

Practical Course presentation

Verified Conversion between NFAs and Regular Expressions

Julian Brunner
julianbrunner@gmail.com

Manuel Eberl
eberlm@in.tum.de

Advisor: Dr. Peter Lammich
lammich@in.tum.de

- 1 Introduction
- 2 NFA to Regular Expression
 - Algorithms
 - The abstract algorithm
 - Refinement
 - Evaluation
- 3 Regular Expression to NFA
 - Algorithms
 - Implementation and Verification
 - Evaluation
- 4 Conclusion

Problem

- Given
 - $r \in \text{RE}(\Sigma)$
 - $\mathcal{A} = (\mathcal{Q}, \Sigma, \Delta, \mathcal{I}, \mathcal{F})$
- Wanted
 - $\text{nfa_to_re} :: \text{NFA}(\mathcal{Q}, \Sigma) \Rightarrow \text{RE}(\Sigma)$
 - $\text{re_to_nfa} :: \text{RE}(\Sigma) \Rightarrow \text{NFA}(\mathcal{Q}, \Sigma)$
- Such that
 - $\mathcal{L}(\text{nfa_to_re}(\mathcal{A})) = \mathcal{L}(\mathcal{A})$
 - $\mathcal{L}(\text{re_to_nfa}(r)) = \mathcal{L}(r)$

Problem

Regex as defined in `Regular_set.thy`:

- Zero with $\mathcal{L}(\text{Zero}) = \{\}$
- One with $\mathcal{L}(\text{One}) = \{\varepsilon\}$
- Atom a with $\mathcal{L}(\text{Atom } a) = \{a\}$
- Plus $r_1 \ r_2$ with $\mathcal{L}(\text{Plus } r_1 \ r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$
- Times $r_1 \ r_2$ with $\mathcal{L}(\text{Times } r_1 \ r_2) = \mathcal{L}(r_1) \cdot \mathcal{L}(r_2)$
- Star r with $\mathcal{L}(\text{Star } r) = \mathcal{L}(r)^*$

Part 1: NFA to Regular Expression

Manuel Eberl

Algorithm 1 – system of equations

- Set up system of equations for accepted words in state i :
$$X_i = \{c_1\} \cdot X_{i_1} \cup \dots \cup \{c_k\} \cdot X_{i_k} \quad (\cup \{\epsilon\})$$
- While \exists unsolved equation for some X_i :
 1. bring equation into form $X_i = A \cdot X_i \cup B$
 2. apply Arden's lemma: $X_i = A^* \cdot B$
 3. propagate solution to other equations

Algorithm 2 – recursive approach

Definition

Let $R_{Q'}^{i,j} \in RE(\Sigma)$

such that $\mathcal{L}(R_{Q'}^{i,j}) = \mathcal{L}_{(Q', \Sigma, \delta, I, F)}(i, j)$

Then: recursive computation of $R_Q^{i,j}$

- $R_{\{\}}^{i,i} = \delta(i, i) \mid \varepsilon$
- $R_{\{\}}^{i,j} = \delta(i, j) \quad (\text{for } i \neq j)$
- $R_{Q' \cup \{j\}}^{i,k} = R_{Q'}^{i,k} \mid R_{Q'}(i, j) \cdot R_{Q'}(j, j)^* \cdot R_{Q'}(j, k)$

Algorithm 3 – matrix method (cf. Dexter Kozen)

System can be seen as $AX = b$
where $X = (X_1, \dots, X_n)^T$ and $A \in \text{RE}(\Sigma)^{n \times n}$

Solution: $X = A^* \cdot b$

Compute A^* by Divide & Conquer

Note: all previous algorithms can be seen as special case of this
Divide & Conquer algorithm

Algorithm 4 – contractions

Definition (GNFA (Generalised NFA))

Properties of a GNFA:

- regexes as labels, $\delta : Q \mapsto Q \mapsto \text{RE}(\Sigma)$
- $I = \{\text{Start}\}$ and $F = \{\text{End}\}$
- $\forall q. \delta(q, \text{Start}) = \{\} \wedge \delta(\text{End}, q) = \{\}$

Idea:

1. convert NFA to GNFA (trivial)
2. remove some state q and compensate with new transitions
3. rinse, repeat

Algorithm 4 – contractions

We have implemented contractions. Why?

- simple, intuitive algorithm
- straightforward correctness proof
- formalisation effort significant, but not higher than the other algorithms
- implementation equivalent to all the other algorithms (except matrix)
- complexity $O(n^3)$ the same for all algorithms
But: worst-case output size $2^{\Omega(n)}$ (Gruber/Holzer, 2008)

Abstract algorithm

Note: on the abstract level, GNFA is labelled with elements from Σ^* not $\text{RE}(\Sigma)$.

Why? all proofs far easier; \cup is commutative and associative, Plus isn't. Regexes have too much “structure” for the abstract level.

Abstract algorithm

```
 $\mathcal{A}' \leftarrow \text{NFA\_to\_GNFA } \mathcal{A}$   
while  $Q_{\mathcal{A}'} \neq \{\text{Start}, \text{End}\}$  do  
  obtain  $q \in Q_{\mathcal{A}'} \setminus \{\text{Start}, \text{End}\}$   
  for each  $u \in \delta(u, q) \neq \{\}$  do  
    for each  $v \in \delta(q, v) \neq \{\}$  do  
       $\delta(u, v) := \delta(u, v) \cup \delta(u, q) \cdot \delta(q, q)^* \cdot \delta(q, v)$   
    end for  
  end for  
   $Q := Q \setminus \{q\}$   
end while  
return  $\delta(\text{Start}, \text{End})$ 
```

Abstract correctness proof

Abstract part of the theory:

1. definition of GNFA
2. inductive for reachability in GNFAs
3. lemma: adding the subsumed transitions of state q does not change behaviour of automaton
4. lemma: removing q thereafter does not change behaviour (reachability) of remaining automaton
5. definition of conversion abstract NFA \rightarrow abstract GNFA

Refinement

1. implementation of algorithm as shown above
 - while loop invariant: $\forall u, v \in Q_{\mathcal{A}'}. \mathcal{L}_{\mathcal{A}'}(u, v) = \mathcal{L}_{\mathcal{A}}(u, v)$
 - after the loop, $Q = \{\text{Start}, \text{End}\}$ and therefore $\mathcal{L}_{\mathcal{A}'} = \delta(\text{Start}, \text{End})$
2. Refinement: caching of predecessors/successors
3. Refinement: δ from $Q \mapsto Q \mapsto \Sigma^*$ to $Q \mapsto (Q \mapsto \text{RE}(\Sigma) \text{ option}) \text{ option}$
Note: now the return value is a regex
4. Refinement: executable data structures

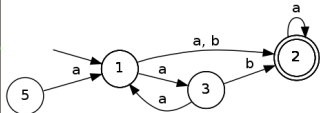
Evaluation

Exported ML code + sugar:

```

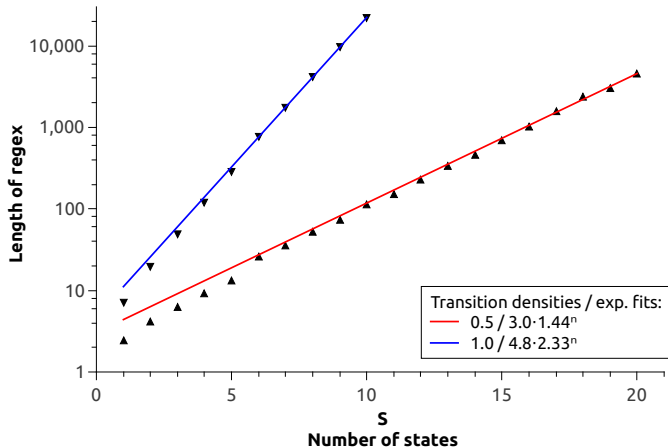
manuel@glados: ~/isaworspace/cava/Regexp/NFAtoRE/c graphviz:
File Edit View Search Terminal Help
(Rbt (Branch (B, Empty, 2, ()), Empty
  Nfa_props_ext (false, false, ())))
> show_NFA automaton1;
Error_Type error in function application.
Function: show_NFA : string option -> NF
Argument: automaton1 : NFA
Reason:
  Can't unify string option to
  (int, unit) RBT.rbt *
  ((string, unit) RBT.rbt *
  (((int, (string, (int, unit) RBT.rbt)
  (int, (string, int) RBT.rbt) RBT.rbt) LTSByLTS_DELT.LTS_choice
  ((int, unit) RBT.rbt *
  ((int, unit) RBT.rbt * unit NFABylTS.nfa_props_ext))))
  (Incompatible types)
Found near show_NFA automaton1
Static Errors
> show_NFA NONE automaton1;
val it = (): unit
> rexp_to_string (nfa_to_rexp automaton1);
val it = "(aa)*(a|b|ab)a*": string
>

```



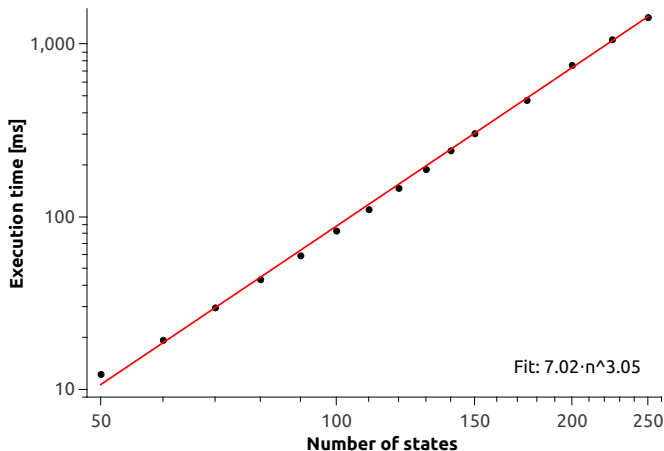
Evaluation

Average size of regex against # of states, log plot



Evaluation

Average computation time against # of states, log/log plot



Part 2: Regular Expression to NFA

Julian Brunner

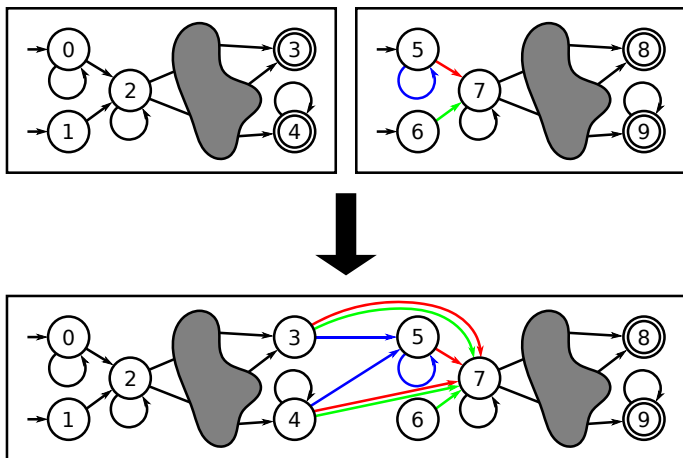
Classic Thompson Algorithm

- Follows the syntactic structure of the regular expression
- Constructs automata for primitives Zero, One and Atom a
- Given a non-primitive expression
 - Recursively constructs automata for the subexpressions
 - Merges these automata according to the original expression
- Usually described in terms of ϵ -NFAs

Algorithms Based on Derivatives

- Derivative of r with respect to u : $\mathcal{L}(D_u(r)) = \{v \mid uv \in \mathcal{L}(r)\}$
- $D_u(r)$ is a regular expression
- Algorithms
 - Brzozowski
 - Glushkov / McNaughton & Yamada
 - Berry & Sethi

Thompson Merging Algorithm



Thompson Merging Realization

- Define algorithm in terms of sets and abstract NFAs
 - Primitives: `TM_zero_NFA` `TM_one_NFA` `TM_atom_NFA`
 - Composite: `TM_plus_NFA` `TM_times_NFA` `TM_star_NFA`
 - Translation: `TM_translate_NFA`
 - States are natural numbers
 - Automata are built using consecutive state indices
- Prove correctness of abstract definitions
 - Example: $\mathcal{L}(\text{TM_star_NFA}(\mathcal{A})) = \mathcal{L}(\mathcal{A})^*$
- Refine to executable data structures

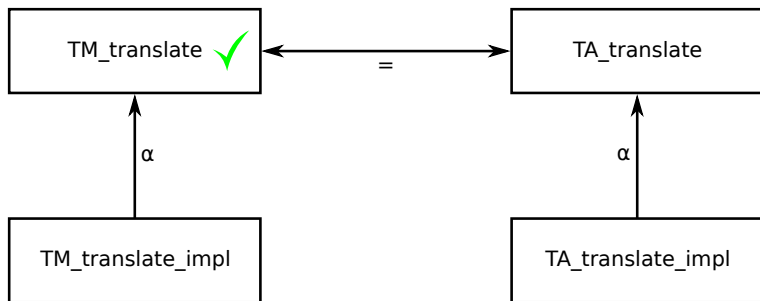
Thompson Accumulating Algorithm

- Union of large sets performs poorly
- Idea: Collect small automata in accumulator automaton
- TM: Merge given automata, return result
- TA: Build merged automaton on top of accumulator, return new accumulator
- Further optimizations: Accumulator state
 - gives the first unused state index
 - indicates whether the accumulator automaton accepts ϵ

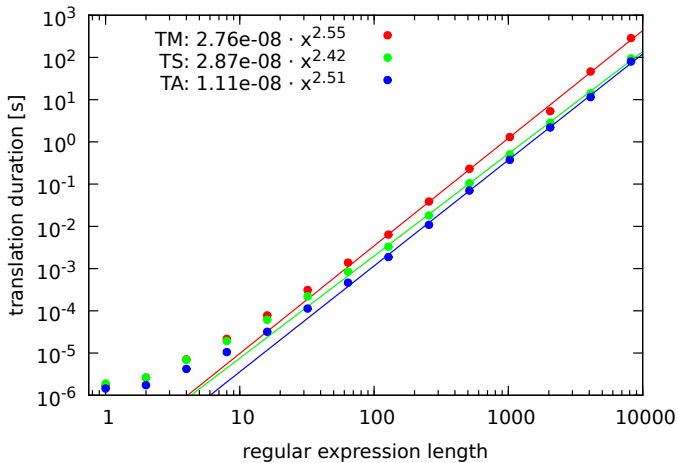
Thompson Accumulating Realization

- Define algorithm in terms of sets and abstract NFAs
- Prove that the TA algorithm builds the same automaton as the TM algorithm
- Refine to executable data structures

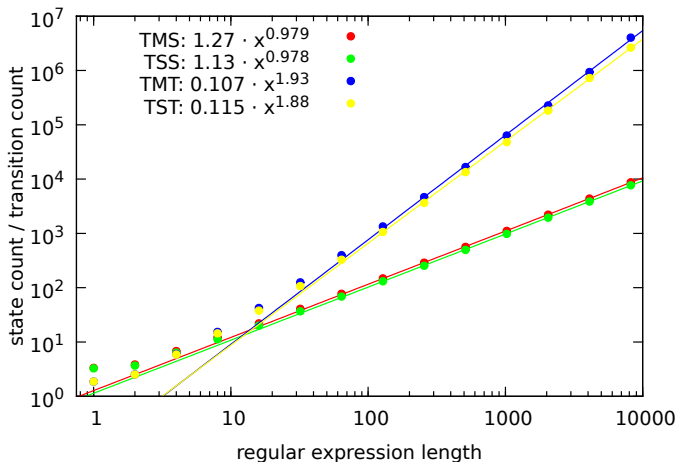
Algorithms and Correctness Proofs



Translation Performance



Automaton Size



Conclusion

Possible improvements / further work:

- General
 - clean up proofs, make an AFP entry
 - what about extended regexes (\cap , \neg)?
- NFA to RE
 - use on-the-fly simplification of regexes to avoid unnecessarily bloated regexes for complex automata
- RE to NFA
 - implement a derivative-based algorithm if smaller automata and faster translation times are desired