A Taxonomy of Algorithms for Constructing Minimal Acyclic Deterministic Finite Automata

Bruce W. Watson

[1] watson@OpenFIRE.org
www.OpenFIRE.org
University of Pretoria
(Department of Computer Science)
Pretoria 0002, South Africa
[2] Ribbit Software Systems Inc.
(IST Technologies Research Group)
Kelowna, B.C., V1Y 9P9, Canada

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 (with more than 10^6 states) 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 most recently applied on a large scale in the author's Ph.D dissertation [5]. The dissertation also contains taxonomies of algorithms for constructing finite automata from regular expressions and for minimizing deterministic finite automata. Here, we assume some familiarity with the common algorithms for automata construction and minimization.

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 Turkey [1] (the present author, Jan Daciuk and Richard Watson), at WIA'98 [6] (the present author) and in [3] (Stoyan Mihov). 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.

1.1 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 $M \in \mathsf{FA}$, $|M|$ 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 $M \in \mathsf{ADFA}$ (indeed, this definition applies to *any* DFA, not just acyclic ones) is: $(\forall\, M' \in \mathsf{MADFA} : \mathcal{L}(M') = \mathcal{L}(M) : |M| \leq |M'|)$.

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

All of the algorithms presented here are in the guarded command language, a type of pseudo-code — see [2].

2 A 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 $W \subset \Gamma^*$, compute some $M \in \mathsf{ADFA}$ 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 $\mathsf{MADFA} \subset \mathsf{ADFA} \subset \mathsf{DFA} \subset FA$. At any point 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
$\{\ \mathcal{L}(M) = W \wedge \mathit{Min}(M)\ \}$

□

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,
- furthermore, it is reasonably 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 X, that is: $X(M) \Rightarrow \mathit{Min}(M)$. In that case, we choose S_1 to be the **skip** statement (which does nothing, since $\mathit{Min}(M)$ already holds), and we are left with a statement S_0 which is as difficult to derive as our first algorithm. For this reason, we abandon strengthenings of *Min* (including the possiblity $X(M) \equiv \mathit{Min}(M)$).

Instead, we turn our attention to weakenings[1] of *Min*. We begin with the extreme of these weakenings: *true*.

[1] We could equally choose some X which is not related by implication to *Min*; this has not been explored and is a topic for future research, since it may lead to interesting algorithms.

3.1 $X(M) \equiv true$

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

Algorithm 3.1:

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

□

For S_0, we can use any algorithm which yields an automaton M such that $\mathcal{L}(M) = W$. In §7, we separately consider algorithms for doing this.

If the expansion of S_0 is an algorithm yielding a DFA, then for S_1 we can use any of the minimization algorithms in [5, Chapter 7] or the one given by [4]. If M delivered by S_0 is not deterministic, we can either use Brzozowski's minimization algorithm (see [5, Chapter 7]) or first apply the subset construction (to determinize M) and then any one of the other minimization algorithms.

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

3.2 $X(M) \equiv M \in$ DFA

This yields:

Algorithm 3.2:

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

□

The choices for S_0 are as in §3.1 and are discussed in §7. Similarly, for S_1, there are a number of choices (see [5, Chapter 7] and [4]) — though we are already certain that M is a DFA and no determinization step is required.

3.3 $X(M)$ as partial minimality

In [6], a partial minimality predicate is introduced and it is shown to be a weakening of *Min*. This yields the following algorithm:

Algorithm 3.3:

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

□

In the original paper, S_0 is derived as an algorithm which constructs M as a *partially minimal* DFA, 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 $f(W) = W^R$

It is no accident that reversal was used in f: it is known to be related to minimality via Brzozowski's minimization algorithm [5] (in that presentation, the history of the algorithm is given, along with full correctness arguments for each part of the algorithm). Brzozowski's algorithm, for some $M \in$ FA (not necessarily a DFA), is:

Algorithm 4.1:

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

□

Thanks to this, the most obvious choice for predicate X is $X(M) \equiv M \in$ DFA. In that case, our program is

Algorithm 4.2:

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

□

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

Algorithm 4.3:

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

□

For S_0, there are a number of algorithms for building a DFA from W (see §7), and we can trivially 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[2]. Armed with this, we choose $X(M) \equiv Min(M)$. This yields

Algorithm 5.1:

$\{\ W \subseteq \Gamma^* \wedge W \text{ is finite }\}$
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:

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

□

Statement S_0 can be further split, giving

[2] We assume a negating mapping which 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.

Algorithm 5.3:

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

□

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 [4, 5]. The running time advantages of including this negation step are not yet clear.

6 *Min*(*M*) as an invariant

In the previous section, we have considered algorithms with two primary 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 $\mathit{Min}(M)$ as a repetition invariant:

Algorithm 6.1:

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

□

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

- *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 [1], and the interested reader is referred to that paper.

– *Unordered choice from* W. This is the easiest way in which to select an element of W. As above, an implementation of S_3 was also derived in [1], and it is not considered in detail here.

These two algorithms are the only two fully incremental MADFA construction algorithms known. Both of them have running time which is linear in the size of W (as does Revuz's algorithm [4] — 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 $w_0+w_1+\cdots+w_{|W|-1}$) and then use one of the general construction algorithms given in [5, Chapter 6]. This algorithm has not yet been benchmarked, although it is likely to be slow due to the 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. Such algorithms are presented in most algorithm texts.

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:

- Although most of the algorithm details are intuitively correct, the full correctness arguments must be provided.
- A number of 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 Saes and the anonymous referees for improving the quality of this paper.

References

1. Daciuk, J.D., Watson, B.W. and R.E. Watson. An Incremental Algorithm for Constructing Acyclic Deterministic Transducers. (Proceedings of the International Workshop on Finite State Methods in Natural Language Processing, Ankara, Turkey, 30 June–1 July 1998).
2. Dijkstra, E.W. *A Discipline of Programming.* (Prentice Hall, Englewood Cliffs, N.J., 1976).
3. Mihov, S. Direct Building of Minimal Automaton for Given List. (Available from `stoyan@lml.acad.bg`).
4. Revuz, D. Minimisation of acyclic deterministic automata in linear time. (Theoretical Computer Science 92, pp. 181-189, 1992).
5. Watson, B.W. *Taxonomies and Toolkits of Regular Language Algorithms.* (Ph.D dissertation, Eindhoven University of Technology, The Netherlands, 1995). See `www.OpenFIRE.org`
6. Watson, B.W. A Fast New Semi-Incremental Algorithm for the Construction of Minimal Acyclic DFAs. (Proceedings of the Third Workshop on Implementing Automata, Rouen, France, September 1998).