Experiments on minimization of finite automata

Jean Berstel Dominique Perrin

March 2, 2005

Contents

1 Introduction

We have described in [3] a number of minimization algorithms for finite automata. We discuss here their implementation and report on the experimentation realized on automata resulting from dictionnaries in french, which represent lists of large size, allowing to compare the efficiency of the different methods and implementations.

2 A myriad of methods

Surprisingly, there is plenty of methods which can be used to minimize a finite automaton. A number of them operate by successive refinements of a partition of the set of states. This is the case of Moore's and of Hopcroft's algorithm. A kind of opposite method, working only for a restricted class of automata works by fusion of equivalent states. Finally, Bzrozowski's method operates by reversal and determinization repeated twice.

2.1 Moore's method

2.2 Hopcroft's algorithm

2.3 Bzrozowski's algorithm

Let $\mathfrak{A} = (Q, i, T)$ be a deterministic automaton recognizing a set X. Let $\mathfrak{B} = (Q, T, \{i\})$ be the nondeterministic automaton obtained by reversing the edges of $\mathfrak{A} = (Q, i, T)$. Let $\mathfrak{C}$ be the automaton obtained from $\mathfrak{B}$ by determinization. Then $\mathfrak{C}$ is the minimal automaton recognizing the reversal of X. This observation can be used

- to minimize an automaton by repeating twice the transformation.
- to compute the minimal automaton of a set X by starting from a deterministic automaton recognizing the reversal $\tilde{X}$ of X.

The complexity is in principle very large since the determinization is exponential. However, the second variant does not produce an exponential blow-up if the automaton obtained for $\tilde{X}$ is of reasonable size.

2.4 Fusion

Two states p, q of a deterministic automaton $\mathfrak{A} = (Q, i, T)$ are mergeable if for all $a \in A$, $p.a = q.a$. The merge of two mergeable states consists in replacing them by a single state. The fusion algorithm consists in repeatedly identify two mergeable states and merge them. It is not true that any finite automaton can be minimized by this method since, for example, the automaton represented below is not minimal although its tawo states are nor mergeable. However, a

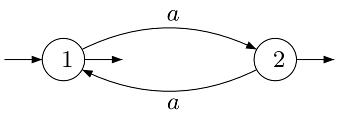

Figure 1: A non minimal automaton whose states cannot be merged.

large class of automata can be minimized by fusion. A deterministic automaton is called *local* if for any word $x \in A^*$ there is at most one cycle labeled x. For example, an acyclic automaton is local. Actually, it is proved in [1] (see p.56) that any local automaton can be minimized by fusion.

3 Results

The Java class `MinAutomaton` implements a command computing the minimal automaton of a set of words given in a text file using one of several possible minimization algorithms. The minimization algorithms are contained in the classes implementing the `Minimizer` interface. Usage: `java MinAutomaton method file (option)` The method can be

- B (Brzozowski double reverse algorithm).
- F (the fusion algorithm).
- N (naive method using `NMinimizer`). This is a naive version of Moore's algorithm using a succession of refinements without lexicographic sort. Each refinement step runs in $O(kn^2)$. The result is a $O(kn^3)$ minimization algorithm. It is of course very slow but the space used is reduced to the minimum $O(e)$ where e is the number of edges.
- Nbis (a variant of the previous using `NbisMinimizer`). A slight improvement on the previous one. The difference is that each refinement step runs in $O(knm)$ where m is the current number of classes. The resulting minimization method can be significantly faster.
- H (Hopcroft's algorithm using `HopcroftMinimizer`). This algorithm uses a succession of refinements operated by a sequence of blocks (the *anvils*) kept in a queue. The result, if correctly implemented, is a $O(kn \log n)$ algorithm (see [2] for an interesting discussion of this point). Its drawbacks are a substantial space required (to store a sophisticated data structure) and the dependence on k, especially because it operates on complete automata.
- R (Revuz algorithm using `RMinimizer` to be used only with acyclic automata represented as `IDFA`).

The option can be `v` (verbose mode).

The files used are all parts of the `delaf` dictionnary. It is a dictionnary of about 800000 words representing all words in french at all grammatical forms. A typical part of the `delaf` looks like

```
activait
activant
activant
activants
activas
activasse
activassent
activasses
activassiez
activassions
activateur
activateur
activateur biologique
activateur tissulaire du plasminogne
```

Compound words (like 'activateur biologique') are also entries of the dictionnary. Some words are repeated because the file is a stripped version of one associating with each word some information giving its grammatical category.

The `delafA` is the set of entries beginning with 'A' and the `delafa` of those beginning with 'a'.

The following table gives the result of experiments on these files using several minimization methods. The first column gives the name of the file. The second one gives the number of words in the file. The next one gives the number of nodes of a trie representing the words. The trie is build using the class `VariableArrayTrie` which represents the nodes as arrays of variable size of pointers on their sons. The column IDFA gives the number of states of an IDFA equivalent to the trie. Its number of nodes is less because all leaves are merged into one state (the difference between columns 3 and 4 is thus equal to the number of distinct words in the dictionary which are not a prefix of another one).

	nbWords	alphabet	Trie	IDFA	Minimal
delafAb.txt	39	29	97	73	64
delafA.txt	731	53	2526	2085	1202
delafB.txt	1556	58	5182	4254	2243
delafC.txt	2550	63	9310	7742	4086
delafa.txt	60324	72	175546	136886	36499
delaf.txt	802009	90	2203261	1683979	273716

Figure 2: Parts of the delaf french dictionnary.

The table below gives the computation time for each of the files and the minimization methods.

	Fusion	Naive	NaiveBis	Brzozo	Hopcroft	Revuz
delafAb.txt	33	30	29	126	58	30
delafA.txt	7188	398	281	202	297	116
delafB.txt	31733	2012	1213	341	693	159
delafC.txt		7427	4135	507	1199	210
delafa.txt		428487	6759	6922		2240
delaf.txt						78550

Figure 3: Computation time in ms. on a Dell portable

References

[1] Marie-Pierre Béal. *Codage Symboloique.* Masson, 1993.

	Naive	NaiveBis	Brzozo	Hopcroft	Revuz
delafA.txt	288	211			
delafa.txt		325681	344149		2179
delaf.txt					19135

Figure 4: Computation time in ms. on Monge

[2] Timo Knuutila. Re-describing an algorithm by Hopcroft. *Theoret. Comput. Sci.*, 250(1-2):333–363, 2001.

[3] M. Lothaire. *Applied Combinatorics on Words*. Cambridge university Press, 2005.