Minimization of acyclic DFAs

Johannes Bubenzer

University of Potsdam
Department of Linguistics

30. August 2011

Overview

- Definitions
- Minimality
- Revuz algorithm
- New algorithm
- Evaluation

Definitions

Deterministic Finite-State Automaton (DFA)
$\mathcal{A} = \langle Q, q_0, F, \delta, \Sigma \rangle$

- Q finite set of states
- Σ finite alphabet
- $q_0 \in Q$ start state
- $F \subseteq Q$ final states
- $\delta : Q \times \Sigma \to Q$ transition function

Acyclic DFA (ADFA)

- DFA which contains no cycles.

Connected DFA

- All states can be reached from the start state
- All states are connected to a final state.

Definitions

Extended transition function δ^*: $(\forall q \in Q, a \in \Sigma, w \in \Sigma^*)$

$$\delta^*(q, \epsilon) = q$$
$$\delta^*(q, a \cdot w) = \delta^*(\delta(q, a), w)$$

Right-Language $\overrightarrow{\mathcal{L}}(q), q \in Q$

- $\overrightarrow{\mathcal{L}}(q) = \{w \mid \delta^*(q, w) \in F\}$

Language $\mathcal{L}(\mathcal{A})$

- $\mathcal{L}(\mathcal{A}) = \overrightarrow{\mathcal{L}}(q_0)$

Definitions

Nerode Equivalence $\sim$ of $q, p \in Q$

- $q \sim p \leftrightarrow \overrightarrow{\mathcal{L}}(q) = \overrightarrow{\mathcal{L}}(p)$
- $[q]$ denotes equivalence class of q wrt. $\sim$.

Right-Language Signature τ of state $q \in Q$

- $\tau(p) = \langle q \in F, \langle a, p \rangle \mid \delta(q, a) = p \rangle \rangle \quad p, q \in Q, a \in \Sigma$
- $\tau(p) = \tau(q) \rightarrow p \sim q$

Note: I assume the transitions to be ordered on the alphabet symbol.

Minimality

Minimal DFA (MDFA) $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}$:

- DFA with the minimal number of states accepting $\mathcal{L}$
- iff $\forall q, p \in Q : q \sim p \rightarrow q = p$

DFA Minimization:

- Create/Determine the MDFA for a given DFA
- Join all $\sim$-equivalent states into one. Adjust transitions.

Minimality

Minimal State $q \in Q$:

- All successor states are minimal
- No other equivalent state exists

Minimal Signature $\tau_{min}(q)$:

- $\tau(q)$ where all successors of q are minimal states.
- $\tau_{min}(p) = \tau_{min}(q) \longleftrightarrow p \sim q$

Minimization

General Idea

- Determine the minimal signatures of some states
- Minimize those states (join the equal, adjust transitions)

ADFA case

- Signature of a state depends on its direct successors
- and we have no cycles
- => minimizing states requires minimizing their successors first

Minimizaion - Two approaches

Revuz (1992) - Minimization of Acyclic Deterministic Automata in Linear Time:

- Process states layerwise
- Starting with final states with no outgoing transitions.

New approach:

- Preorder processing of states.
- All successors are (recursively) minimized before the actual state is

both are O(n).

Example Revuz '92

Start with final states:

- All states without outgoing transition have minimal signatures.
- They are all final, so they are all equivalent.
- Joining them yields one minimal state.

Proceed layerwise:

- All states that have only transitions into previously minimized layers have minimal signatures.
- Minimizing them leads to a new layer of minimal states.
- When the start state is reached all states are minimal

Informal algorithm:

- computes a height for each state (max. distance to a final state)
- process height-levels from low to high:
- sort states of same height according to τ
- sorting requires O(n) wrt. $|\Sigma|$. (radix sort with *tricks*)
- merges states of same height and same τ

Revuz '92

Disadvantages:

- Quite complicated to implement in practice
- Requires precomputation of heights.
- Requires partitioning of states according to height-levels.
- Requires external sorting phase.

Depth-first minimization

Minimize a state q (recursive):

- Minimize all of its successors.
- If a state p with same $\tau(q)$ exists replace q by p.
- Otherwise q is a new representative of class $\tau(q)$.
- Terminates at states with no outgoing transitions.

- Requires a map of $\tau \to Q$ (Register).
- Requires a map of $Q \to Q$ mapping states to class representatives. (StateMap)

Algorithm

```
begin minimize(q)
    foreach trans ∈ q.transitions() do
        if ! StateMap [trans.destination] then
            minimize(trans.destination)
        trans.destination := StateMap [trans.destination]
    if Register [τ(q)] then
        StateMap [q] := Register [τ(q)]
        Q := Q − {q}
    else
        StateMap [q] := Register [τ(q)] := q
end
```

Runnning Time

Algorithm requires linear space

- StateMap contains $|Q|$ states at the end
- Register contains $|Q|$ states at most

Algorithm runs in linear time

- consists in just a pre-order traversal.

Reduced constant factors

- no height-precomputing, no state partitioning
- no sorting

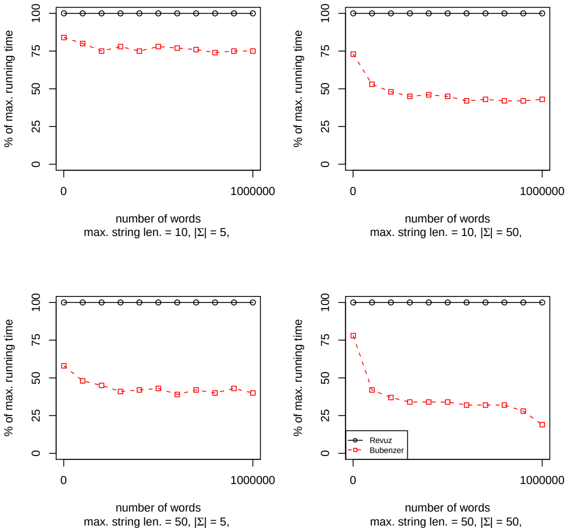
Evaluation

Performed evaluation

- on random-sampled sets of strings (two different distributions)
- variing maximum string lengths
- variing alphabet sizes
- and on natural-language data sets
- compiled into a trie

Implemented new algorithm in a C++ finite-state library. Run against an existing (optimized) Revuz implementation.

Evaluation (uniform distribution)



Conclusion

- faster (in practice)
- simpler (to implement)
- incremental (can be stopped at any time)

Thank you!