Constructive Formalization of Regular Languages

Jan-Oliver Kaiser
Advisors: Christian Doczkal, Gert Smolka
Supervisor: Gert Smolka

April 23, 2012

Our roadmap:

1. RE $\Rightarrow$ FA (**DONE**)
2. Emptiness test on FA (**Easy**)
3. RE equivalence (**Follows from 1 and 2**)
4. FA $\Rightarrow$ RE
5. Myhill-Nerode

Finite automata to regular expressions

- Converting REs to FAs is straight forward and there is really only one algorithm (with slight variations).

Finite automata to regular expressions

- Converting REs to FAs is straight forward and there is really only one algorithm (with slight variations).
- Converting FAs to REs is complicated and there are at least three general algorithms.

Finite automata to regular expressions

- Converting REs to FAs is straight forward and there is really only one algorithm (with slight variations).
- Converting FAs to REs is complicated and there are at least three general algorithms.

Why is that?

My intuition:

- Converting REs to FAs is done by structural recursion on a **tree**. The result is a **flat structure**.

My intuition:

- Converting REs to FAs is done by structural recursion on a **tree**. The result is a **flat structure**.
- Converting FAs to REs **can not be done** by structural recursion. There is no **recursive structure** in FAs. But somehow we need to construct a **tree structure**.

Three methods (+ variations):

1. Transitive Closure
2. State Removal
3. Brzozowski Algebraic Method

Transitive Closure

Given: DFA $A = (\Sigma, Q, s_0, F \subseteq Q, \delta)$.

Idea: Construct regexps r_f for every final states $f \in F$ s.t. r_f matches all words which A accepts with final state f.

$\Rightarrow \mathcal{L}(A) = \mathcal{L}(\sum_{f \in F} r_f)$

How do we construct r_f?

We generalize the idea of r_f to R_{ij}^k which matches all words with which lead from state i to j while passing only through states with index smaller than k.

1. Merge multiple edges between states to one unified edge.
2. Construct regexp R_{ij}^k recursively:

$$R_{ij}^0 := \begin{cases} r & \text{if } i \neq j \wedge i \text{ has edge } r \text{ to j} \\ \varepsilon + r & \text{if } i = j \wedge i \text{ has edge } r \text{ to j} \\ \emptyset & \text{otherwise} \end{cases}$$

$$R_{ij}^k := R_{ik}^{k-1} R_{kk}^{k-1} R_{kj}^{k-1} + R_{ij}^{k-1}$$

$$\Rightarrow \mathcal{L}(A) = \mathcal{L}(\textstyle\sum_{f \in F} r_f) = \mathcal{L}(\textstyle\sum_{f \in F} R_{0f}^{|Q|})$$

Formalization:
Easier than the other methods.
The recursive definition translates quite well.
Some details are quite challenging.

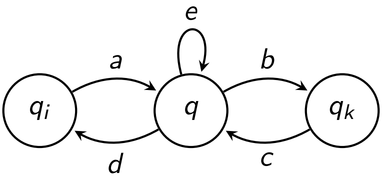
Efficiency: The resulting regular expressions are huge.

Complexity: $O(n^3)$ in the number of states.

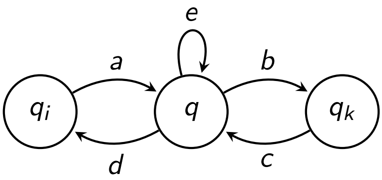
State Removal

Given: FA $A = (\Sigma, Q, s_0, F \subseteq Q, \delta)$.
New concept: Automata that have transitions labeled by RE.
Idea: Remove states until there are two or less states remaining. Update the remaining states' transitions by incorporating the "lost" paths.

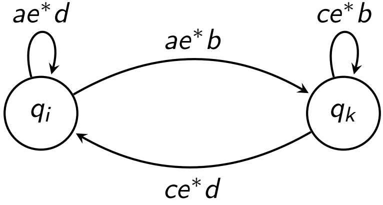
Remove q from



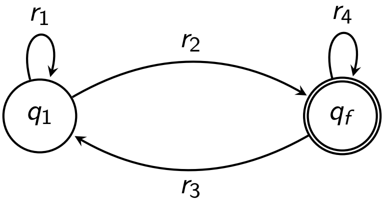
Remove q from



to get



Repeat until A is of this form:



$$\Rightarrow \mathcal{L}(A) = \mathcal{L}(r_1^* r_2 (r_4 + r_3 r_1^* r_2)^*)$$

Caveats

- It looks like we only need to update two edges.

Caveats

- It looks like we only need to update two edges.
  In reality, there can be $|Q| - 1$ states connected to q.

Caveats

- It looks like we only need to update two edges.
  In reality, there can be $|Q| - 1$ states connected to q.
- What about final states?

Caveats

- It looks like we only need to update two edges.
  In reality, there can be $|Q| - 1$ states connected to q.
- What about final states?
  Introduce a new final state without any outgoing edges.
  Introduce ε transitions from all other final states to the final new state.
  Make all other states non-final.
  Never remove the new final state.

Formalization:
Complex.
Lots of details to consider.

Efficiency: The resulting regular expressions are big.
They can be made small in special cases (e.g., acyclic automata).

Complexity: $O(n^3)$ in the number of states.
Less in special cases (e.g. $O(n^2 logn)$ for acyclic automata).

Brzozowski Algebraic Method

Given: FA $A = (\Sigma, Q, q_0, F \subseteq Q, \delta)$.

Idea: Retrieve a RE for FA by solving a system of equations determined by δ.

Construct system of equations:

$$\begin{aligned} r_0 &= \sum_{\substack{a\in\Sigma \\ 0\leq i<|Q|}} \{\, a\, r_i \mid (q_0, a, q_i) \in \delta\} && (+\,\varepsilon \text{ if } r_0 \in F) \\ \vdots &= \vdots \\ r_{|Q|-1} &= \sum_{\substack{a\in\Sigma \\ 0\leq i<|Q|}} \{\, a\, r_i \mid (q_{|Q|-1}, a, q_i) \in \delta\} && (+\,\varepsilon \text{ if } r_{|Q|-1} \in F) \end{aligned}$$

$$\Rightarrow \mathcal{L}(A) = \mathcal{L}(r_0)$$

Solve the system by substitution and **Arden's Lemma** which states that for all regular languages X, Y and Z the equation

$$X = YX + Z \tag{1}$$

has the unique solution

$$X = Y^*Z \tag{2}$$

Formalization:
Complex due to missing infrastructure.

Efficiency: The resulting regular expressions are quite small.

Complexity: $O(n^3)$ in the number of states.
Less in special cases (e.g. $O(n^2 logn)$ for acyclic automata).

- We eliminated the Brzozowski algebraic method due to its heavy overhead in required infrastructure.

- We eliminated the Brzozowski algebraic method due to its heavy overhead in required infrastructure.
- The state removal method requires a new kind of automaton which we would rather avoid.

- We eliminated the Brzozowski algebraic method due to its heavy overhead in required infrastructure.
- The state removal method requires a new kind of automaton which we would rather avoid.
- We settled for Kleene's transitive closure method.