implement [8, 1].

Since the result of S_0 is a DFA, for S_1 we can use any of the minimization algorithms in [12, Chapter 7] or the one given by [10].

Clearly the extensive choices for S_0 and S_1 yield an entire subtree of the taxonomy — and therefore an entire subfamily of algorithms.

3.2 $X(M) \equiv \mathsf{reverse}(M) \in \mathsf{DFA}$

This yields:

Algorithm 3.2:

$\{ W \subseteq \Gamma^* \wedge W \text{ is finite } \}$
S_0;
$\{ \mathcal{L}(M) = W \wedge \mathsf{reverse}(M) \in \mathsf{DFA} \}$
S_1
$\{ \mathcal{L}(M) = W \wedge Min(M) \}$

□

For S_0, we can use any algorithm which yields an automaton M such that $\mathcal{L}(M) = W \wedge \mathsf{reverse}(M) \in \mathsf{DFA}$. The algorithms given in §7 can easily be modified to construct such an automaton.

It is no accident that reversal was used in the above algorithm: it is known to be related to minimality via Brzozowski's minimization algorithm [3, 2, 12, 15] (in those presentations, the history of the algorithm is given, along with full correctness arguments for each part of the algorithm). Brzozowski's algorithm, for some $M \in \mathsf{FA}$ (not necessarily a DFA), is:

Algorithm 3.3:

$M' := \mathsf{reverse}(M)$;
$M' := \mathsf{determinize}(M')$;
$\{ \mathcal{L}(M') = \mathcal{L}(M)^R \wedge M' \in \mathsf{DFA} \}$
$M' := \mathsf{reverse}(M')$;
$\{ \mathcal{L}(M') = \mathcal{L}(M) \wedge \mathsf{reverse}(M') \in \mathsf{DFA} \}$
$M' := \mathsf{determinize}(M')$
$\{ \mathcal{L}(M') = \mathcal{L}(M) \wedge Min(M') \}$

□

Given $X(M)$ and Brzozowski's minimization algorithm, it is sufficient to determinize M, yielding $M : Min(M)$.

The resulting algorithm is an acyclic version of the one given in [14].

3.3 $X(M) \equiv true$

By writing our choices of f and X in full, our program becomes:

Algorithm 3.4:

$\{ W \subseteq \Gamma^* \wedge W \text{ is finite } \}$
S_0;
$\{ \mathcal{L}(M) = W \}$
S_1
$\{ \mathcal{L}(M) = W \wedge Min(M) \}$

□

As in §3.1, for S_0, we can use any algorithm which yields an automaton M such that $\mathcal{L}(M) = W$ — see §7. In particular, we are not restricted to only those algorithms yielding a DFA.

If S_0 is an algorithm yielding a DFA, then for S_1 we can use any of the minimization algorithms in [12, Chapter 7] or the one given by [10]. If M delivered by S_0 is not deterministic, we can either use Brzozowski's minimization algorithm (see [12, Chapter 7]) or first apply the subset construction (to determinize M) and then any one of the other minimization algorithms.

3.4 $X(M)$ as partial minimality

In [13, 16], a partial minimality predicate is introduced and it is shown to be a weakening of *Min*. This yields the following algorithm:

Algorithm 3.5:

$\{ W \subseteq \Gamma^* \wedge W \text{ is finite } \}$
S_0;
$\{ \mathcal{L}(M) = W \wedge X(M) \}$
S_1
$\{ \mathcal{L}(M) = W \wedge Min(M) \}$

□

In the original paper, S_0 is derived as an algorithm which constructs M as a *partially minimal* DFA (in which many, but not all, of the redundant states are already combined), while S_1 is derived as a 'cleanup' phase to finalize the minimization. The interested reader is referred to the presentation in that paper.

4 $f(W) = W^R$

In §3.2, we made use of the relationship between reversal and minimality — namely, through Brzozowski's minimization algorithm. Thanks to this, the most obvious choice for predicate X is $X(M) \equiv M \in \mathsf{DFA}$. In that case, our program is

Algorithm 4.1:

$\{ W \subseteq \Gamma^* \wedge W \text{ is finite } \}$
S_0;
$\{ \mathcal{L}(M) = W^R \wedge M \in \mathsf{DFA} \}$
S_1
$\{ \mathcal{L}(M) = W \wedge Min(M) \}$

□

Based on Brzozowski's algorithm, we expand S_1 in the above program:

Algorithm 4.2:

$\{ W \subseteq \Gamma^* \wedge W$ is finite $\}$
S_0;
$\{ \mathcal{L}(M) = W^R \wedge M \in \mathsf{DFA} \}$
$M := \mathsf{reverse}(M)$;
$M := \mathsf{determinize}(M)$
$\{ \mathcal{L}(M) = W \wedge Min(M) \}$

□

For S_0, there are a number of algorithms for building a DFA for W (see §7), and we can easily modify them to deal with W^R.

5 $f(W) = \neg W$

It is known that DFA minimality is preserved under negation of the DFA, at least using most reasonable definitions of a negating mapping[1]. With this, we choose $X(M) \equiv Min(M)$. This yields

Algorithm 5.1:

$\{ W \subseteq \Gamma^* \wedge W$ is finite $\}$
S_0;
$\{ \mathcal{L}(M) = \neg W \wedge Min(M) \}$
S_1
$\{ \mathcal{L}(M) = W \wedge Min(M) \}$

□

With the preservation of minimality under negation, we select S_1 to be the negation function, giving

Algorithm 5.2:

$\{ W \subseteq \Gamma^* \wedge W$ is finite $\}$
S_0;
$\{ \mathcal{L}(M) = \neg W \wedge Min(M) \}$
$M := \mathsf{negate}(M)$
$\{ \mathcal{L}(M) = W \wedge Min(M) \}$

□

Statement S_0 can be further split, giving

Algorithm 5.3:

$\{ W \subseteq \Gamma^* \wedge W$ is finite $\}$
S_0';
$\{ \mathcal{L}(M) = \neg W \}$
S_0'';
$\{ \mathcal{L}(M) = \neg W \wedge Min(M) \}$
$M := \mathsf{negate}(M)$
$\{ \mathcal{L}(M) = W \wedge Min(M) \}$

□

In this case, S_0' builds M corresponding to $\neg W$; this can be accomplished by first building M corresponding to W and then applying negate. (There may, of course, be other algorithms still to be derived.) Subsequently, S_0'' corresponds to some minimization algorithm, for example, those given in [10, 12]. The running time advantages of including this negation step are not yet clear.

[1] We assume that negate is able to work on DFAs with partial transition functions, since a total transition function would (in the case of non-empty DFAs) be cyclic.

6 $Min(M)$ as an invariant

In the previous sections, we have considered algorithms with two parts: S_0 and S_1. We return to Algorithm 2.1 — the root of the taxonomy — to obtain the following algorithm, where we use $Min(M)$ as a repetition invariant:

Algorithm 6.1:

$\{ W \subseteq \Gamma^* \wedge W$ is finite $\}$
$M := empty\_DFA$;
$Done, To\_do := \emptyset, W$;
$\{$ invariant: $\mathcal{L}(M) = Done \wedge Min(M) \wedge Done \cup To\_do = W \wedge Done \cap To\_do = \emptyset$
variant: $|To\_do|$ $\}$
do $To\_do \neq \emptyset \rightarrow$
  S_2; $\{$ choose some word w in To_do $\}$
  $\{ w \in To\_do \}$
  $Done, To\_do := Done \cup \{w\}, To\_do - \{w\}$;
  S_3
od
$\{ Done = W \}$
$\{ \mathcal{L}(M) = W \wedge Min(M) \}$

□

We now consider possible versions of statements S_2 and S_3. There are two straightforward ways to proceed with S_2:

1. *Lexicographically order the words in W.* Obtaining the elements of W in lexicographic order is easily implemented. To implement statement S_3, a derivation was recently given in [6, 5, 4], and the interested reader is referred to that paper.

2. *Unordered choice from W.* This is the easiest way in which to select an element of W. An implementation of S_3 was also derived independently in [5, 4] and [11]. In [17], a recursive algorithm implementing the specification of S_3 is presented. None of them are considered in detail here.

These algorithms are the only known fully incremental MADFA construction algorithms. Interestingly, they have running time which is linear in the size of W (as does Revuz's algorithm [10] — an algorithm related to the two mentioned here).

7 Constructing a (not necessarily minimal) finite automaton

In this section, we briefly discuss some algorithms for constructing a finite automaton from W:

1. One obvious (though not very efficient) method is to first build a regular expression from W (as $w_0 + w_1 + \cdots + w_{|W|-1}$ for words $w_i \in W$) and then use one of the

although it is likely to be slow due to its generality. It is possible, however, that some improvements could be made based upon the simple (star-free) structure of the regular expressions.

2. For each $w \in W$, we build a simple linear finite automaton with $|w| + 1$ states (the transitions are respectively labeled with the letters from w). The final (nondeterministic) finite automaton is built by combining all of the individual automata and adding a new start state with ε-transitions to the individual start states. As with the above algorithm, this one is very likely to be slow.

3. For each $w \in W$, we apply the standard algorithm for adding a word to a *trie*-structured DFA [8, 1, 18].

8 Conclusions

We have presented a straightforward taxonomy of algorithms for constructing minimal acyclic deterministic finite automata. The taxonomy begins with an algorithm which has unelaborated statements, postulated to be correct. Each of the subsequent algorithms is derived by applying correctness-preserving transformations to the initial algorithm. In the course of constructing the taxonomy, all of the pre-existing algorithms were derived — including some of the most recently presented incremental algorithms. Furthermore, the taxonomy elaborated on two other groups of algorithms:

- Many of the original, and efficient, algorithms were previously only known as trade secrets in industry.
- Some of the intermediate algorithms contain dead-ends or have derivation possibilities which are unexplored.

There are a number of areas of future research:

- Several unexplored directions were highlighted in the taxonomy. Some of these may, in fact, lead to new algorithms of practical importance.
- The theoretical and benchmarked running time of the algorithms has not been adequately explored and are not given in this paper. This will allow the careful choice of an algorithm to apply in practice.

Acknowledgements: I would like to thank Nanette Y. Saes and the anonymous referees for improving the quality of this paper.

References

[1] Alfred V. Aho. *Algorithms for finding patterns in strings*, volume A, pages 257–300. North-Holland, 1990.

[2] Janusz A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. volume 12 of *MRI Symposia Series*, pages 529–561, Polytechnic Institute of Brooklyn, 1962. Polytechnic Press.

[3] Janusz A. Brzozowski. *Regular Expression Techniques for Sequential Circuits*. PhD thesis, Princeton University, Princeton, New Jersey, June 1962.

[4] Jan Daciuk. *Incremental Construction of Finite-State Automata and Transducers, and their Use in the Natural Language Processing*. PhD thesis, Technical University of Gdańsk, Poland, 1998.

[5] Jan Daciuk, Stoyan Mihov, Bruce W. Watson, and Richard E. Watson. Incremental construction of minimal acyclic finite state automata. *Computational Linguistics*, 26(1):3–16, April 2000.

[6] Jan Daciuk, Bruce W. Watson, and Richard E. Watson. Incremental construction of minimal acyclic finite state automata and transducers. In Lauri Karttunen and Kemal Oflazer, editors, *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pages 48–56, Ankara, Turkey, June 1998.

[7] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[8] E. Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[9] Stoyan Mihov. *Direct Building of Minimal Automaton for Given List*. PhD thesis, Bulgarian Academy of Science, 1999.

[10] Dominique Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92:181–189, 1992.

[11] K.N. Sgarbas, N.D. Fakotakis, and G.K. Kokkinakis. Two algorithms for incremental construction of directed acyclic word graphs. *International Journal of Artificial Intelligence Tools*, 4:369–381, 1995.

[12] Bruce W. Watson. *Taxonomies and Toolkits of Regular Language Algorithms*. PhD thesis, Division of Computer Science, Eindhoven University of Technology, the Netherlands, September 1995.

[13] Bruce W. Watson. A fast new semi-incremental algorithm for the construction of minimal acyclic DFAs. In Derick Wood and Denis Maurel, editors, *Proceedings of the Third Workshop on Implementing Automata*, volume 1660 of *Lecture Notes in Computer Science*, pages 91–98, Rouen, France, September 1998. Springer-Verlag.

[14] Bruce W. Watson. Combining two algorithms by Brzozowski. *submitted to South African Computer Journal*, 2001.

[15] Bruce W. Watson. A history of Brzozowski's DFA minimization algorithm. Technical report, Department of Computer Science, University of Pretoria, South Africa, 2001.

[16] Bruce W. Watson. A new algorithm for the construction of minimal acyclic DFAs. *to appear in Science of Computer Programming*, 2001.

[17] Bruce W. Watson. A new recursive algorithm for building minimal acyclic deterministic finite automata. Technical report, Department of Computer Science, University of Pretoria, South Africa, 2001.

[18] Bruce W. Watson and Gerard Zwaan. A taxonomy of keyword pattern matching algorithms. Technical Report 27, Division of Computer Science, Eindhoven University of Technology, the Netherlands, 1992.