Rational Languages

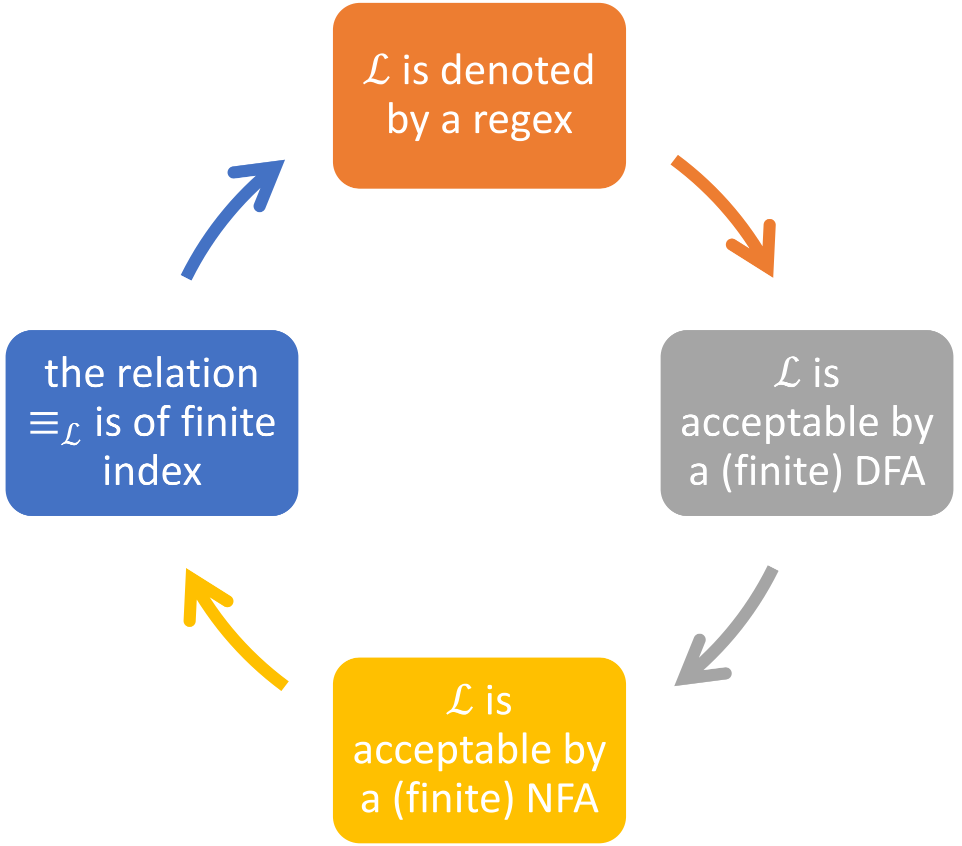
reminder

Equivalent characterization

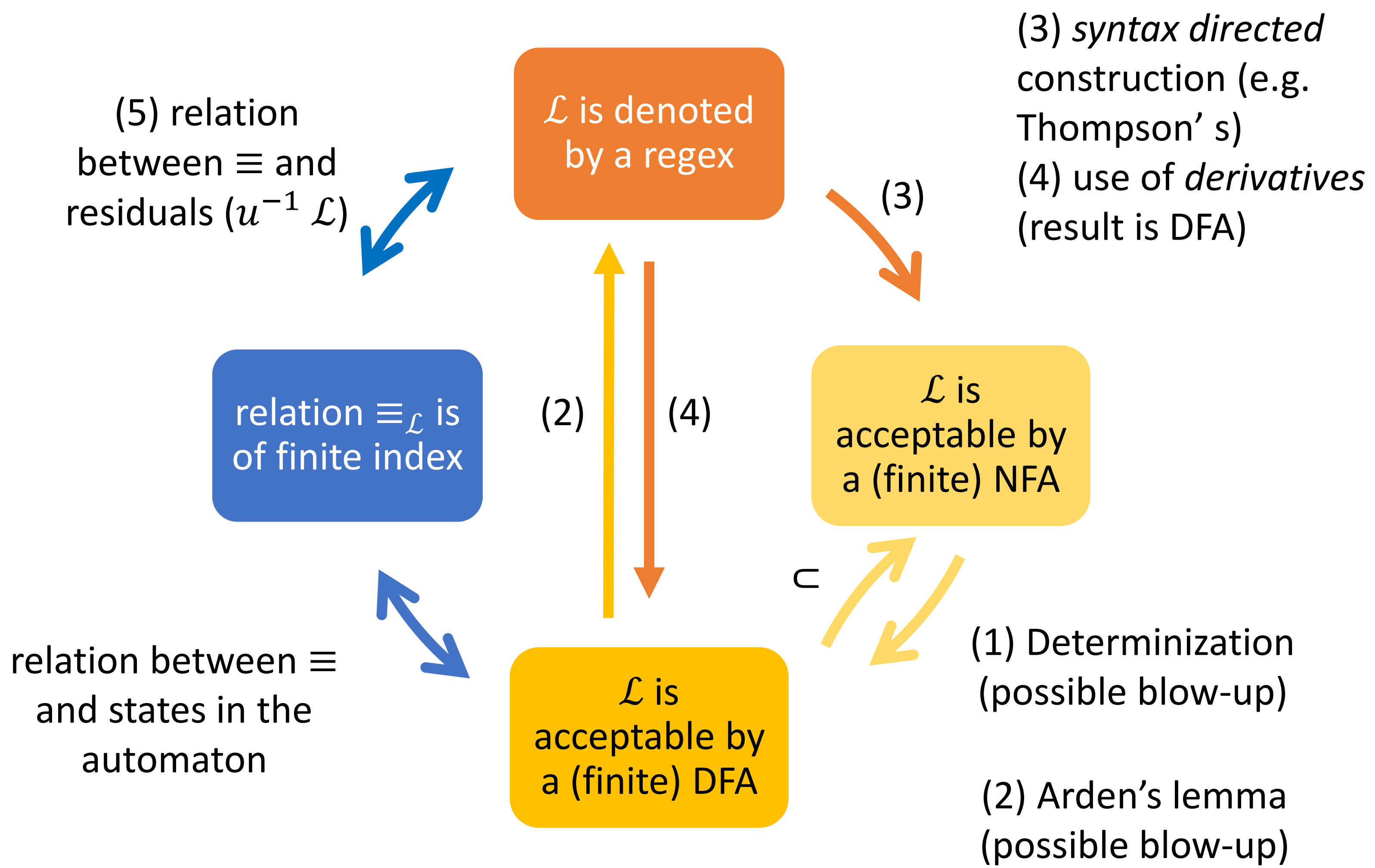
For a language $\mathcal{L}$, the following conditions are equivalent:

1. $\mathcal{L}$ is denoted by a regex
2. $\mathcal{L}$ is acceptable by a (finite) DFA
3. $\mathcal{L}$ is acceptable by a (finite) NFA
4. the relation $\equiv_{\mathcal{L}}$ is of finite index
5. $\mathcal{L}$ is generated by a right-linear grammar

Equivalent characterizations



Equivalent characterizations



Application of Myhill-Nerode th.

Are those languages rational ?

1. $\{ a^n b^n \mid n \geq 0 \}$
2. $\{ a^n b^n \mid n < 1\,000\,000 \}$
3. $\{ a^n b^m \mid n, m \geq 0 \}$
4. $\{ a^n b^m \mid n + m \equiv 0 \; [5] \}$
5. Dyck languages, containing (well-parenthesized) words of the form $\epsilon, a\,\bar{a}, a\,\bar{a}\,a\,\bar{a}, a\,a\,\bar{a}\,\bar{a}$, …

   if $u, v \in \mathcal{L}$ then $a\,u\,\bar{a} \in \mathcal{L}$ et $u\,v \in \mathcal{L}$

   imagine that $a = '('$ and $\bar{a} = ')'$

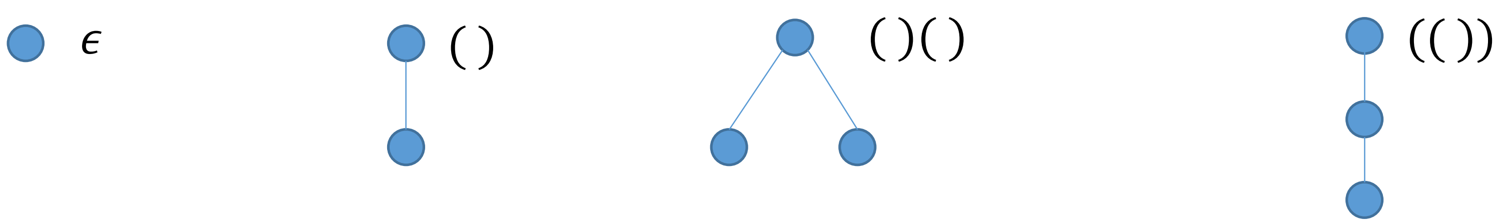
Dyck languages

Dyck languages $\equiv$ containing (well-parenthesized) words of the form $\epsilon, a\,\bar{a}, a\,\bar{a}\,a\,\bar{a}, a\,a\,\bar{a}\,\bar{a}, \ldots$

Imagine that $a = '(' $ and $\bar{a} = ')' $

words of the form ϵ, ( ), ( )( ), (( )), …

Equivalently: $u, v \in \mathcal{L} \Rightarrow a\,u\,\bar{a} \in \mathcal{L}$ and $u\,v \in \mathcal{L}$

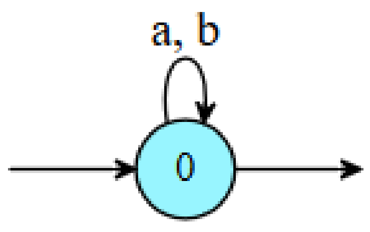


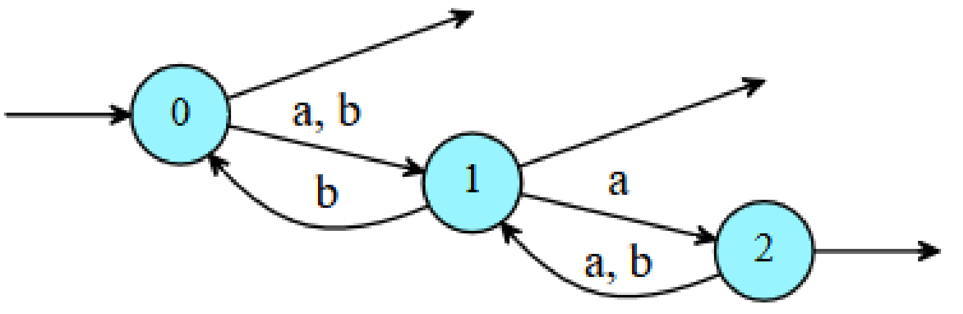
Automata

Equivalence and minimization

Minimality and DFA

- There can be many equivalent DFA





- A sureway method for minimality is to use $\equiv_L$
- We can also use *partition refining* methods to find equivalent states in a DFA

Minimization ≡ Equivalence

- Minimal DFA ⇒ we have a *canonical DFA* for each rational language (up-to naming of the states)

- Hence, to test if two NFA $\mathcal{A}$ and $\mathcal{B}$ are equivalent (they accept the same language), we can just test:

$$\min\big(\det(\mathcal{A})\big) =^{?} \min\big(\det(\mathcal{B})\big)$$

- also work with regex !

Minimizing a DFA

- We want to find equivalent states, $\equiv$

$p \equiv q$ meaning $\mathcal{A}(p) = \mathcal{A}(q)$

- We have two necessary conditions
  1. If $p \equiv q$ and p a final state ($\epsilon \in \mathcal{A}(p)$) then q final
  2. if $p \equiv q$ then $\delta(p, a) \equiv \delta(q, a)$ (same residuals)

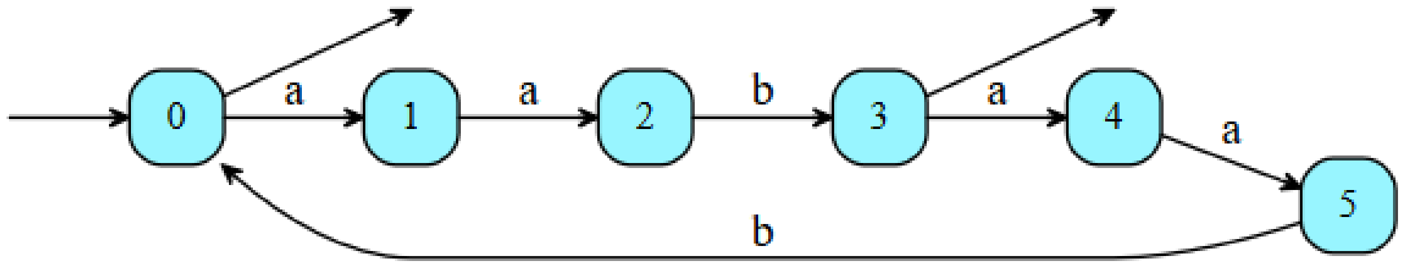
Minimizing a DFA: Idea

We try to compute the equivalence classes for $\equiv$

1. start: assume all states are equivalent
2. repeat: split a class when you find two states with different transitions (up-to $\equiv$)
3. end: until we rich a fix point

$\Rightarrow$ all the states in the same partition are *non distinguishable*

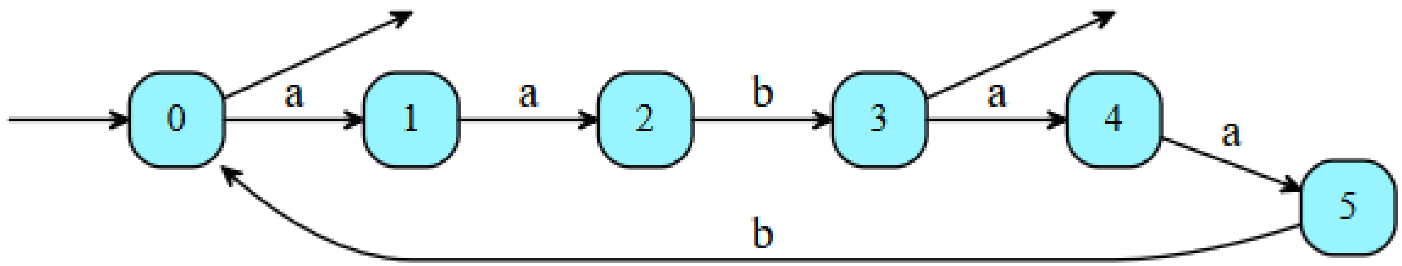
Minimization: example



This automaton is deterministic

states: {0, 1, 2, 3, 4, 5}

Minimization: example

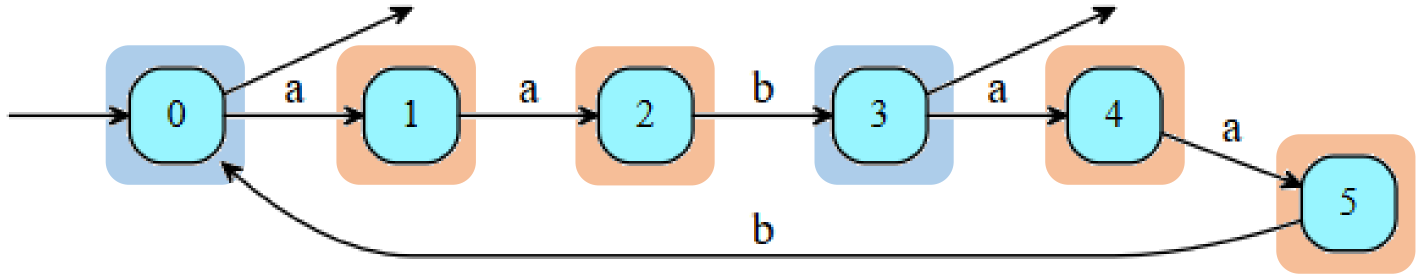


states : {0, 1, 2, 3, 4, 5}

we have $\{0, 3\}$ finals and $\{1, 2, 4, 5\}$ not finals

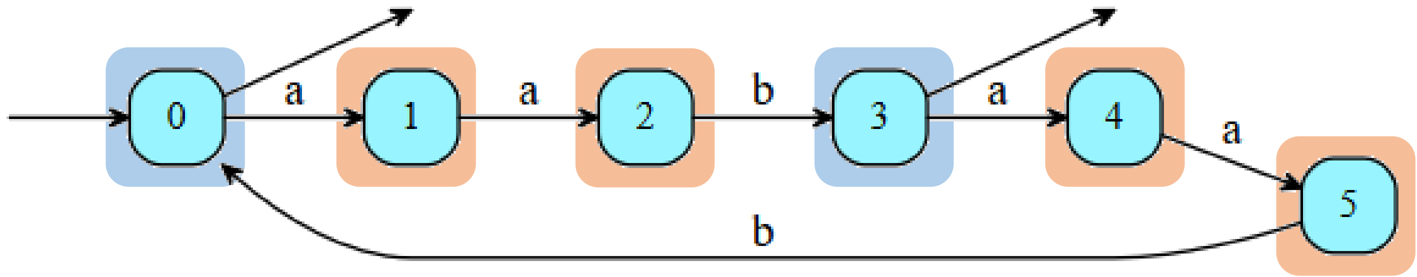
⇒ we need to split the initial partition in two.

Minimization: example



states: {0, 3} {1, 2, 4, 5} (2 partitions)

Minimization: example

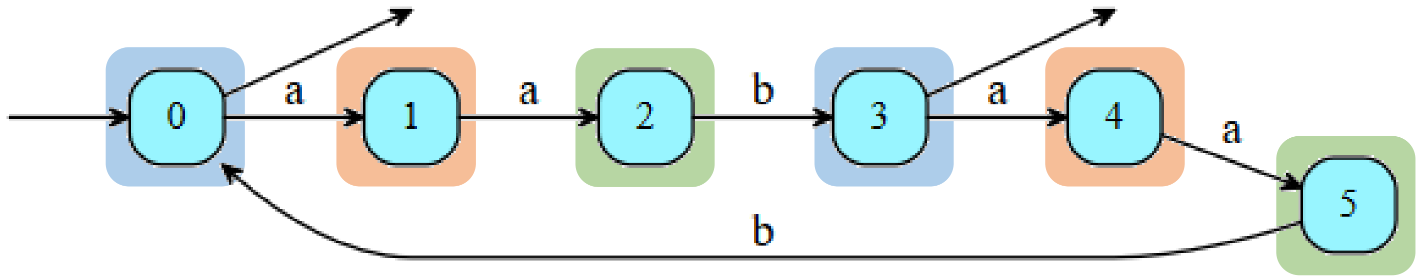


states: $\{0, 3\}$ $\{1, 2, 4, 5\}$

$0 \xrightarrow{a}$ ■ and $1 \xrightarrow{a}$ ■, same for $\xrightarrow{b}$: OK

$2 \xrightarrow{b}$ ■ and $5 \xrightarrow{b}$ ■, but $1, 4$ block : need to split ■

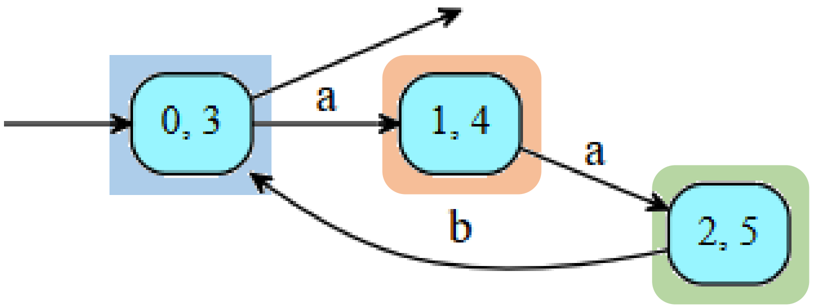
Minimization: example



states: {0, 3} {1, 4} {2, 5} (3 partitions)

we have no more partitions to split

Minimization: example



states: {0, 3} {1, 4} {2, 5}

we obtain the minimal DFA by *fusing* states in the same partition together

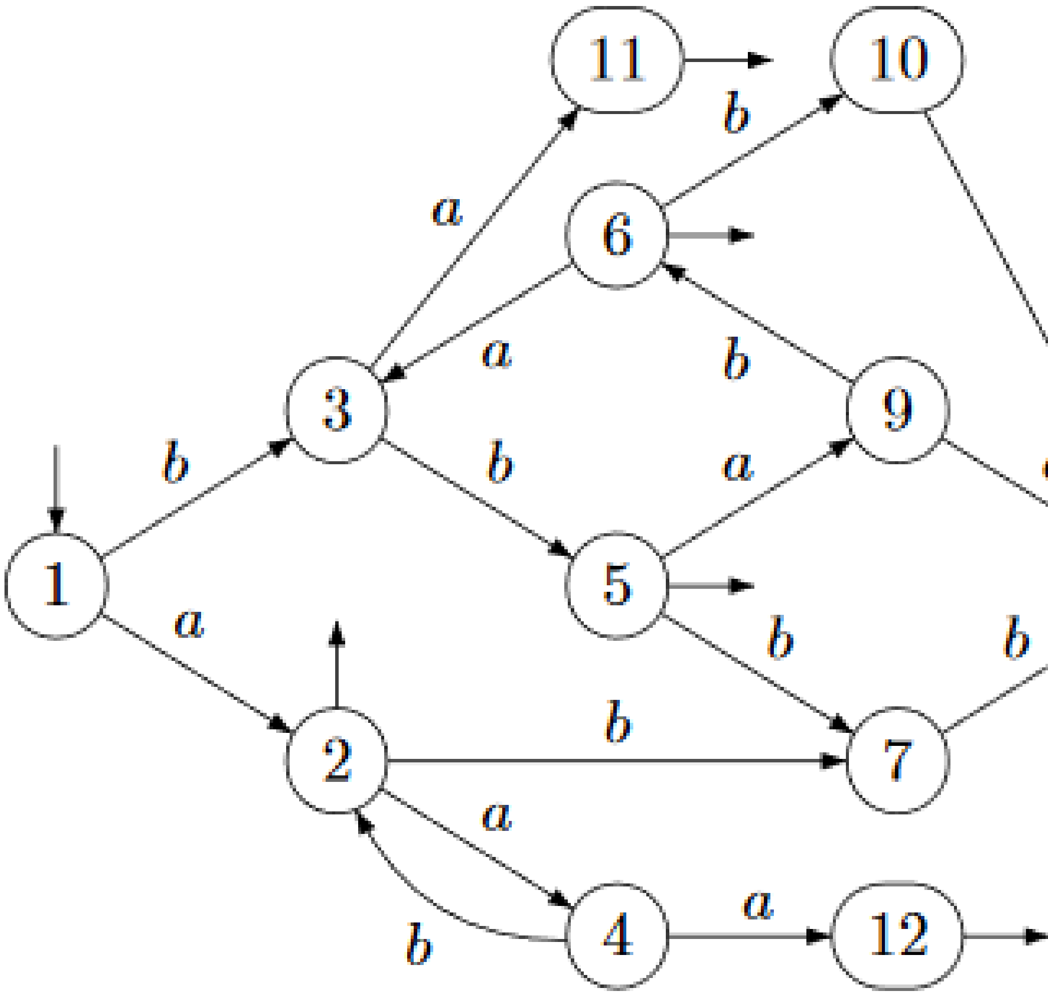
Minimization: Hopcroft

- This algorithm is due to Hopcroft (1971)
  - requires a complete automaton
- We start from the *coarsest* relation $\{Q\}$ and stop when there are no more partition to split $\Rightarrow$ so we split at most n times ($n = |Q|$)

  this uses a *coinductive proof principle*.
- complexity is $n\ |\Sigma| \log n$

- There is another $\sim$ algorithm due to Moore
- In each case, it requires to work on DFA

Example



Example: ≡ between regexes

- $R_1 = (a + b)^\star$
- $R_2 = (a^\star + b^\star)^\star$
- $R_3 = (a^\star + b.a^\star)^\star$
- $R_4 = (a^\star + b)^\star.a^\star$

$D(R(D(R(\mathcal{A}))))$

Minimization: Brzozowski

- Start from a DFA $\mathcal{A}$ that is complete
- Reverse $\mathcal{A}$ (take its mirror image)
  - switch the direction of the "arcs"
  - the result may not be deterministic
- Determize the result
  - the result is a minimal DFA for $\widetilde{\mathcal{L}(\mathcal{A})}$
- Reverse the automata and determinize again

$\Rightarrow$ produces a minimal automata equivalent to $\mathcal{A}$

Equivalence between NFA

- Minimization is not a solution: too complex

- Finding a solution without determinization is complex, but some recent advances: Raskin (2006), Pous (2013)

Automata

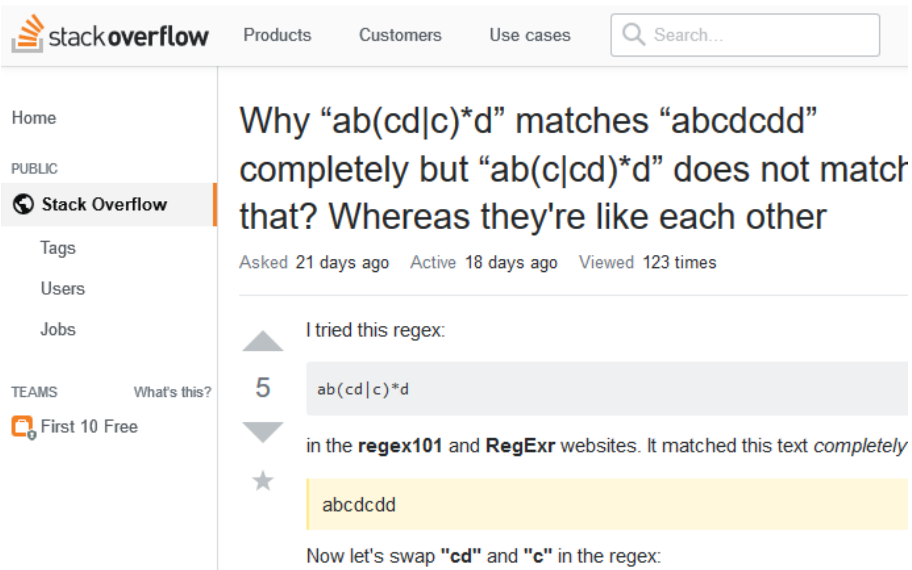
Conclusion

What you should have learned

- There are different views to the same problem: *operational, declarative, denotational*
- We can better understand a problem when we have different ways to look at it
- C. S. problems can (sometimes) be answered using simple abstractions (discrete maths) and lead to interesting questions about complexity

  $\Rightarrow$ what are the limits ?
- It has applications in practice, …

... or not



Why does $ab(cd + c)^{\star}d$ matches $a\ b\ c\ d\ c\ d\ d$ completely

But $ab(c + cd)^{\star}d$ matches $a\ b\ c\ d\ c\ d\ d$

This is a problem/question about ambiguity

Languages and complexity

Some complexity theory

Languages

- (Formal) Languages are subsets of $\Sigma^\star$
  - *this is a very general definition*
  - we have concentrated on *regular languages*, but not all languages are regular
- Regular/Rational languages $\equiv$ a class of languages with good properties
  - multiple (unrelated!) ways to define regularity
  - closed by various operations: $+$, $\times$, $\div$, $\cap$, $\star$, $\overline{\cdot}$, $\tilde{\cdot}$, ...
  - many decidable properties
  - a "meaningful" complexity class, REG

Languages: questions

- What are some examples of properties and problems ?
- What do you mean by complexity class of a set of languages
- Can we define classes that are simpler, or more complex than REG ?

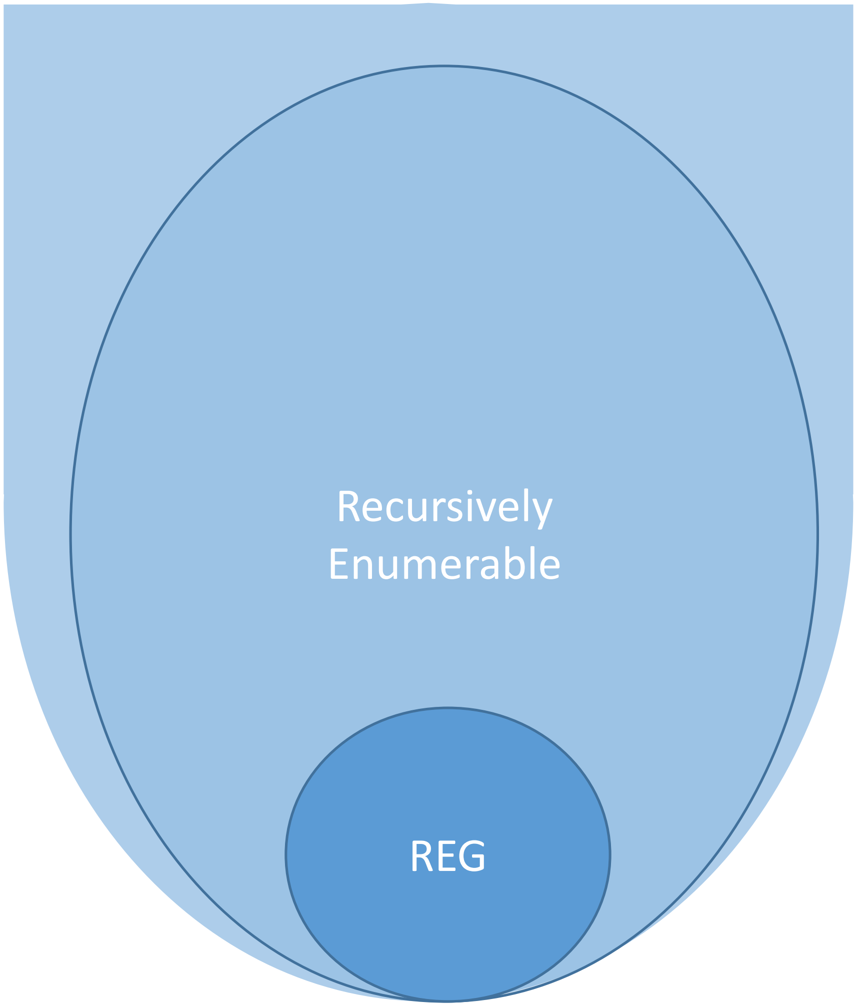
Languages: hierarchy

$\mathcal{P}(\Sigma^\star)$

in between ?

REG

even simpler ?



Languages: problems

- We have often tried to prove whether two regex / FSA "are the same"
- A more basic question if *containment* : $\mathcal{L}_A \subseteq^? \mathcal{L}_B$?

Languages: problems

There are simpler questions:

membership: is a given word u in $\mathcal{L}_A$?

emptiness: is it true that $\mathcal{L}_A = \emptyset$?

does $\mathcal{L}_A$ contains ∞^{ly} many words ?

universality: does $\mathcal{L}_A = \Sigma^\star$?

- **Good news**: almost all problems of interest are decidable in REG

REG: complexity

Membership problem: $u \in \mathcal{L}_A$?

```
func member(u []byte) bool {
    st := q0
    for i = 0; i < len(u); i++ {
        st = delta(st, u[i])
    }
    return isfinal(st)
}
```

We can test membership using a “machine” that has one register (storing the current state) and a table for storing δ and F

REG: complexity of membership

Assume $\mathcal{A}$ is a DFA with n states.

We can test membership using a "machine" that has one register (storing the current state) and a table for storing δ and F

$\log n$ bits of writable data

non-writable data

Hence problem is in DLOGSPACE (also called L)

REG: complexity of membership

Membership for DFA is in DLOGSPACE-c

Membership for NFA is in NLOGSPACE (L $=^?$ NL)
$\equiv$ membership for regex

Same complexity than deciding whether there exists a path between given vertices in a directed graph

Considered feasible

Complexity

- Emptiness: $\mathcal{L} =^? \emptyset$
  is in NLOGSPACE-c for both NFA and DFA

- Equivalence: $\mathcal{L}_1 =^? \mathcal{L}_2$
  is PSPACE-c for DFA, NFA and regex

Suspected infeasible

Languages hierarchy: star-free

star free languages ≡ corresponds to generalized regex ($\bar{e}$) without $\star$

- example: $\overline{\overline{\mathbf{0}}\ a\ a\ \overline{\mathbf{0}}}$
- count.-ex.: $(aa)^{\star}$