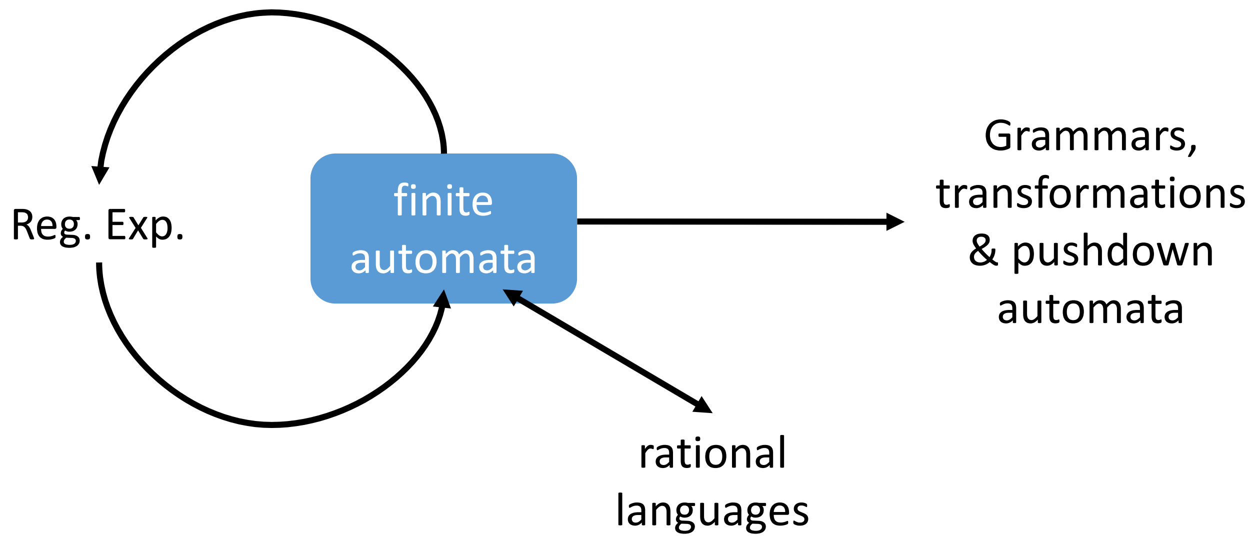
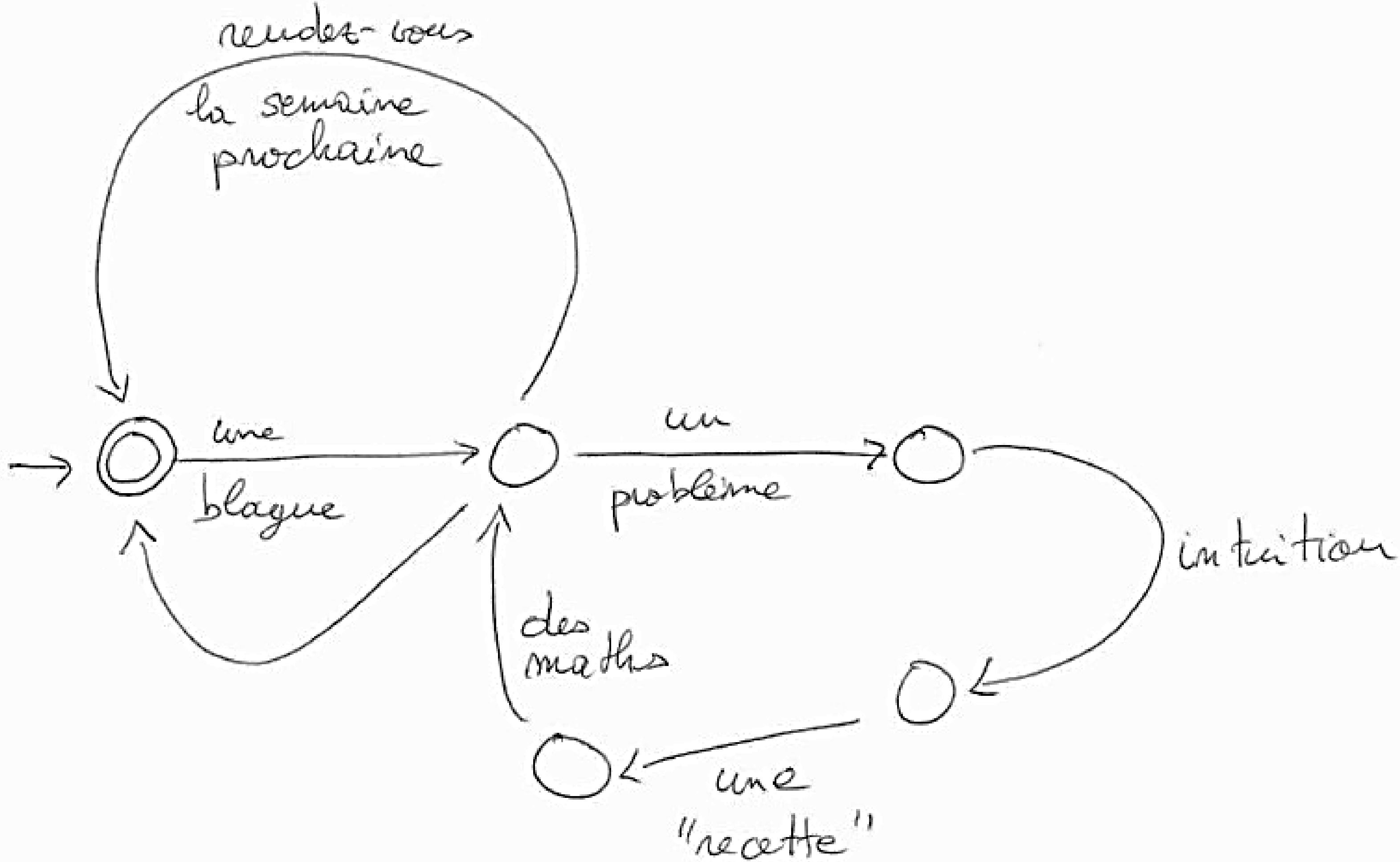
Automates et Langages



http://homepages.laas.fr/dalzilio/courses/ag/

2019—2020



rendez-vous
la semaine
prochaine
une
blague
un
problème
intuition
une
"recette"
des
maths

méta-blague

Automata

quick refresher + an application to string searching

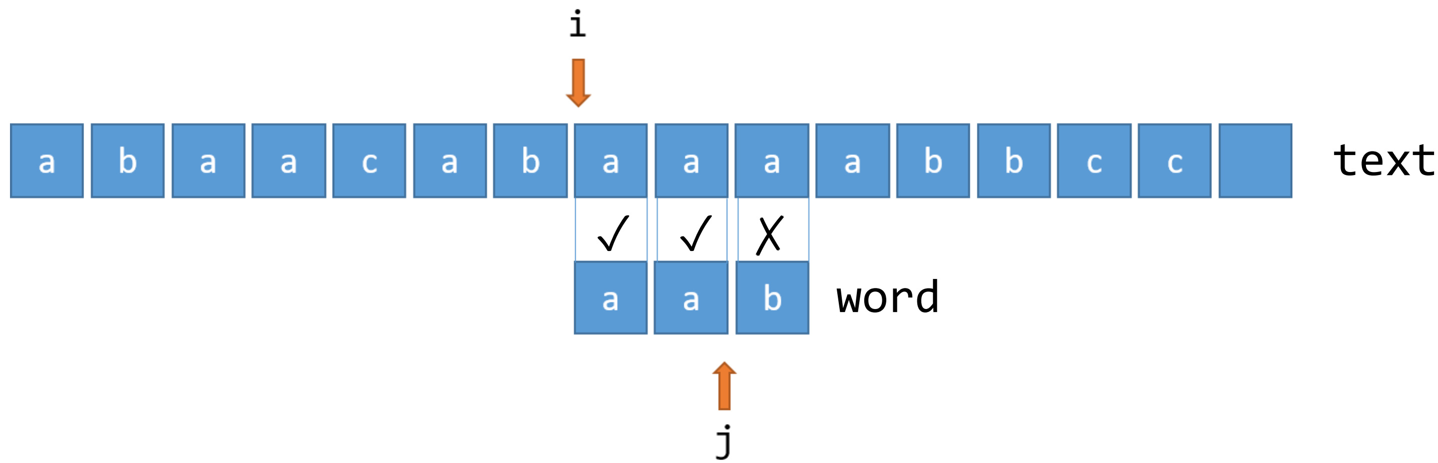
String searching

Problem: find the occurrences of a given word (the needle) inside a, usually much larger, text (the haystack)

- a simple problem (linear complexity)
- an important problem, with many applications: log analysis ; security ; DNA and protein sequences, string mining

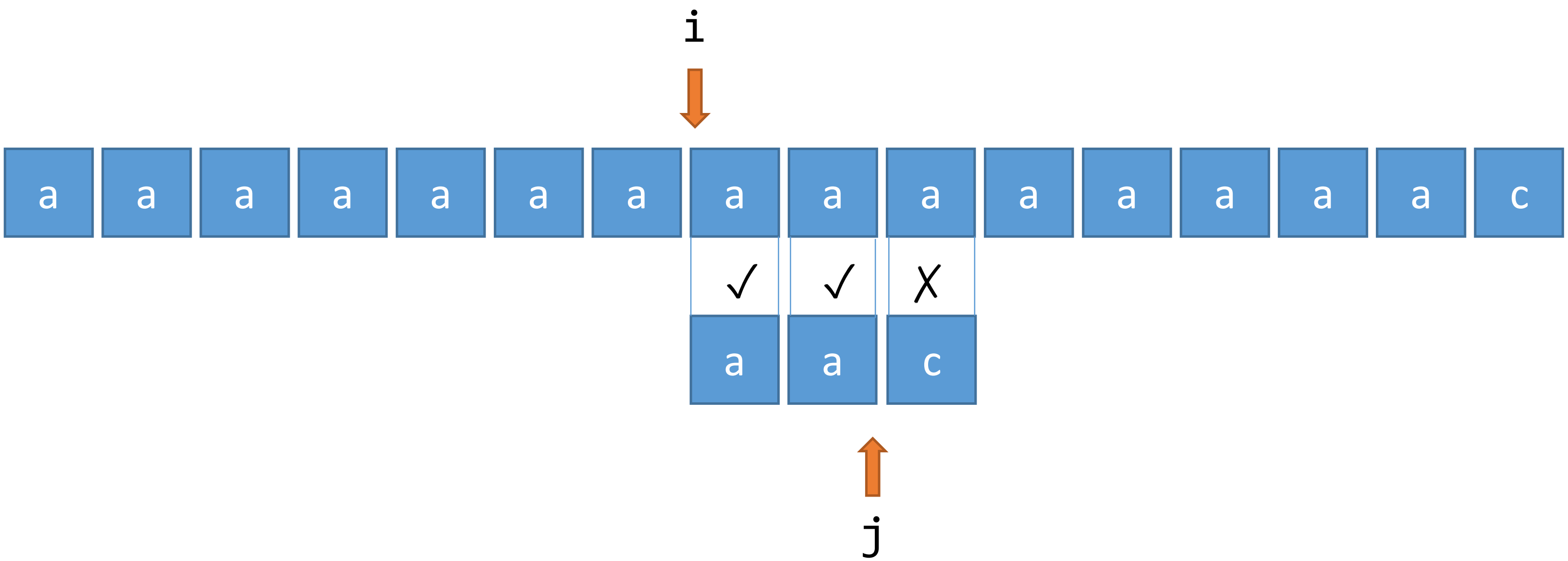
An instance of a more general problem ⇒ pattern matching

String searching: $O(n)$




```
func simple(word, text []byte) int {
    for i = 0; i < len(text); i++ {
        for j = 0; j < len(word); j++ {
            if i+j == len(text) { return -1 }
            if text[i+j] != pattern[j] { break }
     }
    return i }
}
```

String searching: $k \times n$

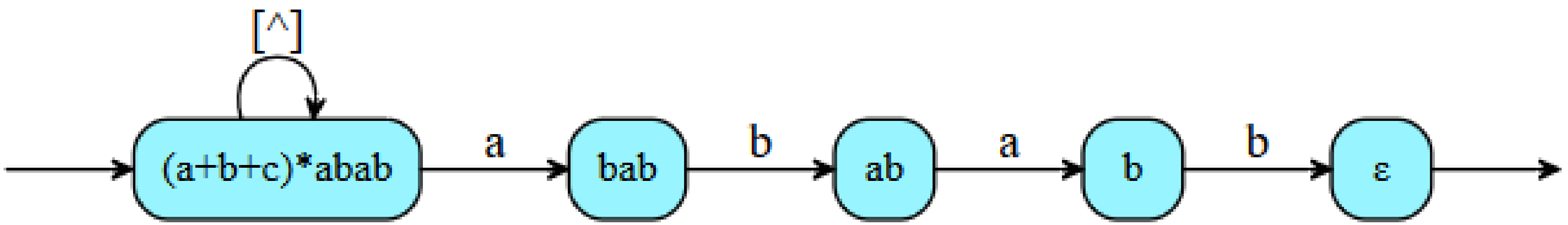


$\Sigma = \{ A, C, G, T \}$

We are in a case where the constants are important; two $O(n)$ algorithms are not equivalent.
Can we do it in time n ?

Idea: use a DFA

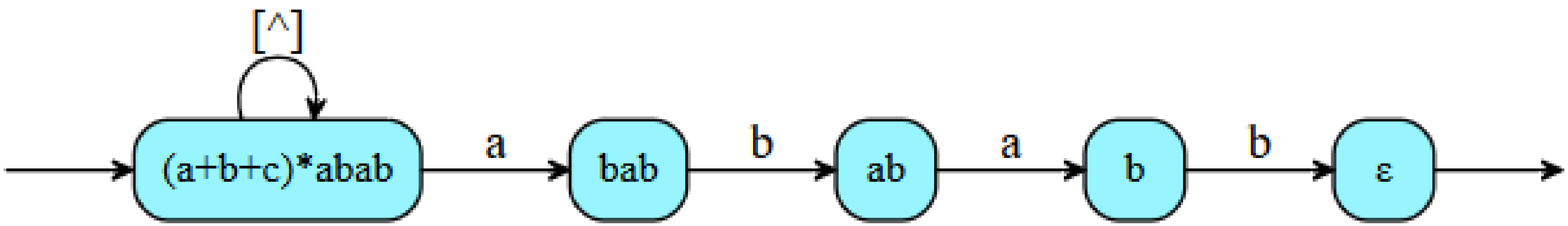
- Imagine we want to match the string $a\ b\ a\ b$
- We want to match every possible occurrences



here [^] means \Sigma (i.e. $\Sigma \setminus \emptyset$)

Idea: use a DFA

- Imagine we want to match the string $a\ b\ a\ b$
- We want to match every possible occurrences

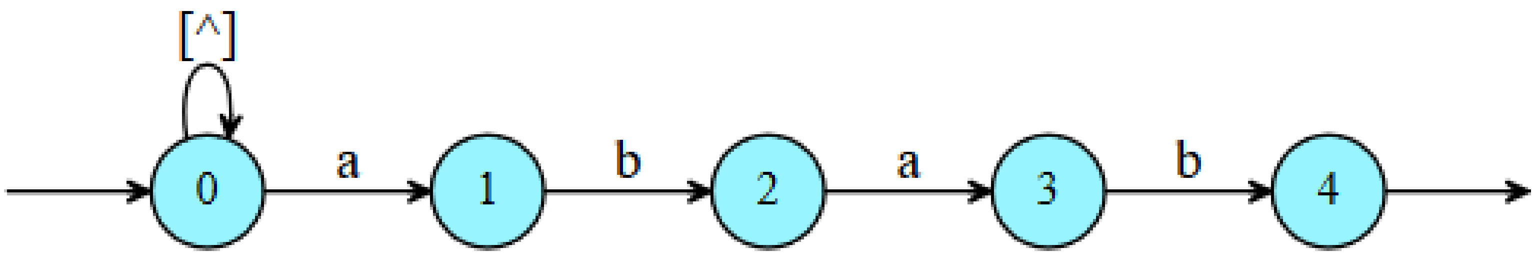


Drawbacks:

- are we sure to match all occurrences? Imagine the haystack $a\ b\ a\ b\ a\ b$ (that has 2 occurrences)
- non-deterministic ⇒ we need to use "4 registers" (not optimal !?) or to use backtracking (not practical)

here [^] means Σ (i.e. $\Sigma \setminus \emptyset$)

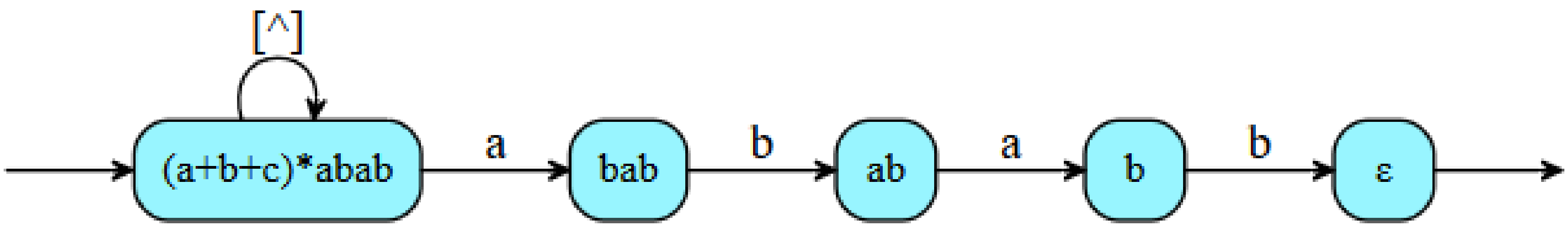
Déterminization

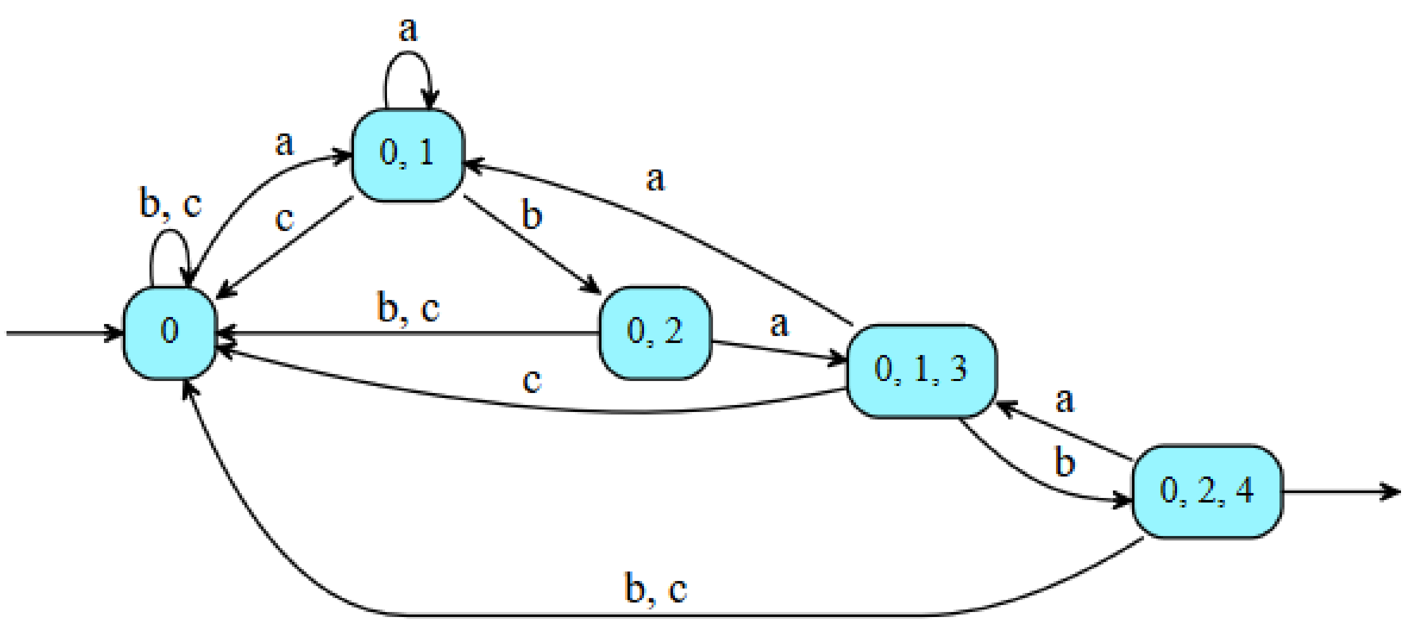


$\det(\mathcal{A})$ est un tuple $(Q',\ \Sigma,\ \delta',\ q'_I, F')$ où :

- $Q' = 2^Q = \mathcal{P}(Q)$ (powerset)
- $\Sigma =$ même alphabet que $\mathcal{A}$
- $q'_I = \epsilon F(I)$
- $F' = \{\, S \mid S \cap F \neq \emptyset \,\}$
- $\delta' \in (Q' \times \Sigma) \to Q'$: fonction de transition

Idea: determinization

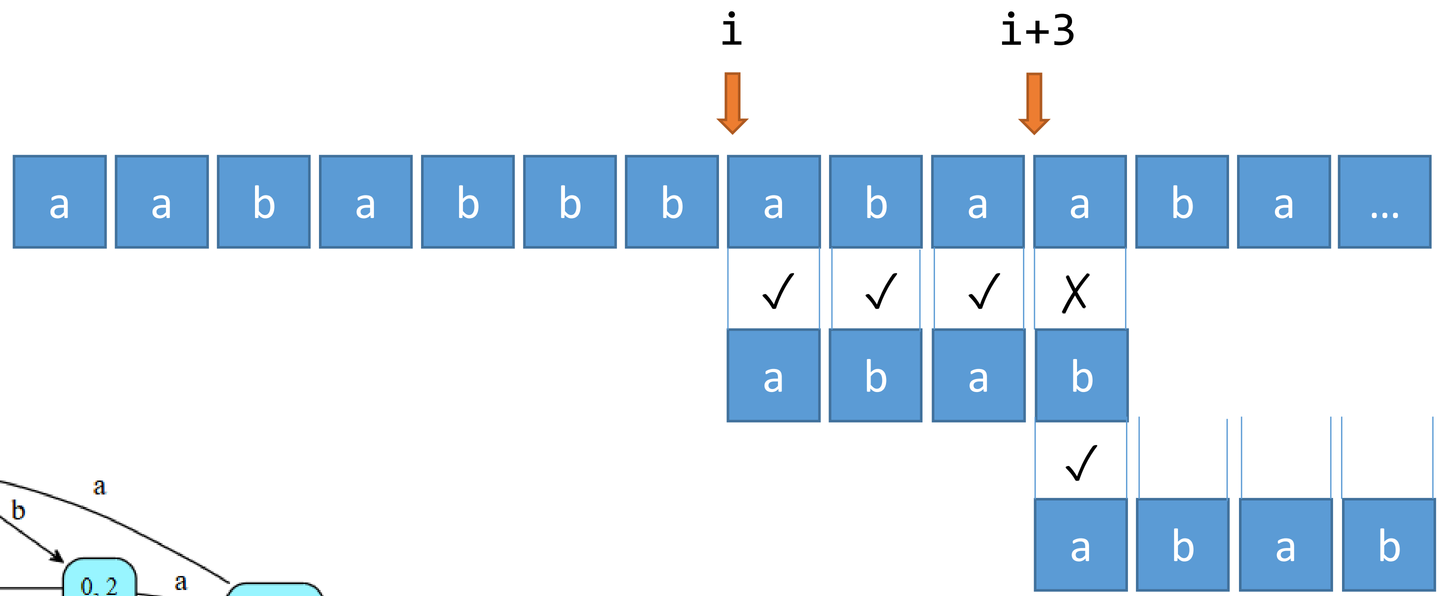


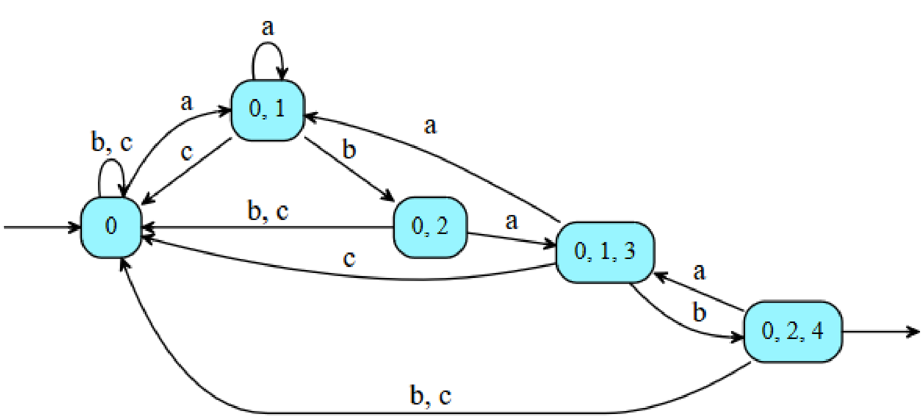


preprocessing time in $k.|\Sigma|$

String matching: an observation

when we “fail” to match the next symbol, we do not always need to start over





String matching: an observation

- When matching fails at position j, it is enough to remember the longest prefix of word that is a suffix of word[0..j]

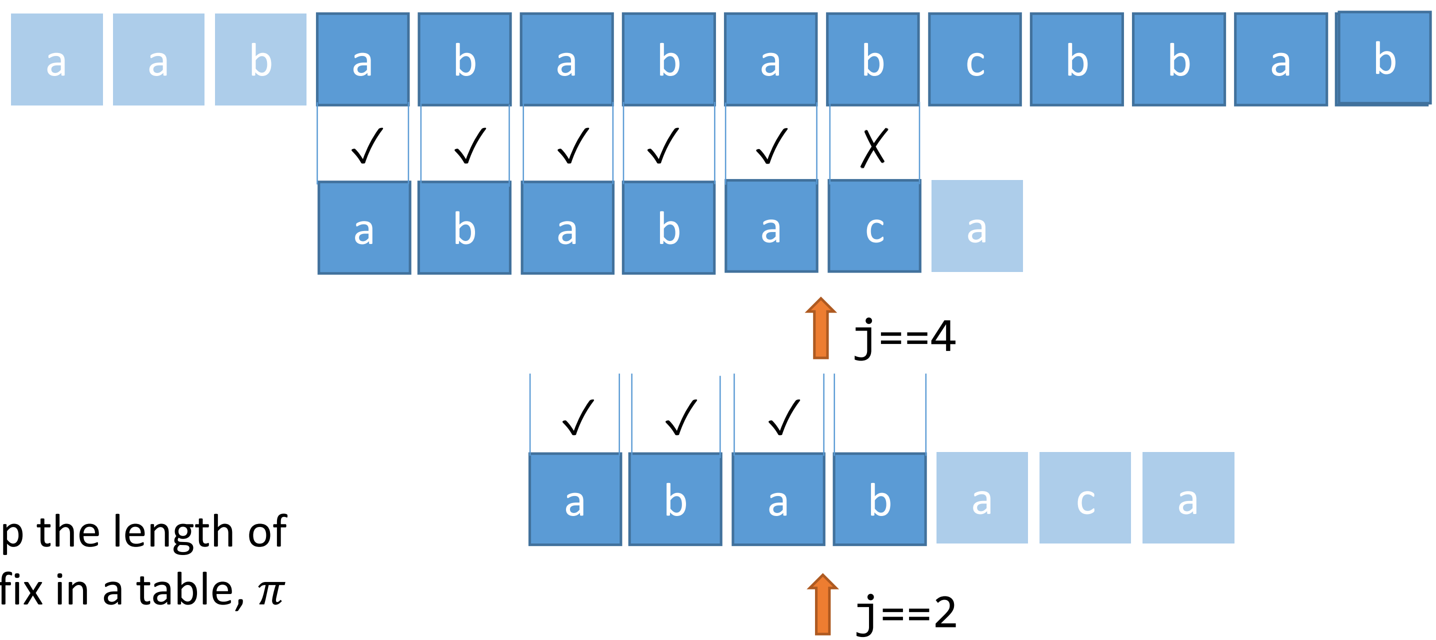
$b\ c\ a$ suffix

$$\underbrace{a\ b\ a\ b\ a}\ b\ a\ \overbrace{b\ c\ a}$$

prefix $a\ b\ a\ b\ a$

String matching: an observation

- When matching fails at position *j*, it is enough to remember the longest prefix of word that is a suffix of word[0..j]



We keep the length of the prefix in a table, π

String matching: an observation

- When matching fails at position j, it is enough to remember the longest prefix of word that is a suffix of word[0..j]

word	$a\ b\ a\ b\ a\ b\ a\ b\ c\ a$
π	$0\ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 0\ 1$

if we have already matched $a\ b\ a\ b\ a\ b$ and fail at matching the next a (when j==6), the next "possibly useful" shift is j==4

String matching: an observation

- When matching fails at position j, it is enough to remember the longest prefix of word that is a suffix of word[0..j]

word	$a\ b\ a\ b\ a\ b\ a\ b\ c\ a$
π	$0\ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 0\ 1$

$a\ b\ a\ b\ a\ b\ d$

$a\ b\ a\ b\ a \qquad \pi(6) = 4$

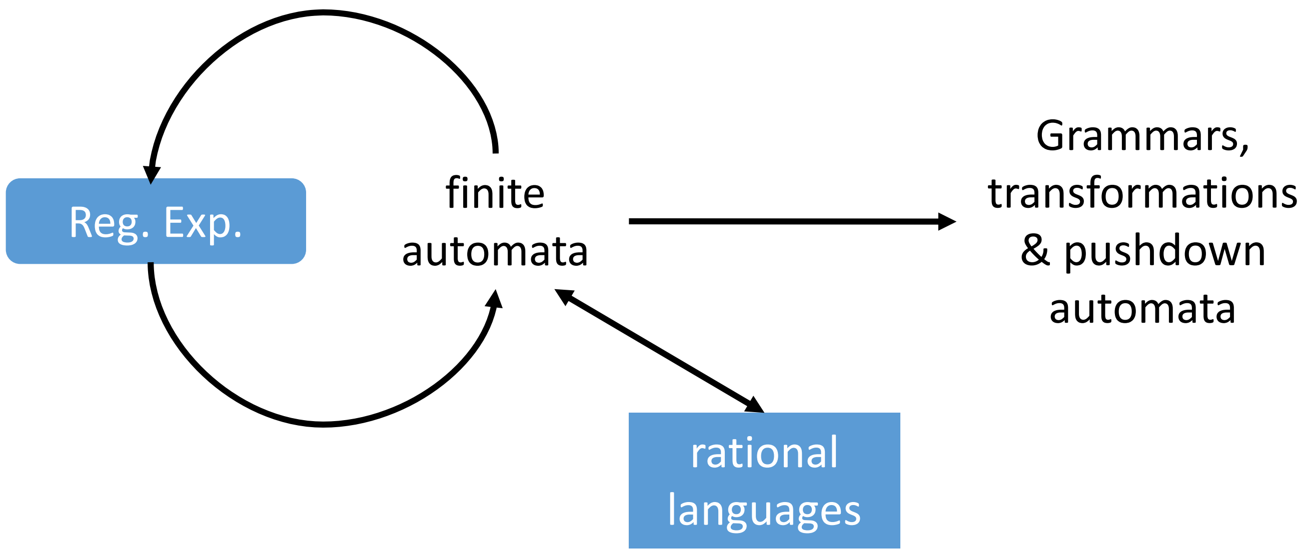
$a\ b\ a \qquad \pi(4) = 2$

$a \qquad \pi(2) = 0$

String matching

- This approach ≡ *Knuth-Morris-Pratt* algorithm (1970)
- Other algorithms use hashing ⇒ Karp
  - good solution for approx./randomized pattern matching
- A state of the art algorithm is Boyer-Moore (1977)
  - scans the needle from right → left
  - part of the C++ STL, still used in grep, …
- Aho–Corasick algorithm
  - search multiple strings: grep -F
  - “[…] constructs a finite-state machine that resembles a trie with additional links between internal nodes.”

Regular Expressions and Rational Languages



http://homepages.laas.fr/dalzilio/courses/ag/

2019—2020

Languages

quick refresher on words and languages

Monoid $\Sigma^\star$

- a word w is a sequence of symbols in Σ
- we can concatenate two words $w_1 . w_2$
- $|w|$ is the length of w
- ϵ is the empty sequence, $|\epsilon| = 0$

  ϵ is the identity element, $w . \epsilon = \epsilon . w = w$

- $(\Sigma^\star, \ . \ , \epsilon)$ is the *free monoid* of Σ
- * is often referred to as the Kleene star

Properties of $\Sigma^\star$ (ex.)

When we have a total order between elements of Σ, (say $a \leq b \leq$ ...), we obtain a (lexicographic) total order on $\Sigma^\star$, e.g. $\epsilon \leq a\ a\ b \leq a\ b$

u is a *prefix of* w if there is v such that $u\ v = w$

Likewise, v is said to be a *suffix of* w

We can denote* the prefix order $u \sqsubseteq w$

Lemma (ex.): if $u \sqsubseteq w$ and $v \sqsubseteq w$ (u et v are both prefixes of w), then $u \sqsubseteq v$ iff $|\mathrm{u}| \leq |v|$.

[*]: no need to order Σ

Languages over Σ

Languages are subsets of $\Sigma^\star$

Some examples of languages:

$\emptyset$	also denoted **0**
$\{ \epsilon \}$	also denoted Λ or **1**
$\{ a \}$	also denoted a

Natural operations between languages

union:	$\mathcal{L}_1 + \mathcal{L}_2$
concatenation:	$\mathcal{L}_1 . \mathcal{L}_2 = \{ u_1\, u_2 \mid u_1 \in \mathcal{L}_1, u_2 \in \mathcal{L}_2 \}$
exponentiation:	$\mathcal{L}^n = \{ u_1 \ldots u_n \mid u_i \in \mathcal{L} \}$
	$\mathcal{L}^0 = \mathbf{1}$
Kleene star:	$\mathcal{L}^\star = \bigcup_{n \geq 0} \mathcal{L}^n$

Languages over Σ: algebraic laws

$$\mathcal{L} + \mathbf{0} = \mathbf{0} + \mathcal{L} = \mathcal{L}$$

$$\mathcal{L} \, . \, \mathbf{1} = \mathbf{1} \, . \, \mathcal{L} = \mathcal{L}$$

$$\mathcal{L} \, . \, \mathbf{0} = \mathbf{0} \, . \, \mathcal{L} = \mathbf{0}$$

$$a^\star = \mathbf{1} + a \, . \, a^\star = \{\epsilon\} + \{a\} + \{aa\} + \ \ldots$$

$$(1 + \mathcal{L})^\star = \mathcal{L}^\star$$

$\emptyset$ also denoted $\mathbf{0}$

$\{ \epsilon \}$ also denoted Λ or $\mathbf{1}$

$\{ a \}$ also denoted a

Languages over Σ: algebraic laws

$$\mathcal{L} + \mathcal{L} = \mathcal{L}$$

$$(\mathcal{L}_1 + \mathcal{L}_2).\mathcal{L}_3 = (\mathcal{L}_1.\mathcal{L}_3) + (\mathcal{L}_2.\mathcal{L}_3)$$

$$\mathcal{L}^{\star^\star}a \ \ = \mathcal{L}^\star$$

…

Languages: other operations

intersection: $\mathcal{L}_1 \cap \mathcal{L}_2$

complement: $\bar{\mathcal{L}} = \Sigma \setminus \mathcal{L}$

mirror image: $\widetilde{\mathcal{L}} = \{ \tilde{u} \mid u \in \mathcal{L} \}$

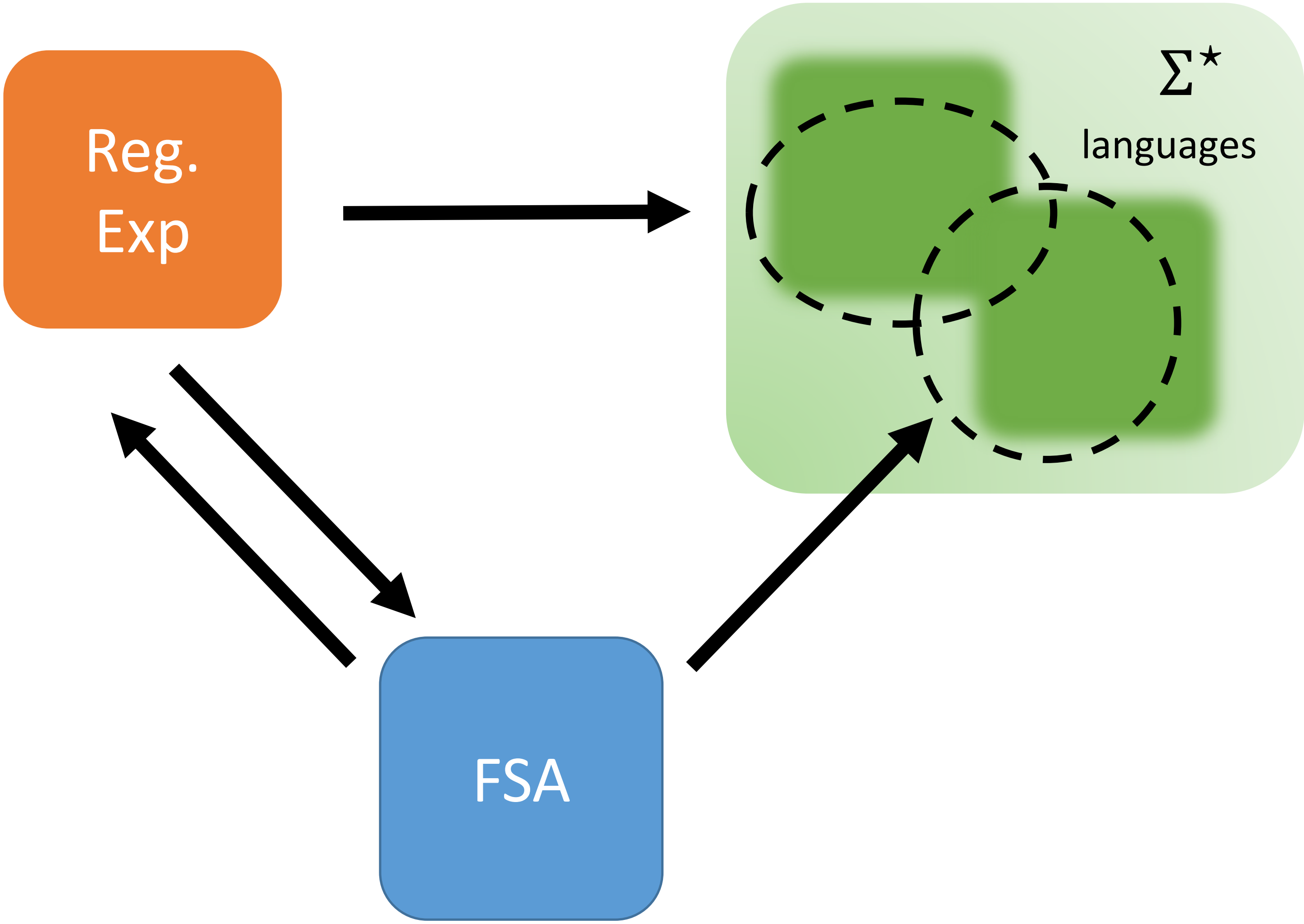
residual: $u^{-1}\mathcal{L} = \{ v \mid u.v \in \mathcal{L} \}$

The notion of residuals is very important to understand the power of DFA.

$(con)^{-1} . \mathcal{L}$

computing
concept
concern
concurrency
concurrent
conference
confuses
connectivity
consistent
consortium
constraint
constructed
context
continues
contradicting
contrast
contribution
control
conventional
cooperates

What are we proving today ?



Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.

[alt.religion.emacs]

Regular Expressions

Regex 101

Regular Expression

A *regular expression* (regex) is either:

- the constant $\emptyset$ (matching nothing)
- the constant Λ (matching the empty word)
- a symbol a in Σ
- a sequential composition: $R_1\ R_2$
- a union: $R_1 + R_2$
- a repetition: $R^\star$

R_1, R_2 two regexes

Regular Expression

It is an expression built from the syntax:

$$
\begin{aligned}
R, R_1, R_2, \ldots \quad := \quad & \mathbf{0} \\
\mid\ & \mathbf{1} \\
\mid\ & a \\
\mid\ & R_1 \ . \ R_2 \\
\mid\ & R_1 + R_2 \\
\mid\ & {R_1}^*
\end{aligned}
$$

Example: $a^* b + b\ a$

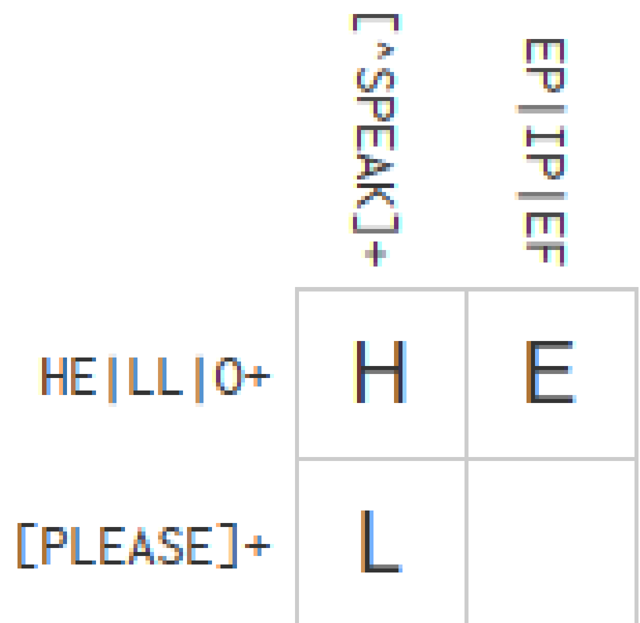
Regular Expression: notations

- we use indifferently $+$ and | for choice
- we simply write $R_1 R_2$ instead of $R_1 . R_2$
- [abc]   means $a + b + c$
- [a-zA-Z]   for range of symbols
- [^ab]   every symbol but a or b
- [^]   every symbol in Σ
- R^+ stands for $R . R^\star$ (at least one R)
- $R^?$ stands for $\epsilon + R$ (zero or one occ.)

"syntactic sugar"

Regex Crosswords

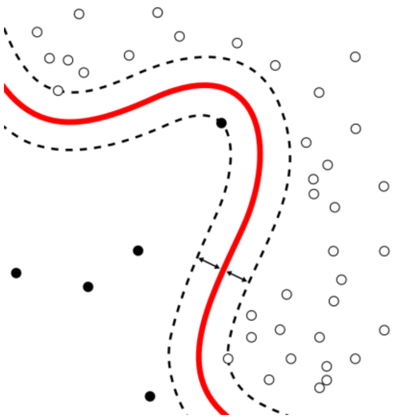
You should all know about regular expression by now, but this is a good opportunity for a refresher



https://regexcrossword.com/

Regex Golf

XKCD #1313



the expression:

/bu|[rn]t|[coy]e|[mtg]a|j|iso|n[hl]|[ae]d|lev|sh|[lnd]i|[po]o|ls/

matches the last names of elected US presidents but not their opponents

Regex Golf

XKCD #1313

The Motion Picture	The Phantom Menace	`/m␣/`
The Wrath of Khan	Attack of the Clones	`/␣t/`
The Search For Spock	Revenge of the Sith	`/␣t/`
The Voyage Home	A New Hope	`/␣n/`
The Final Frontier	The Empire Strikes Back	`/b/`
The Undiscovered Country	Return of the Jedi	`/␣t/`

Regex `/m␣|␣[tn]|b/` separates these two lists

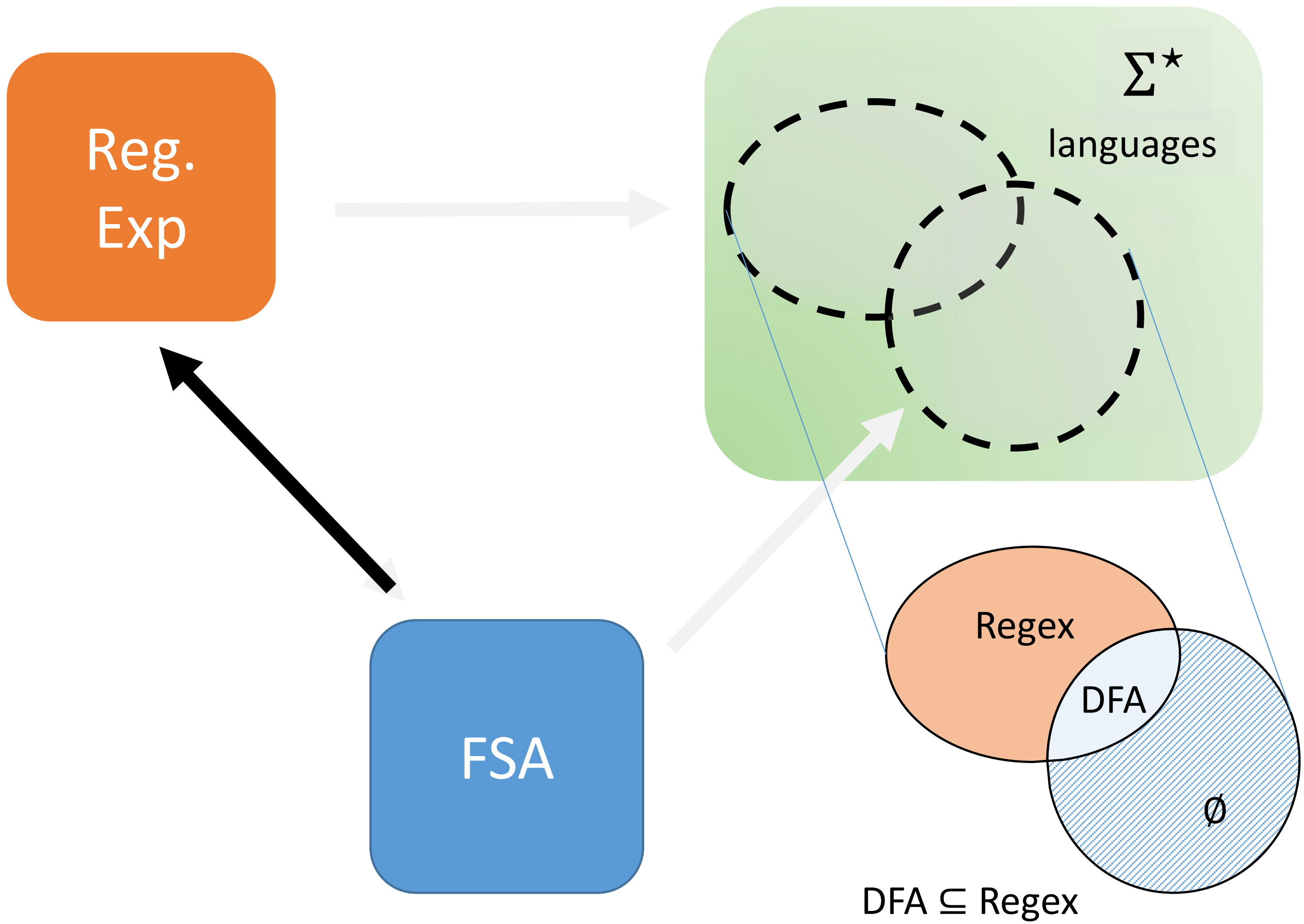
Regular Expression: semantics

A regex e defines a language $\mathcal{L}(e)$ over $\Sigma^\star$ with the following interpretation:

$$\begin{aligned}
\mathcal{L}(\,\mathbf{0}\,) &= \emptyset \\
\mathcal{L}(\,\mathbf{1}\,) &= \Lambda \\
\mathcal{L}(\,a\,) &= \{\,a\,\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{L}(\,R_1\,R_2\,) &= \mathcal{L}(R_1)\,.\,\mathcal{L}(R_2) \\
\mathcal{L}(\,R_1 + R_2\,) &= \mathcal{L}(R_1) \cup \mathcal{L}(R_2) \\
\mathcal{L}(\,R^\star\,) &= \mathcal{L}(R)^\star
\end{aligned}$$

What are we proving next ?



NFA $\Rightarrow$ Regex

sémantique

Intuition

- Call $\mathcal{A}(i,j)$ the language of words recognized by $\mathcal{A}$ when starting in state i and ending in j
- Call $\mathcal{A}(i)$ the language of words recognized by $\mathcal{A}$ if i was the initial state

$$\mathcal{A}(i) = \bigcup_{q_f \in F} \mathcal{A}(i, q_f)$$

- Then: $\mathcal{L}(\mathcal{A}) = \mathcal{A}(q_i)$

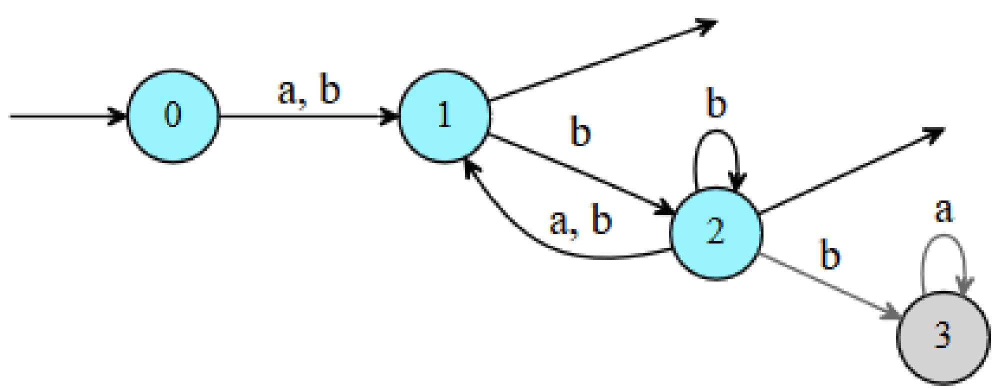
Intuition: matrix format

	\$	0	1	2	3
\$	ϵ	T	-	-	-
0	-	ϵ	a, b	-	-
1	T	-	ϵ	b	-
2	T	-	a, b	ϵ, b	b
3	-	-	-	-	ϵ, a

$\times$ is concatenation

$+$ is set union ($\cup$)

$\star$ is transitive closure



Intuition: matrix format

$\mathcal{A}[i,j] =$ words "accepted" starting from state i, ending in j, in one step ($|w| = 1$)

	$	0	1	2	3
$	ϵ	T	-	-	-
0	-	ϵ	a, b	-	-
1	T	-	ϵ	b	-
2	T	-	a, b	ϵ, b	b
3	-	-	-	-	ϵ, a

Intuition: “linear algebra”

solution to
$X = \mathbf{1}_F + \mathcal{A}.X$ $\Rightarrow$ the X_i’s are the words “accepted” starting from state i.

the result in X_i is the set $\mathcal{A}(i)$ defined before.

we have ϵ in X_i iff state i is final.

	\$	0	1	2	3
\$	ϵ	T	-	-	-
0	-	ϵ	a, b	-	-
1	T	-	ϵ	b	-
2	T	-	a, b	ϵ, b	b
3	-	-	-	-	ϵ, a

Arden’s rule

Theorem: the language $A^{\star}.B$ is the smallest language that is a solution for X in the (linear) equation:

$$X = A.X + B$$

The solution is unique as soon as $\epsilon \notin A$

This can be extended to sets of equations of the form

$$X_i = A_{i,1}.X_1 + \ \dots + A_{i,n}.X_n + B$$

From NFA to Regex

- Use one variable X_i for every state $i \in Q$
- For every transition $\delta(i, a) = j$ add the constraint

$$X_i = a.X_j + X_i$$

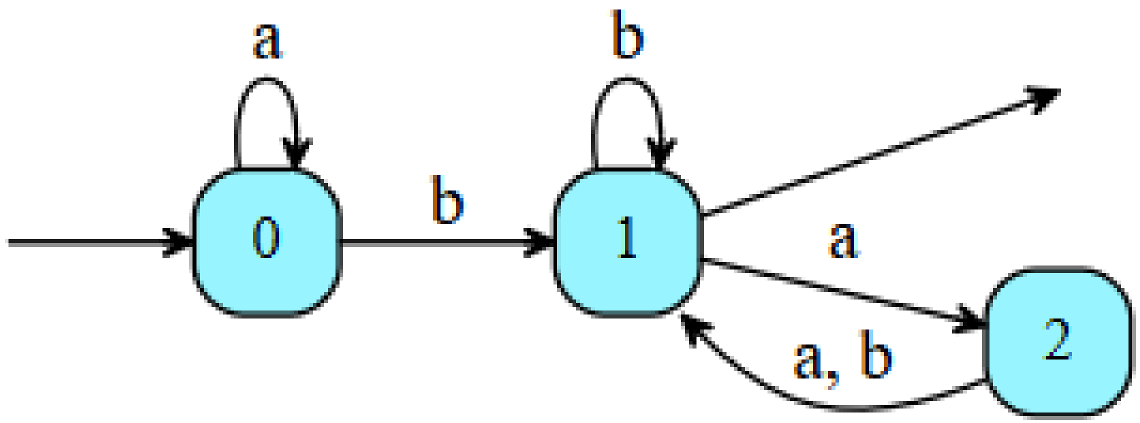
- For every state $i \in F$ add the constraint

$$X_i = X_i + \Lambda$$

- Solve for X

we can make use of simplifications in order to have one occurrence of each X_i as a *lhs, e.g.* $X = X + X$, or $R.X + R'.X = (R + R').X$, etc.

From NFA to Regex



$$X_0 = a.X_0 + b.X_1$$

$$X_1 = b.X_1 + a\,X_2 + \Lambda$$

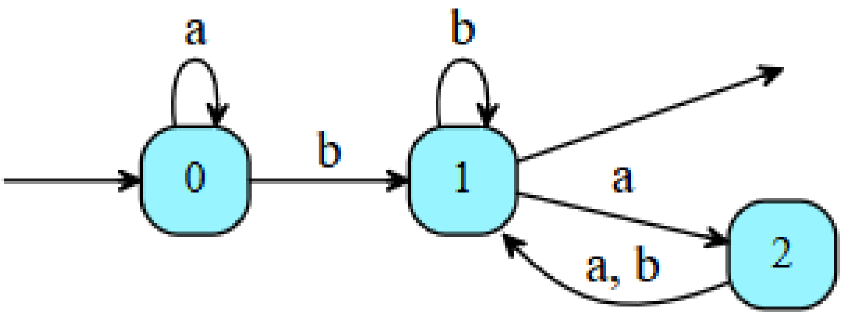
$$X_2 = (a + b).X_1$$

From NFA to Regex

$$X_0 = a.X_0 + b.X_1$$

$$X_1 = b.X_1 + a\,X_2 + \Lambda$$

$$X_2 = (a+b).X_1$$

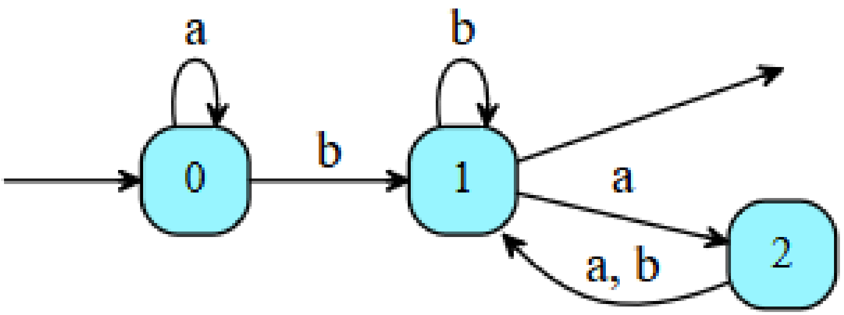


From NFA to Regex

$X_0 = a.X_0 + b.X_1$

$X_1 = b.X_1 + a\,X_2 + \Lambda$

$X_2 = (a + b).X_1$



$\Downarrow$ substitution + factorization

$X_0 = a.X_0 + b.X_1$

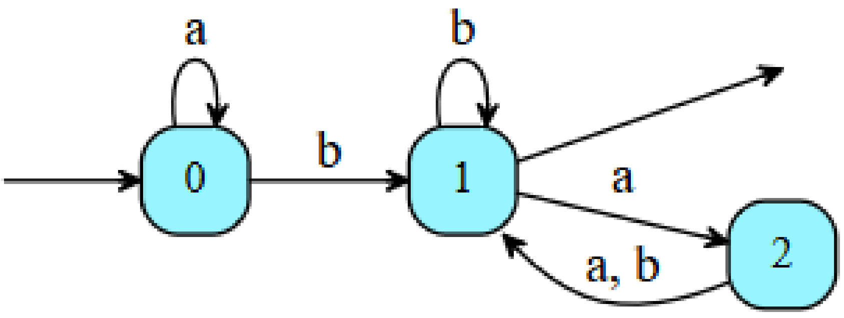
$X_1 = b.X_1 + a(a + b).X_1 + \Lambda = R.X_1 + \Lambda$

$X_2 = (a + b).X_1$

where $R = b + a(a + b)$

From NFA to Regex

$$X_0 = a.X_0 + b.X_1$$
$$X_1 = R.X_1 + \Lambda$$
$$X_2 = (a+b).X_1$$



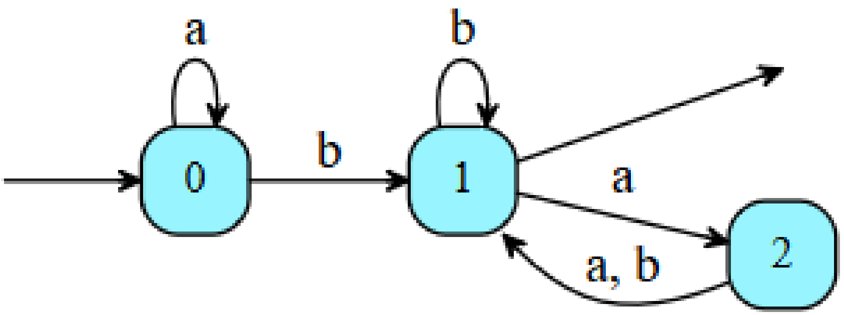
$\Downarrow$ Arden's rule

$$X_0 = a.X_0 + b.X_1$$
$$X_1 = R^\star$$
$$X_2 = (a+b).X_1$$

where $R = b + a(a+b)$

From NFA to Regex

$X_0 = a.X_0 + b.X_1$
$X_1 = R^\star$
$X_2 = (a+b).X_1$



$\Downarrow$ substitution

$X_0 = a.X_0 + b.R^\star$
$X_1 = R^\star$
$X_2 = (a+b).R^\star$

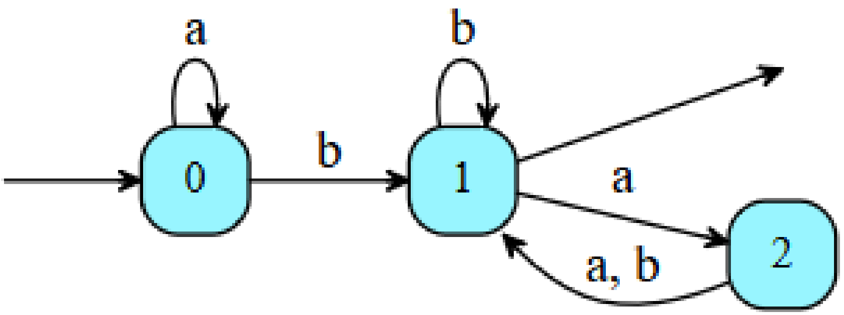
where $R = b + a(a+b)$

From NFA to Regex

$X_0 = a.X_0 + b.R^\star$

$X_1 = R^\star$

$X_2 = (a + b).R^\star$



$\Downarrow$ Arden's rule

$X_0 = a^\star.b.(b + a(a + b))^\star$

$X_1 = R^\star$

$X_2 = (a + b).R^\star$

where $R = b + a(a + b)$

From NFA to Regex: remarks

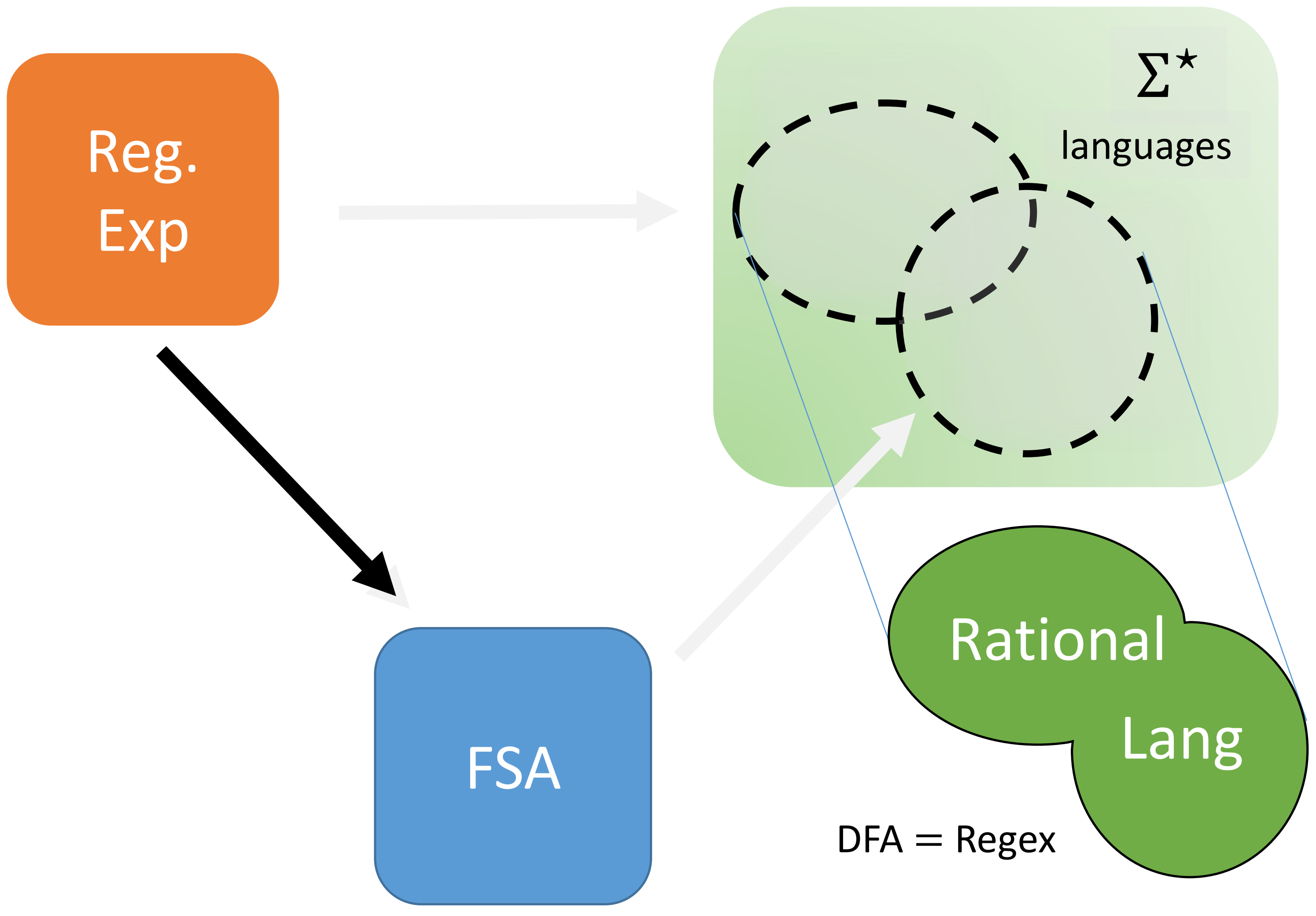
- This construction can be used on automata that are not deterministic and/or not complete
- The size of the resulting regex depends on the order in which we "eliminates" variables
- The resulting regex can be exponentially larger than the size of $\mathcal{A}$

  but there are DFA with "small" regex: $(a|b)^{\star}.a.(a|b)^{n}$

Theorem: we have NFA = DFA ⊆ Regex

no real applications in practice ?

What are we proving next ?



Automata &
Closure properties

Product of automata

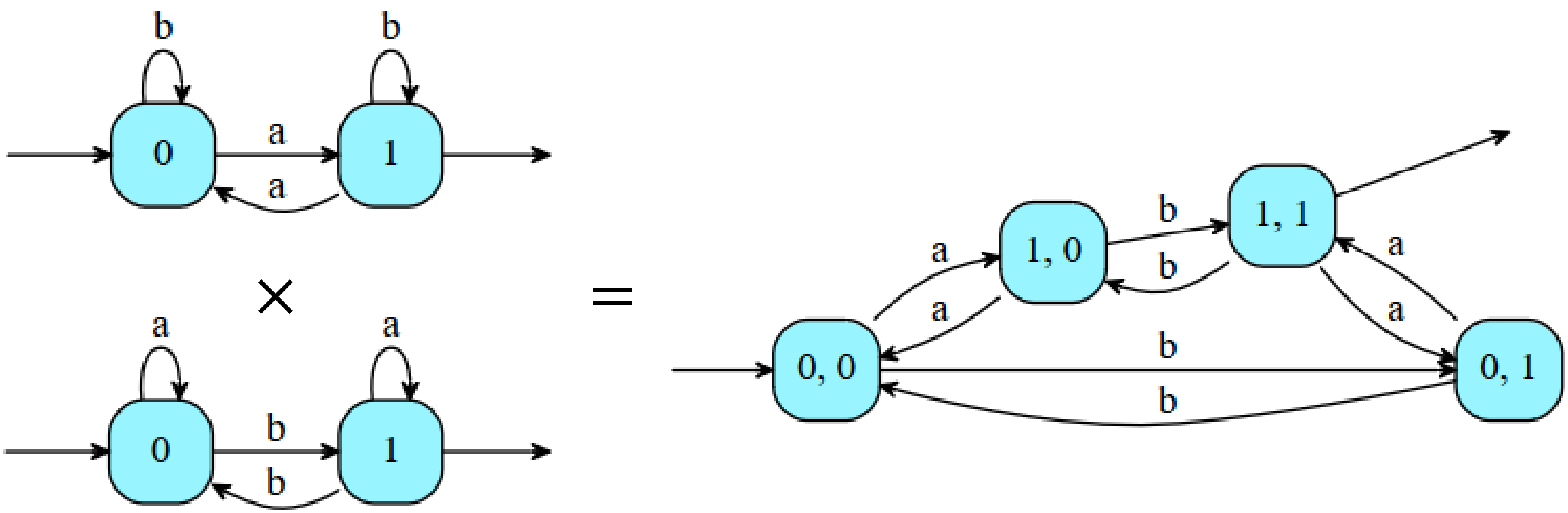
Given two NFA $\mathcal{A}_1$ and $\mathcal{A}_2$, we define the product NFA $\mathcal{A}_1 \times \mathcal{A}_2$ such that

- set of states $Q_1 \times Q_2$
- initial state is (q_I^1, q_I^2)
- final states $F_1 \times F_2$
- $\delta\big((q_1, q_2), a\big) = (q_1', q_2')$ whenever both $\delta_1(q_1, a) = q_1'$ and $\delta_2(q_2, a) = q_2'$

$\mathcal{A}_i = \big(Q_i, \Sigma, \delta_i, q_I^i, F_i\big), i \in 1..2$

Product of automata

We can prove that a word u is in $\mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2)$ iff both $u \in \mathcal{L}(\mathcal{A}_1)$ and $u \in \mathcal{L}(\mathcal{A}_2)$



This gives a simple construction for computing the "intersection" of two languages

The definition still works when $\Sigma_1 \neq \Sigma_2$ ($\Rightarrow$ synchronous product)

Product of automata

Theorem: DFA are closed by intersection.

Therefore Regex are also closed by intersection.

not obvious by direct means !

Complement of an automata

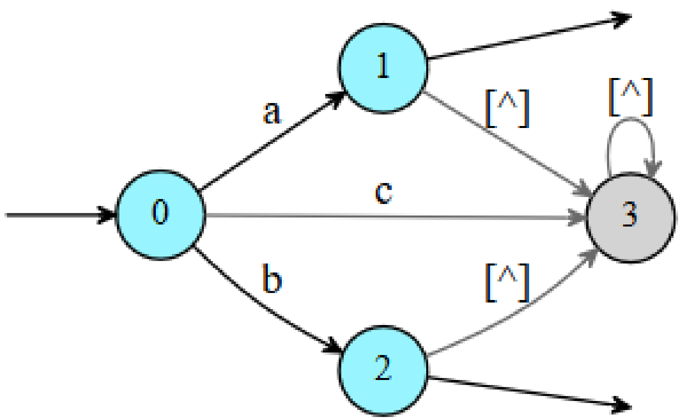
Given a complete + deterministic FSA $\mathcal{A}$, we can define its complement $\mathcal{A}^c$ such that

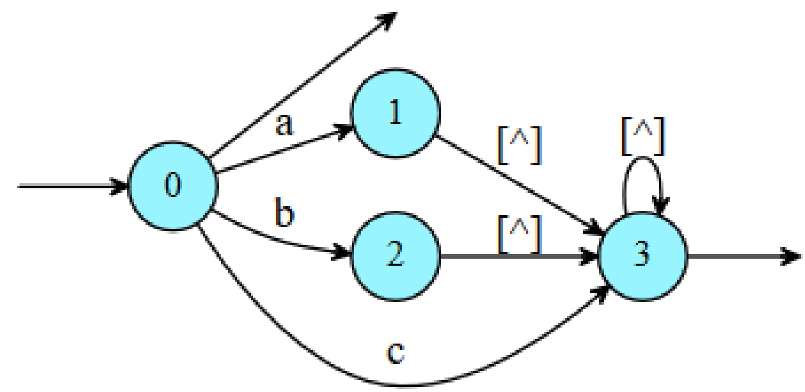
- set of states Q
- initial states is q_I
- final states $Q \setminus F$
- same δ

$$\mathcal{A} = (Q, \Sigma, \delta, q_I, F)$$

Complement of an automata

We can prove that a word u is in $\mathcal{L}(\mathcal{A})$ iff $u \notin \mathcal{L}(\mathcal{A}^c)$





$\mathcal{A} \equiv a + b$

$\mathcal{A}^c \equiv \epsilon + c + [\hat{}].[\hat{}]^\star$

This construction entails $\mathcal{A}$ complete (easy) AND deterministic (costly)

The result is also complete and deterministic

Complement of an automata

Theorem: DFA are closed by complement.

Therefore Regex are also closed by negation.

even less obvious by direct means !

Mirror image of automata

Given a complete DFA $\mathcal{A}$, we can define its mirror image $\tilde{\mathcal{A}}$ such that

- set of states Q
- initial states is ...
- final states ...
- transition function is $\tilde{\delta}$ such that ...

left has an exercise !

$\mathcal{A} = (Q, \Sigma, \delta, q_I, F)$

Shuffle of two automata

- The shuffle of two words, $u \# v$, is the set of words obtained by interlacing u and v
  - $\approx$ concatenation for concurrent activities
- The shuffle of two languages $\mathcal{L}_1 \# \mathcal{L}_2$ is the set of words $u_1 \# u_2$ with $u_i \in \mathcal{L}_i, i \in 1..2$
- "DFA" are closed by shuffle

left has an exercise !

Union of automata

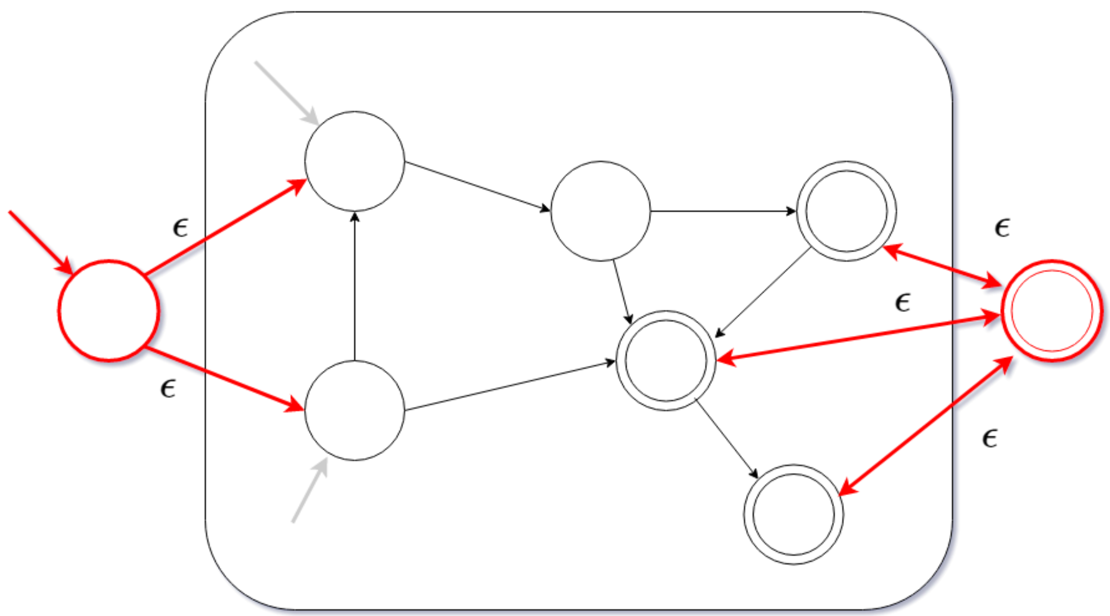
Given two NFA $\mathcal{A}_1$ and $\mathcal{A}_2$, we define the union NFA $\mathcal{A}_1 \cup \mathcal{A}_2$ such that

- set of states $Q_1 \cup Q_2 \cup \{ q_I \}$
- initial state is q_I
- final states $F_1 \cup F_2$
- $\delta(q, a) = q'$ if $\delta_i(q, a) = q'$ for some $i \in 1..2$
- $\delta(q_I, \epsilon) = q_I^1$ and $\delta(q_I, \epsilon) = q_I^2$

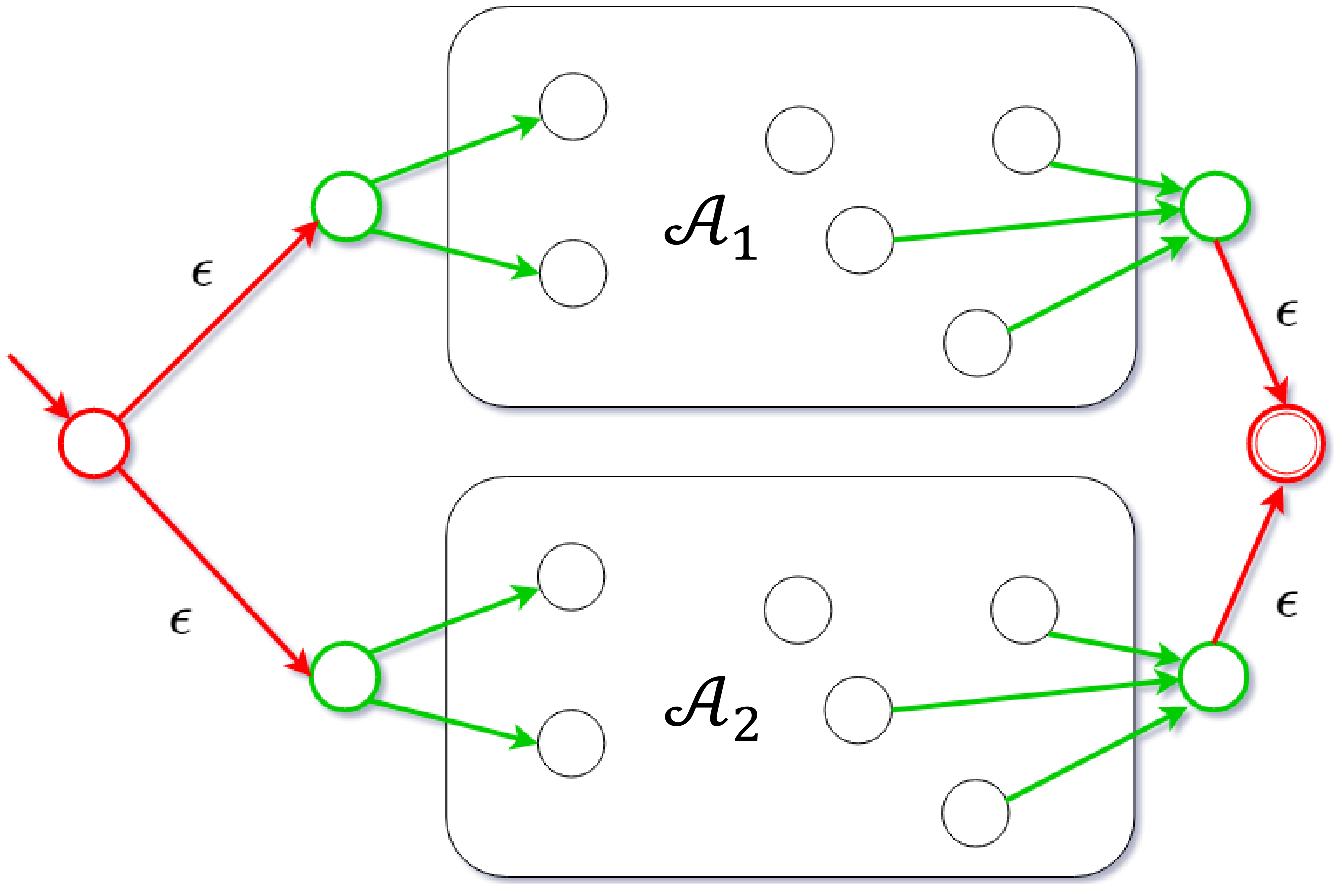
$$\mathcal{A}_i = \left(Q_i, \Sigma, \delta_i, q_I^i, F_i\right), i \in 1..2$$

Standard form

- We can always assume a unique initial state
- no incoming transition on the initial state
- a single final state and no back transitions from it



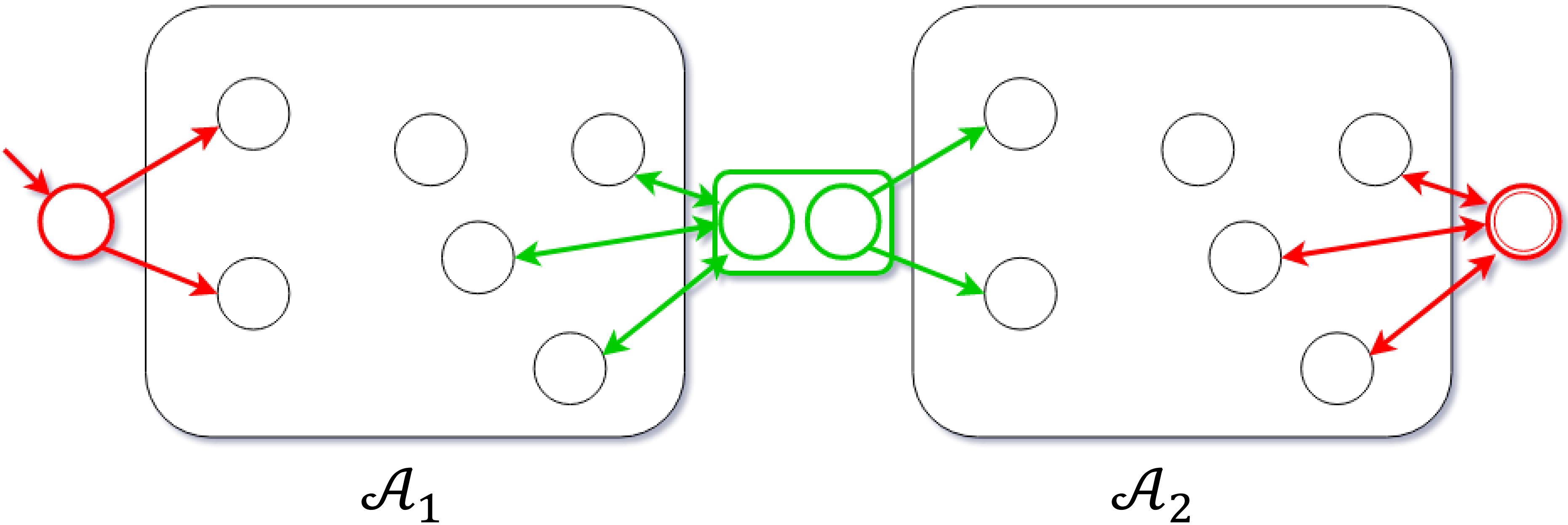
Union of automata



Union of automata

Theorem: DFA are closed by $+$.

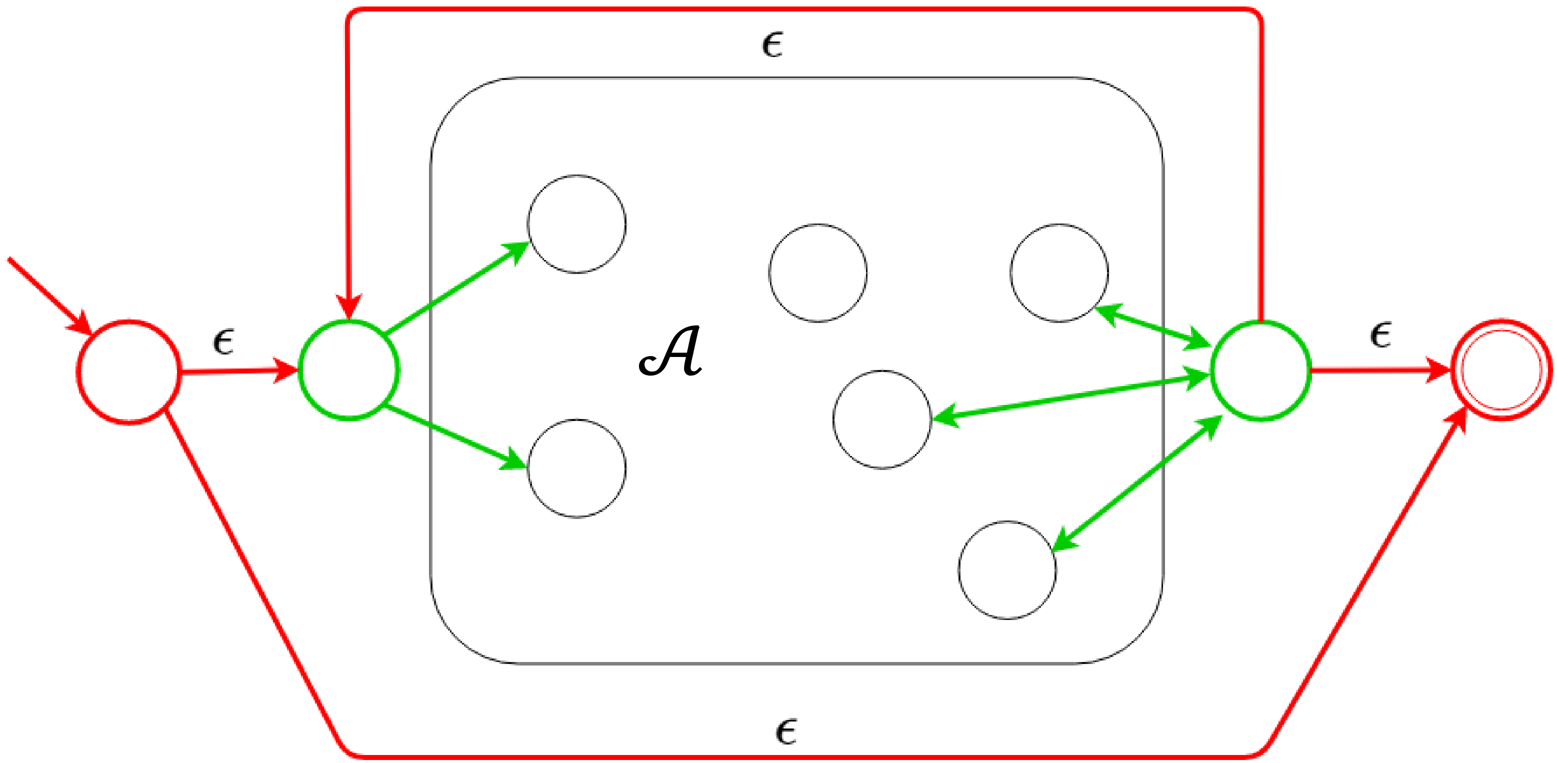
Concatenation of automata



Concatenation of automata

Theorem: DFA are closed by concatenation.

Iteration of automata



Iteration of automata

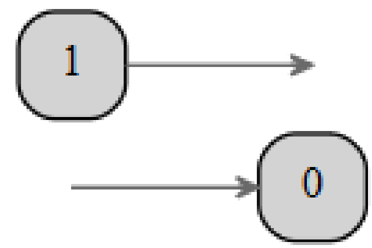
Theorem: DFA are closed by $\star$.

⇒ we prove that Regex ⊆ DFA

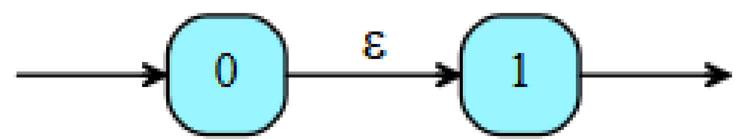
Closure properties

DFA are closed by $.$, $+$, and $\star$

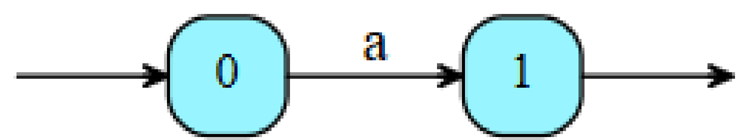
There is a DFA for $\mathbf{0}$



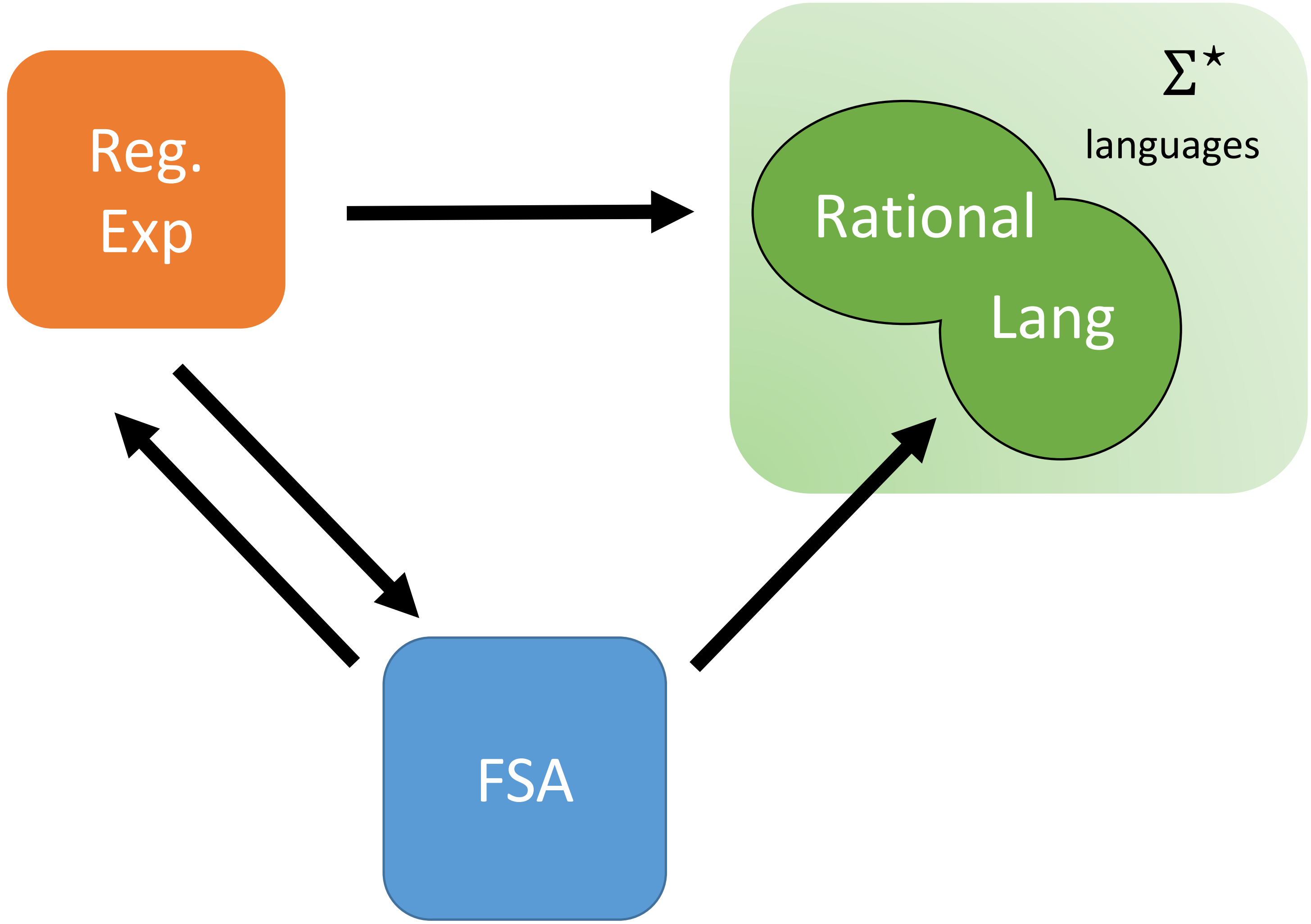
There is a DFA for $\mathbf{1}$



There is a DFA for a



What we have proved so far



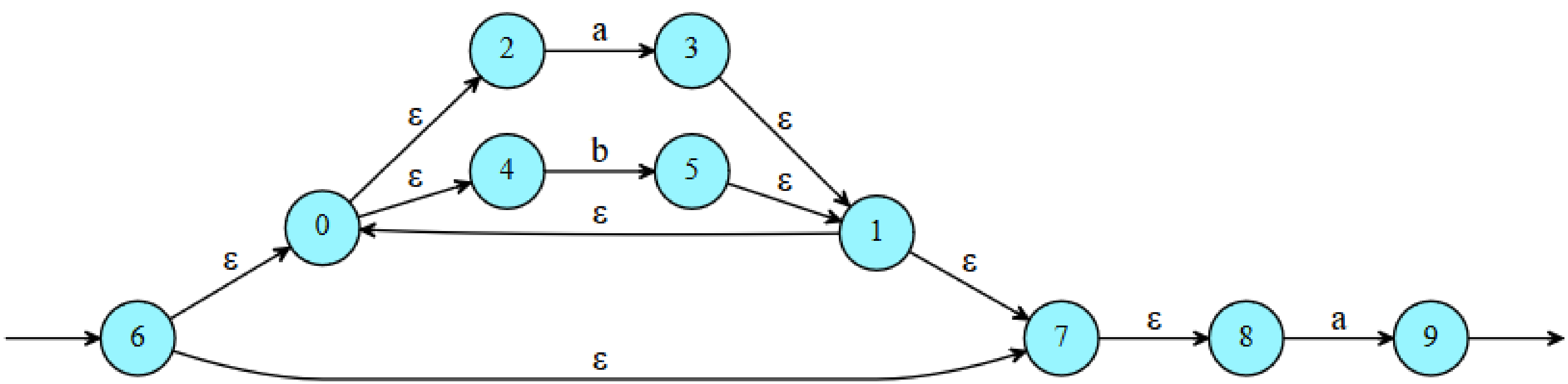
Regex $\Rightarrow$ NFA

Regex ⇒ NFA: compositionally

- The previous results give a simple and *compositional* method for computing a NFA equivalent to a regex ⇒ Thompson's method
- The result is in standard form
- If regex e has c concatenation and s symbols then the resulting NFA has $2\,s\; - c$ states
- Hence we can do pattern matching of e on a word u with "*linear*" complexity $O(|e|^2.\,|u|)$

Regex ⇒ NFA: compositionally

- The resulting NFA has a lot of ϵ-transitions ⇒ we can do better (see Glushkov's construction)
- The result is non-det. ⇒ we can do better



Thompson's construction for $(a + b)^{\star}. a$

Regex $\Rightarrow$ DFA

Brzozowski derivative

Residuals and derivative

Residual: $u^{-1}\mathcal{L} = \{ v \mid u.v \in \mathcal{L} \}$

we have $(a.u)^{-1}\mathcal{L} = u^{-1}(a^{-1}\mathcal{L})$

the language $a^{-1}\mathcal{L}$ is called a *derivative*

computing
concept
concern
concurrency
concurrent
conference
confuses
connectivity
consistent
consortium
constraint
constructed
context
continues
contradicting
contrast
contribution
control
conventional
cooperates

$(con)^{-1} . \mathcal{L}$

think $(f \circ g)^{-1} = g^{-1} \circ f^{-1}$

Intuition

- Take a complete DFA $\mathcal{A}$ with n states
- Each word u leads to one state, say q_u, meaning that $(q_I, u) \Rightarrow q_u$
- The set of words accepted by $\mathcal{A}$ when starting from q_u, denoted $\mathcal{A}(q_u)$ before, is exactly $u^{-1}\mathcal{L}(\mathcal{A})$

There is a mapping (surjection) from Q
$\mapsto$ the set of residuals of $\mathcal{A}$

Intuition (2)

- The set of words accepted by $\mathcal{A}$ when starting from q_u, denoted $\mathcal{A}(q_u)$ before, is exactly $u^{-1}\mathcal{L}(\mathcal{A})$
- If $\delta(q, a) = q'$ then: $\mathcal{A}(q) = a.\mathcal{A}(q')$ and
  $\mathcal{A}(q') = a^{-1}\mathcal{A}(q)$

There is a mapping (bijection) between the transitions in $\mathcal{A}$ $\mapsto$ and the set of residuals of $\mathcal{A}$

the relation is $\propto$ in the case of a NFA

Derivates of a Regex

- We can define the notion of residuals/derivates directly at the level of Regex
- $\mathrm{D}_a(e) =$ regex matching the words in $a^{-1}\mathcal{L}(e)$
- $\mathrm{D}_a(e) =$ "*what we match in* e after reading an a"

Derivatives of a Regex

- $\mathrm{D}_a(\mathbf{0}) = \mathbf{0}$
- $\mathrm{D}_a(\mathbf{1}) = \mathbf{0}$
- $\mathrm{D}_a(\mathrm{a}) = \mathbf{1}$
- $\mathrm{D}_a(\mathrm{b}) = \mathbf{0}$
- $\mathrm{D}_a(\mathrm{e}_1 + \mathrm{e}_2) = \mathrm{D}_a(\mathrm{e}_1) + \mathrm{D}_a(\mathrm{e}_2)$
- $\mathrm{D}_a(\mathrm{e}_1 . \mathrm{e}_2) \quad = \mathrm{D}_a(\mathrm{e}_1) . \mathrm{e}_2 + \epsilon^?(\mathrm{e}_1) . \mathrm{D}_a(\mathrm{e}_2)$
- $\mathrm{D}_a(\mathrm{e}^\star) \quad = \mathrm{D}_a(\mathrm{e}) . \mathrm{e}^\star$

where $\epsilon^?(\mathrm{e})$ = **1** if $\epsilon \in \mathrm{e}$, otherwise $\mathbf{0}$

Derivatives ≈ Differentials

- f constant $\qquad \frac{dc}{dx} = \mathbf{0}$
- f linear $\qquad \frac{df}{dx} = \mathbf{1}$ or $\mathbf{0}$ $\qquad$ (think of $f(x, y) = y$)
- addition: $\qquad \frac{d(f+g)}{dx} = \frac{df}{dx} + \frac{dg}{dx}$
- product: $\qquad \frac{d(f.g)}{dx} = \frac{df}{dx}.g + f.\frac{dg}{dx}$
- iteration: $\qquad f^{\star} = \mathbf{1} + f.f^{\star}$

Derivatives: rule for .

- What are the words, matched by $e_1 . e_2$, starting with an a (this is $\mathrm{D}_a(e_1 . e_2)$).
- These are words of the form $u = a\ v_1\ v_2$

(1) e_1 matches a (and the start of u)

$$\begin{array}{c|c|c} a & v_1 & v_2 \\ a & D_a(e_1) & e_2 \end{array}$$

(2) e_1 can match ϵ, then we can also match u with e_2

$$\begin{array}{c|c|cc} & a & v_1 & v_2 \\ \epsilon & a & D_a(e_2) & \end{array}$$

Derivatives: examples

- $\mathrm{D}_a\big((a+b)^n\big) = (a+b)^{n-1}$ with $n \geq 1$
- $\mathrm{D}_a\big((a+b)^\star\big) = (a+b)^\star$

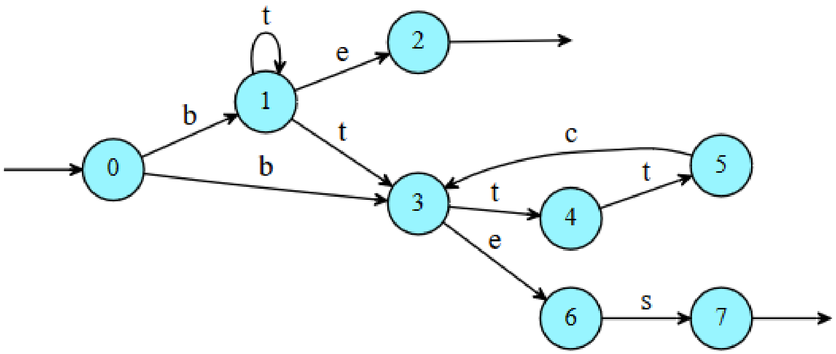
- De Bruijn expression: $J_n = (a+b)^\star a(a+b)^n$
- $\mathrm{D}_a(J_n) = J_n + \mathbf{1}\,.\,\mathrm{D}_a(a(a+b)^n)$
  $= J_n + (a+b)^n$
- $\mathrm{D}_b(J_n) = J_n + \mathbf{1}\,.\,\mathrm{D}_b(a(a+b)^n)$
  $= J_n + \mathbf{0}$

Regex ⇒ DFA: Brzozowski

- We can build an DFA from e_0 using derivatives
- We have an initial state $\equiv e_0$
- We have one state for each residuals of e_0
  - there are finitely many (TBD)
- We have a transition from e_i to e_j, labeled a, whenever $\mathrm{D}_a(e_i) = e_j$
  - ensure both deterministic + complete
- Final states are regex matching ϵ (i.e. $\epsilon^?(e_i) = \mathbf{1}$)
  - because we can stop there !

drawback: two derivatives may correspond to = languages

Example



communication protocol:

$$b\ t^{\star}\ e + (b + b\ t^{+})(t\ t\ c)^{\star} es$$

Rational Languages

petit rappel sur le cours précédent

Language: $\equiv$ modulo $\mathcal{L}$

residual: $u^{-1}\mathcal{L} = \{\, v \mid u.v \in \mathcal{L} \,\}$

Definition: We say that $u \equiv_L v$ if u and v have the same residuals in $\mathcal{L}$.

$$u \equiv_L v \quad \text{iff} \quad u^{-1}\mathcal{L} = v^{-1}\mathcal{L}$$

Rational Languages

Definition: a language $\mathcal{L}$ is said to be *rational,* $\mathcal{L} \in Rat$, if $\exists\ \mathcal{A}$ such that $\mathcal{L} = \mathcal{L}(\mathcal{A})$

or equivalently

Definition: a language $\mathcal{L}$ is said to be *rational,* $\mathcal{L} \in Rat$, if $\exists\ e$ such that $\mathcal{L} = \mathcal{L}(e)$

Rational Languages

Definition: a language $\mathcal{L}$ is said to be *rational,* $\mathcal{L} \in Rat$, if $\exists\ \mathcal{A}$ such that $\mathcal{L} = \mathcal{L}(\mathcal{A})$

Theorem (Myhill-Nerode): a language $\mathcal{L}$ is *rational* iff the relation $\equiv_L$ has a finite number of equivalent classes.

moreover each class $\mapsto$ (bijectively) the states of the *minimal DFA* accepting $\mathcal{L}$.

Why is it called rational

$$^{1}/_{7} = 0\,.\,\overbrace{142857}^{p}\,142857\,142\ldots$$