Languages and Machines (Module 7 TCS+IAM)

L&M 2: Non-deterministic Finite Automata

Ch 5:4-6, Ch 6:2

Jaco van de Pol

Formal Methods and Tools, Computer Science, University of Twente

Lecture 2

Contents

Contents

Deterministic Finite Automata (DFA)

Recap

A Deterministic Finite Automaton (DFA) is a 5-tuple:

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

Where

- Q is a set of states and Σ is the alphabet.
- The initial state is $q_0 \in Q$ and $F \subseteq Q$ is the set of accepting (final) states.
- $\delta : Q \times \Sigma \to Q$ is a total function .....(deterministic)

Incomplete DFA

- Has a partial function: $\delta : Q \times \Sigma \rightharpoonup Q$

Acceptance by DFA through configurations ($\vdash$)

A configuration of a DFA is a pair $[q, w] \in Q \times \Sigma^*$

- q is the current state
- w is the word that is left to be read

Define the one-step-relation ($\vdash$) as follows

- $[q_i, aw] \vdash [\delta(q_i, a), w]$
- $\vdash^*$ means applying $\vdash$ zero or more times

Acceptance

- Now define: M accepts w, if a $q \in F$ exists, with $[q_0, w] \vdash^* [q, \lambda]$.

Accepted language

$$\mathcal{L}(M) = \{w \in \Sigma^* \mid M \text{ accepts the word } w\}$$

State of affairs

Known already

- Regular languages are closed under union (by definition $E_1 \cup E_2$)
- DFA languages are closed under complement (complement accepting states)

Open problem

- Are these classes of languages equivalent?
- If yes: then they are also closed under intersection: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

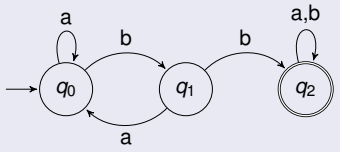
Solution

- Non-deterministic Finite Automata (NFA): Guess!

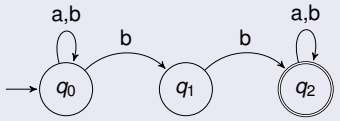
Contents

Guessing can be helpful

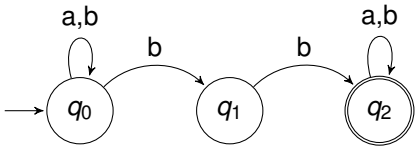
DFA (deterministic) for $(a \cup b)^* bb(a \cup b)^*$



NFA (non-deterministic) for $(a \cup b)^* bb(a \cup b)^*$



Non-determinism



There are four ways of tracing *bbb*

$[q0, bbb]$	$[q0, bbb]$	$[q0, bbb]$	$[q0, bbb]$
$\vdash [q0, bb]$	$\vdash [q0, bb]$	$\vdash [q0, bb]$	$\vdash [q1, bb]$
$\vdash [q0, b]$	$\vdash [q0, b]$	$\vdash [q1, b]$	$\vdash [q2, b]$
$\vdash [q0, \lambda]$	$\vdash [q1, \lambda]$	$\vdash [q2, \lambda]$	$\vdash [q2, \lambda]$

How does this work exactly?

Is *bbb* now accepted??

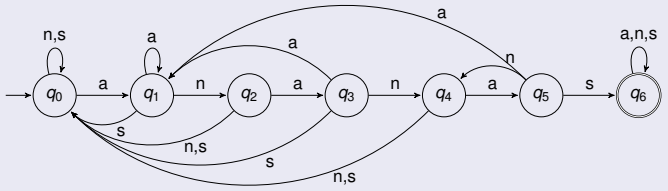
Non-determinism (intuitive explanation)

When does an NFA accept a word?

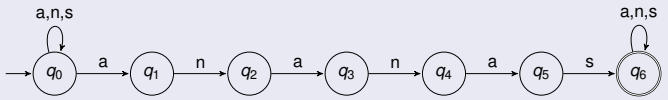
- In a non-deterministic automation, a state can have multiple outgoing arrows with the same symbol.
- So, a word can correspond to multiple paths in the automaton. It might be the case that:
  - all paths lead to an accepting state
  - no path leads to an accepting state
  - some paths lead to an accepting state, while others do not.
- Definition: a word is accepted if there exists at least one path that leads to an accepting state
- (so non-deterministic automata always 'guess' right)

Guessing can be really helpful

string contains 'ananas' as DFA



string contains 'ananas' as NFA



Non-deterministic Finite Automaton (NFA)

A Non-deterministic Finite Automaton (NFA) is a 5-tuple

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

In this case

- Q is a finite set of states
- $q_0 \in Q$ is the unique initial state
- Σ is a finite set of symbols, the alphabet.
- $F \subseteq Q$ is the set of accepting state.
- δ is a total function, $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ (transition function)

Nota bene

$\delta(q, a) \in \mathcal{P}(Q)$ is equivalent with $\delta(q, a) \subseteq Q$.

Every DFA "is" an NFA

Difference between DFA and NFA

- In a NFA, $\delta(q, a)$ is a set of states.
- This set can:
  - be empty (as in an incomplete DFA)
  - be singleton (as in a DFA)
  - contain multiple elements: real non-determinism

Interpret DFA as NFA (formally:)

- Given a DFA $M_D = (Q, \Sigma, q_0, \delta, F)$,
  - define an NFA $M_N = (Q, \Sigma, q_0, \delta', F)$,
  - with $\delta'(q, a) := \{\delta(q, a)\}$.
- Note: $\delta : Q \times \Sigma \to Q$ and $\delta' : Q \times \Sigma \to \mathcal{P}(Q)$.
- So the class of DFA-definable languages is contained in the class of NFA-definable languages. And the other way around??

Language accepted by an NFA

Acceptance through configurations

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA.
- Define $[q, aw] \vdash [q', w]$ as $q' \in \delta(q, a)$
- M accepts w if there exists some computation of the form $[q_0, w] \vdash^* [q, \lambda]$ with $q \in F$.

In other words

$$\mathcal{L}(M) = \{w \in \Sigma^* \mid \exists q \in F.\ [q_0, w] \vdash^* [q, \lambda]\}$$

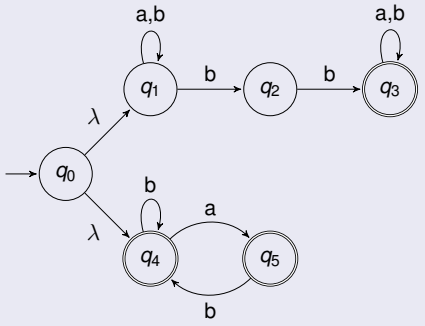
Contents

λ-steps can also be helpful

Helpful for the union of overlapping languages

All words that:

- either contain *bb* as substring;
- or do not contain *aa*



Automata with λ-transitions (NFA-λ)

Definition NFA-λ

A Nn-deterministic Finite Automaton with λ-transitions (NFA-λ) is a 5-tuple:

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

In this case

- Q is a finite set of states.
- $q_0 \in Q$ is the unique initial state.
- Σ is a finite set of symbols, the alphabet.
- $F \subseteq Q$ is the set of accepting states.
- δ is a total function, $\delta : Q \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q)$

So automata with λ-transitions can either read a symbol, or "simply" perform steps spontaneously.

Semantics of NFA-λ (through λ-closure)

λ-steps

- Automata with λ-transitions can read a symbol,or "simply" perform steps spontaneously.
- This results in the following new one-step-relation:
  - $[q, aw] \vdash [q', w]$ if $q' \in \delta(q, a)$
  - $[q, w] \vdash [q', w]$ if $q' \in \delta(q, \lambda)$
- To describe all states q that can be reached with λ steps, we introduce λ-closure.

Contents

Defining the λ-closure

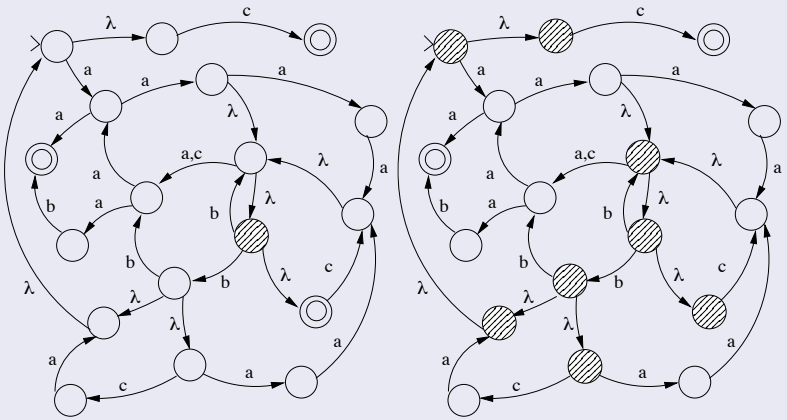
The λ-closure from q is the set of states that can be reached from q through taking a number of λ-steps only.

Inductive definition of λ-closure

- $q \in \lambda\text{-closure}(q)$.
- If $q_1 \in \lambda\text{-closure}(q)$, and $q_2 \in \delta(q_1, \lambda)$, then also $q_2 \in \lambda\text{-closure}(q)$.
- $\lambda\text{-closure}(q)$ is the smallest set that meets this requirement.

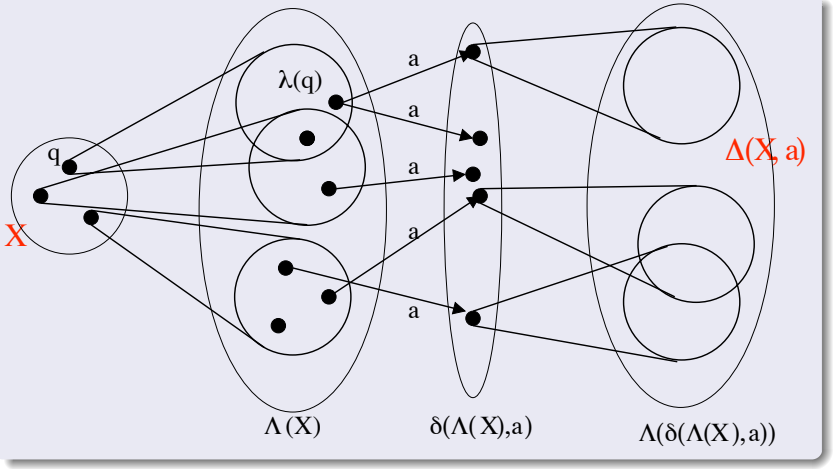
λ-closure (example)

illustration of λ-closure



- Imagine a machine with buttons a, b, c and internal steps λ.
- What is the λ-closure after performing a b-step?
- Does this automaton accept b from the grey state?

Impression of $\Delta(X, a)$



Extending λ-closure on sets

- The λ-closure of a state is a set
- Applying δ on one state also produces a set
- We now extend λ and δ on sets as input.
- Suppose you are in a state of $X \subseteq Q$ and you perform an a:
  - First take the λ-closure of all elements of X:

$$\Lambda(X) := \bigcup \{\lambda\text{-closure}(q) \mid q \in X\}$$

  - Perform an a from those states and take the λ-closure again

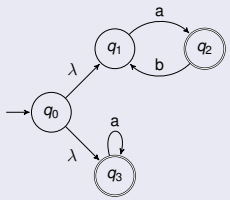
$$\Delta(X, a) := \bigcup \{\Lambda(\delta(q, a)) \mid q \in \Lambda(X)\}$$

Input Transition Function: $t : Q \times \Sigma \to \mathcal{P}(Q)$

Connection:

- $t(q, a) = \Delta(\{q\}, a)$
- $\Delta(X, a) = \bigcup \{t(q, a) \mid q \in X\}$

Example of input transition function



Transition function $t(q, a)$

	λ-closure	a	b
q_0	$\{q_0, q_1, q_3\}$	$\{q_2, q_3\}$	$\varnothing$
q_1	$\{q_1\}$	$\{q_2\}$	$\varnothing$
q_2	$\{q_2\}$	$\varnothing$	$\{q_1\}$
q_3	$\{q_3\}$	$\{q_3\}$	$\varnothing$

$\Delta(\{q_1, q_3\}, a) = \{q_2, q_3\}$

From NFA-λ to DFA (1)

- Start with the NFA-λ: $M_N = (Q, \Sigma, \delta, q_0, F)$. We now have:

$$\Lambda(X) := \bigcup\{\lambda\text{-closure}(q) \mid q \in X\}$$

$$\Delta(X, a) = \bigcup\{\Lambda(\delta(q', a)) \mid q' \in \Lambda(X)\}$$

- Note: the last one is a function $\Delta : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$
- Now the brain-twist: this is exactly the transition function of a DFA with states $\mathcal{P}(Q)$:

$$M_D := (\mathcal{P}(Q), \Sigma, \Delta, \Lambda(\{q_0\}), F')$$

$$F' := \{X \subseteq Q \mid X \cap F \neq \varnothing\}$$

- The construction is such that $\mathcal{L}(M_N) = \mathcal{L}(M_D)$. Conclusion:
  - for every NFA-λ, a DFA exists that accepts the same language
  - If the NFA contains n states, the DFA contains 2^n states

From NFA-λ to DFA: determinization

Idea to avoid "state space explosion" if this is possible

- Many of the 2^n states might be unreachable
- Alg. 5.6.3 in [Sudkamp]: only add states if necessary

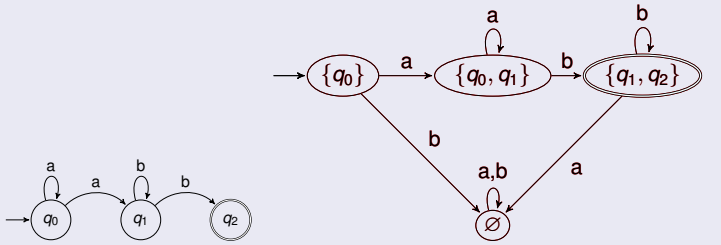
Algorithm

input: NFA-λ: $M_N = (Q, \Sigma, \delta, q_0, F)$
output: DFA $M_D = (Q', \Sigma, \Delta \cap (Q' \times Q'), \lambda\text{-closure}(q_0), F')$

$Q' := \{\lambda\text{-closure}(q_0)\}$; $U := Q'$;
while $(U \neq \varnothing)$ **do**
  choose X **from** U; $U := U - \{X\}$;
  forall $a \in \Sigma$ **do**
    $Y := \Delta(X, a)$;
    if $(Y \notin Q')$
      $Q' := Q' \cup \{Y\}$;
      $U := U \cup \{Y\}$;
$F' := \{X \in Q' \mid X \cap F \neq \varnothing\}$;

Example (1) of determinization: a^+b^+

Example



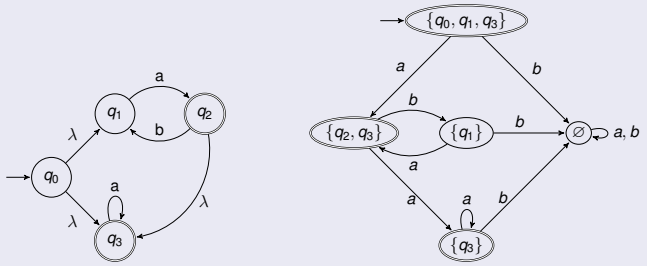
Run of the algorithm

Q'	U	X	$\Delta(X, a)$	$\Delta(X, b)$
$\{\{q_0\}\}$	$\{\{q_0\}\}$	$\{q_0\}$	$\{q_0, q_1\}$	$\varnothing$
$\{\{q_0\}, \varnothing, \{q_0, q_1\}\}$	$\{\varnothing, \{q_0, q_1\}\}$	$\varnothing$	$\varnothing$	$\varnothing$
$\{\{q_0\}, \varnothing, \{q_0, q_1\}\}$	$\{\{q_0, q_1\}\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_1, q_2\}$
$\{\{q_0\}, \varnothing, \{q_0, q_1\}, \{q_1, q_2\}\}$	$\{\{q_1, q_2\}\}$	$\{q_1, q_2\}$	$\varnothing$	$\{q_1, q_2\}$
$\{\{q_0\}, \varnothing, \{q_0, q_1\}, \{q_1, q_2\}\}$	$\varnothing$	–	–	–

Example (2) of determinization: with λ's

Input Transition Function t (almost as before)

	λ-closure	a	b
q_0	$\{q_0, q_1, q_3\}$	$\{q_2, q_3\}$	$\varnothing$
q_1	$\{q_1\}$	$\{q_2, q_3\}$	$\varnothing$
q_2	$\{q_2, q_3\}$	$\{q_3\}$	$\{q_1\}$
q_3	$\{q_3\}$	$\{q_3\}$	$\varnothing$



Contents

Regular expressions (recap)

Regular expressions over Σ are defined inductively:

- Basis: $\varnothing$, λ and a (for $a \in \Sigma$) are regular expressions
- If E_1 and E_2 are regular expressions, then $(E_1 E_2)$ and $(E_1 \cup E_2)$ and (E_1^*) are also regular expressions
- There exist no other regular expressions

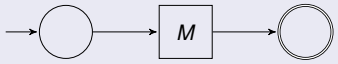
A regular expression E defines a language $\mathcal{L}(E)$ as follows:

Expression E	Language $\mathcal{L}(E)$	Alternative notation
$\varnothing$	$\varnothing$	0
λ	$\{\lambda\}$	1 (of: ε)
a	$\{a\}$	a
$E_1 \cup E_2$	$\mathcal{L}(E_1) \cup \mathcal{L}(E_2)$	$E_1 + E_2$
$E_1 E_2$	$\mathcal{L}(E_1)\mathcal{L}(E_2)$	$E_1 \cdot E_2$
E^*	$\mathcal{L}(E)^*$	E^*

Composition of NFA-λ's

Example with λ's

- With NFA-λ we have sufficient ingredients to compose automata.
- To simplify the composition, we work with composable NFA-λ, this means of the following form:
  1. The initial state q_0 has no incoming arrows (in-degree 0)
  2. There is a unique final state
  3. The final state has no outgoing arrows (out-degree is 0)
- This can be visualized as:



- You can always make an NFA composable by introducing a new initial and final state, with some λ-steps

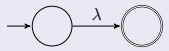
From Regular Expressions to NFA-λ

RE to NFA

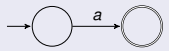
- Theorem: For every regular expression E an equivalent NFA-λ exists, M, such that $\mathcal{L}(E) = \mathcal{L}(M)$.
- Proof: this can even be done with composable NFA-λ's. Induction over the construction of E.

Base case: $\varnothing$

Base case: λ



Base case: a



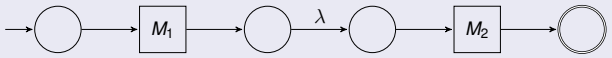
Regular Expressions to NFA-λ: Induction step

Induction step

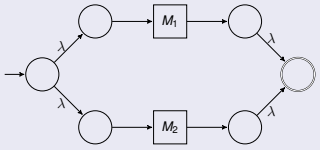
- Suppose: we already have composable M_1 and M_2 for E_1 and E_2, with $\mathcal{L}(M_1) = \mathcal{L}(E_1)$ and $\mathcal{L}(M_2) = \mathcal{L}(E_2)$ (induction hypothesis).

Composable NFA-λ's for E_1E_2, $E_1 \cup E_2$ and E_1^*

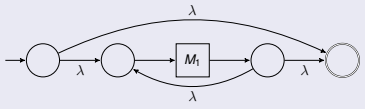
E_1E_2:



$E_1 \cup E_2$:



E_1^*:



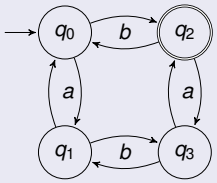
Contents

Example: even number of a's, odd number of b's

Complex example for regular expressions

- Try to write this as a regular expression (exercise *2.38 from the book)
- Write this as a DFA (example 5.3.7 from the book)
- Translate the DFA to a regular expression (exercise 6.3)

Reasonably simple DFA



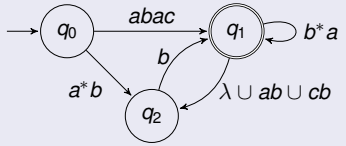
From automata to regular expressions

The global idea

- Ensure that there is one accepting state (through extra λ's)
- Omit states step by step (elimination)
- This complicates the labels: regular expressions
- At the end, there are at most two states left:
  one initial and one accepting state; then retrieve the expression.

Expression graphs (= extended automata)

- NFA's with arbitrary regular expressions on the transitions.



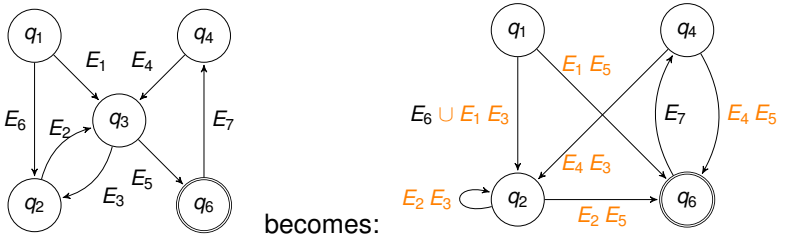
- Note: Every NFA-λ is a special expression graph

Elimination of states (3 cases)

Before elimination	After elimination
If there is not a loop on q_i:	
$q_j \xrightarrow{E_1} q_i \xrightarrow{E_2} q_k$	$q_j \xrightarrow{E_1\,E_2} q_k$
If there is a loop on q_i:	
$q_j \xrightarrow{E_1} q_i \xrightarrow{E_2} q_k$, $q_i \xrightarrow{E_3} q_i$	$q_j \xrightarrow{E_1\,E_3^*\,E_2} q_k$
Combining parallel transitions:	
$q_j \xrightarrow{E_1} q_k$, $q_j \xrightarrow{E_2} q_k$	$q_j \xrightarrow{E_1 \cup E_2} q_k$

Elimination of states (procedure)

- Remove all nodes q_i one by one, that are not initial or accepting.
- To remove q_i, you have to:
  - Consider all combinations $q_j \to q_i \to q_k$ with $j, k \neq i$;
  - Even those where $j = k$.
  - Add an arrow/label with $E_{ji}E_{ik}$ of $E_{ji}E_{ii}^*E_{ik}$



From Automata to Regular Expressions: final chord

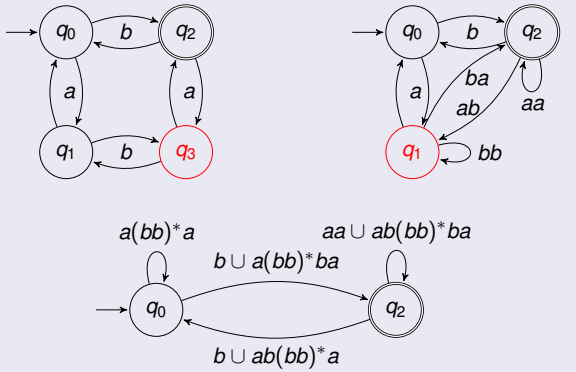
- Expression graphs with 1 or 2 states can be directly converted into a regular expression. Some examples:

Automaton	Regular expression
q_0, E_1	E_1^*
$q_0 \xrightarrow{E_2} q_1$, E_3	$E_2\,E_3^*$
$q_0 \xrightarrow{E_1} q_1$, E_2, E_3, E_4	$E_1^*E_2(E_3 \cup E_4E_1^*E_2)^*$

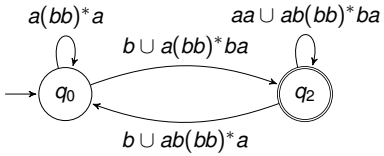
Example: even number of a's, odd number of b's

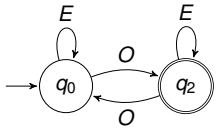
Complex example

- Try to write as regular expression (exercise 2.38 from the book)
- Write this as a DFA (example 5.3.7 from the book)
- Translate the DFA to a regular expression (exercise 6.3)



Example: even number of a's, odd number of b's





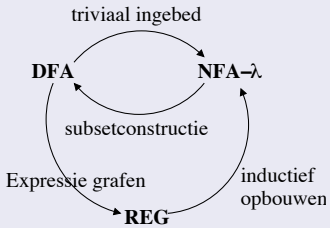
Retrieve the regular expression (with some use of imagination)

- Define $E := a(bb)^*a$ and $O = b \cup ab(bb)^*a$
- Use equivalences: $b(bb)^* = (bb)^*b$ and $aa \cup ab(bb)^*ba = aa \cup a(bb)^+a = a(bb)^*a$
- Then the requested expression is: $E^*O(E \cup OE^*O)^*$

Contents

Conclusions so far: Regular Languages

- We define:
  - **REG** the class of regular languages (= definable with regexp)
  - **DFA** the class of DFA-definable languages
  - **NFA** the class of NFA-λ definable languages
- The following diagram shows that:
  - The class **DFA** is contained in the class **REG**.
  - The class **REG** is contained in the class **NFA**.
  - So, the classes **DFA**, **NFA** and **REG** are all equal!

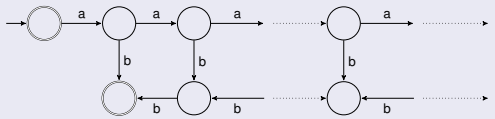


Open problems

We are still going to solve the following open problems

- How can you minimalize DFAs? (project: Partition Refinement)
- Do non-regular languages exist; prove how? (Pumping Lemma)
- How should you specify non-regular languages?(Grammars)

example: $\{a^n b^n \mid n \geq 0\}$ is non-regular



So what? . . . . . Recursion in natural and artificial languages!

- Are the braces correct? ((((()))(()()))(()))())
- Mathematical expressions, valid Python program, . . .
- I beat the dog that gets the stick that is on the ground with another stick that it brought to me before.