Lecture 3 – Nondeterminism, closure properties

NTIN071 Automata and Grammars

Jakub Bulín (KTIML MFF UK)

Spring 2025

** Adapted from the Czech-lecture slides by Marta Vomlelová with gratitude. The translation, some modifications, and all errors are mine.*

Recap of Lecture 2

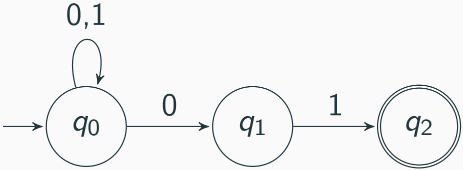
- Mihyll–Nerode theorem (DFAs $\leftrightarrow$ right congruences of Σ^* of finite index where L is a union of classes)
- Equivalent automata (recognize the same language), automata homomorphism (implies automata equivalence).
- Finding reachable states: BFS on the state diagram
- Finding equivalent (indistinguishable) states: a table-filling algorithm
- Testing equivalence of DFAs, equality of regular languages
- Reduced (minimum–state) DFA, an algorithm to reduce a given DFA (using the equivalent states algorithm)

1.6 Nondeterminism

Nondeterministic finite automata

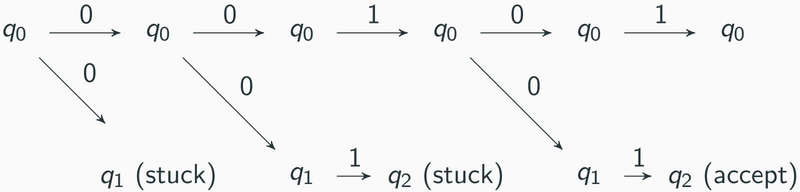
- more general but still recognize only regular languages
- can be in multiple states at once
- able to "guess" information about the input
- smaller representation, easier to construct
- but harder to test acceptance
- can be converted to a DFA (subset construction, worst case exponentially larger)

Example: accepting strings ending in 01



δ	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$*q_2$	$\emptyset$	$\emptyset$

Processing the input $w = 00101$:



The definition

Definition (Nondeterministic finite automation)

An NFA is a structure $A = (Q, \Sigma, \delta, S_0, F)$ consisting of:

- A finite set of states, often denoted Q.
- A finite set of input symbols, denoted Σ.
- A transition function $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ which returns a subset of Q.
- A set of starting states $S_0 \subseteq Q$ (alternatively, only $q_0 \in Q$).
- A set accepting states (final states) $F \subseteq Q$.

Extended transition function

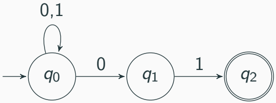
Definition (δ^* for NFA)

$\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$, i.e. takes a state q and a word w and returns a set of states, and is defined by induction:

- $\delta^*(q, \epsilon) = \{q\}$.
- $\delta^*(q, ua) = \bigcup_{p \in \delta^*(q,u)} \delta(p, a)$ for $u \in \Sigma^*, a \in \Sigma$

(That is, it outputs the set of states to which there exists some path from q with edges labelled w.)

$\delta^*(q_0, \epsilon)$	$=$		$=\{q_0\}$
$\delta^*(q_0, 0)$	$=$	$\delta(q_0, 0)$	$=\{q_0, q_1\}$
$\delta^*(q_0, 00)$	$=$	$\delta(q_0, 0) \cup \delta(q_1, 0)$	$=\{q_0, q_1\}$
$\delta^*(q_0, 001)$	$=$	$\delta(q_0, 1) \cup \delta(q_1, 1)$	$=\{q_0, q_2\}$
$\delta^*(q_0, 0010)$	$=$	$\delta(q_0, 0) \cup \delta(q_2, 0)$	$=\{q_0, q_1\}$
$\delta^*(q_0, 00101)$	$=$	$\delta(q_0, 1) \cup \delta(q_1, 1)$	$=\{q_0, q_2\}$



The language recognized

Definition (Language of an NFA)

The language recognized by an NFA $A = (Q, \Sigma, \delta, S_0, F)$:

$$L(A) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \emptyset \text{ for some } q_0 \in S_0\}$$

That is, we can get from some starting to some accepting state.

Example

The NFA from above indeed recognizes $L = \{w \mid w \text{ ends in } 01\}$.
Prove by induction that $\delta^*(q_0, w)$:

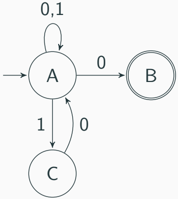
- contains q_0 for every w
- contains q_1 iff w ends in 0
- contains q_2 iff w ends in 01

Remarks

- Abusing notation, for $S \subseteq Q$ we could (but won't) write $\delta^*(S, w)$ meaning $\bigcup_{q \in S} \delta^*(q, w)$. Then we would have:

$$\delta^*(S, ua) = \delta(\delta^*(S, u), a)$$
$$L(A) = \{w \in \Sigma^* \mid \delta^*(S_0, w) \cap F \neq \emptyset\}$$

- The indistinguishable states/reduction algorithm fails for NFA:



we can remove C but $\{A, C\}$ are distinguishable by $w = 0$

- Minimizing NFA is not easy, we could use exhaustive search

Computation graph of a [D/N]FA

- a configuration is a pair (q, v) where $q \in Q$ is the current state and $v \in \Sigma^*$ is the remaining (unread) input
- the computation graph has all configurations as nodes and its oriented edges denote possible 1-step transitions, i.e. for NFA:

$$(p, au) \to (q, u) \;\text{ iff }\; q \in \delta(p, a)$$

- accept iff path from some initial to some accepting config
- useful theoretical concept, not to be explicitly constructed
- later for other types of automata (configs more complex)
- similarly the computation tree for input w: root is (q_0, w), nodes labelled by configs (but do not identify same labels)

Equivalence of NFA and DFA

Every DFA $D = (Q, \Sigma, \delta, q_0, F)$ can be trivially transformed to an equivalent NFA $N = (Q, \Sigma, \delta', \{q_0\}, F)$, where $\delta'(q, a) = \{\delta(q, a)\}$

Every NFA can also be transformed to an equivalent DFA albeit with a different, potentially exponentially bigger set of states: using the subset construction

Why NFA? Easier to design, usually no need to explicitly transform.

Subset construction

Given $N = (Q_N, \Sigma, \delta_N, S_0, F_N)$ construct $D = (Q_D, \Sigma, \delta_D, q_0, F_D)$

- $Q_D = \mathcal{P}(Q_N)$ (all subsets of Q_N) or discard those that would be unreachable: start constructing from the initial state
- $\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$ for $S \subseteq Q_N$, $a \in \Sigma$
- $q_0 = S_0$ (which is an element of Q_D)
- $F_D = \{S \subseteq Q_N \mid S \cap F_N \neq \emptyset\}$ (accept if contains accepting)

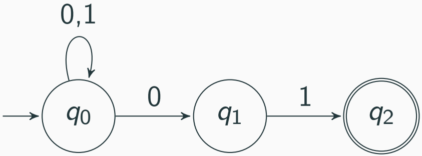
Theorem

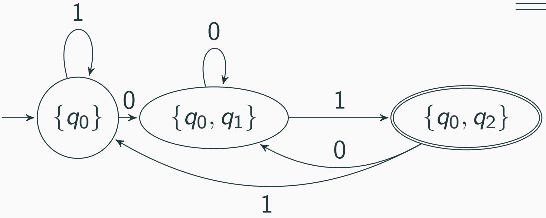
The resulting DFA D is indeed equivalent to the original NFA N.

Proof.

By induction, show that $\delta^*_D(q_0, w) = \bigcup_{q \in S_0} \delta^*_N(q, w)$. $\square$

Example of the subset construction





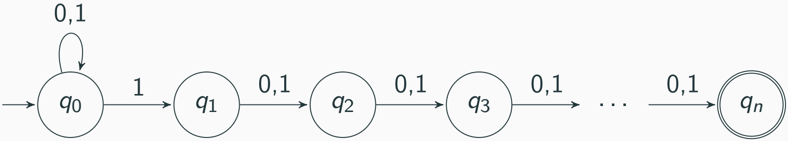
	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$*\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$*\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Sometimes it blows up

Example (Hard case for the subset construction)

Words over $\{0, 1\}$ where the nth symbol from the end is 1.

Intuitively, a DFA must remember the last n symbols it has read.



Exercise

Prove that any DFA recognizing the langauge has $\Omega(2^n)$ states.

(Hint: Use the Myhill–Nerode theorem.)

Adding ϵ-transitions

It is sometimes useful to further generalize NFAs by allowing ϵ-transitions, i.e., change state without reading any input symbol.

In an ϵ-NFA, the transition function is $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$

The subset construction still works, if we restrict to subsets closed under ϵ-transitions.

Definition (ϵ-NFA)

A ϵ-NFA is $E = (Q, \Sigma, \delta, S_0, F)$, where all components have the same interpretation as for NFAs, except that δ is now a function that takes arguments from $Q \times (\Sigma \cup \{\epsilon\})$. (We require $\epsilon \notin \Sigma$.)

Example: decimal numbers

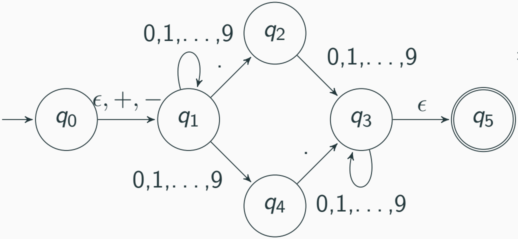
(1) Optionally a + or - sign, then

(2) a string of digits, then

(3) a decimal point, and

(4) another string of digits.

At least one of strings (2) and (4) must be nonempty.



	ϵ	'+','-'	'.'	'0','1',. . . ,'9'
q_0	$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_1, q_4\}$
q_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_3	$\{q_5\}$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_4	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$
q_5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

ϵ-closure and δ^*

For $S \subseteq Q$ define the ϵ-closure of S recursively as follows:

- $S \subseteq \epsilon CLOSE(S)$
- if $p \in \epsilon CLOSE(S)$ and $r \in \delta(p, \epsilon)$ then $r \in \epsilon CLOSE(S)$

The extended transition function is then naturally defined:

Definition (δ^* for ϵ-NFA)

For an ϵ-NFA $E = (Q, \Sigma, \delta, S_0, F)$ define δ^* inductively:

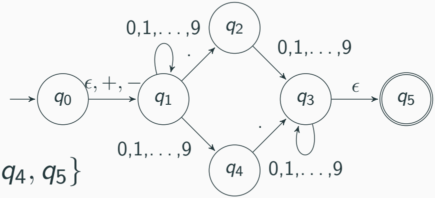
- $\delta^*(q, \epsilon) = \epsilon CLOSE(\{q\})$
- $\delta^*(q, ua) = \epsilon CLOSE\left(\bigcup_{p \in \delta^*(q,u)} \delta(p, a)\right)$ for $u \in \Sigma^*, a \in \Sigma$

$\delta^*(q, w)$ still means states we can be in if start from q and read w

Example continued

ϵ-closure:

- $\epsilon CLOSE(\{q_0\}) = \{q_0, q_1\}$
- $\epsilon CLOSE(\{q_1\}) = \{q_1\}$
- $\epsilon CLOSE(\{q_3\}) = \{q_3, q_5\}$
- $\epsilon CLOSE(\{q_3, q_4\}) = \{q_3, q_4, q_5\}$



Extended transition function: $\delta^*(q_0, 5.6)$

- $\delta^*(q_0, \epsilon) = \epsilon CLOSE(\{q_0\}) = \{q_0, q_1\}$
- $\delta^*(q_0, 5) = \epsilon CLOSE(\bigcup_{q \in \delta^*(q,\epsilon)} \delta(q, 5)) =$
  $\epsilon CLOSE(\delta(q_0, 5) \cup \delta(q_1, 5)) = \{q_1, q_4\}$
- $\delta^*(q_0, 5.) = \epsilon CLOSE(\delta(q_1, .) \cup \delta(q_4, .)) = \{q_2, q_3, q_5\}$
- $\delta^*(q_0, 5.6) = \epsilon CLOSE(\delta(q_2, 6) \cup \delta(q_3, 6) \cup \delta(q_5, 6)) = \{q_3, q_5\}$

Equivalence of ϵ-NFA and DFA

Add ϵ-closure to the subset construction:

Given an ϵ-NFA $E = (Q_E, \Sigma, \delta_E, S_0, F_E)$ construct a DFA $D = (Q_D, \Sigma, \delta_D, q_0, F_D)$

- $Q_D = \{S \subseteq Q_E \mid S = \epsilon CLOSE(S)\}$, i.e., only ϵ-closed subsets
- $\delta_D(S, a) = \epsilon CLOSE(\bigcup_{p \in S} \delta_E(p, a))$
- $q_0 = \epsilon CLOSE(S_0)$
- $F_D = \{S \in Q_D \mid S \cap F_E \neq \emptyset\}$, i.e., ϵ-closed subsets containing some accepting state

Theorem

A language L is recognized by an ϵ-NFA, iff L is regular.

(Proof similar as for NFA.)

1.7 Closure properties

Set operations

Set operations preserving regularity

For languages L, M:

- complement $\overline{L} = -L = \{w \mid w \notin L\} = \Sigma^* \setminus L$
- intersection $L \cap M = \{w \mid w \in L \text{ and } w \in M\}$
- union $L \cup M = \{w \mid w \in L \text{ or } w \in M\}$
- difference $L - M = \{w \mid w \in L \text{ and } w \notin M\}$

Theorem (Closure under set operations)

Given a pair of regular languages L and M, the languages $\overline{L}$, $L \cap M$, $L \cup M$, and $L - M$ are also regular.

Note: union/intersection of infinitely many regular languages is generally not regular!

Proof

We can assume that L, M are over the same alphabet Σ. Let $L = L(A), M = L(B)$ for DFA A, B. Ensure that their transition functions are total (if not, add a fail state).

- complement: accepted by the DFA A' obtained from A by switching accepting and nonaccepting states: $F_{A'} = Q_A \setminus F_A$
- intersection: accepted by the product automaton

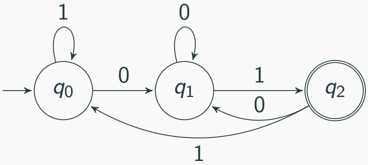
$$C = A \times B = (Q_A \times Q_B, \Sigma, \delta_C, (q_{0A}, q_{0B}), F_A \times F_B)$$
$$\delta_C((q_A, q_B), a) = (\delta_A(q_A, a), \delta_B(q_B, a))$$

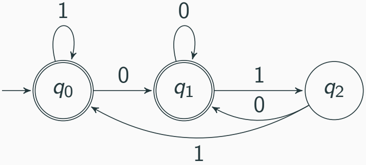
- union: by De Morgan laws, $L \cup M = \overline{\overline{L} \cap \overline{M}}$
- difference: $L - M = L \cap \overline{M}$ $\square$

Note: For union and difference we can also directly construct the product automaton but with $F_C = (F_A \times Q_B) \cup (Q_A \times F_B)$ and $F_C = F_A \times Q_B$, respectively.

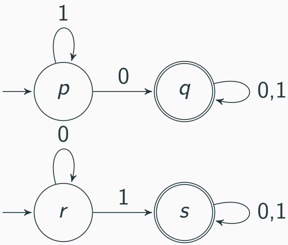
Example of the constructions

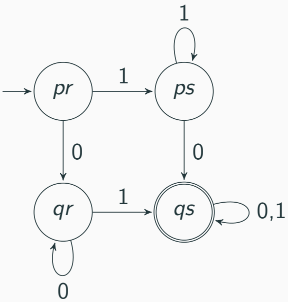
Complement: words not ending in 01





Intersection: words containing both 0 and 1





Applications

Example

Accepting words with $3k+2$ of 1's and no substring 11.

- The direct construction is complicated.
- $L_1 = \{w \in \{0,1\}^* \mid |w|_1 = 3k+2\}$
- $L_2 = \{w \in \{0,1\}^* \mid w = u11v \text{ for some } u, v \in \{0,1\}^*\}$
- $L = L_1 - L_2$.

Example

The language $M = \{w \in \{0,1\}^* \mid |w|_0 \neq |w|_1\}$ is not regular.

- If M is regular, then $\overline{M}$ is also regular.
- We know $\overline{M}$ is not regular (Pumping lemma).
- So M cannot be regular.

One more application

Example

The language $L_{0\neq 1} = \{0^i 1^j \mid i \neq j, i, j \in \mathbb{N}\}$ is not regular:

- The language $L_{01} = \{0^i 1^j \mid i, j \in \mathbb{N}\}$ is regular (we can construct a DFA directly).
- A difference of two regular languages is regular.
- L_{01} is regular. Assume that $L_{0\neq 1}$ is regular, then $L_{01} - L_{0\neq 1} = \{0^i 1^i \mid i \in \mathbb{N}\}$ is also regular.
- But it is not regular (Pumping lemma)—a contradiction.

String operations

String operations preserving regularity

- concatenation $L.M = \{uv \mid u \in L \text{ and } v \in M\}$, we also write $L.w = L.\{w\}$ and $w.L = \{w\}.L$ for $w \in \Sigma^*$
- powers of languages $L^0 = \{\epsilon\}$, $L^{i+1} = L^i.L$
- iteration $L^* = L^0 \cup L^1 \cup L^2 \ldots = \bigcup_{i \geq 0} L^i$
- positive iteration $L^+ = L^1 \cup L^2 \ldots = \bigcup_{i \geq 1} L^i$
  that is, $L^* = L^+ \cup \{\epsilon\}$
- reverse $L^R = \{u^R | u \in L\}$, $(x_1 x_2 \ldots x_n)^R = x_n x_{n-1} \ldots x_2 x_1$
- left quotient of L with M, $M \setminus L = \{v | uv \in L \text{ and } u \in M\}$
- left derivation of L with w, $\partial_w L = \{w\} \setminus L$
- (right) quotient of L with M, $L/M = \{u | uv \in L \text{ and } v \in M\}$
- right derivation of L with $w \partial_w^R L = L/\{w\}$

Regular languages are closed under those

Theorem (Closure under string operations)

Given a pair of regular languages L and M, the languages $L.M$, L^*, L^+, L^R, $M \backslash L$, and L/M are also regular.

We assume that we have DFA for L and M with disjoint sets of states and that any newly added states are indeed new (otherwise rename states)

We give constructions of ϵ-NFA or NFA for each operation.

Proof for concatenation $L.M$

Let $A_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be DFA such that $L = L(A_1)$ and $M = L(A_2)$. Assume that δ_1, δ_2 are total functions and $Q_1 \cap Q_2 = \emptyset$.

Define an ϵ-NFA $B = (Q_1 \cup Q_2, \Sigma, \delta, \{q_1\}, F_2)$, where:

- $\delta(q, a) = \{\delta_1(q, a)\}$ for $q \in Q_1, a \in \Sigma$,
- $\delta(q, a) = \{\delta_2(q, a)\}$ for $q \in Q_2, a \in \Sigma$,
- $\delta(q, \epsilon) = \{q_2\}$ for $q \in F_1$

(And $\delta(q, x) = \emptyset$ in all other cases.)

It is straightforward to verify that $L(B) = L(A_1).L(A_2)$. $\square$

Proof for iteration L^*, L^+

Let $A = (Q, \Sigma, \delta, q_0, F)$ be DFA such that $L = L(A)$.

- **Idea:** repeated computation of $A = (Q, \Sigma, \delta, q_0, F)$, a nondeterministic decision whether to restart or continue.
- New (re-)start state q_R, accepting for L^* but not for L^+.

Define an NFA $B = (Q \,\dot\cup\, \{q_R\}, \Sigma, \delta_B, \{q_R\}, F_B)$ where:

- $F_B = F \cup \{q_R\}$ for L^*, $F_B = F$ for L^+
- $\delta_B(q_R, \epsilon) = \{q_0\}$ — start computation on A
- $\delta_B(f, \epsilon) = \{q_R\}$ for $f \in F$ — nondeterministic restart
- $\delta_B(q, a) = \{\delta(q, a)\}$ for $q \in Q, a \in \Sigma$ — computation of A

Then $L(B) = L(A)^*$ (if $q_R \in F_B$), or $L(B) = L(A)^+$ (if $q_R \notin F_B$).

□

Proof for reverse L^R

Idea: reverse edges in the state diagram; we get an NFA

Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ such that $L = L(A)$, define an NFA $B = (Q, \Sigma, \delta_B, F, \{q_0\})$, where:

$$\delta_B(q, x) = \{p \mid \delta(p, x) = q\}$$

Then for any word $w = x_1 x_2 \ldots x_n$:

$$q_0, q_1, q_2, \ldots, q_n \text{ is an accepting computation for } w \text{ in } A$$

if and only if

$$q_n, q_{n-1}, \ldots, q_2, q_1, q_0 \text{ is an accepting computation for } w^R \text{ in } B$$

$\square$

Note: L is regular if and only if L^R is regular.

Proof for quotients $M \setminus L$ and L/M

Idea for $M \setminus L$: use an automaton for A but start in states reachable from the initial state by a word in M.

Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ such that $L = L(A)$, define an NFA $B = (Q, \Sigma, \delta', S, F)$ where $\delta'(q, a) = \{\delta(q, a)\}$ where $\delta'(q, a) = \{\delta(q, a)\}$ and

$$S = \{q \in Q \mid q = \delta(q_0, u) \text{ for some } u \in M\}$$

Note: S can be found algorithmically: use $A_q = (Q, \Sigma, \delta, q_0, \{q\}))$, check if $L(A_q) \cap M \neq \emptyset$.

$v \in M \setminus L \Leftrightarrow (\exists u \in M) uv \in L$
$\Leftrightarrow (\exists u \in M)(\exists q \in Q)(\delta(q_0, u) \;\&\; \delta(q, v) \in F)$
$\Leftrightarrow (\exists q \in S)\delta(q, v) \in F$
$\Leftrightarrow v \in L(B)$

To prove the claim for L/M, note that $L/M = (M^R \setminus L^R)^R$. $\square$

Summary of Lecture 3

- Nondeterministic finite automata (NFA): can ‘guess’ the right path to accepting, computation described by a state tree.
- ϵ-transitions: allow to change states without reading any input
- Subset construction: every NFA and ϵ-NFA is equivalent to a DFA (but can be easier to design, much smaller).
- Regular languages are closed under set operations (union, intersection, complement, difference)
- And under string operations (concatenation, iteration and positive iteration, reverse, left and right quotient)