

- Recap
- Goal
- Finite automata to regular expressions
- State Removal
- Brzozowski Algebraic Method
- Transitive Closure
- Our Approach
- Summary

Constructive Formalization of Regular Languages

Jan-Oliver Kaiser

Advisors: Christian Doczkal, Gert Smolka

Supervisor: Gert Smolka

April 27, 2012

1 Goal

2 State Removal

3 Brzozowski Algebraic Method

4 Transitive Closure

5 Our Approach

6 Summary

Definitions:

- We use extended regular Expressions (RE):

$$r, s ::= \emptyset \mid \varepsilon \mid a \mid rs \mid r + s \mid r \& s \mid r^* \mid \neg r$$

$$\text{Def.: } \bigoplus_{x \in X} r_x := r_{x_0} + \dots + r_{x_{|X|-1}}$$

- Our Finite Automata (FA) are

$$(\Sigma, Q, q_0, F, \delta)$$

(The transition relation δ of deterministic FA is total).

Goal:

Find simple proofs for the decidability of regular expression equivalence and the Myhill-Nerode theorem.

Roadmap:

- 1 RE \Rightarrow FA (**DONE**)
- 2 Emptiness test on FA (**Easy**)
- 3 RE equivalence (**Follows from 1 and 2**)
- 4 FA \Rightarrow RE (**Work in progress**)
- 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):
We mirror the constructors of RE in operations on FAs.

Finite automata to regular expressions

- Converting REs to FAs is straight-forward and there is really only one algorithm (with slight variations):
We mirror the constructors of RE in operations on FAs.
- Converting FAs to REs is complicated and there are at least three algorithms found in textbooks.

Finite automata to regular expressions

- Converting REs to FAs is straight-forward and there is really only one algorithm (with slight variations):
We mirror the constructors of RE in operations on FAs.
- Converting FAs to REs is complicated and there are at least three algorithms found in textbooks.

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** of REs.

Three methods (+ variations):

- 1 Transitive Closure (Kleene [3])
- 2 State Removal (Du, Ko [2], simplified in Linz [4])
- 3 Brzozowski Algebraic Method (Brzozowski [1])

Recap
Goal
Finite automata to regular expressions
State Removal
Brzowski Algebraic Method
Transitive Closure
Our Approach
Summary

Approach
Algorithm
Caveats
Properties

State Removal

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

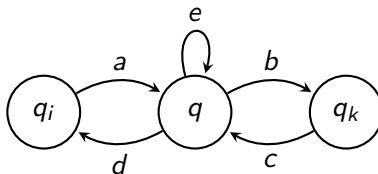
New concept: Automata that have transitions labeled by RE.

Idea: Remove states until there are two or less states remaining.
Update the remaining states' transitions by incorporating the
"lost" paths.

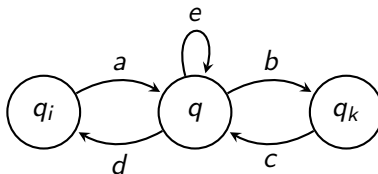
Recap
Goal
Finite automata to regular expressions
State Removal
Brzozowski Algebraic Method
Transitive Closure
Our Approach
Summary

Approach
Algorithm
Caveats
Properties

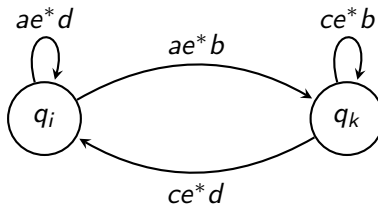
Remove q from



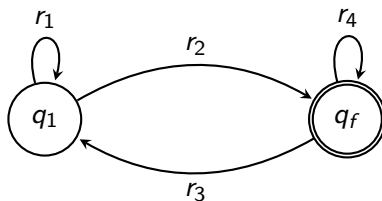
Remove q from



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

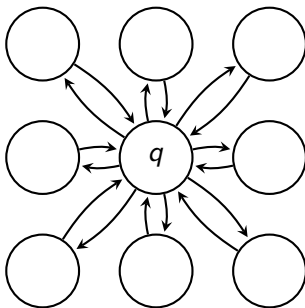
- It looks like we only need to update two edges.

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 .



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?
 - 1 Introduce a new final state without any outgoing edges.
 - 2 Introduce ε transitions from all other final states to the final new state.
 - 3 Make all other states non-final.
 - 4 Never remove the new final state.

Formalization:

- Requires a new kind of finite automaton that has RE transitions.
- Lots of details to consider.
- Induction on the number of states.

	Recap
	Goal
	Finite automata to regular expressions
	State Removal
	Brzozowski Algebraic Method
	Transitive Closure
	Our Approach
	Summary

Approach
Algorithm
Properties

Brzozowski Algebraic Method

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

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

Construct system of equations:

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

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

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

$$X = YX + Z \quad (1)$$

has the unique solution

$$X = Y^*Z \quad (2)$$

Formalization:

- Requires a formalization of these equations and operations on them.
- We would need to prove Arden's Lemma.
- We would also need to prove that Arden's Lemma (and substitution) is enough to solve these systems of equations.

Transitive Closure

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

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

$$\Rightarrow \mathcal{L}(A) = \mathcal{L}\left(\bigoplus_{f \in F} r_f\right)$$

How do we construct r_f ?

We generalize the idea of r_f to R_{ij}^k which matches all words 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:

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

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

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

Formalization:

- Easier than the other methods.
- The recursive definition translates quite well.
- The details are quite challenging.

Formalization:

- Easier than the other methods.
- The recursive definition translates quite well.
- The details are quite challenging.

This appears to be the simplest to formalize.

Our Approach:

There are different ways of formalizing R_{ij}^k itself, especially its parameters. Most practical so far:

k is of type `nat`, i and j are ordinals from $[0..|Q| - 1]$.

Our Approach:

There are different ways of formalizing R_{ij}^k itself, especially its parameters. Most practical so far:

k is of type nat, i and j are ordinals from $[0..|Q| - 1]$.

This gives us easy recursion and matching on k .

Our Approach:

There are different ways of formalizing R_{ij}^k itself, especially its parameters. Most practical so far:

k is of type nat, i and j are ordinals from $[0..|Q| - 1]$.

This gives us easy recursion and matching on k .

But we have to use $\min k (|Q| - 1)$ to map k to the corresponding ordinal (and then to a state).

To get a FA counterpart to R_{ij}^k , we introduce A_{ij}^k s.t.

$$\mathcal{L}(A_{ij}^k) = \mathcal{L}(R_{ij}^k).$$

A_{ij}^k is similar to A . It has one additional state, which has all incoming edges of state j and no outgoing edges. It only leaves states that are i or $< k$. It only enters states $< k$ or the new state. We can then show that

$$\mathcal{L}\left(\bigcup_{f \in F} A_{q_0 f}^{|Q|}\right) = \mathcal{L}(A).$$

Lemmas:

- 1 The language of an automaton is the union of the words with runs on A that end in any final state, i.e.

$\mathcal{L}(A) = \bigcup_{f \in F} \mathcal{L}(A_f)$, where A_f accepts only in f . (Not difficult)

- 2 A path in A_{ij}^{k+1} that is not in A_{ij}^k can be decomposed into:

- a sub path from index 0 to the first occurrence of k
- a sub path from the first to the last occurrence of k
- the remaining sub path from the last occurrence of k to the end.

(Complex; requires new operations on lists and paths)

- 3 If a path on A passes through a final state more than once, the corresponding word is in $\mathcal{L}(A)^*$ (Relies on the same operations we need for lemma 2)
- 4 $\mathcal{L}(A_{ij}^k) = \mathcal{L}(R_{ij}^k)$ (By recursion on k , using lemmas 1, 2 and 3)

- 5 Any path on A_{ij}^k that ends in the new accepting state can be mapped to an equivalent path that ends state j (and vice-versa). (Probably easy)
- 6 Any path on A_{ij}^k that does not end in the new accepting state is also a path on A . (Follows from lemma 5 and definition of A_{ij}^k)
- 7 $\mathcal{L}(A_{q_0f}^{|Q|}) = \mathcal{L}(A_f)$ (Follows from lemma 6)

Theorem:

$$\mathcal{L}\left(\bigoplus_{f \in F} R_{q_0 f}^{|Q|}\right) = \mathcal{L}(A).$$

Proof:

- By lemma 1: $\mathcal{L}\left(\bigoplus_{f \in F} R_{q_0 f}^{|Q|}\right) = \bigcup_{f \in F} \mathcal{L}(A_f).$
- By lemma 7: $\mathcal{L}\left(\bigoplus_{f \in F} R_{q_0 f}^{|Q|}\right) = \bigcup_{f \in F} \mathcal{L}(A^{|Q|}_{q_0 f}).$
- By lemma 4: $\mathcal{L}\left(\bigoplus_{f \in F} R_{q_0 f}^{|Q|}\right) = \bigcup_{f \in F} \mathcal{L}(R^{|Q|}_{q_0 f}).$
- Holds by definition of \oplus .

Summary:

- Creating recursive from non-recursive structures is difficult.
- Existing algorithms to construct RE from FA differ vastly in how easily and elegantly we can formalize them.
- We benefit from thinking of the R_{ij}^k invariant in terms of existing infrastructure (A_{ij}^k is an ordinary NFA).

Recap
Goal
Finite automata to regular expressions
State Removal
Brzozowski Algebraic Method
Transitive Closure
Our Approach
Summary

Thank you for your attention

Reference I

- [1] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.
- [2] Ding-Shu Du and Ker-I Ko. *Problem Solving in Automata, Languages, and Complexity*. 2001.
- [3] S. C. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, 1965.
- [4] Peter Linz. *An introduction to formal languages and automata (4. ed.)*. Jones and Bartlett Publishers, 2006.