

Minimal Acyclic Deterministic Finite Automata (MADFAs)

Bruce W. Watson
watson@win.tue.nl

13 December 2001

Contents

1. TABASCO.
2. Preliminaries.
3. An algorithm skeleton.
4. Three algorithms.
5. Implementation.
6. Closing comments.

TABASCO

TAXonomy BAsed Software CONstruction.

Primary research activity of SoC.

Four phases:

1. Taxonomy: used for navigating, understanding, teaching, and comparing algorithms.
2. Toolkit: used in research software and also in industry.
3. Domain-specific language + processor: enables non-expert users.
4. Benchmark: facilitates choosing an algorithm from the toolkit.

Application areas done:

- Garbage collection.
- Attribute evaluation.
- Single and multiple-keyword pattern matching.
- Automata construction.
- Deterministic automata minimization.

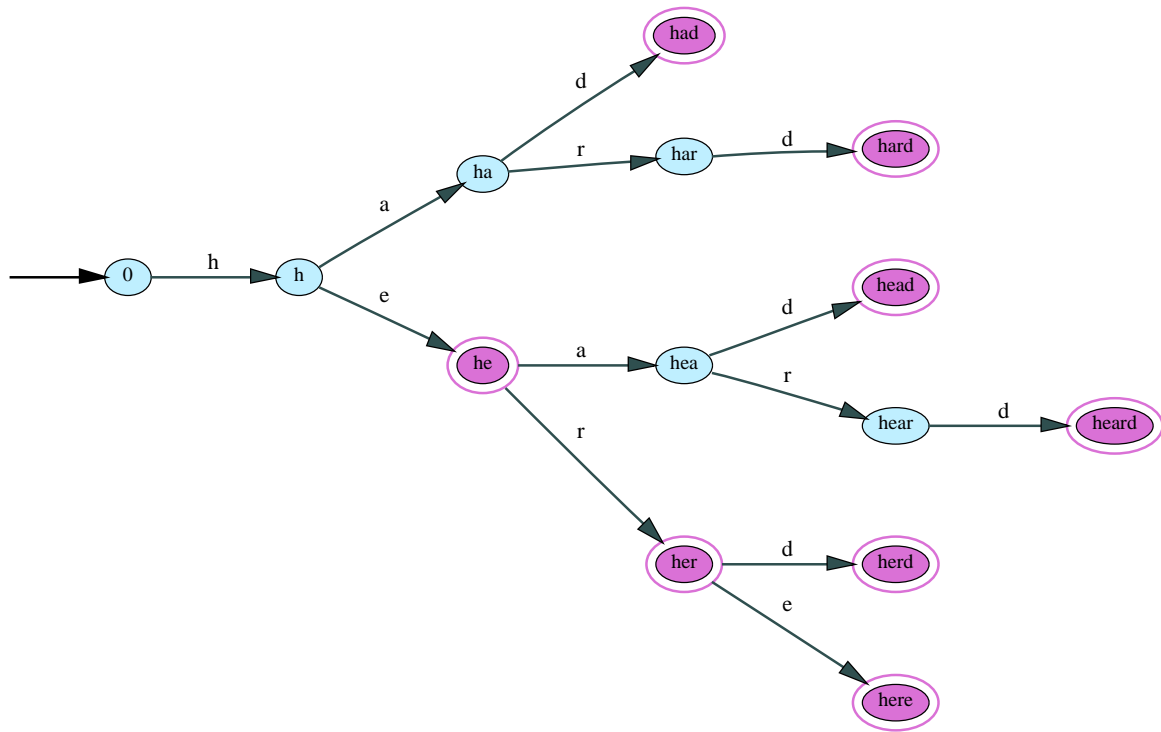
Upcoming application areas:

- MADFAs.
- Matrix representations.

- Graph representations.
- Compression algorithms.
- Tree parsing and pattern matching.
- String parsing.
- Approximate pattern matching.

Introduction

What does an acyclic deterministic finite automaton look like?



Represents a finite language (a finite set of words/strings).

Among the most compact known representations, eps. when minimal.

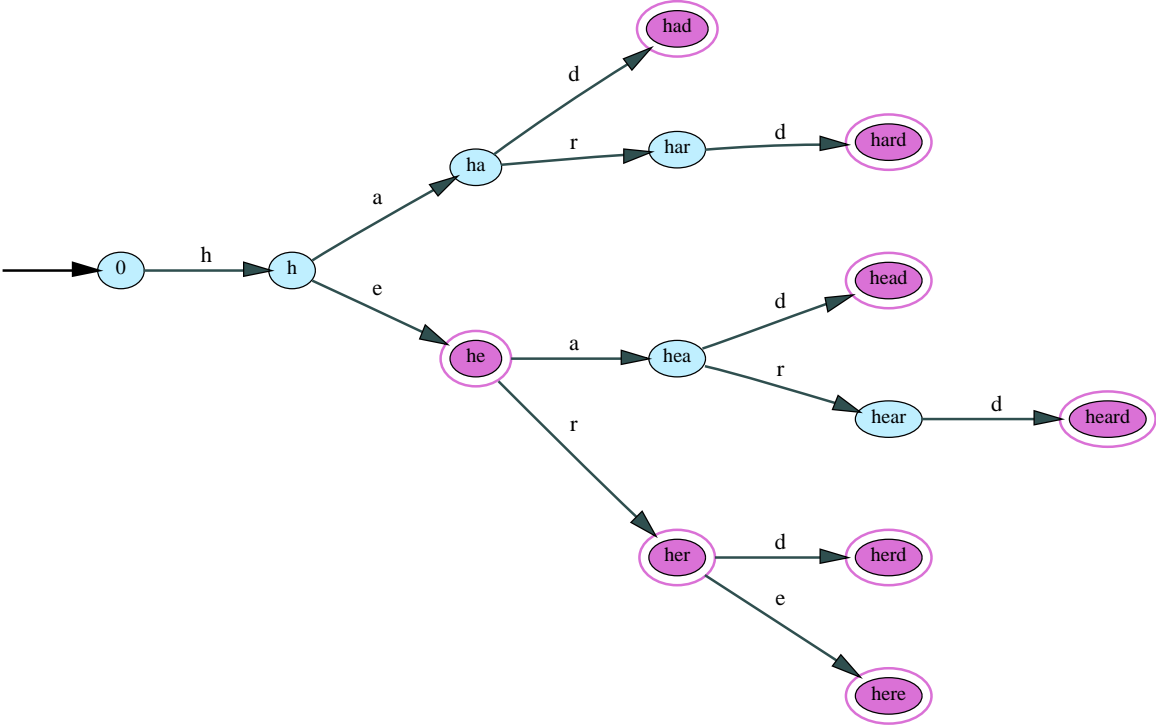
Very efficient test for language membership.

States usually represented as integers.

Application areas:

- Spell-checking.
- Checking for keywords in a compiler.
- Network intrusion detection.
- T9 text entry (SMS phones, Palm Pilot, ...).

Running example



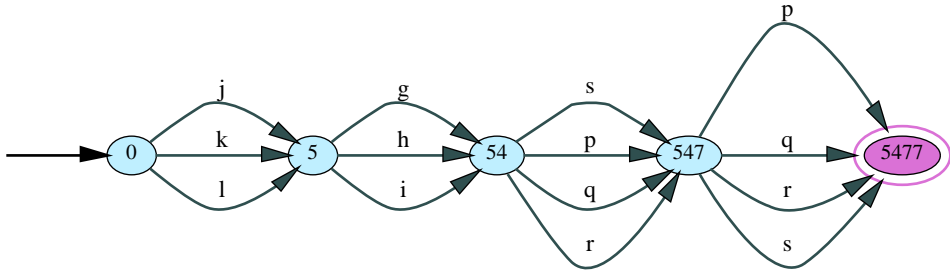
Example: The T9 text entry system

User enters using number keypad.

Multiple letters appear on each key.

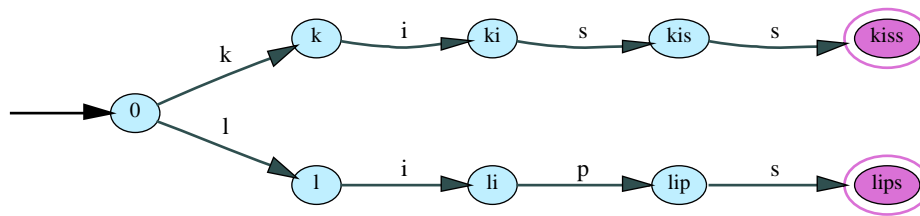
E.g. 5477 represents strings

$$Entry = \{jkl\}\{ghi\}\{pqrs\}\{pqrs\}$$



T9 computes

$$Entry \cap English = \{kiss, lips\}$$



Observations:

- In fact, backtracking is used.
- All states in the dictionary are *final* (allows prefixes of valid words).

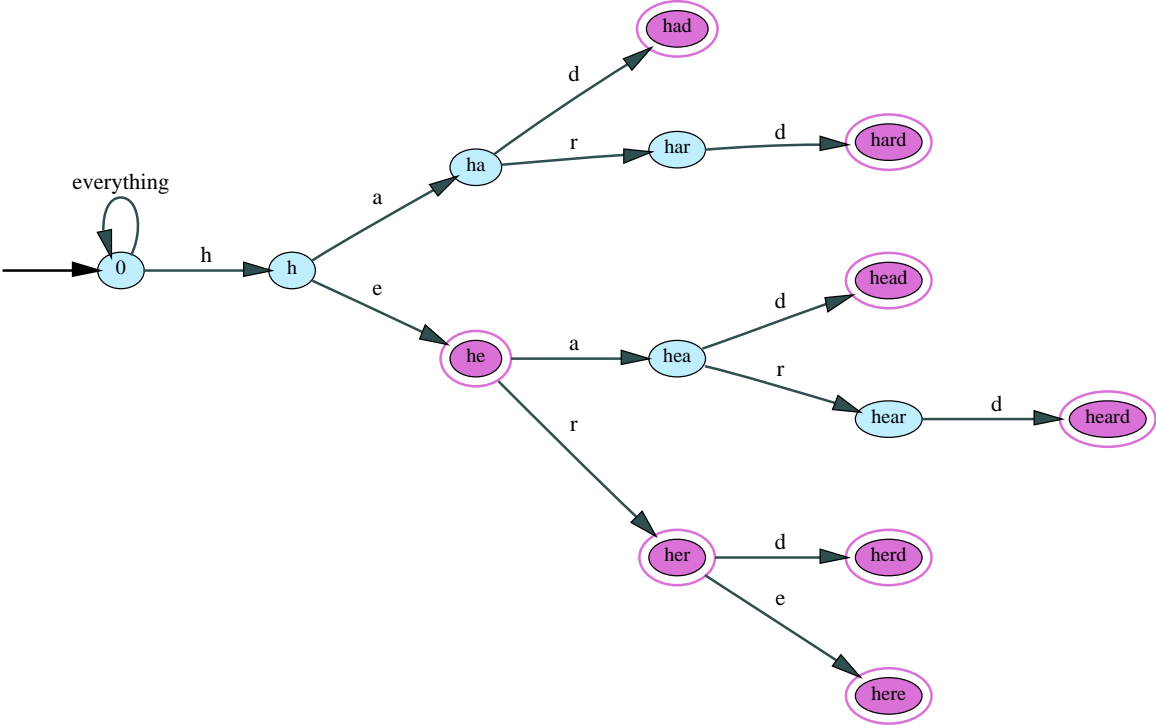
Automata with outputs

Instead of 'final' states, include an 'output'.
(Implements a mapping from a finite domain
(the strings).)

Application areas:

- Search engines. Outputs are the 'hits'.
- Grammar checking. Outputs are 'parts of speech'.
- Modern spell-checking: Outputs are 'valid suffixes'.

Related application: pattern matching



Preliminaries

A deterministic finite automaton (DFA) is a five-tuple $(Q, \Sigma, \delta, s, F)$ where

- Q is a finite set of states.
- Σ is an alphabet.
- $\delta \in Q \times \Sigma \rightarrow Q$ is a (perhaps partial) transition function.
- s is the start state.
- $F \subseteq Q$ is a set of final states.

(We assume there are no unreachable states.)

An ADFA is an acyclic DFA.

The *right language* of a state p is the set of all words on paths from p to a final state.

Minimality

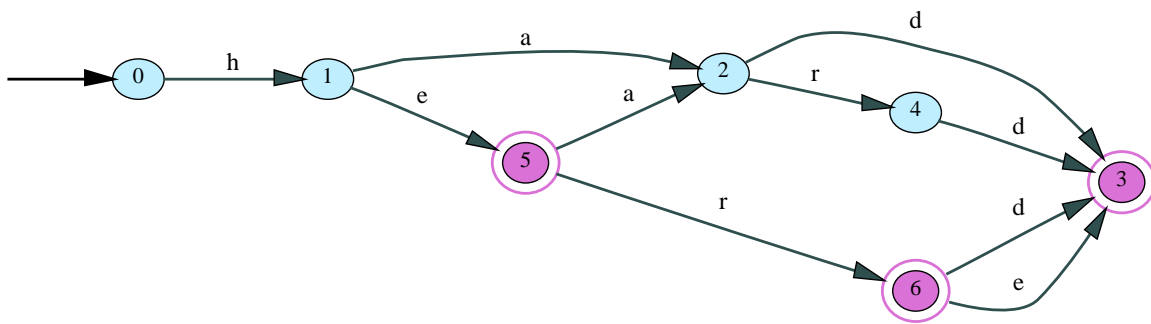
- Minimality. (A minimal ADFA is a MADFA.)
- Right language of a state: the right language, $RL(p)$, of state p is the set of words on p -final-state paths.
- Right languages and minimality: a DFA is minimal iff its states have pairwise unequal right languages.

- Equivalence of states:

$$E(p, q) \equiv (RL(p) = RL(q))$$

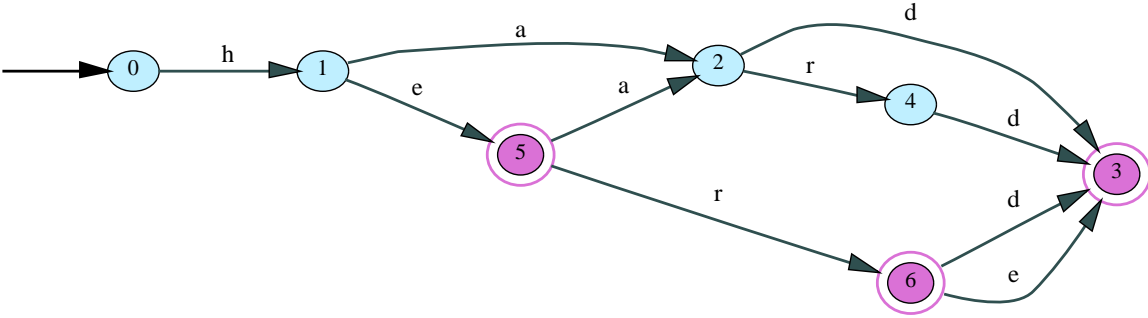
- Recursive equivalence of states $E(p, q) \equiv (p \in F \equiv q \in F) \wedge (\forall a \in \Sigma : E(\delta(p, a), \delta(q, a)))$

E.g. consider our first (tree-shaped) ADFA.
Minimal equivalent is:



Observation: compression techniques and clever representations are also important, but separate issues.

Running example



An algorithm skeleton

Most of the existing algorithms can be commonly presented.

Criteria:

- Adding words one-at-a-time.
- Maintaining a structural invariant.
- May require an ordering on the words.
- After all words added, final step may be necessary.

$D, T \doteq \emptyset$, word set;

do $T \neq \emptyset \rightarrow$

let $w : w$ is next in T under the ordering;

$aw(w)$;

$D, T \doteq D \cup \{w\}, T - \{w\}$

od;

complete

Trie algorithm

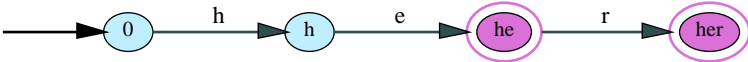
What is the invariant?

Why maintain a trie?

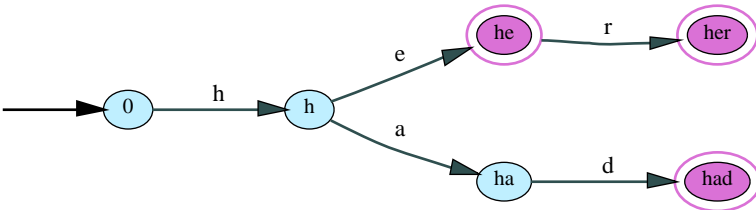
Adding a word which is a prefix:



Add *he*: follow the path and make the last state final.



Adding *had* (requires new states):



Minimization: isomorphic subtrees.

E as a recursive procedure.

- Bottom-up solutions.
- Hashing.
- Ordering the words.

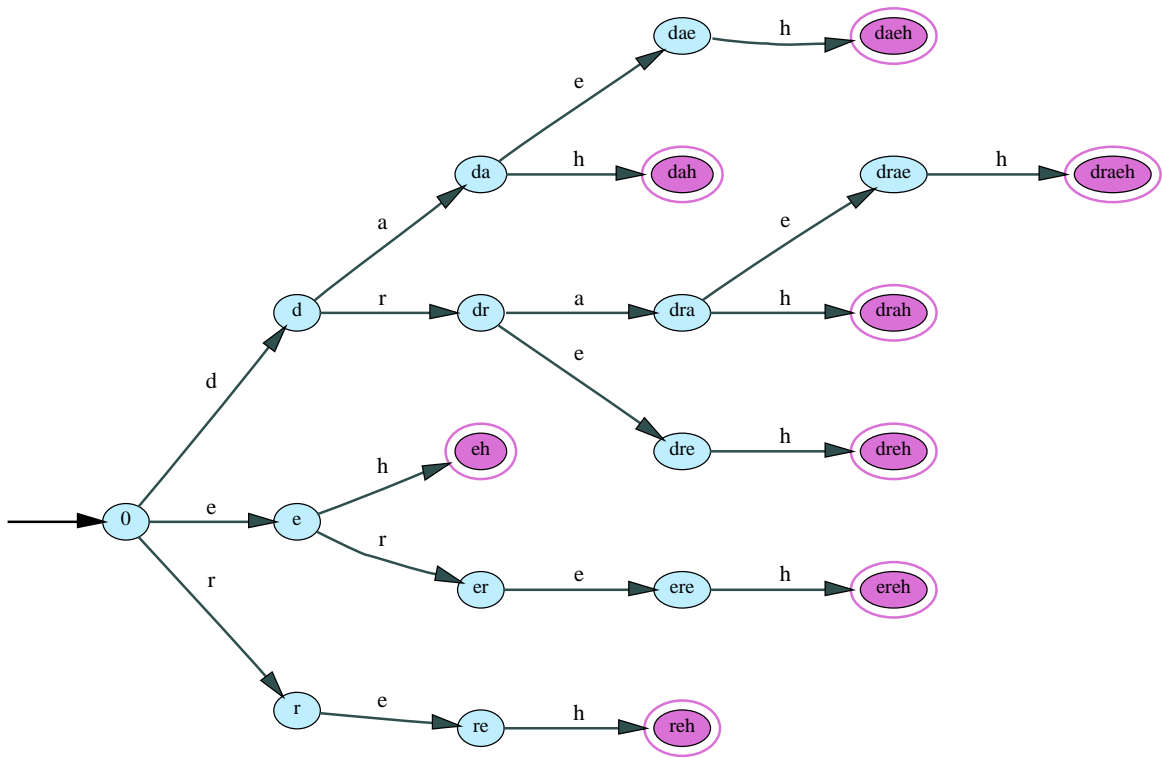
Double reversal

How does Brzozowski's algorithm work:

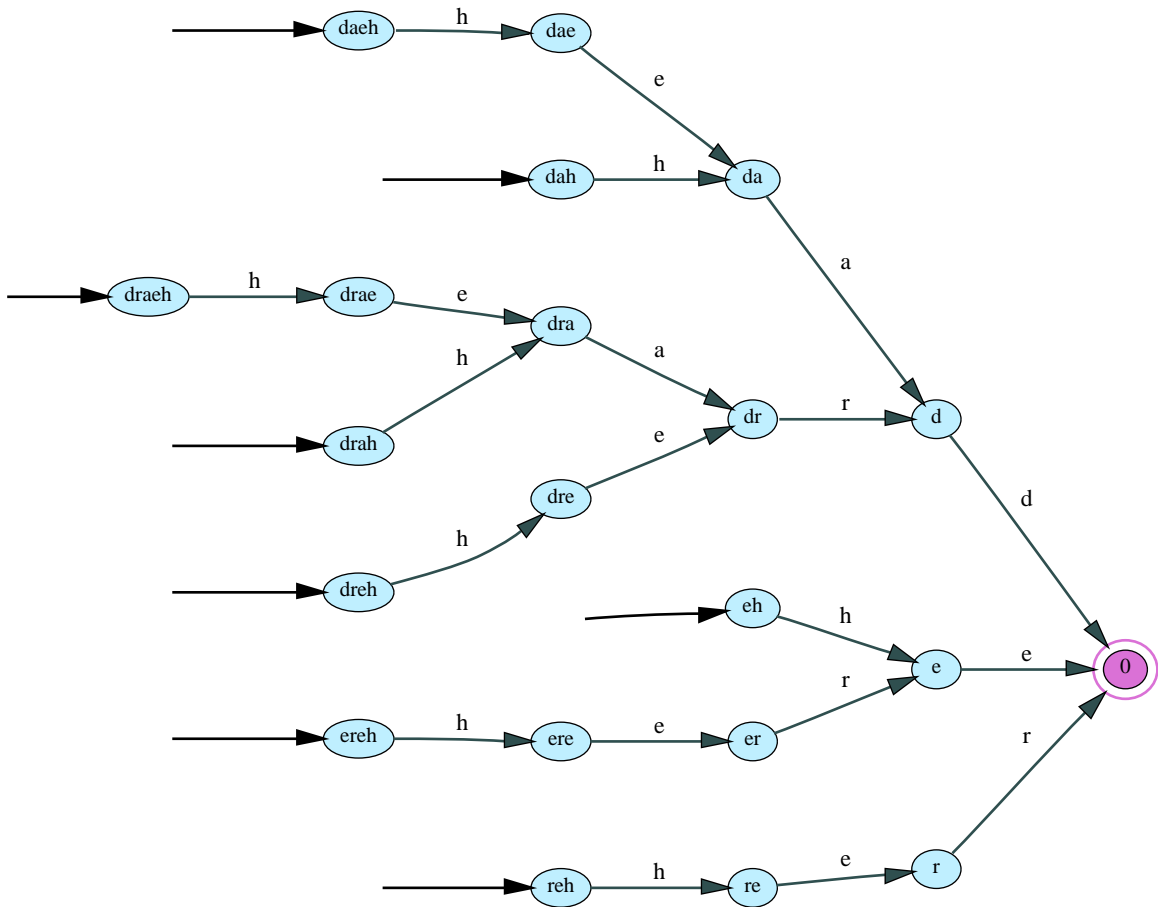
1. Reverse the DFA.
2. Determinize it.
3. Reverse the resulting DFA.
4. Determinize it.

We can directly perform the first two steps by building a *reverse trie*.

Structural invariant: the A DFA is a reverse trie for the words.



complete reverses it again



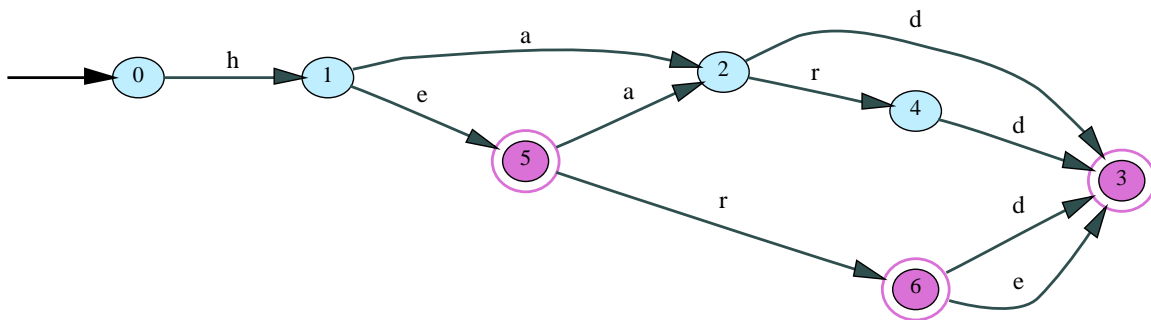
and determinizes.

Incremental algorithm

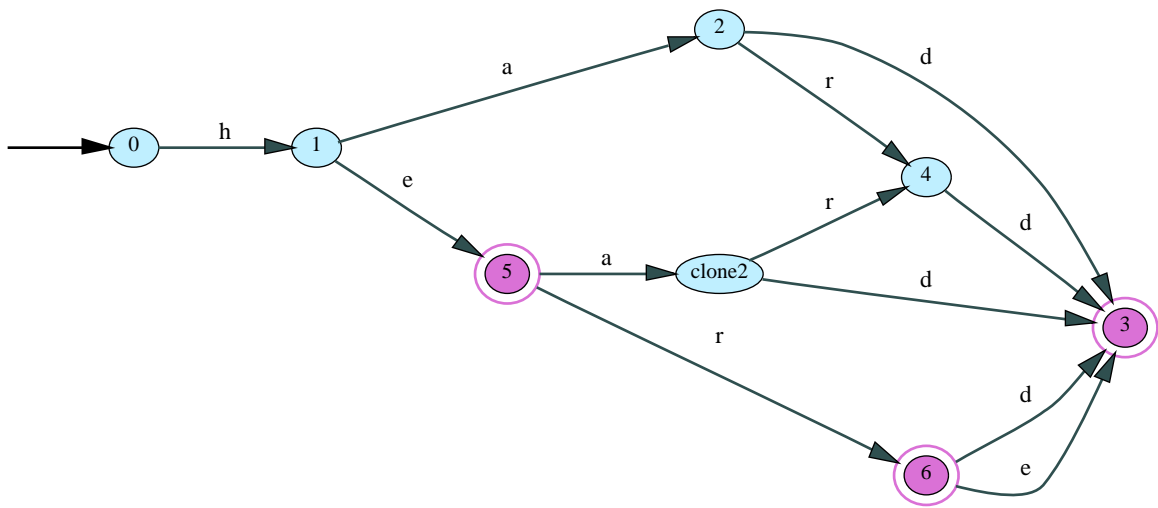
Structural invariant: maintaining minimality.

Real-life memory issues.

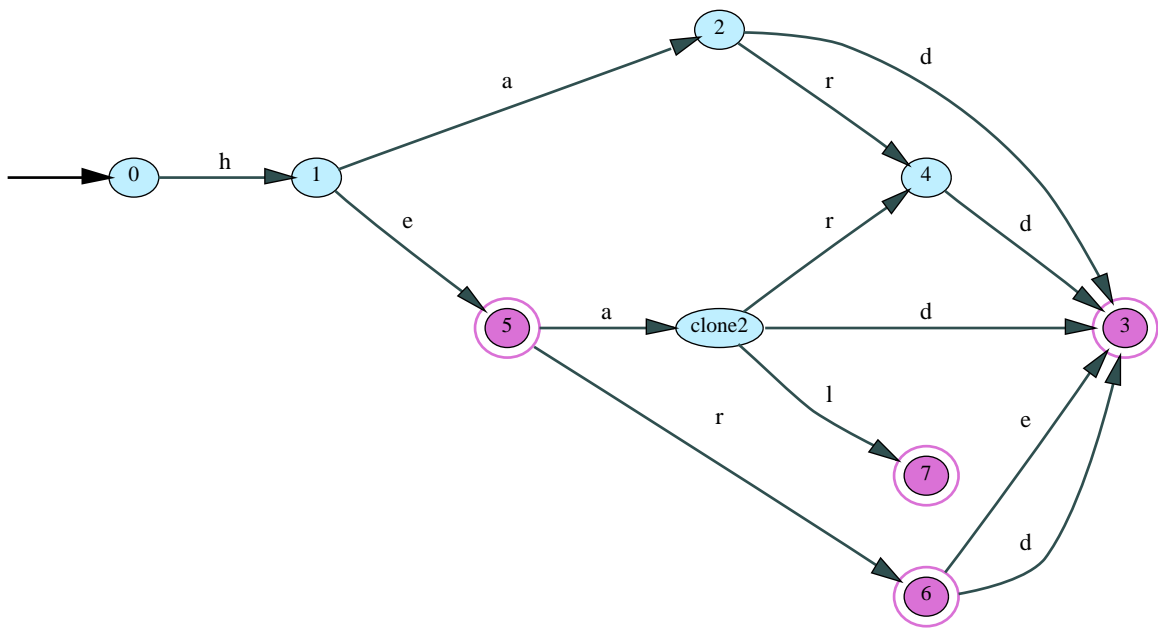
E.g. add *heal* to the MADFA



Following *hea*, state 2 is a 'confluence' and must be cloned, giving



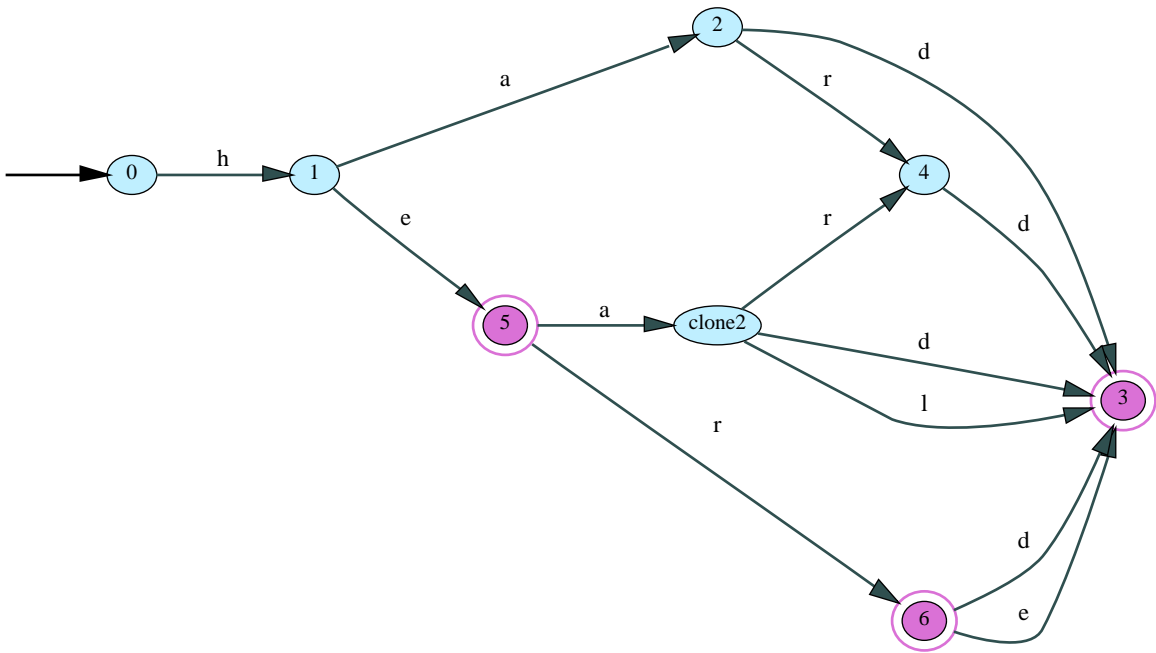
Following *heal*, a new state is created.



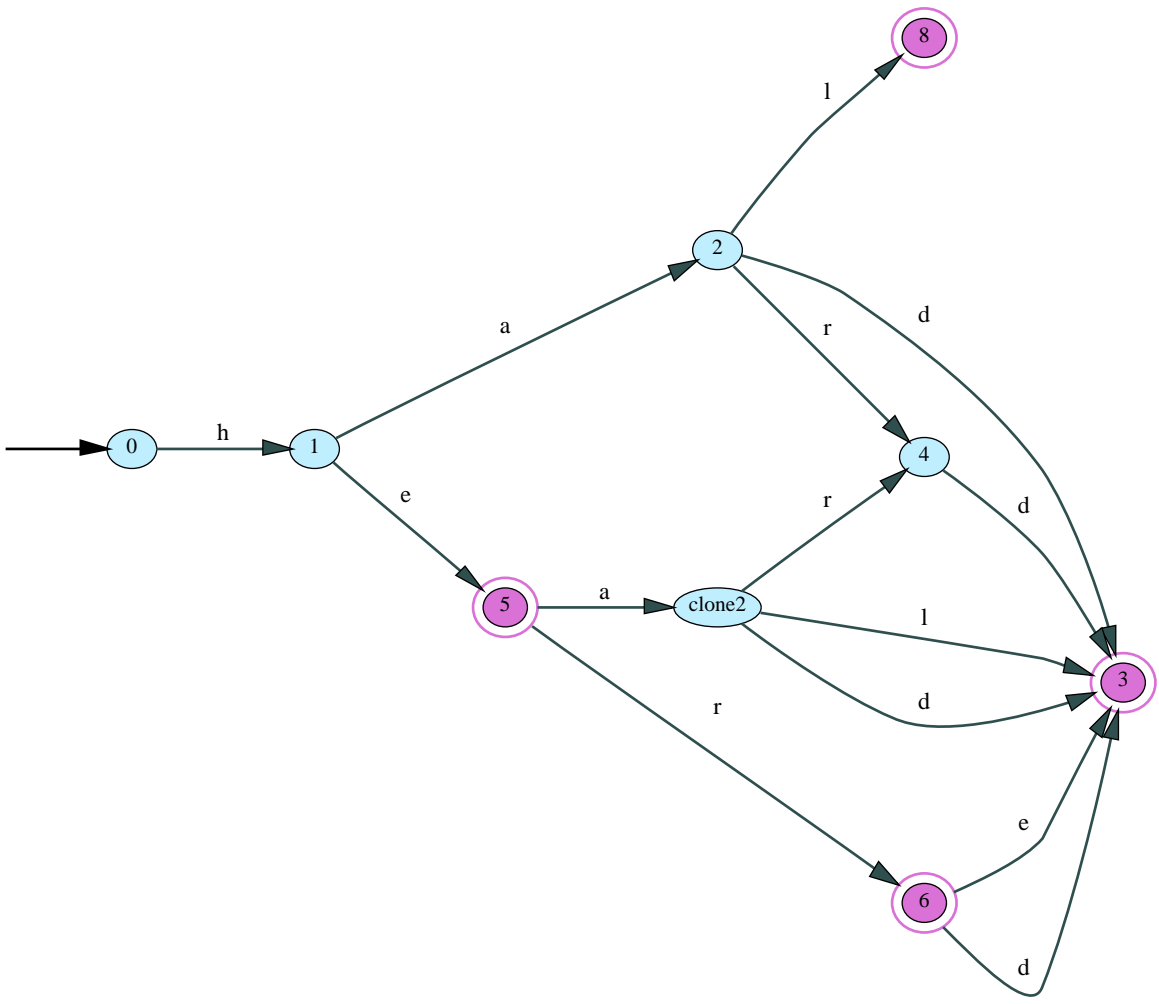
Which right languages have changed?

The path of *heal*.

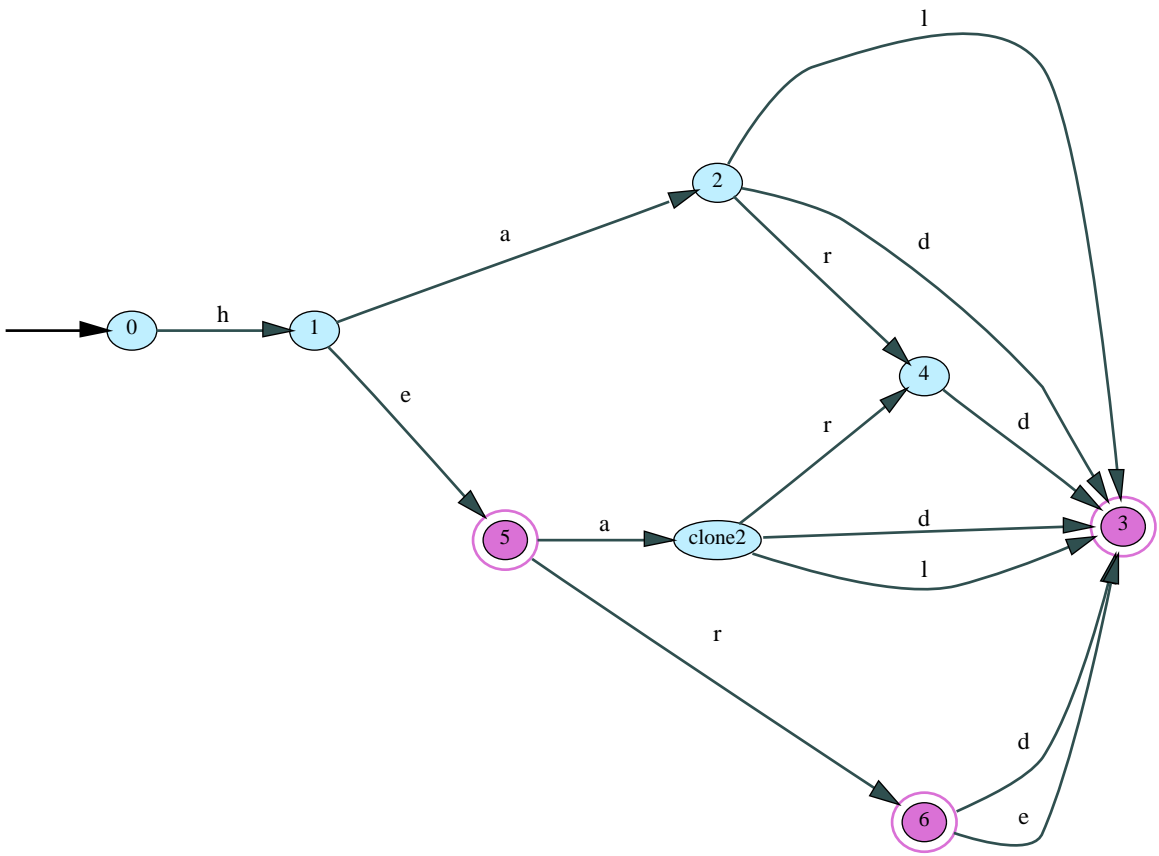
States 3 and 7 can be merged.



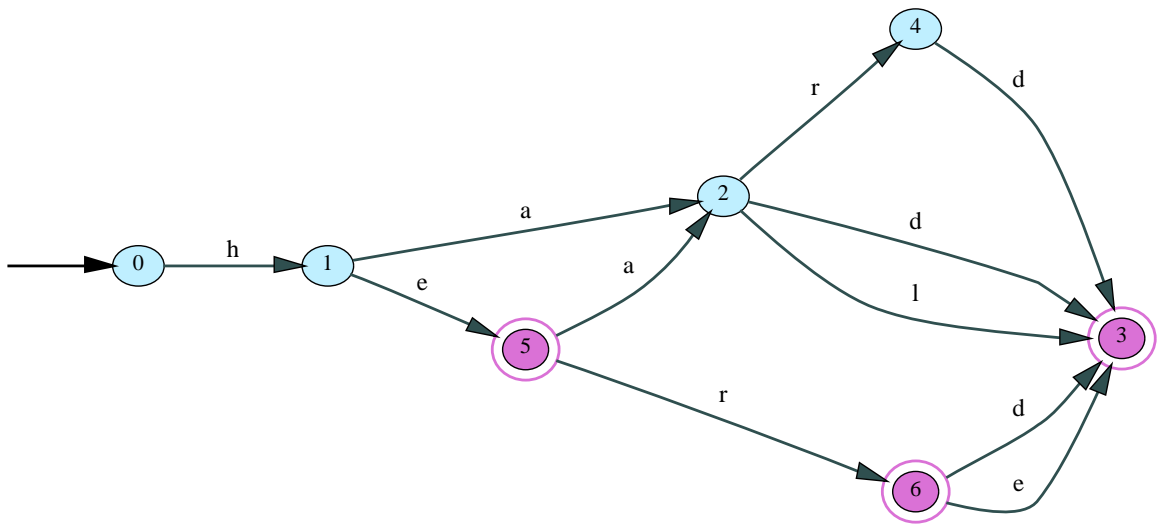
Another example: adding word *hal* would initially yield



States 3 and 8 can be merged



Finally, 2 and clone2 can be merged:



Implementation: FiniteFIRE

Automaton class.

aw and *complete* are procedures.

\leq is an iterator.

C++ implementation (of FiniteFIRE) binds things at compile-time.

Performance ranges (1 million words) from 30 minutes (trie-version) down to 40 seconds (incremental version). Used by: Nokia, Cisco, . . .

Closing comments

- Numerous articles since late 80's, my involvement since 1996.
- Taxonomies and toolkits have enjoyed significant industry success.
- Monograph nearing completion, with short history of the field.
- Transducers (automata with output) remain to be done.