

Constructing minimal acyclic deterministic finite automata

Bruce W. Watson

bruce@bruce-watson.com

FASTAR Research Group

www.fastar.org

Chair and Professor of Software Construction
Software Construction Research Group
Faculty of Mathematics and Computing Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600MB Eindhoven
The Netherlands
watson@win.tue.nl

Professor of Computer Science
Department of Computer Science
School for Information Technology
University of Pretoria
Lynwood Road, Pretoria 0002
Republic of South Africa
watson@cs.up.ac.za

Contents

Preface	6
1 Introduction	9
1.1 Problem statement	10
1.2 To the reader	10
1.3 Related work and a short history	11
1.4 Links to the literature	13
2 Preliminaries	15
2.1 General definitions	15
2.2 Strings and languages	16
2.3 Automata	17
2.4 Minimality of automata	24
2.4.1 Computing E	28
2.4.2 Computing $\neg E$ in <i>ADFAs</i>	28
2.4.3 Computing $E(p, q)$ pointwise	28
2.4.4 A more efficient computation of E	29
3 A <i>MADFA</i>-construction algorithm skeleton	31
3.1 Commentary	34
4 Trie intermediate <i>ADFA</i>	35
4.1 Procedure add_word_T	35
4.1.1 Adding only prefix words	36
4.1.2 Adding a nonprefix word in a trie	37
4.2 Procedure $cleanup_T$	38
4.2.1 Selecting $N : N \subseteq T \wedge N \neq \emptyset \wedge Intern_PI(N)$	40
4.2.1.1 Selecting a single state	40
4.2.1.2 Selecting a path of states	42
4.2.2 Selecting $N : N \subseteq T \wedge N \neq \emptyset \wedge Mutual_PI(D, N)$	42
4.3 An example	45

4.4	Time and space performance	47
4.4.1	Improvements	47
4.5	Commentary	48
5	Arbitrary intermediate ADFA	49
5.1	Procedure add_word_N	49
5.2	Procedure $cleanup_N$	53
5.3	Time and space performance	53
5.3.1	Improvements	53
5.4	Commentary	53
6	Minimal intermediate ADFA	55
6.1	Procedure add_word_I	55
6.2	Procedure $cleanup_I$	61
6.3	An example	61
6.4	Time and space performance	66
6.4.1	Improvements	66
6.5	Commentary	67
7	Reversed trie intermediate ADFA	69
7.1	Procedure add_word_R	69
7.2	Procedure $cleanup_R$	70
7.3	An example	70
7.4	Time and space performance	71
7.4.1	Improvements	72
7.5	Commentary	72
8	Avoiding cloning while adding words	73
9	Words in lexicographic order	75
9.1	Procedure add_word_S	75
9.2	Procedure $cleanup_S$	77
9.3	An example	78
9.4	Time and space performance	80
9.4.1	Improvements	80
9.5	Commentary	80
10	Minimizing depth layers	81
10.1	Procedure add_word_D	81
10.2	Procedure $cleanup_D$	83
10.3	An example	83
10.4	Time and space performance	87

<i>CONTENTS</i>	5
10.4.1 Improvements	87
10.5 Commentary	87
11 Minimizing semi-incrementally	89
11.1 Procedure <i>add_word_W</i>	89
11.2 Procedure <i>cleanup_W</i>	90
11.3 An example	90
11.4 Time and space performance	93
11.4.1 Improvements	93
11.5 Commentary	93
Bibliography	94
Index	107
Colophon	110

Chapter 7

Reversed trie intermediate *ADFA*

In this chapter, we maintain M as a trie corresponding to the reverse of the words added so far. Formally, the invariant is

$$\text{Struct}_R(M) \equiv \text{Is_trie}(M) \wedge \mathcal{L}(M) = D^R$$

The resulting *ADFA* accepts W^R . Minimality of M is achieved by reversing M (usually yielding a nondeterministic automaton) and determinizing it. The reversal and determinization steps are combined in this chapter into cleanup_R . We will only present the procedures and examples — a full derivation of this algorithm can be found in [Wat01f, Wat02a] and most recently in [Wat02b], where an alternative derivation is given.

7.1 Procedure add_word_R

As our specification, we get:

```
proc  $\text{add\_word}_R(\text{inout } (Q, \delta, s, F) : \text{ADFA}; \text{in } w : \Sigma^*) \rightarrow$   
  { pre  $\text{Is\_trie}((Q, \delta, s, F)) \wedge L = \mathcal{L}((Q, \delta, s, F))$  }  
   $(Q, \delta, s, F) : S_{7.1}$   
  { post  $\text{Is\_trie}((Q, \delta, s, F)) \wedge \mathcal{L}((Q, \delta, s, F)) = L \cup \{w^R\}$  }  
corp
```

Given the specification of add_word_T (in Chapter 4), if we assume that argument w can be reversed as a primitive operation (it can be done in $\mathcal{O}(|w|)$ time and constant space), we implement add_word_R as

```
proc  $\text{add\_word}_R(\text{inout } (Q, \delta, s, F) : \text{ADFA}; \text{in } w : \Sigma^*) \rightarrow$   
  { pre  $\text{Is\_trie}((Q, \delta, s, F)) \wedge L = \mathcal{L}((Q, \delta, s, F))$  }  
   $\text{add\_word}_T((Q, \delta, s, F), w^R)$ 
```

```

    { post  $Is\_trie((Q, \delta, s, F)) \wedge \mathcal{L}((Q, \delta, s, F)) = L \cup \{w^R\}$  }
  corp

```

Naturally, it would also be easy to specialize add_word_T explicitly by expanding its body.

7.2 Procedure $cleanup_R$

We now require a minimization procedure with specification:

```

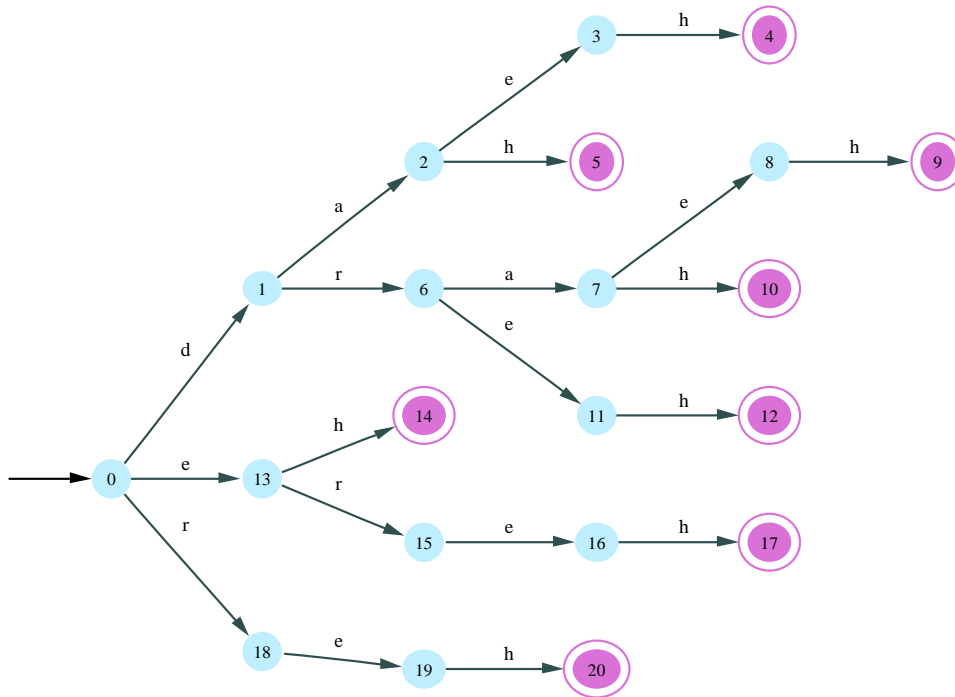
proc  $cleanup_R$ (inout  $M : ADFA$ )  $\rightarrow$ 
  { pre  $Is\_trie(M) \wedge L = \mathcal{L}(M)$  }
   $(Q, \delta, s, F) : S_{7.2}$ 
  { post  $\mathcal{L}(M) = L^R \wedge M \in MADFA$  }
  corp

```

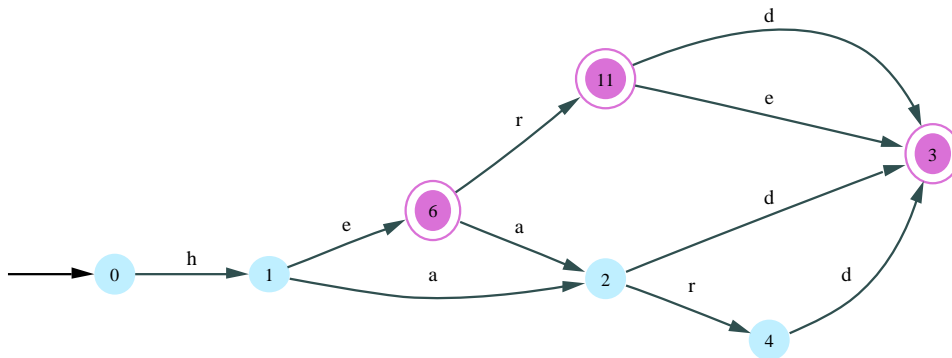
Without discussing the details (which are given extensively originally in [Brz62a] and again in [Wat95c, Wat00b, Wat02a]), $cleanup_R$ can be implemented by reversing M and determinizing the result using the automata determinization (also known as the ‘subset construction’) algorithm (see [HU79] for a detailed treatment of automata determinization).

7.3 An example

The reverse trie corresponding to he, her, had, head, hard, herd, here, heard is



The *MADFA* resulting from applying $cleanup_R$ to the above reverse trie is



This *ADFA* is minimal.

7.4 Time and space performance

As a simple variant of procedure add_word_T (in Chapter 4), procedure add_word_R has the same running time and space, also yielding a trie of

size $\mathcal{O}(\sum_{w \in W} |w|)$. I conjecture that procedure $cleanup_R$ takes time and space $\mathcal{O}(|M|)$. Assuming this conjecture, the construction and minimization takes $\mathcal{O}(\sum_{w \in W} |w|)$ space and time.

7.4.1 Improvements

The efficiency of this algorithm hinges on a good encoding of the *ADFA* which is reversible — usually by storing the reversed transitions in addition to the forward transitions — and an efficient implementation of the determinization algorithm. Aspects of efficient determinization implementations are discussed in [JW96].

7.5 Commentary

This algorithm is a specialization (to acyclic *DFAs*) of Brzozowski's *DFA* minimization algorithm [Brz62a, Brz62b]. More recently, it is described in [Wat00a, Wat02a]. Brzozowski's minimization algorithm has an interesting history, described in [Wat00b, Wat01a].

Chapter 10

Words by decreasing length: minimizing depth layers

In this chapter, we derive a simple semi-incremental algorithm which, like the one in Chapter 9, depends upon an ordering of the words to avoid the relatively expensive cloning operation. The words are added in *any* order of decreasing length. No confluence states are encountered while adding a word w , and states below depth $|w|$ are maintained pairwise inequivalent, while those at or above depth $|w|$ are not confluence states. As in Chapter 9, this will enable us to use add_word_D . Our first structural invariant is $Struct_D((Q, \delta, s, F), D) \equiv$

$$\begin{aligned} & Intern_PI(depth_level_{>\mathcal{L}_{minlen}}) \\ \wedge & Confl_free(depth_level_{\leq\mathcal{L}_{minlen}}) \\ \wedge & \mathcal{L} = D \end{aligned}$$

10.1 Procedure add_word_D

Our starting point is

```
proc  $add\_word_D$ (inout  $(Q, \delta, s, F) : ADFA$ ; in  $w : \Sigma^*$ )  $\rightarrow$ 
  { pre  $|w| \leq \mathcal{L}_{minlen}$ 
     $\wedge Intern\_PI(depth\_level_{>\mathcal{L}_{minlen}})$ 
     $\wedge Confl\_free(depth\_level_{\leq\mathcal{L}_{minlen}})$ 
     $\wedge L = \mathcal{L}$  }
   $(Q, \delta, s, F) : S_{10.1}$ 
  { post  $|w| = \mathcal{L}_{minlen}$ 
     $\wedge Intern\_PI(depth\_level_{>\mathcal{L}_{minlen}})$ 
     $\wedge Confl\_free(depth\_level_{\leq\mathcal{L}_{minlen}})$ 
```

$$\wedge \mathcal{L} = L \cup \{w\} \}$$

corp

Clearly, in $S_{10.1}$ we can use add_word_T .

```

proc  $add\_word_D$ (inout  $(Q, \delta, s, F) : ADFA$ ; in  $w : \Sigma^*$ )  $\rightarrow$ 
  { pre  $|w| \leq \mathcal{L}_{minlen}$ 
     $\wedge Intern\_PI(depth\_level_{>\mathcal{L}_{minlen}})$ 
     $\wedge Confl\_free(depth\_level_{\leq\mathcal{L}_{minlen}})$ 
     $\wedge L = \mathcal{L}$  }
  {  $Intern\_PI(depth\_level_{>\mathcal{L}_{minlen}})$ 
     $\wedge Confl\_free(depth\_level_{\leq\mathcal{L}_{minlen}})$ 
     $\wedge Confl\_free([s \xrightarrow{w}])$  }
   $add\_word_T((Q, \delta, s, F), w)$ ;
   $(Q, \delta, s, F) : S'_{10.1}$ 
  { post  $|w| = \mathcal{L}_{minlen}$ 
     $\wedge Intern\_PI(depth\_level_{>\mathcal{L}_{minlen}})$ 
     $\wedge Confl\_free(depth\_level_{\leq\mathcal{L}_{minlen}})$ 
     $\wedge \mathcal{L} = L \cup \{w\}$  }

```

corp

Given that the word-lengths are monotonically decreasing, while adding w we are assured that no states deeper than $|w|$ will be visited during future word-adding operations. We can thus minimize all states $p : \overleftarrow{\mathcal{L}}_{minlen}(p) > |w|$ after adding w . Statement $S'_{10.1}$ can be implemented naïvely, by examining all states, comparing their depth against $|w|$ — though such a solution is costly.

```

proc  $add\_word_D$ (inout  $(Q, \delta, s, F) : ADFA$ ; in  $w : \Sigma^*$ )  $\rightarrow$ 
  { pre  $|w| \leq \mathcal{L}_{minlen}$ 
     $\wedge Intern\_PI(depth\_level_{>\mathcal{L}_{minlen}})$ 
     $\wedge Confl\_free(depth\_level_{\leq\mathcal{L}_{minlen}})$ 
     $\wedge L = \mathcal{L}$  }
  [| var  $k : \mathbb{N}$ 
  | {  $Intern\_PI(depth\_level_{>\mathcal{L}_{minlen}})$ 
     $\wedge Confl\_free(depth\_level_{\leq\mathcal{L}_{minlen}})$ 
     $\wedge Confl\_free([s \xrightarrow{w}])$  }
   $k := \mathcal{L}_{minlen}$ ;
   $add\_word_T((Q, \delta, s, F), w)$ ;
  as  $|w| < k \rightarrow$ 
    [| var  $p, q : State$ 
    | for  $p : p \in depth\_level_k \rightarrow$ 

```

```

    as  $\langle \exists q : q \in \text{depth\_level}_{>k} : \text{eq}(p, q) \rangle \rightarrow$ 
      let  $q : q \in \text{depth\_level}_{>k} \wedge \text{eq}(p, q);$ 
      merge( $p, q$ )
    sa
  rof
]]
sa
]]
{ post  $|w| = \mathcal{L}_{\text{minlen}}$ 
   $\wedge \text{Intern\_PI}(\text{depth\_level}_{>\mathcal{L}_{\text{minlen}}})$ 
   $\wedge \text{Confl\_free}(\text{depth\_level}_{\leq \mathcal{L}_{\text{minlen}}})$ 
   $\wedge \mathcal{L} = L \cup \{w\}$  }
corp

```

10.2 Procedure *cleanup_D*

Once the last word has been added, a final minimization step deals with those states at depths less than $\mathcal{L}_{\text{minlen}}$ in:

```

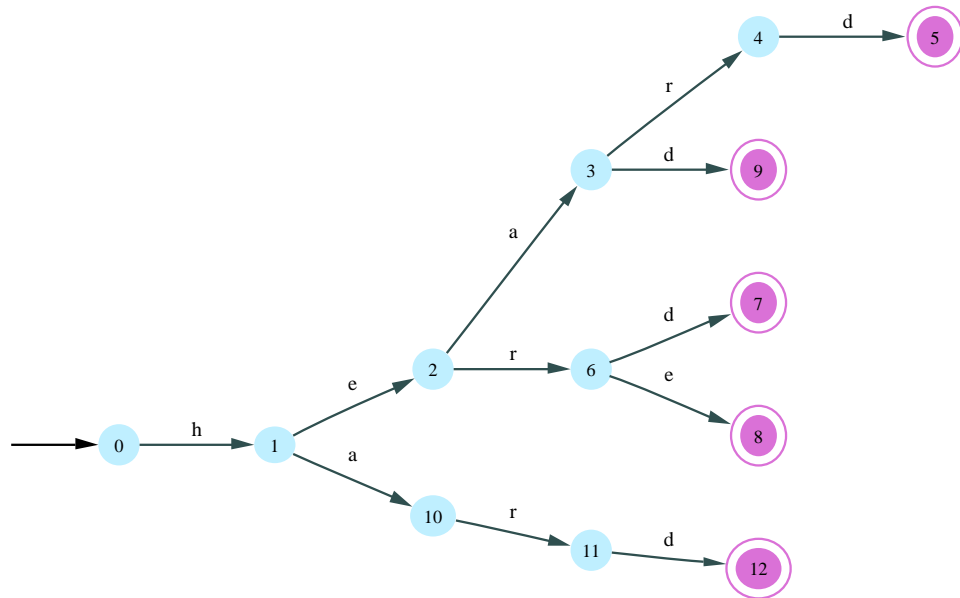
proc cleanupD(inout  $(Q, \delta, s, F) : \text{ADFA}$ )  $\rightarrow$ 
{ pre  $\text{Struct}_D((Q, \delta, s, F), L)$  }
[[ var  $p, q : \text{State}$ 
  for  $p : p \in \text{depth\_level}_{<\mathcal{L}_{\text{minlen}}} \rightarrow$ 
    as  $\langle \exists q : q \in Q : \text{eq}(p, q) \rangle \rightarrow$ 
      let  $q : q \in Q \wedge \text{eq}(p, q);$ 
      merge( $p, q$ )
    sa
  rof
]]
{ post  $\mathcal{L} = L \wedge (Q, \delta, s, F) \in \text{MADFA}$  }
corp

```

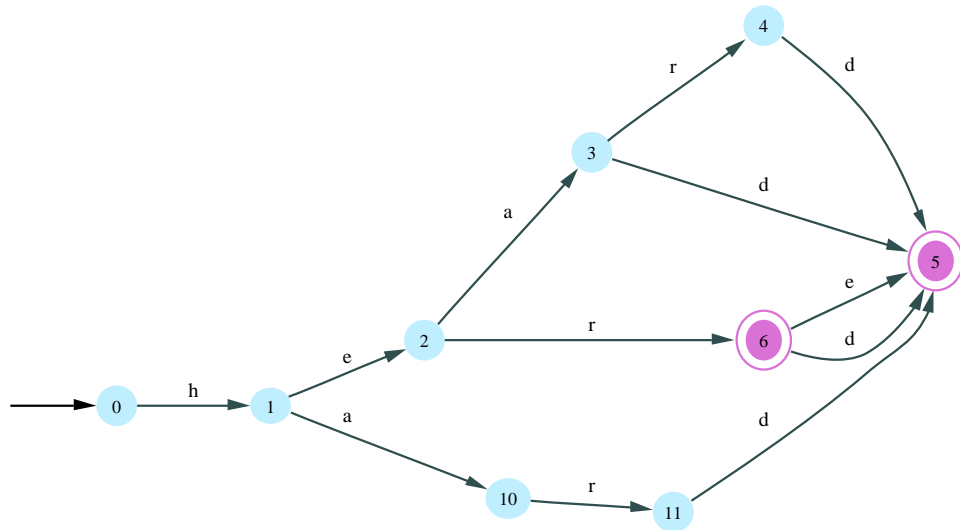
[ZZZ: Can I use *eq'* in this chapter?]

10.3 An example

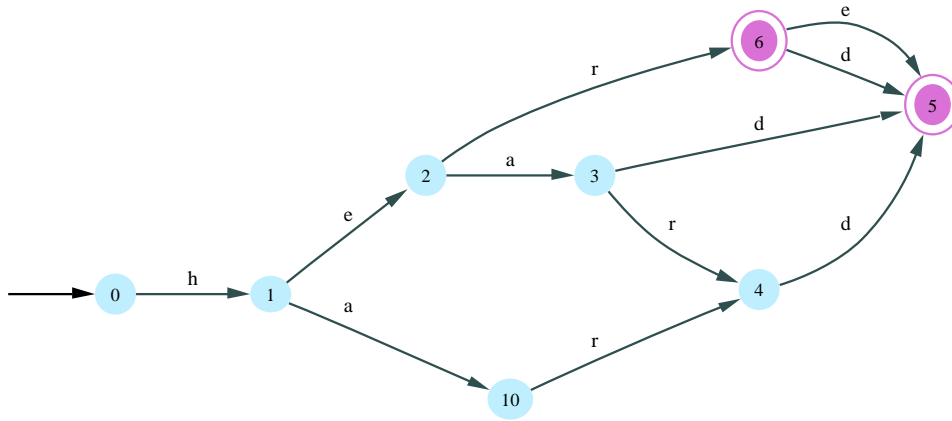
Consider adding the words (using *add_word_w*) in the order heard, herd, here, head, hard, her, had, he. After adding hard, we have



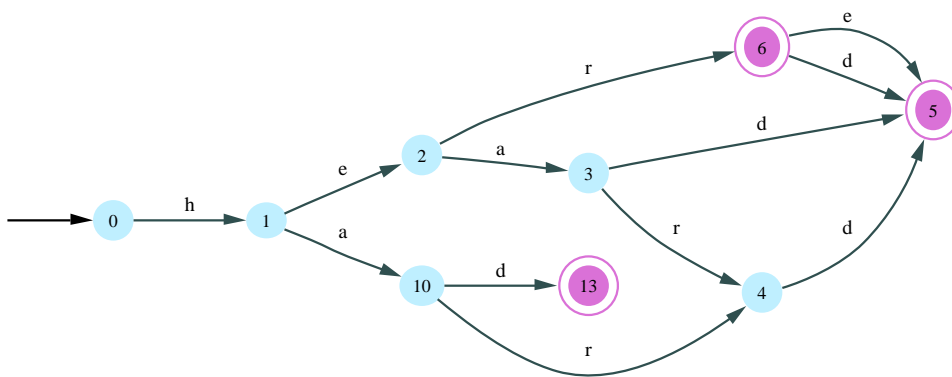
Once her is added, we can minimize states at depths $(3, 4]$ (depth 5 states, namely state 5 itself, is trivially minimized already in this example). This means that final states 5, 7, 8, 9, and 12 are combined, giving



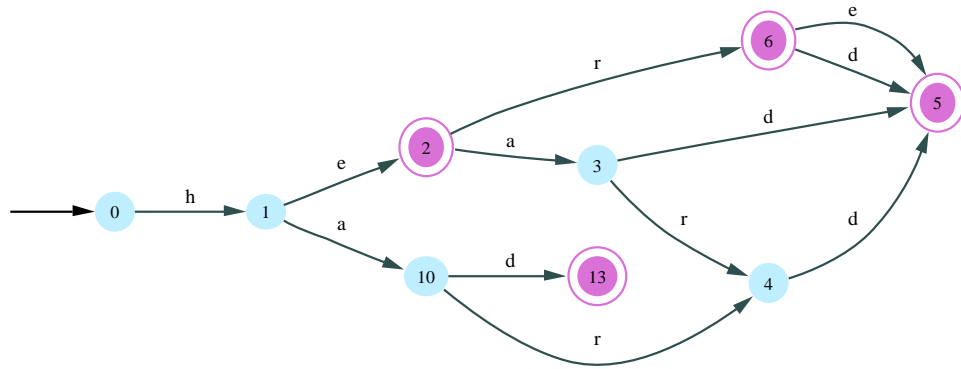
Subsequently, we see that states 4 and 11 can be merged, giving



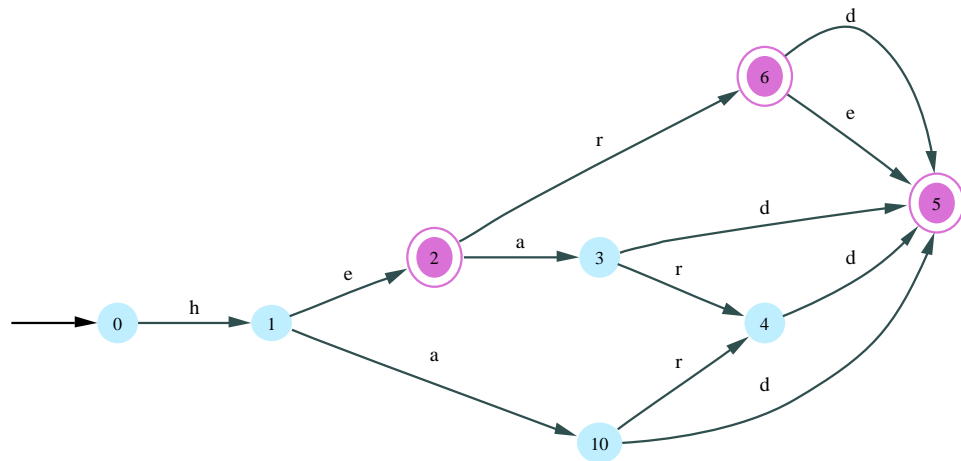
Adding had gives



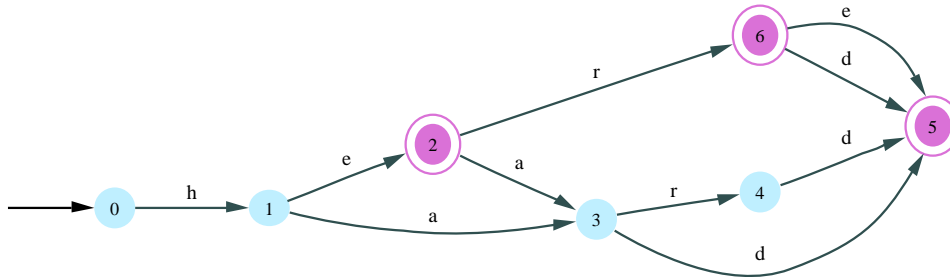
Finally, adding he gives



This means that we can minimize states at depths $(2, 3]$. Initially, we merge state 13 into state 5, giving



We can then merge states 10 and 3



Since this is already minimal, the final $cleanup_D$ step does not change the automaton.

10.4 Time and space performance

Word set W can be sorted into *some* order of decreasing length in time and space $\mathcal{O}(|W|)$. Procedure add_word_D requires time and space $|w|$ while adding w . Finally, $cleanup_D$ takes time and space $|M|$.

10.4.1 Improvements

This algorithm is not particularly efficient, with add_word_D containing a repetition over states at a particular depth, the body of which includes invocations of eq . I conjecture that there may well be a strengthening of the invariant which would allow the use of eq' , which is much more efficient. More straightforward improvements are immediately possible, including maintaining \mathcal{L}_{minlen} in a global variable instead of recomputing it. $cleanup_D$ can also be implemented to traverse the remaining states layer-wise.

10.5 Commentary

The algorithm presented here is a new one, not previously appearing in the literature.

Chapter 11

Words by decreasing length: minimizing semi-incrementally

In this chapter, we derive another algorithm which (as in Chapter 10) relies on the words being added in an order of decreasing length. To avoid the cloning operation (and therefore use add_word_T), we will maintain invariant $Struct_W((Q, \delta, s, F), D) \equiv$

$$Intern_PI(Reach^*(F)) \wedge Confl_free(Q - Reach^*(F)) \wedge \mathcal{L} = D$$

[ZZZ: Explain this.]

11.1 Procedure add_word_W

Our starting point is

```
proc  $add\_word_W$ (inout  $(Q, \delta, s, F) : ADFA$ ; in  $w : \Sigma^*$ )  $\rightarrow$ 
  { pre  $|w| \leq \overleftarrow{\mathcal{L}}_{minlen}$ 
     $\wedge Intern\_PI(Reach^*(F)) \wedge Confl\_free(Q - Reach^*(F))$ 
     $\wedge L = \mathcal{L}$ 
     $\wedge F' = F$  }
   $(Q, \delta, s, F) : S_{11.1}$ 
  { post  $Intern\_PI(Reach^*(F)) \wedge Confl\_free(Q - Reach^*(F))$ 
     $\wedge \mathcal{L} = L \cup \{w\}$  }
corp
```

After adding word w , we will have made state $\delta^*(s, w)$ final. Since we are adding words in order of decreasing length, and will never pass through state $\delta^*(s, w)$ again while adding another word, we can minimize states

$Reach^*(\delta^*(s, w))$, keeping in mind that states $Reach^*(F - \delta^*(s, w))$ will already be minimized according to $Struct_W$.

Our first algorithm is

```

proc  $add\_word_W(\mathbf{inout} (Q, \delta, s, F) : ADFA; \mathbf{in} w : \Sigma^*) \rightarrow$ 
  { pre  $|w| \leq \overleftarrow{\mathcal{L}}_{minlen}$ 
     $\wedge Intern\_PI(Reach^*(F)) \wedge Confl\_free(Q - Reach^*(F))$ 
     $\wedge L = \mathcal{L}$ 
     $\wedge F' = F$  }
   $add\_word_T((Q, \delta, s, F), w);$ 
  {  $F = F' \cup \{\delta^*(s, w)\}$  }
   $(Q, \delta, s, F) : S'_{11.1}$ 
  { post  $Intern\_PI(Reach^*(F)) \wedge Confl\_free(Q - Reach^*(F))$ 
     $\wedge \mathcal{L} = L \cup \{w\}$  }
corp

```

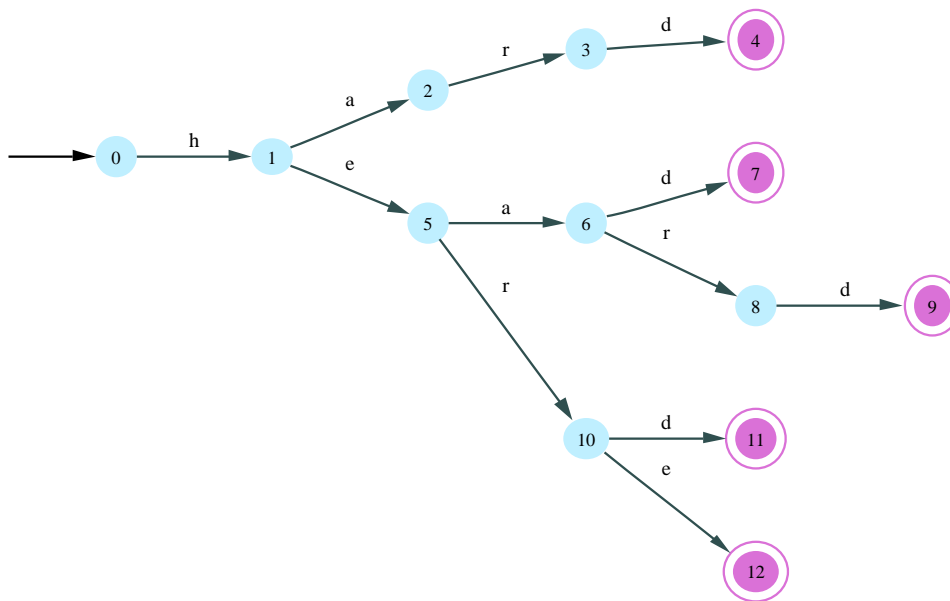
To implement $S'_{11.1}$, we rewrite the first postcondition conjunct:

$$\begin{aligned}
& Intern_PI(Reach^*(F)) \wedge Confl_free(Q - Reach^*(F)) \\
\equiv & \text{“ after } add_word_T \text{ invocation } F = F' \cup \{\delta^*(s, w)\} \text{”} \\
& Intern_PI(Reach^*(F' \cup \{\delta^*(s, w)\})) \\
& \wedge Confl_free(Q - Reach^*(F' \cup \{\delta^*(s, w)\})) \\
\equiv & \text{“ } Reach^*(X \cup Y) = Reach^*(X) \cup Reach^*(Y) \text{”} \\
& Intern_PI(Reach^*(F') \cup Reach^*(\{\delta^*(s, w)\})) \\
& \wedge Confl_free(Q - (Reach^*(F') \cup Reach^*(\{\delta^*(s, w)\})))
\end{aligned}$$

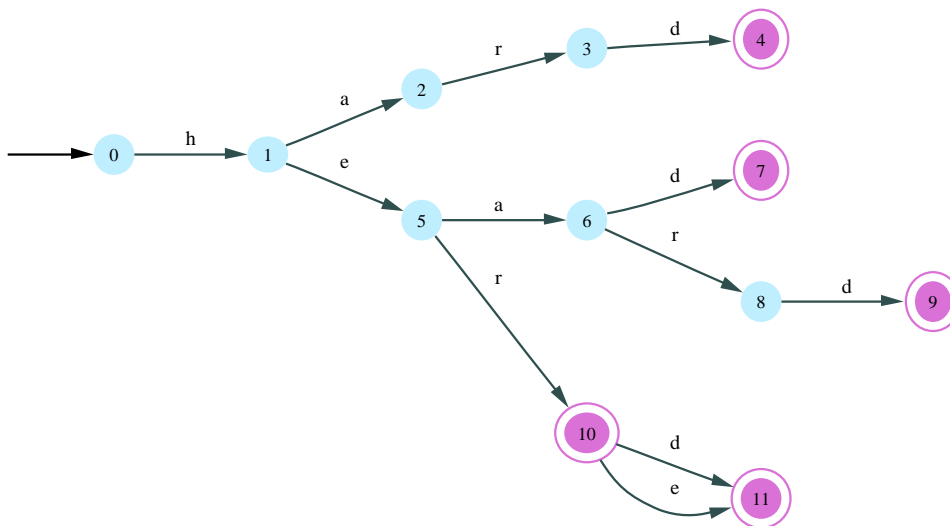
11.2 Procedure $cleanup_W$

11.3 An example

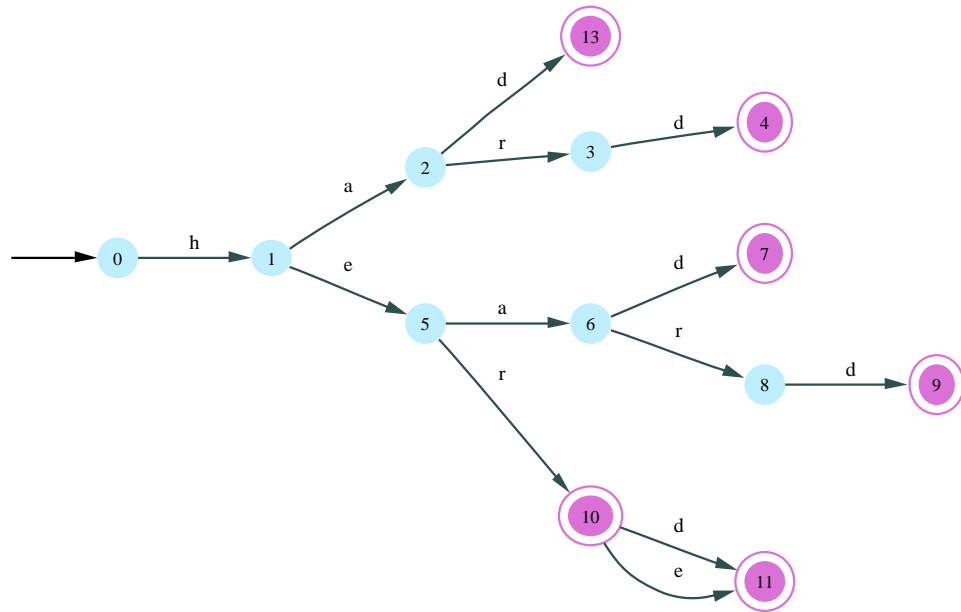
[ZZZ: Give some examples of Reach here.] Consider adding the words (using add_word_W) in the order heard, herd, here, head, hard, her, had, he. After adding hard, we have



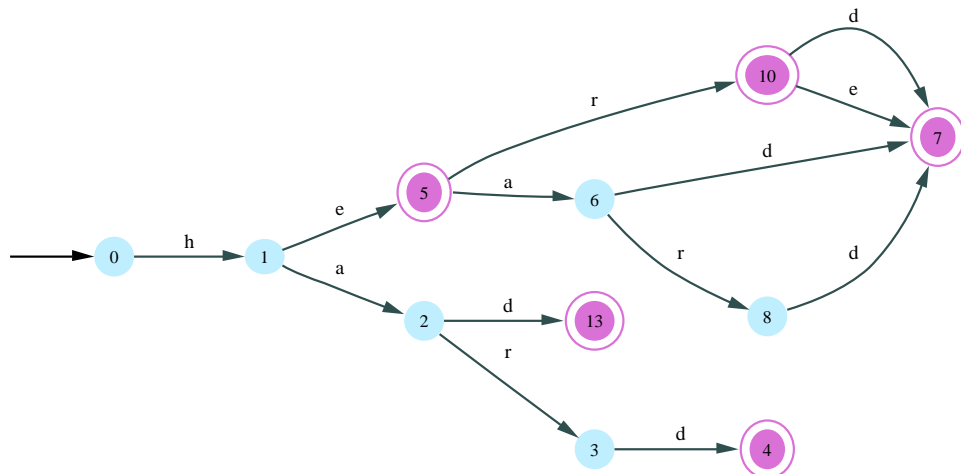
While adding her, we make state 10 final, and subsequently merge states 11 and 12 in



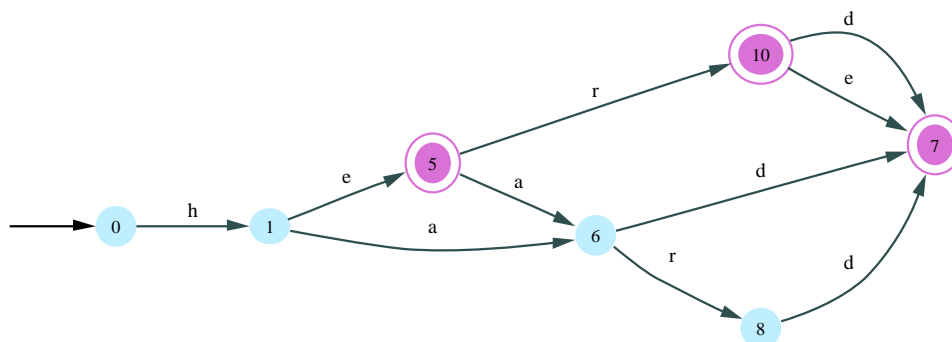
After adding had we get a new state and



Finally, after adding he , we make state 5 final, and merge states 7, 9, and 11



Continuing from this last *ADFA*, the $cleanup_W$ merges states 4 and 13 into 7, and then state 3 into 8 and finally state 2 into 6, giving the *MADFA*



11.4 Time and space performance

Word set W can be sorted into an order of decreasing length in time and space $\mathcal{O}(|W|)$. As is shown in [Wat98b, Wat03c], procedure add_word_W requires time and space $|w|$ while adding w , and $cleanup_W$ takes time and space $|M|$.

Word set W can be sorted into *any* order of decreasing length in time $\mathcal{O}(|W|)$. In [Wat03c], further performance improvements are discussed.

11.4.1 Improvements

A simple implementation of this algorithm has proven to be efficient in practice [Wat03c]. In that paper, several other optimizations are discussed.

11.5 Commentary

As mentioned in Chapter 1, this algorithm was actually accidentally derived. It is presented in detail in [Wat98b] and in [Wat03c].

Bibliography

- [ASU88] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1988.
- [Brz62a] 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.
- [Brz62b] Janusz A. Brzozowski. *Regular Expression Techniques for Sequential Circuits*. PhD thesis, Princeton University, Princeton, New Jersey, June 1962.
- [BSWK04] Gabor Barla-Szabo, Bruce W. Watson, and Derrick G. Kourie. A taxonomy of directed graph representations. *submitted to South African Computer Journal*, 2004.
- [BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.
- [CD99] Marcin Ciura and Sebastian Deorowicz. Experimental study of finite automata storing static lexicons. Technical report, Silesian Technical University, Poland, November 1999.
- [CF02] Rafael C. Carrasco and Mikel L. Forcada. Incremental construction and maintenance of minimal finite-state automata. *Computational Linguistics*, 28(2):207–216, june 2002.
- [CR94] Maxime A. Crochemore and Wojciech Rytter. *Text Algorithms*. Oxford University Press, 1994.
- [CR03] Maxime A. Crochemore and Wojciech Rytter. *Jewels of Stringology*. World Scientific Publishing Company, 2003.

- [CWZ03] Loek Cleophas, Bruce W. Watson, and Gerard Zwaan. On factor oracles. In Jan Holub, editor, *Proceedings of the Eighth Prague Stringologic Workshop*, Prague, Czech Republic, September 2003. Czech Technical University.
- [Dac98] 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.
- [dBW03] Noud de Beijer and Bruce W. Watson. Stretching and jamming automata. In Eloff [Elo03].
- [Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.
- [DMWW00] 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.
- [DWW98] 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.
- [Elo03] M.M. Eloff, editor. *Proceedings of the Postgraduate Symposium of the South African Institute for Computer Scientists and Information Technologists*, South Africa, September 2003.
- [Fre60] E. Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.
- [FWC03] Michiel Frishert, Bruce W. Watson, and Loek Cleophas. The effects of rewriting regular expressions on their accepting automata. In Ibarra and Zhu [IZ03].
- [GBA01] Jorge Graña, Fco. Mario Barcala, and Miguel A. Alonso. Compilation methods of minimal acyclic finite-state automata for large dictionaries. In Watson and Wood [WW01b], pages 116–129.

- [GBY91] Gaston H. Gonnet and Ricardo Baeza-Yates. *Handbook of Algorithms and Data Structures (In Pascal and C)*. Addison-Wesley, second edition, 1991.
- [Gri73] David Gries. Describing an algorithm by Hopcroft. *Acta Informatica*, 2:97–109, 1973.
- [Gri80] David Gries. *The Science of Computer Programming*. Springer-Verlag, second edition, 1980.
- [Gus97] Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
- [Hol97] Jan Holub, editor. *Proceedings of the Second Prague Stringologic Workshop, Prague, Czech Republic, July 1997*. Czech Technical University.
- [Hop71] John E. Hopcroft. *An $n \log n$ algorithm for minimizing the states in a finite automaton*, pages 189–196. Academic Press, 1971.
- [HP98] Gerard J. Holzmann and Anuj Puri. A minimized automaton representation of reachable states. *Software Tools for Technology Transfer*, 3 (1998)(1), 1998.
- [HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- [IZ03] Oscar Ibarra and Deng Zhu, editors. *Proceedings of the Eighth Conference on Implementations and Applications of Automata*, Santa Barbara, USA, July 2003. Springer-Verlag.
- [JM00] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, 2000.
- [JW96] J. Howard Johnson and Derick Wood. Instruction computation in subset construction. In Raymond et al. [RWY96], pages 64–71.
- [Kor99] András Kornai, editor. *Extended Finite State Models of Language*. Cambridge University Press, 1999.
- [KWK03a] Ernest Ketcha, Bruce W. Watson, and Derrick G. Kourie. Hardcoding finite automata processing. In Eloff [Elo03].

- [KWK03b] Ernest Ketcha, Bruce W. Watson, and Derrick G. Kourie. Hardcoding finite automata processing. In Ibarra and Zhu [IZ03].
- [Mih99a] Stoyan Mihov. Direct building of minimal automaton for given list. Technical report, Bulgarian Academy of Science, 1999.
- [Mih99b] Stoyan Mihov. *Direct Building of Minimal Automaton for Given List*. PhD thesis, Bulgarian Academy of Science, 1999.
- [PAMS94] K.-H. Park, Jun-Ichi Aoe, K. Morimoto, and M. Shishibori. An algorithm for dynamic processing of DAWGs. *International Journal of Computational Mathematics*, 54:155–173, 1994.
- [Pev00] Pavel Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. MIT Press, 2000.
- [Rev91] Dominique Revuz. *Dictionnaires et lexiques: méthodes et algorithmes*. PhD thesis, Institut Blaise Pascal, LITP 91.44, Paris, France, 1991.
- [Rev92] Dominique Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92:181–189, 1992.
- [Rev00] Dominique Revuz. Dynamic acyclic minimal automaton. In Yu and Păun [YP00], pages 226–232.
- [RWY96] Darrell Raymond, Derick Wood, and Sheng Yu, editors. *Proceedings of the First Workshop on Implementing Automata*, volume 1260 of *Lecture Notes in Computer Science*, London, Canada, August 1996. Springer-Verlag.
- [SFK95] 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.
- [TKW03] W.H. Morkel Theunissen, Derrick G. Kourie, and Bruce W. Watson. Standards and agile software development. In Eloff [Elo03].

- [vAW04] Markus van Aardt and Bruce W. Watson. Experimental software engineering. *submitted to South African Computer Journal*, 2004.
- [vEW01] Stephan van Eeden and Bruce W. Watson. The DIESEL project — Missing Link: Design of a linker. In Paula Kotzé, editor, *Proceedings of the Postgraduate Symposium of the South African Institute for Computer Scientists and Information Technologists*, pages 80–85, Pretoria, South Africa, September 2001. University of South Africa Press.
- [Wat90] Bruce W. Watson. The design of the C-Processor instruction set architecture. Technical report, Digital Engineering, Eindhoven University of Technology, the Netherlands, 1990.
- [Wat93a] Bruce W. Watson. A taxonomy of deterministic finite automata minimization algorithms. Technical Report 44, Faculty of Computing Science, Eindhoven University of Technology, the Netherlands, 1993.
- [Wat93b] Bruce W. Watson. A taxonomy of finite automata construction algorithms. Technical Report 43, Faculty of Computing Science, Eindhoven University of Technology, the Netherlands, 1993.
- [Wat94a] Bruce W. Watson. The design of the FIRE Engine: A C++ toolkit for FInite automata and Regular Expressions. Technical Report 22, Faculty of Computing Science, Eindhoven University of Technology, the Netherlands, 1994.
- [Wat94b] Bruce W. Watson. The Eindhoven Pattern Kit (C implementation). Available via <http://fastar.cs.up.ac.za>, 1994.
- [Wat94c] Bruce W. Watson. The FIRE Engine (C++ implementation). Available via <http://fastar.cs.up.ac.za>, 1994.
- [Wat94d] Bruce W. Watson. An introduction to the FIRE Engine: A C++ toolkit for FInite automata and Regular Expressions. Technical Report 21, Faculty of Computing Science, Eindhoven University of Technology, the Netherlands, 1994.
- [Wat94e] Bruce W. Watson. The performance of single-keyword and multiple-keyword pattern matching algorithms. Technical

- Report 19, Faculty of Computing Science, Eindhoven University of Technology, the Netherlands, 1994.
- [Wat95a] Bruce W. Watson. A Boyer-Moore (or Watson-Watson) type algorithm for regular tree pattern matching. In E.H.L. Aarts, H.M.M. ten Eikelder, C. Hemerik, and M. Rem, editors, *Simplex Sigillum Veri: Een Liber Amicorum voor prof.dr. F.E.J. Kruseman Aretz*, pages 315–320. Faculty of Computing Science, Eindhoven University of Technology, the Netherlands, 1995.
- [Wat95b] Bruce W. Watson. The SPARE Parts (C++ implementation). Available via <http://fastar.cs.up.ac.za>, 1995.
- [Wat95c] Bruce W. Watson. *Taxonomies and Toolkits of Regular Language Algorithms*. PhD thesis, Faculty of Computing Science, Eindhoven University of Technology, the Netherlands, September 1995.
- [Wat95d] Bruce W. Watson. Trends in compiler construction. In A. Lerie Steenkamp, editor, *Proceedings of the SAICSIT Symposium*, pages 3–12, Pretoria, South Africa, May 1995. University of South Africa Press.
- [Wat96a] Bruce W. Watson. A collection of new regular grammar pattern matching algorithms. In Jan Holub, editor, *Proceedings of the First Prague Stringologic Workshop*, pages 64–83, Prague, Czech Republic, August 1996. Czech Technical University.
- [Wat96b] Bruce W. Watson. The FIRE Lite: FAs and REs in C++. In Raymond et al. [RWY96], pages 167–188.
- [Wat96c] Bruce W. Watson. Implementing and using finite automata toolkits. In András Kornai, editor, *Proceedings of the Twelfth European Conference on Artificial Intelligence*, pages 97–100, Budapest, Hungary, August 1996.
- [Wat96d] Bruce W. Watson. Implementing and using finite automata toolkits. *Journal of Natural Language Engineering*, 2(4):295–302, December 1996.
- [Wat96e] Bruce W. Watson. A new regular grammar pattern matching algorithm. In J. Diaz and M. Serna, editors, *Proceedings of the European Symposium on Algorithms*, volume 1136 of *Lecture*

Notes in Computer Science, pages 364–377, Barcelona, Spain, September 1996. Springer-Verlag.

- [Wat96f] Bruce W. Watson. The performance of single and multiple keyword pattern matching algorithms. In Nivio Ziviani, Ricardo Baeza-Yates, and Katia Guimaraes, editors, *Proceedings of the Third South American Workshop on String Processing*, volume 4 of *International Informatics Series*, pages 280–294, Recife, Brazil, August 1996. Carleton University Press.
- [Wat97a] Bruce W. Watson. A Boyer-Moore (or Watson-Watson) type algorithm for regular tree pattern matching. In Holub [Hol97], pages 33–38.
- [Wat97b] Bruce W. Watson. A new family of string pattern matching algorithms. In Holub [Hol97], pages 12–23.
- [Wat97c] Bruce W. Watson. A new family of string pattern matching algorithms. Technical Report 97-1, IST Technologies Research Group, Ribbit Software Systems Inc., 1997.
- [Wat97d] Bruce W. Watson. Practical optimizations for automata. In Derick Wood and Sheng Yu, editors, *Proceedings of the Second Workshop on Implementing Automata*, volume 1436 of *Lecture Notes in Computer Science*, pages 232–240, London, Canada, September 1997. Springer-Verlag.
- [Wat97e] Bruce W. Watson. SPARE Parts: A C++ toolkit for String PAttern REcognition. In Holub [Hol97], pages 47–60.
- [Wat97f] Bruce W. Watson. The SPARE Parts: A C++ toolkit for String PAttern REcognition. Technical Report 97-2, IST Technologies Research Group, Ribbit Software Systems Inc., 1997.
- [Wat98a] Bruce W. Watson. An early-retirement plan for the states. In Jan Holub, editor, *Proceedings of the Third Prague Stringologic Workshop*, pages 119–124, Prague, Czech Republic, September 1998. Czech Technical University.
- [Wat98b] 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.
- [Wat98c] Bruce W. Watson. The SPARE Parts software. In Steven S. Skiena, editor, *The Algorithm Design Manual*. Springer-Verlag, 1998. Software on CD-ROM.
- [Wat99a] Bruce W. Watson. The FIRE Engine software. In Kornai [Kor99]. Software on CD-ROM.
- [Wat99b] Bruce W. Watson. Implementing and using finite automata toolkits. In Kornai [Kor99].
- [Wat99c] Bruce W. Watson. The OpenFIRE initiative. In Jun-Ichi Aoe, editor, *Proceedings of the International Conference on Computer Processing of Oriental Languages*, volume 2, pages 421–424, Tokushima, Japan, March 1999.
- [Wat99d] Bruce W. Watson. A taxonomy of algorithms for constructing minimal acyclic deterministic automata. In Helmut Jürgensen, editor, *Proceedings of the Fourth Workshop on Implementing Automata*, Potsdam, Germany, July 1999. Springer-Verlag.
- [Wat00a] Bruce W. Watson. Combining two algorithms by Brzozowski. In Yu and Păun [YP00], pages 242–249.
- [Wat00b] Bruce W. Watson. A history of Brzozowski’s DFA minimization algorithm. In Yu and Păun [YP00].
- [Wat00c] Bruce W. Watson. A new approach to teaching compiler and interpreter construction. In Stan Shear, editor, *Proceedings of the South African Computer Lecturers Association Annual Conference*, Cape Town, South Africa, June 2000.
- [Wat00d] Bruce W. Watson. A new family of Commentz-Walter-style multiple-keyword pattern matching algorithms. In Borivoj Melichar, editor, *Proceedings of the Fifth Prague Stringologic Workshop*, Bratislava, Slovakia, September 2000. Czech Technical University.
- [Wat01a] Bruce W. Watson. A history of Brzozowski’s DFA minimization algorithm. Technical report, Department of Computer Science, University of Pretoria, South Africa, 2001.

- [Wat01b] Bruce W. Watson. An incremental DFA minimization algorithm. Technical report, Department of Computer Science, University of Pretoria, South Africa, 2001.
- [Wat01c] Bruce W. Watson. An incremental DFA minimization algorithm. In Lauri Karttunen, Kimmo Koskenniemi, and Gertjan van Noord, editors, *Proceedings of the Second International Workshop on Finite State Methods in Natural Language Processing*, Helsinki, Finland, August 2001.
- [Wat01d] Bruce W. Watson. A new family of Commentz-Walter-style multiple-keyword pattern matching algorithms. Technical report, Department of Computer Science, University of Pretoria, South Africa, 2001.
- [Wat01e] 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.
- [Wat01f] Bruce W. Watson. A taxonomy of algorithms for constructing minimal acyclic deterministic finite automata. *South African Computer Journal*, (27):12–17, 2001.
- [Wat02a] Bruce W. Watson. Combining two algorithms by Brzozowski. *South African Computer Journal*, 29:17–23, December 2002.
- [Wat02b] Bruce W. Watson. A fast and simple algorithm for constructing minimal acyclic deterministic finite automata. *Journal of Universal Computer Science*, 8(2), 2002.
- [Wat03a] Bruce W. Watson. Compiling DSLs for the .NET platform. May 2003.
- [Wat03b] Bruce W. Watson. FIRE Station: A unifying environment for finite state objects. In Marc Joliat, editor, *IFIP Working Group 2.4*, Santa Cruz, USA, August 2003.
- [Wat03c] Bruce W. Watson. A new algorithm for the construction of minimal acyclic DFAs. *Science of Computer Programming*, 48:81–97, 2003.

- [Wat03d] Bruce W. Watson. A new family and structure for Commentz-Walter-style multiple-keyword pattern matching algorithms. *South African Computer Journal*, 30:29–33, June 2003.
- [Wat03e] Bruce W. Watson. A new family of string pattern matching algorithms. *South African Computer Journal*, 30:34–41, June 2003.
- [Wat03f] Bruce W. Watson. A new recursive incremental algorithm for building minimal acyclic deterministic finite automata. In Carlos Martin-Vide and Victor Mitrana, editors, *Grammars and Automata for String Processing: From Mathematics and Computer Science to Biology, and Back*, pages 189–200. Taylor and Francis, 2003.
- [Wat03g] Bruce W. Watson. A new regular grammar pattern matching algorithm. *Theoretical Computer Science*, 299(1–3):509–521, 2003.
- [Wat03h] Bruce W. Watson. Reducing memory requirements during finite automata construction. *to appear in Software — Practice & Experience*, 2003.
- [Wat04a] Bruce W. Watson. *Algorithms for Constructing Minimal Acyclic Deterministic Finite Automata*. PhD thesis, Department of Computer Science, University of Pretoria, South Africa, 2004.
- [Wat04b] Bruce W. Watson. *Finite automata algorithms*. World Scientific Press, Singapore, 2004.
- [WC03] Bruce W. Watson and Loek Cleophas. SPARE Parts: A C++ toolkit for String Pattern REcognition. *to appear in Software — Practice & Experience*, 2003.
- [WD03] Bruce W. Watson and Jan Daciuk. An efficient incremental DFA minimization algorithm. *Journal of Natural Language Engineering*, 9(1):49–64, 2003.
- [WW94] Bruce W. Watson and Richard E. Watson. A Boyer-Moore type algorithm for regular expression pattern matching. Technical Report 31, Faculty of Computing Science, Eindhoven University of Technology, the Netherlands, 1994.

- [WW01a] Bruce W. Watson and Derick Wood, editors. *Preproceedings of the Sixth Conference on Implementations and Applications of Automata*, Pretoria, South Africa, July 2001. University of Pretoria Press.
- [WW01b] Bruce W. Watson and Derick Wood, editors. *Proceedings of the Sixth Conference on Implementations and Applications of Automata*, volume 2494, Pretoria, South Africa, July 2001. Springer-Verlag.
- [WW03] Bruce W. Watson and Richard E. Watson. A Boyer-Moore-style algorithm for regular expression pattern matching. *Science of Computer Programming*, 48:99–117, 2003.
- [WW04] Bruce W. Watson and Derick Wood, editors. *Special Issue: Implementations and Applications of Automata*, volume 313. *Journal of Theoretical Computer Science*, 2004.
- [WWS91a] Bruce W. Watson, Willem-Jan Withagen, and Mario P.J. Stevens. Compilation techniques for a high-level language processor. In A. Nuñez, editor, *Proceedings of the EuroMicro 91 Conference*, pages 29–36, Vienna, Austria, September 1991.
- [WWS91b] Bruce W. Watson, Willem-Jan Withagen, and Mario P.J. Stevens. Compilation techniques for a high-level language processor. *Journal of Microprocessing and Microprogramming*, 32(1–5):29–36, 1991.
- [WZ92] Bruce W. Watson and Gerard Zwaan. A taxonomy of keyword pattern matching algorithms. Technical Report 27, Faculty of Computing Science, Eindhoven University of Technology, the Netherlands, 1992.
- [WZ93] Bruce W. Watson and Gerard Zwaan. A taxonomy of keyword pattern matching algorithms. In Harry Wijshoff, editor, *Proceedings of the Symposium on Computing Science in the Netherlands*, pages 25–39, Utrecht, The Netherlands, November 1993.
- [WZ95] Bruce W. Watson and Gerard Zwaan. A taxonomy of sub-linear keyword pattern matching algorithms. Technical Report 13, Faculty of Computing Science, Eindhoven University of Technology, the Netherlands, 1995.

- [WZ96] Bruce W. Watson and Gerard Zwaan. A taxonomy of sublinear multiple keyword pattern matching algorithms. *Science of Computer Programming*, 27(2):85–118, 1996.
- [YP00] Sheng Yu and Andre Păun, editors. *Proceedings of the Fifth Conference on Implementations and Applications of Automata*, volume 2088, London, Canada, July 2000. Springer-Verlag.
- [Zwa01] Gerard Zwaan. Personal communication. 2001.

Index

- w -path, 82
- alphabet, 69, 81, 82, 89, 90
- 'big-oh', 69, 72, 87, 93
- confluence-free, 81–83, 89, 90
- depth level, 81–83
- equivalent state function, 83, 87
- finite automata
 - deterministic, 72
 - acyclic, 69–72, 81–83, 89, 90, 92
 - minimal acyclic, 70, 71, 83, 92
- function, add word, 69–71, 81–83, 87, 89, 90, 93
- function, cleanup, 69–72, 83, 87, 90, 92, 93
- function, state merging, 83
- internally pairwise inequivalent set, 81–83, 89, 90
- language, of an automaton, 69, 70, 81–83, 87, 89, 90
- left language, 82, 89, 90
- naturals, 82
- quantification
 - existential, 83
- reachability of states, 89, 90
- set cardinality, 69, 72, 81–83, 87, 89, 90, 93
- shortest left word length function, 82, 89, 90
- shortest word length, 81–83, 87
- state universe, 82, 83
- structural invariant, depth-based, 81, 83
- structural invariant, double-reversal, 69
- structural invariant, Watson, 89, 90
- transition function, 69, 70, 81–83, 89, 90
- trie, 69, 70