To transform is to understand:

an experiment of building an interactive regular language converter

Antonina Nepeivoda

$\mathcal{STEP}$, *Innopolis, March 8th*
Program Systems Institute of RAS

Background (BMSTU, IU9)

- Formal Language theory course, 5th semester, 3 to 4 lectures on regular languages.
- The students have elementary theoretical background of the automata, and most of them use regexes in the programming practice.
- The basic course is too repetitive; the abstract algebraic course is too challenging.
- Intention: to make theoretical concepts more tangible by using an interactive regular language converter.

Two approaches to automata theory

Classical Approach

- Base: DFA representation
- Study of the classical FA transformations (e.g., determinization, minimization...)
- Focus on parsing and algorithms

Modern Approach

- Base: algebraic structures (equivalence classes, monoids...)
- Study of the uniform and specific properties of different language representations
- Focus on generic analysis techniques rather than algorithms (e.g., simulation, closure, rewriting)

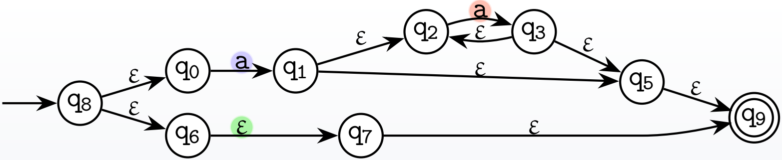
Example: Unified Glushkov construction

(C.Allauzen, M.Mohri: Math. Found. of Comp. Sci. 2006)

Aims at unifying several NFA constructions using closure+rewriting approaches over the classical Thompson automaton.

Thompson automaton

Stepwise construction from the subregexes.
Running example: regex $aa^* \mid \varepsilon$



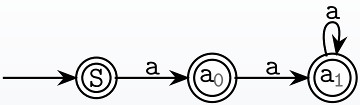
Example: Unified Glushkov construction

(C.Allauzen, M.Mohri: Math. Found. of Comp. Sci. 2006)

Aims at unifying several NFA constructions using closure+rewriting approaches over the classical Thompson automaton.

Glushkov automaton

Making use of Follow-relation. The NFA states correspond to the letters in the linearized regex.
The linearized regex: $a_0 a_1^* \mid \varepsilon$
The Follow set: $\{(a_0, a_1)\ (a_1, a_1)\ \}$

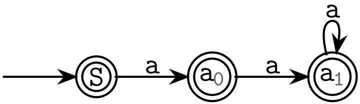


Example: Unified Glushkov construction

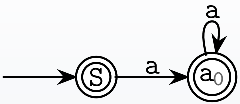
(C.Allauzen, M.Mohri: Math. Found. of Comp. Sci. 2006)

Aims at unifying several NFA constructions using closure+rewriting approaches over the classical Thompson automaton.

Glushkov automaton



Merging the Follow-equivalent states results in so-called Follow automaton:



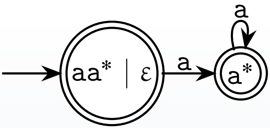
Example: Unified Glushkov construction

(C.Allauzen, M.Mohri: Math. Found. of Comp. Sci. 2006)

Aims at unifying several NFA constructions using closure+rewriting approaches over the classical Thompson automaton.

Antimirov automaton

Uses partial derivatives as states.



Example: Unified Glushkov construction

(C.Allauzen, M.Mohri: Math. Found. of Comp. Sci. 2006)

Aims at unifying several NFA constructions using closure+rewriting approaches over the classical Thompson automaton.

Unified construction

All the three automata can be constructed from `Thompson` using the following basis:

- merging language-equivalent classes (minimisation);
- merging epsilon-equivalent classes (epsilon closure);
- annotation and linearization (together with the reverse operations).

Main idea

WANTED

A converter of the regular languages representations that is:

- **Generic:** support a larger class of operations;
- **Trackable:** add more visual information and logs to help a user to track transformations;
- **Experiment friendly:** add possibility to automatically generate counterexamples by the random search (i.e. to verify statements experimentally).

(additionally: encourage students to practice in collaborative projects)

Three student groups: PYTHON, C++, LUA.

Main idea

WANTED

A converter of the regular languages representations that is:

- **Generic:** support a larger class of operations;
- **Trackable:** add more visual information and logs to help a user to track transformations;
- **Experiment friendly:** add possibility to automatically generate counterexamples by the random search (i.e. to verify statements experimentally).

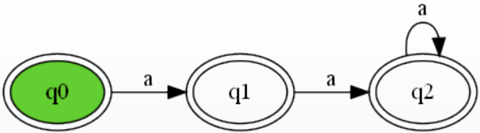
(additionally: encourage students to practice in collaborative projects)

What about the existing solutions? (still, not claiming to be exhaustive)

Python frameworks

AUTOMATA library:

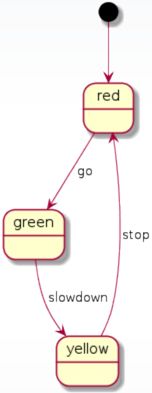
- Boolean operations on automata, equivalence and subset relations, parsing features, some basic automata generators, classical FA transformations (minimization, determinization), to-regex transformation.
- Plain GRAPHVIZ visualization.
- Not hard to implement testing feature, provided a random FA generator with the same API.



Python frameworks

AUTOMATON API:

- High-level API for designing and supporting automata structures. Supports for modifying machines and runner classes.
- Customized GRAPHVIZ and PLANTUML visualization supported.
- Nice library as a starting point.

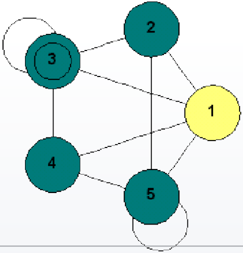


(still, the PYTHON implementation group failed; and the most successful and enthusiastic was the C++ group)

Automata library in Wolfram

MATHEMATICA notebook ALGORITHMS ON FINITE AUTOMATA:

- More algebraic-fashioned library: supports finding equivalence relations, equation systems, together with the traditional FA machinery.
- Graph visualisation is poor (was designed more than 20 years ago); but supports other discrete structures (matrices, systems) in nice LaTeX-grained format.



C++ converters

(nothing is new in the Universe)

- *Grail* project, implementing all the classical FA operations (in 1992)...
- ...and other, more modern, projects (with GRAPHVIZ support) doing almost the same that was done 30+ years ago :(

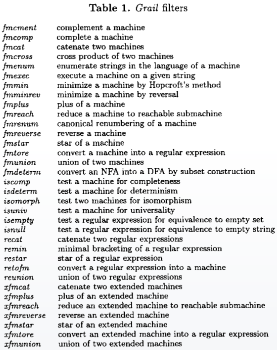


Table 1. *Grail* filters

fmcment	complement a machine
fmcomp	complete a machine
fmcat	catenate two machines
fmcross	cross product of two machines
fmenum	enumerate strings in the language of a machine
fmexec	execute a machine on a given string
fmmin	minimize a machine by Hopcroft's method
fmminrev	minimize a machine by reversal
fmplus	plus of a machine
fmreach	reduce a machine to reachable submachine
fmrenum	canonical renumbering of a machine
fmreverse	reverse a machine
fmstar	star of a machine
fmtore	convert a machine into a regular expression
fmunion	union of two machines
fmdeterm	convert an NFA into a DFA by subset construction
iscomp	test a machine for completeness
isdeterm	test a machine for determinism
isomorph	test two machines for isomorphism
isuniv	test a machine for universality
isempty	test a regular expression for equivalence to empty set
isnull	test a regular expression for equivalence to empty string
recat	catenate two regular expressions
remin	minimal bracketing of a regular expression
restar	star of a regular expression
retofm	convert a regular expression into a machine
reunion	union of two regular expressions
xfmcat	catenate two extended machines
xfmplus	plus of an extended machine
xfmreach	reduce an extended machine to reachable submachine
xfmreverse	reverse an extended machine
xfmstar	star of an extended machine
xfmtore	convert an extended machine into a regular expression
xfmunion	union of two extended machines

Overall design

- Main classes: finite automata, regular expressions, regular grammars, transformation monoids.
- *Type* support: the operations can be chained by a user, and the inconsistent chains execution is blocked by the typechecker.
- The generic `Language` class, to cache unique *language* properties (minimal DFA, syntactic monoid, pump length).
- Input: a simple program consisting of function chains and assignments.
- Output: a LaTeX(BEAMER) source file with the stepwise logs of the transformations; the graphs are processed with DOT2TEX utility and then modified in TIKZ vector graphics format.

Some supported functions

Representation changing

Thompson: Regex -> NFA
IlieYu: Regex -> NFA
Antimirov: Regex -> NFA
Arden: NFA -> Regex
Glushkov: Regex -> NFA

Representation preserving

Determinize: NFA -> DFA
RemEps: NFA -> NFA
Linearize: Regex -> Regex
Minimize: NFA -> DFA
Reverse: NFA -> NFA
Annote: NFA -> DFA
DeLinearize: NFA ->NFA
DeLinearize: Regex -> Regex
Complement: DFA -> DFA
DeAnnote: NFA -> NFA
DeAnnote: Regex -> Regex
MergeBisim: NFA -> NFA

Many-Sorted functions

PumpLength: Regex -> Int
ClassLength: DFA -> Int
States: NFA -> Int
ClassCard: DFA -> Int
Ambiguity: NFA -> Value
GlaisterShallit: NFA ->
MyhillNerode: DFA -> Int
Normalize: (Regex,Array) -> Regex

Predicates

Bisimilar: (NFA,NFA) -> t/f
Minimal: DFA -> t/f
Subset: (Regex,Regex) -> t/f
Equiv: (NFA,NFA) -> t/f
Minimal: NFA -> t/f/u
Subset: (NFA, NFA) -> t/f
Equal: (NFA,NFA) -> t/f
SemDet: NFA -> t/f
Equiv: (Regex,Regex) ->

Special forms

Test: (NFA|Regex, Regex,Int) -> IO
Verify: (Predicate,Int) -> Bool

Some project details

Input Example

```
Verify (Equal (DeLinearize (Minimize.Thompson.Linearize *)) (IlieYu *))
R1 = SemDet.RemEps.Thompson {(aa*|)}
R4 = Determinize.Reverse.Determinize.Reverse.Thompson {(aa*|)}
R5 = MergeBisim.Antimirov {(aa*|)}
R6 = RemEps.DeAnnote.Minimize.RemEps.Annote.Thompson {(aa*|)}
Test (Thompson {(a*)*}) {a*b} 10
```

Frontend

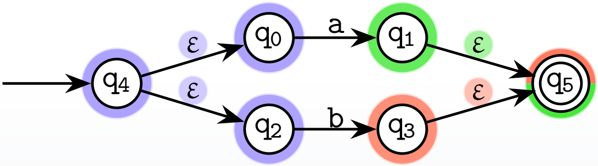
- C++ string processing is pain (project members almost gave up on this point).
- The rendering phase is done in a string processing functional language REFAL (tiny interpreter + rapid and natural string processing features development in terms of generic patterns).
- The logs replace the placeholders in the logger patterns designed in LaTeX with the meta-variables in comments.

Unified Glushkov: ε-removal paradox

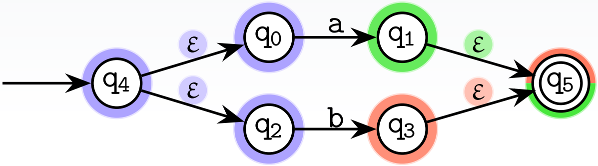
First candidate: `(Equal(RemEps.Thompson ✶)(Glushkov ✶))`

The hypothesis failed almost for all random regexes!

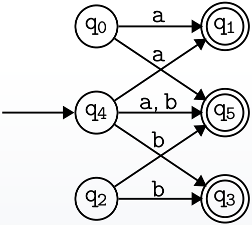
Let us consider a simple ε-NFA and compute its ε-closures.



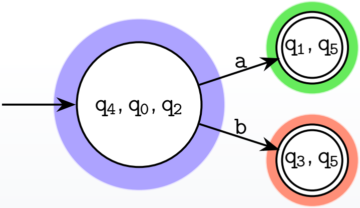
Unified Glushkov: ε-removal paradox



Classical algorithm changing only transitions in ε-closures:

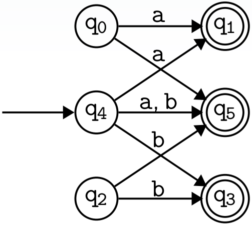


Algorithm merging ε-closures in the new classes:

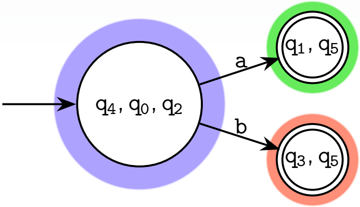


Unified Glushkov: ε-removal paradox

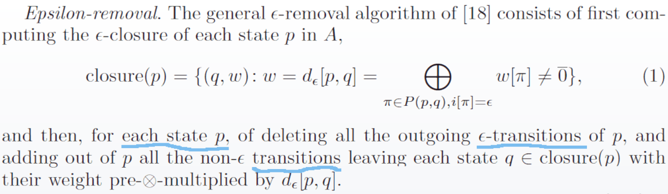
Classical algorithm changing only transitions in ε-closures:



Algorithm merging ε-closures in the new classes:



The algorithm described in the paper:



Epsilon-removal. The general ϵ-removal algorithm of [18] consists of first computing the ϵ-closure of each state p in A,

$$\text{closure}(p) = \{(q, w) \colon w = d_\epsilon[p, q] = \bigoplus_{\pi \in P(p,q), i[\pi]=\epsilon} w[\pi] \neq \overline{0}\}, \qquad (1)$$

and then, for each state p, of deleting all the outgoing ϵ-transitions of p, and adding out of p all the non-ϵ transitions leaving each state $q \in \text{closure}(p)$ with their weight pre-$\otimes$-multiplied by $d_\epsilon[p, q]$.

Paradox solution

Do not believe if they say it is elementary and everyone knows it...

- Despite the paper describes the usual epsilon-removal algorithm, it uses the closure epsilon-removal algorithm instead (which is of no means so well-known).
- The algorithm used by the authors is stronger and results in smaller automata. Using this algorithm, the hypothesis holds.

Symmetry and Brzozowski minimization

Second candidate:

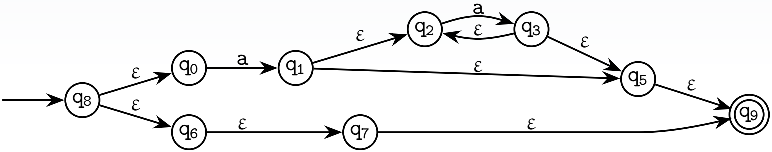
`(Equal(Determinize.Reverse.Determinize.Reverse.Thompson ✶)`
`(Minimize.Thompson ✶))`

Fails in $\approx$ 20% cases. Recall the running example: $\mathsf{aa}^* \mid \varepsilon$. The regex defines the language a^*, thus its minDFA consists of a single state.

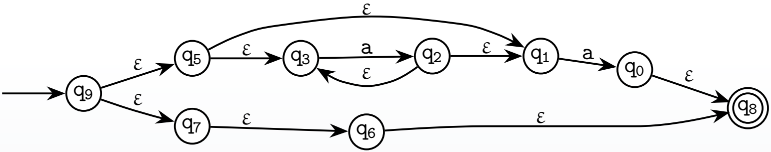
Let us track the transformations given above for it.

Reverse :: NFA → NFA

The initial automaton:



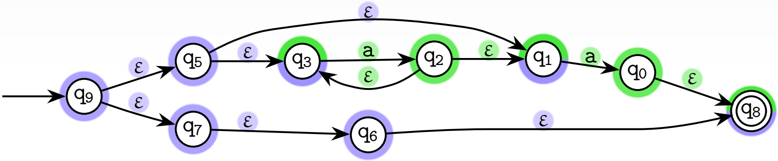
The resulting automaton:



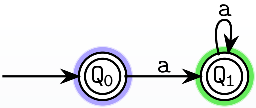
This reversal only switches arrow directions and initial and final states, because the set of these have cardinality 1.

Determinize :: NFA → DFA

The initial automaton:



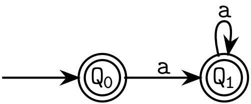
The resulting automaton:



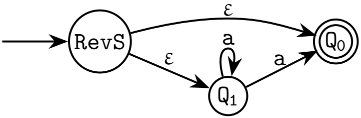
Everything goes well at this point.

Reverse :: NFA → NFA

The initial automaton:



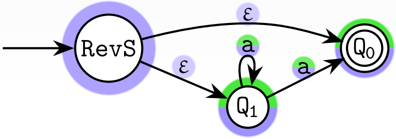
The resulting automaton:



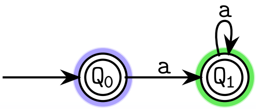
The reverse operation is forced to add the new initial state, since there are multiple final states in the given automaton.

Determinize :: NFA → DFA

The initial automaton:



The resulting automaton:



And now the subset construction cannot merge this initial state with anything else, so two states are produced instead of a single one. Additional merging by bisimilarity is needed to model the effect of considering multiple starting states.

Mystery solution

Practice imposes some constraints on theory...

- Brzozowski considered two completely symmetric structures: algebra and coalgebra. Thus, multiple initial states are allowed in his construction (and then everything works).
- Real-life FA have a single initial state, so additional merging by bisimilarity is required to achieve the 100% verification result.

Normalization magic

Third candidate

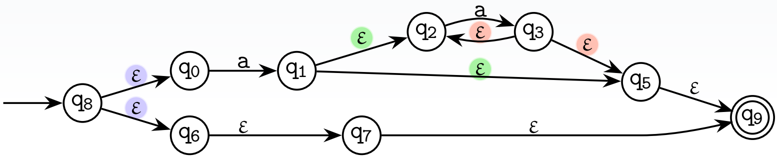
`(Equal(RemEps.DeAnnote.Minimize.RemEps.Annote.Thompson ✶)(Antimirov ✶))`

Fails in ≈20% cases... All the operations are canonical, `RemEps` is adequate.

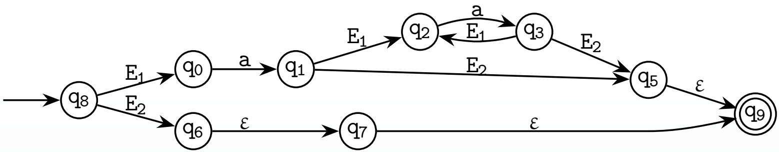
Observation: all the counterexamples contain either $(\mathtt{w}^*)^*$ or $(\varepsilon \mid \mathtt{w}(\mathtt{w})^*)$ subexpression.

Annote :: NFA → DFA

Initial automaton:



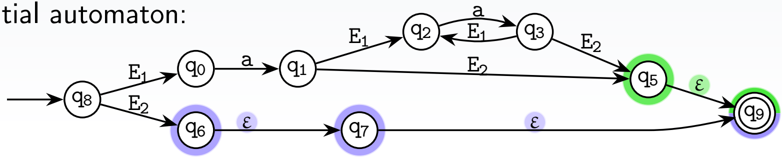
Determinized automaton:



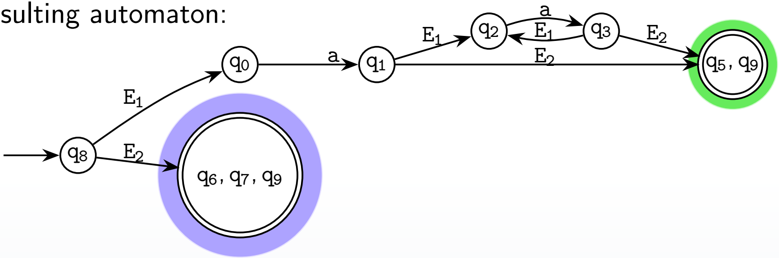
In the case of Thompson automaton, it is sufficient to enrich alphabet only by the two annoted epsilon symbols $\mathtt{E}_1$, $\mathtt{E}_2$, since there are at most two non-deterministic transitions from any state.

RemEps :: NFA → NFA

Initial automaton:



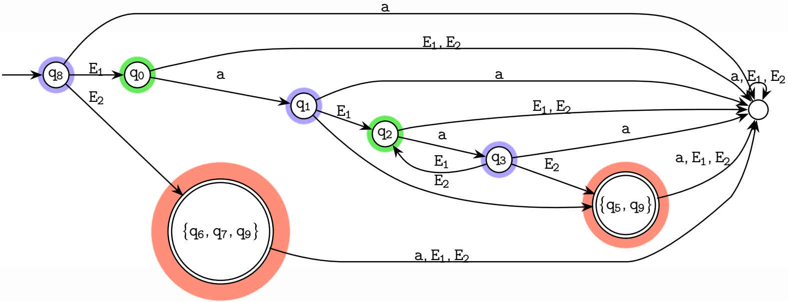
Resulting automaton:



The remaining ε-transitions are removed by the closure. It would be done by minimisation as well, but we are trying to follow the given sequence (suggested also for weighted FA) precisely.

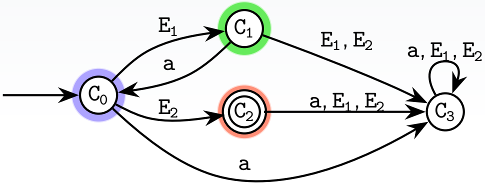
Minimize :: NFA → DFA

Initial automaton:



Alphabet is no more only a's, now it is $\{a, E_1, E_2\}$. So the trap state is added at this point to visualise the remaining possible transitions.

Resulting automaton:



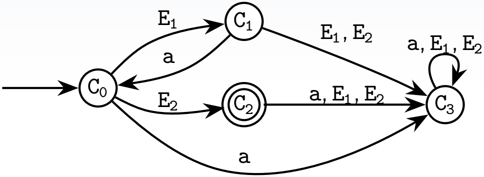
Equivalence classes:

$C_0 = \{q_8, q_1, q_3\}$; $C_1 = \{q_0, q_2\}$; $C_2 = \{\{q_6, q_7, q_9\}, \{q_5, q_9\}\}$;
$C_3 = \{\}$;

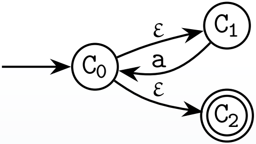
The trap state is shown also in the minimal automaton.

DeAnnote :: NFA → NFA

Initial automaton:



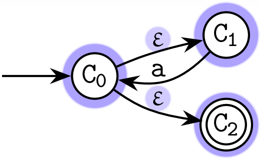
Resulting automaton:



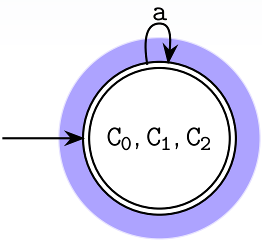
Deannotation collapses alphabet to $\{a\}$.

`RemEps :: NFA → NFA`

Initial automaton:



Resulting automaton:

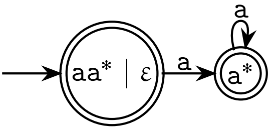


The given automaton is minimal, so it is expected that the Antimirov automaton has the single state either.

Antimirov :: Regex → NFA

Regular expression: $\mathtt{aa}^* \mid \varepsilon$

Resulting automaton:



Partial derivatives:

$\delta_{\mathtt{a}}(\mathtt{aa}^* \mid \varepsilon) \ = \ \mathtt{a}^*$ $\qquad\qquad$ $\delta_{\mathtt{a}}(\mathtt{a}^*) \ = \ \mathtt{a}^*$

No mistake: there are two states, not the single one…

Paradox solution

Always check the data set first...

- The initial regular expressions are normalized (the author mentions only distributivity, but it seems they used the normal form, e.g. simplifying $(\mathtt{w}^*)^*$ to $\mathtt{w}^*$).
- Slightly desorienting assumption, since the main advantage of the Antimirov derivatives (versus Brzozowski's) is their robustness without simplifications.

WIWtK starting a collaborative student project

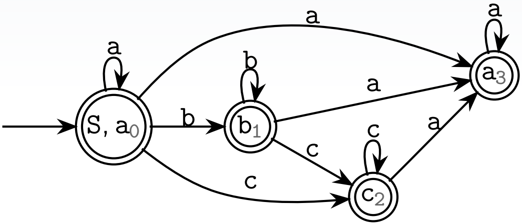
- Give preferences to the languages everyone knows not quite well.
- Choose a project leader basing on stability, not on enthusiasm.
- Testing is crucial: make the testing engine first (and do not rely on the code reviews too much).
- Force the students to release the project before the exam!

That's all!

Thank you for your attention!

And infinitely many thanks to the students who made it possible: A. Delman, D. Knyazihin, A. Terentyeva, K. Shevchenko, M. Teriykha, A. Ilyin, A. Chibizova, and V. Lysenko for the slave labor of doing the log templates.

Weaker one beats stronger one: Glaister–Shallit paradox

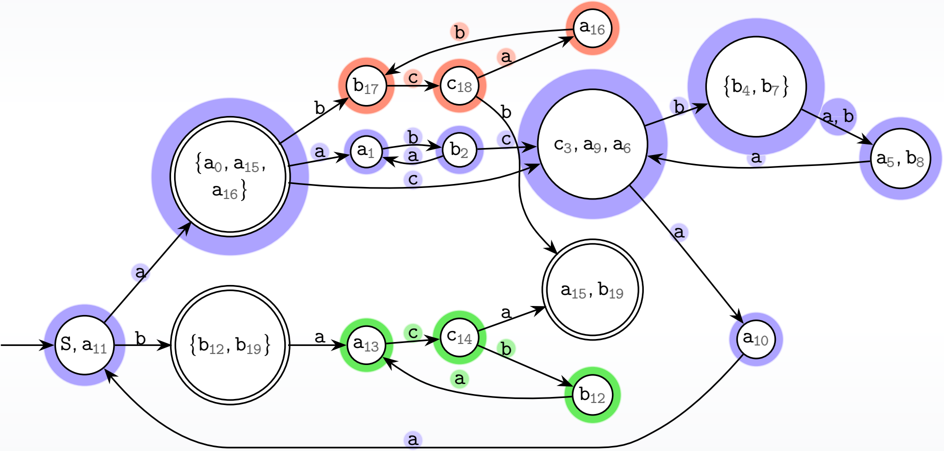


Equivalence classes and distinguishing suffixes:

	ab	ε	b	c
ε	1	1	1	1
ba	0	1	0	0
b	0	1	1	1
c	0	1	0	1

Lower bound on the states in NFA: 4

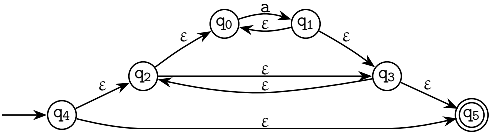
Orbits and ambiguity



Modelling ReDoS

Regexp for testing: `a*b`.

The parse automaton:



	Length	Parse Time	Result
1	1	0.000000	`false`
2	51	0.015000	`false`
3	101	0.024000	`false`
4	151	0.042000	`false`
5	201	0.048000	`false`
6	251	0.060000	`false`
7	301	0.082000	`false`
8	351	0.089000	`false`
9	401	0.103000	`false`
10	451	0.103000	`false`
11	501	0.116000	`false`
12	551	0.122000	`false`
13	601	0.141000	`false`