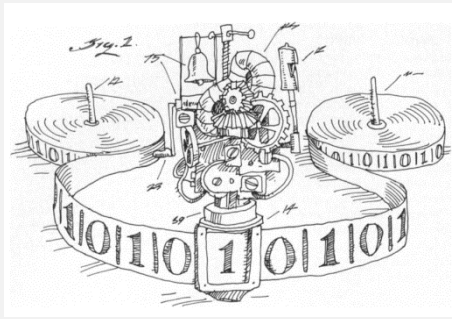
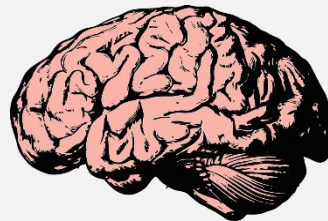
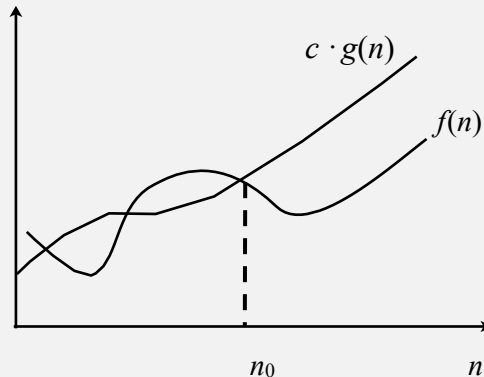


COMP 3602

Intro to Theory of Computation

Lecture 4



Logistics

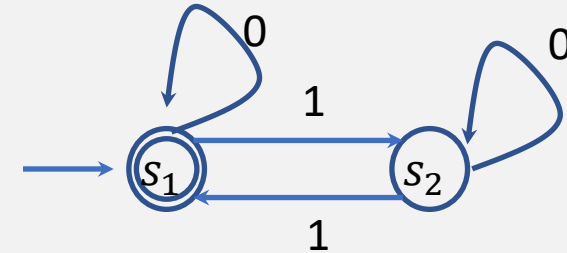
- New room: EN-2007
 - For the rest of the semester
 - From Jan 19 to Feb 6, lectures will be 4pm to 5:15pm
 - Office hours times: after lectures?
 - Both before and after OK, tell me what you prefer.
- Theory reading group
 - This week: Friday 1pm, from next week Monday 1pm.
 - In the seminar room EN-2022
- Drills are up
 - You can do Turing machines and regular expressions drills by now
 - Do the sample drills first!
 - You can do the drills multiple times, but the grade is an average of all attempts
 - So if you open it and let it time out, it will add a 0 to your average!

Questions/comments?



Definition of DFAs

- Formally: a **deterministic finite automaton (DFA)** consists of $(\Sigma, Q, q_0, F, \delta)$, where
 - Σ is the input alphabet
 - Q is the set of states with
 - q_0 the start state
 - F a set of accepting (final) states.
 - $\delta: Q \times \Sigma \rightarrow Q$ a transition function
- A DFA **accepts** a string $w = w_1 \dots w_n$ iff there is a sequence of states $r_0 \dots r_n$ in Q where
 - $r_0 = q_0$ is the start state
 - $\forall i < n \quad \delta(r_i, w_{i+1}) = r_{i+1}$
 - $r_n \in F$
- A DFA **recognizes** (accepts) a language L iff it accepts all and only strings in L .



Regular expressions $(0/1)^*$

- A regular expression R is defined recursively as follows:
 - Basis: R is a symbol a for some $a \in \Sigma$, or $R = \lambda$, or $R = \emptyset$ $()$ $\emptyset$ λ
 - Recursion: if R_1, R_2 are regular expressions, then so are $R_1 \cup R_2$, $R_1 \circ R_2$, R_1^* .
When writing regular expressions, often skip $\circ$, just write $R_1 R_2$. Also, use $|$ for $\cup$.
- Examples:
 - Binary strings with even number of 1s: $(0^* \cup 0^* 1)^* 0^*$
 - Binary strings containing a substring "00": $(0/1)^* 00 (0/1)^*$
 - Binary strings ending with a 1: $(0/1)^* 1$

Regular expressions

- A regular expression recognizes (or describes) a language (matches strings that can be built using this recursive definition).
 - The class of languages recognized by regular expressions is called **regular languages**.
- Theorem: the class of languages recognized by DFAs is exactly the class of regular languages.
 - We will do a proof later.



Definition of NFA

- A **non-deterministic finite automata (NFA)** is defined similarly as $(\Sigma, Q, q_0, F, \delta)$ except with
 - $\delta: Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q)$ as a transition function
 - Where ε is a special empty (null) symbol, allowing for a transition without reading any input symbols.
 - If no transition, $(q, s) \rightarrow \emptyset$: stop and reject.
 - Accept if there exists **some** valid sequence of states $r_0 \dots r_n$ defined as before.
- Example: strings over $\{0,1\}$ where the last symbol is 1.
 - $\Sigma = \{0,1\}$. $Q = \{q_0, q_1\}$, with q_0 the start state. $F = \{q_1\}$

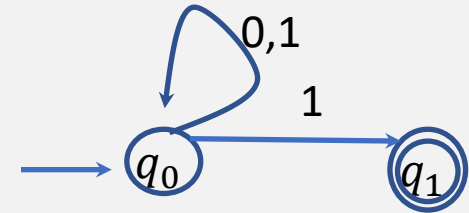


Examples of NFAs

- Example: strings over $\{0,1\}$ where the last symbol is 1.

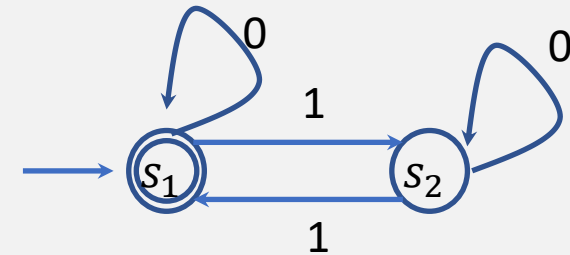
– $\Sigma = \{0,1\}$. $Q = \{q_0, q_1\}$, with q_0 the start state. $F = \{q_1\}$, $\delta =$

	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\emptyset$	$\emptyset$



- Example: strings over $\{0,1\}$ where the last symbol is 1 or with an even number of 1s.

– Recall: DFA for “even number of 1s” was



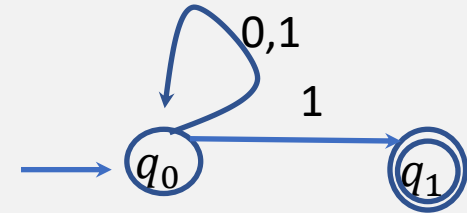


Examples of NFAs

- Example: strings over $\{0,1\}$ where the last symbol is 1.

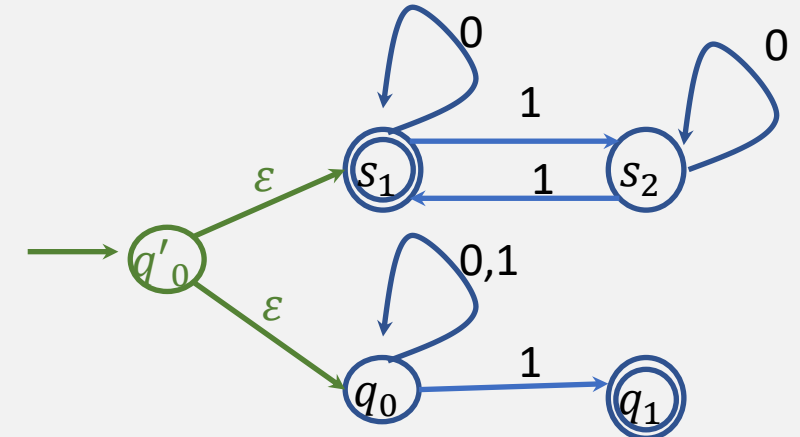
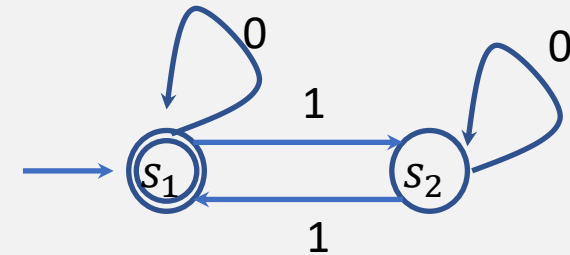
– $\Sigma = \{0,1\}$. $Q = \{q_0, q_1\}$, with q_0 the start state. $F = \{q_1\}$, $\delta =$

	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\emptyset$	$\emptyset$



- Example: strings over $\{0,1\}$ where the last symbol is 1 or with an even number of 1s.

– Recall: DFA for “even number of 1s” was





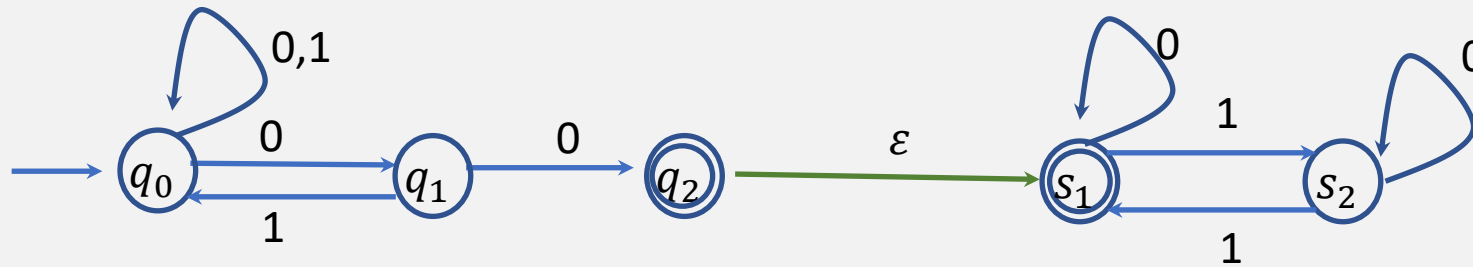
Examples of NFAs

- Example: $\{0,1\}^*00\{0^*10^*10^*\}^*$
- Example: Strings that do not contain 00.
 - Note: if you have a DFA computing a language, you can make a DFA for its complement by flipping accept/reject states
 - that is, a DFA for $\overline{L}$ is the same as a DFA for L except $F' = Q - F$
 - This does not work with NFAs!



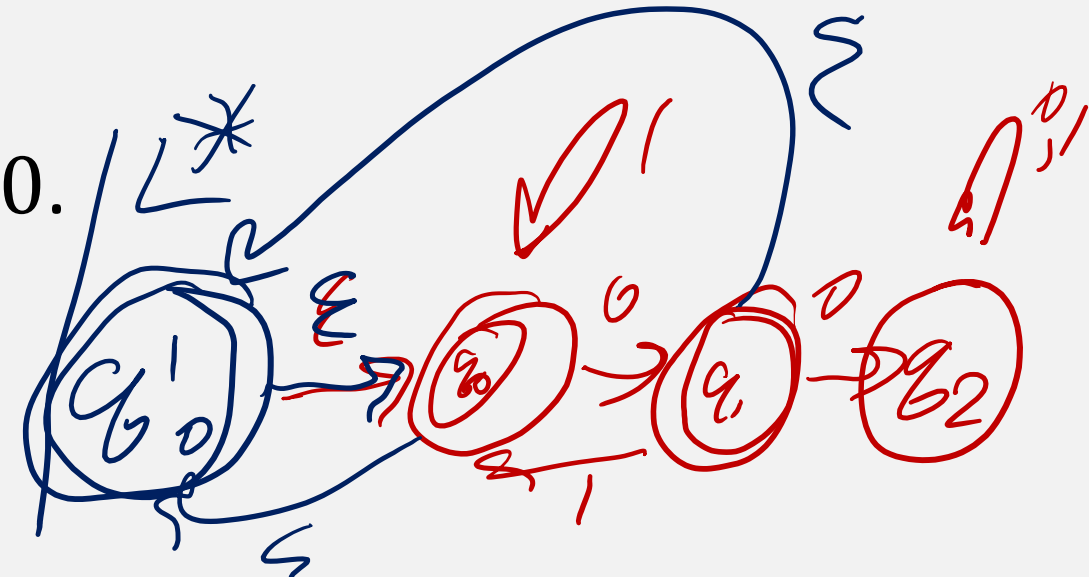
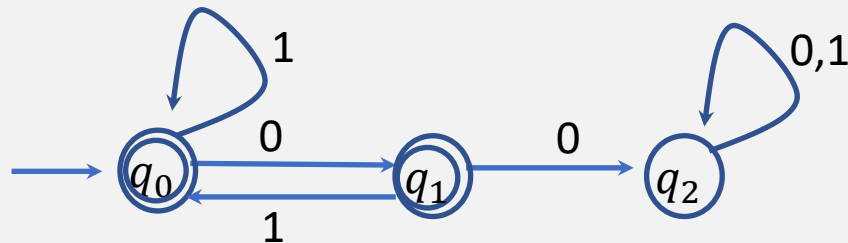
Examples of NFAs

- Example: $\{0,1\}^*00\{0^*10^*1\}^*0^*$



Here, $L^* = \{0,1\}^*$

- Example: Strings that do not contain 00.



Theorem: The class of languages recognized by NFAs is closed under regular operations.

- Let L_1, L_2 be recognized by NFAs $N_1 = (\Sigma_1, Q_1, q_0^1, \delta_1, F_1)$, $N_2 = (\Sigma_2, Q_2, q_0^2, \delta_2, F_2)$
 - Union: construct an NFA N' recognizing $L_1 \cup L_2$ from N_1 and N_2
 - Concatenation: construct an NFA N'' recognizing $L_1 \circ L_2$ from N_1 and N_2
 - Kleene star: construct an NFA N''' recognizing L_1^* from N_1

Theorem: The class of languages recognized by NFAs is closed under regular operations.

- Let L_1, L_2 be recognized by NFAs $N_1 = (\Sigma_1, Q_1, q_0^1, \delta_1, F_1)$, $N_2 = (\Sigma_2, Q_2, q_0^2, \delta_2, F_2)$
- **Union:** introduce a new (non-final) start state with ε -arrows to start states of N_1 and N_2
 - $L_1 \cup L_2$ is recognized by an NFA $N' = (\Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2 \cup \{q'_0\}, q'_0, \delta', F_1 \cup F_2)$ where $\delta' = \delta_1 \cup \delta_2 \cup \{(q'_0, \varepsilon) \rightarrow q_0^1, (q'_0, \varepsilon) \rightarrow q_0^2\}$

Theorem: The class of languages recognized by NFAs is closed under regular operations.

- Let L_1, L_2 be recognized by NFAs $N_1 = (\Sigma_1, Q_1, q_0^1, \delta_1, F_1)$, $N_2 = (\Sigma_2, Q_2, q_0^2, \delta_2, F_2)$
- **Concatenation:** Start in start state of N_1 . When reached any N_1 accept state, add an option to jump to start state of N_2 with ε -arrows. Accept when reached any accept state of N_2
 - $L_1 \circ L_2$ is recognized by $N'' = (\Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2, q_0^1, \delta', F_2)$, with $\delta' = \delta_1 \cup \delta_2 \cup \{(q, \varepsilon) \rightarrow q_0^2 \mid q \in F_1\}$

Theorem: The class of languages recognized by NFAs is closed under regular operations.

- Let L_1, L_2 be recognized by NFAs $N_1 = (\Sigma_1, Q_1, q_0^1, \delta_1, F_1)$, $N_2 = (\Sigma_2, Q_2, q_0^2, \delta_2, F_2)$
- **Kleene star:** Add a new (accepting) start and an ε -loops back to it from any accept state.
- L_1^* is recognized by an NFA $N''' = (\Sigma_1, Q_1 \cup \{q_0'\}, q_0', \delta', F_1 \cup \{q_0'\})$, with $\delta' = \delta_1 \cup \{(q_0', \varepsilon) \rightarrow q_0^1, \} \cup \{(q, \varepsilon) \rightarrow q_0' \mid q \in F_1\}$



Definitions: reminder

- A **regular expression** R is defined recursively as follows:
 - Basis: R is a symbol a for some $a \in \Sigma$, or $R = \lambda$, or $R = \emptyset$
 - Recursion: if R_1, R_2 are regular expressions, then so are $R_1 \cup R_2$, $R_1 \circ R_2$, R_1^* .
When writing regular expressions, often skip $\circ$, just write $R_1 R_2$. Also, use $|$ for $\cup$.
- A **non-deterministic finite automata (NFA)** is defined similarly as $(\Sigma, Q, q_0, F, \delta)$ except with
 - $\delta: Q \times \Sigma \cup \{\varepsilon\} \rightarrow \mathcal{P}(Q)$ as a transition function
 - Where ε is a special empty (null) symbol, allowing for a transition without reading any input symbols.
 - If no transition, $(q, s) \rightarrow \emptyset$: stop and reject.
 - Accept if there exists **some** valid sequence of states $r_0 \dots r_n$ defined as before



Finite automata compute what regular expressions match

- Each regular expression can be computed by an NFA
- Proof (structural induction)

- Base case:

- Compute the empty language: 
- Accept just the empty string: 
- Accept just the string with one symbol a: 

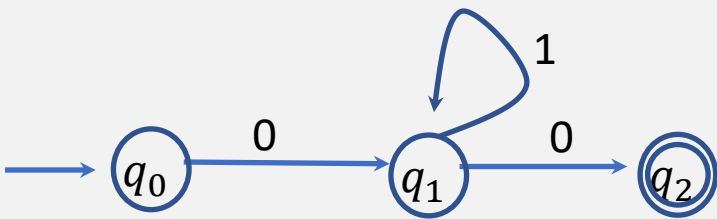
- Recursion step:

- Let regular expression R be of the form $R_1 \mid R_2$. By induction hypothesis, there are NFAs N_1 , N_2 recognizing the languages of R_1, R_2 respectively.
- Now, construct N for R out of N_1, N_2 as in the proof of the closure under regular operations.
- Same for $R = R_1 \circ R_2$ and $R = R_1^*$. That is, apply induction hypothesis and use the construction from the closure under regular operations.



Regular expressions simulate NFAs

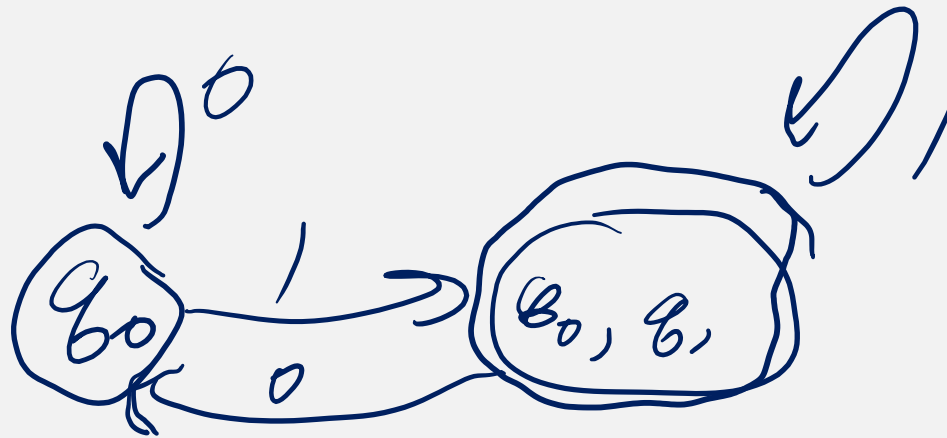
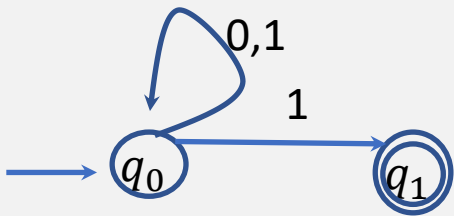
- *Theorem:* For every NFA, there is a regular expression describing the same language.
- Proof idea: given an NFA N , construct a regular expression R_N for the same language.
 - We'll skip the full proof. See Sipser's book if you'd like to read it.



Equivalence of NFAs and DFAs

- *Theorem.* For every NFA there is a DFA recognizing the same language.
 - Since a DFA is a special case of NFA, the other direction is trivial
- Proof: given an NFA $N = (\Sigma, Q, q_0, F, \delta)$, we will construct a DFA $D_N = (\Sigma, Q', q'_0, F', \delta')$ recognizing the same language.
 - Idea: “subset construction”: states of D_N will be sets of states of N .

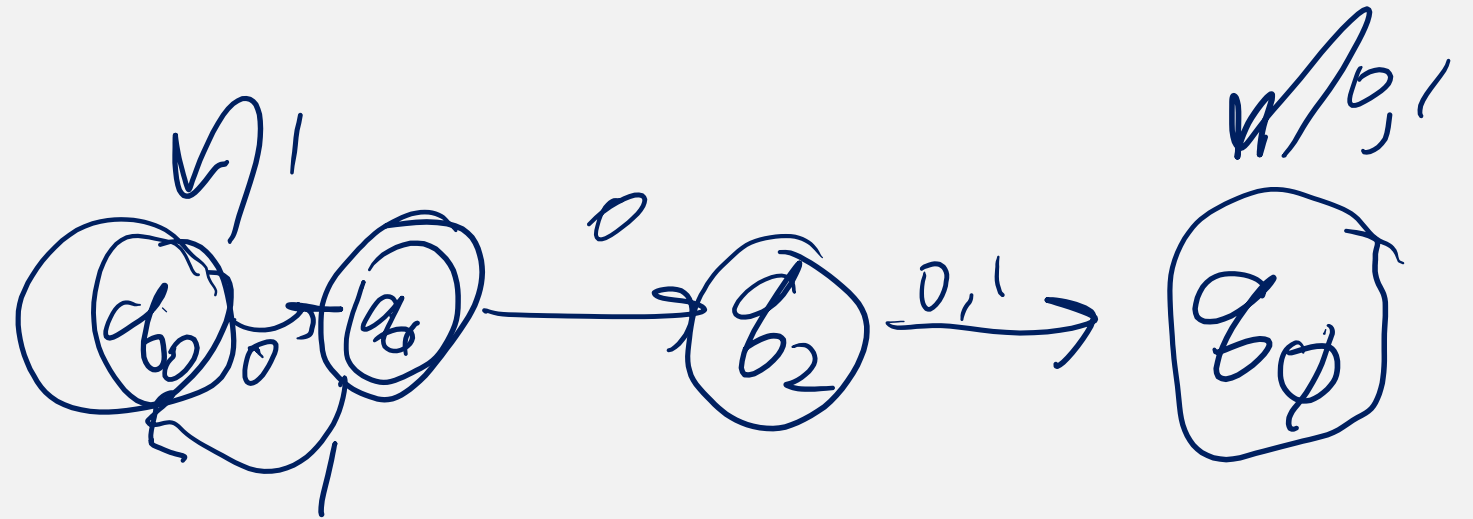
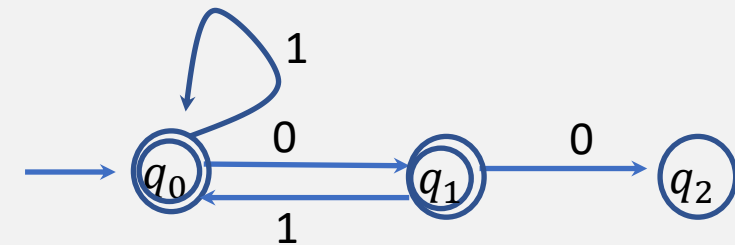
- Example:



Equivalence of NFAs and DFAs

- *Theorem.* For every NFA there is a DFA recognizing the same language.
 - Since a DFA is a special case of NFA, the other direction is trivial
- Proof: given an NFA $N = (\Sigma, Q, q_0, F, \delta)$, we will construct a DFA $D_N = (\Sigma, Q', q'_0, F', \delta')$ recognizing the same language.
 - Idea: “subset construction”: states of D_N will be sets of states of N .

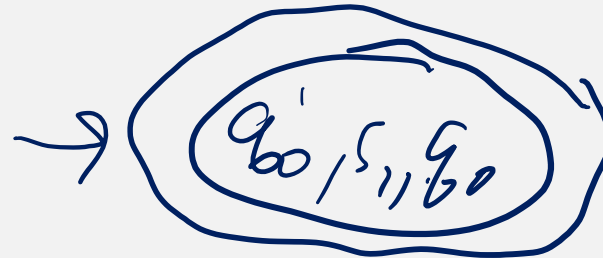
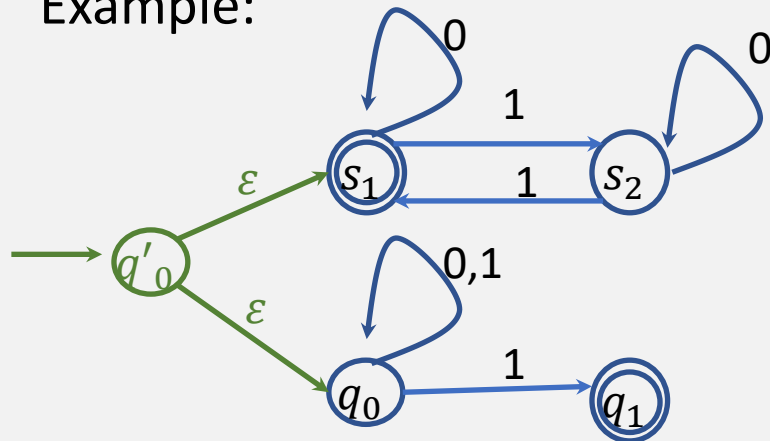
- Example:



Equivalence of NFAs and DFAs

- *Theorem.* For every NFA there is a DFA recognizing the same language.
 - Since a DFA is a special case of NFA, the other direction is trivial
- Proof: given an NFA $N = (\Sigma, Q, q_0, F, \delta)$, we will construct a DFA $D_N = (\Sigma, Q', q'_0, F', \delta')$ recognizing the same language.
 - Idea: “subset construction”: states of D_N will be sets of states of N .
 - Put all states reachable from q by (possibly a sequence of) ε -arrows into the same state of D_N as q
 - If a subset contains a final state of N , make the corresponding state of D_N final.
 - $\emptyset$ state is never final.

- Example:



Theorem. For every NFA there is a DFA recognizing the same language.

- Proof: given an NFA $N = (\Sigma, Q, q_0, F, \delta)$, we will construct a DFA $D_N = (\Sigma, Q', q'_0, F', \delta')$ recognizing the same language.

– Idea: “subset construction”: states of D_N will be sets of states of N .

- That is, $Q' \subseteq \mathcal{P}(Q)$

For a state q , let $E(q)$ be a set of all states reachable from q by following a sequence of 0 or more ε -arrows. For $S \in \mathcal{P}(Q)$, let $E(S) = \bigcup_{q \in S} E(q)$

- $q'_0 = E(q_0)$
- $F' = \{S \in Q' \mid \exists q \ q \in F \wedge q \in S\}$
- For a state $S \in Q'$, symbol $a \in \Sigma$, define $\delta'(S, a) = \bigcup_{q \in S} E(\delta(q, a))$
 - For each $q \in S$, take all states q' such that $(q, a) \rightarrow q'$ and add their $E(q')$
- Now, only include in D_N those subsets of $\mathcal{P}(Q)$ that are reachable from q'_0
 - Along any kind of arrows. Includes q'_0 itself.

Tradeoff: determinism vs. size

There are languages any DFAs for which must be exponentially larger than the smallest NFA.

- Proof (constructive):
 - $L_k = \{w \mid w_{|w|-k} = 1\}$ That is, $(k + 1)^{st}$ symbol from the end is 1.
 - NFA: $k + 1$ symbols



- Every DFA for L_k has size at least 2^k
 - Note: the subset construction gives us a DFA of size 2^{k+1}

Tradeoff: determinism vs. size

There are languages any DFAs for which must be exponentially larger than the smallest NFA.

- Proof (constructive):
 - $L_k = \{w \mid w_{|w|-k} = 1\}$ That is, $(k + 1)^{st}$ symbol from the end is 1.
 - NFA: $k + 1$ symbols
 - Every DFA for L_k has size at least 2^k
 - Note: the subset construction gives us a DFA of size 2^{k+1}

– $L_k = \{w \mid w_{|w|-k} = 1\}$ That is, $(k+1)^{st}$ symbol from the end is 1.

- Every DFA for L_k has size at least 2^k

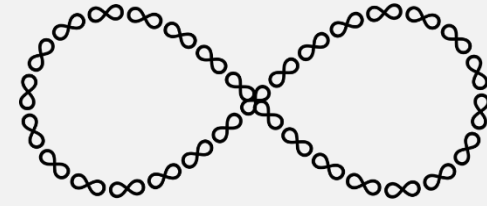
- $L_k = \{w \mid w_{|w|-k} = 1\}$ That is, $(k + 1)^{st}$ symbol from the end is 1.
- Every DFA for L_k has size at least 2^k
 - Proof: Suppose for the sake of contradiction that there is a smaller DFA.
 - By the PigeonHolePrinciple, then at least two strings of length k , w and w' , end up in the same state q .
 - Let i be an index such that $w_i \neq w'_i$
 - As $w \neq w'$, such i exists.
 - Consider strings $w0^i$ and $w'0^i$.
 - Exactly one of them is in L_k , since $w_i \neq w'_i$, and $|w0^i| - k = (k + i) - k = i$.
 - But the DFA computes exactly the same way from q on 0^i , so it will reach the same state for both $w0^i$ and $w'0^i$.
 - Contradiction!
 - Therefore, there is no DFA for this language smaller than 2^k states.



Corollaries of what we proved so far.

- Corollary 1: Regular expressions, NFAs and DFA are all equivalent:
 - They all recognize regular languages.
 - Though DFAs are exponentially larger than NFAs in the worst case.
- Corollary 2: Regular languages are closed under complementation and intersection.
 - Every language is regular iff it is computable by a DFA, and a DFA computing its complement just has $F' = Q - F$.
 - By De Morgan's law for sets, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$
 - Given D_1 and D_2 for L_1 and L_2 ,
 - first build DFAs for their complements by flipping states,
 - then take the union getting an NFA,
 - then convert that NFA into a DFA,
 - then flip the states again.

Structure of regular languages



- If a language is finite, it must be regular
 - Just make DFAs for every string in the language and take their union.
- What kind of structure do infinite regular languages have?
 - Hint: the only regular operation that can create an infinite language out of a finite one is the Kleene star.

