CS 483 - Introduction to the Theory of Computation

Lecture 2

Operations on Languages

Recall that a language is a set of strings over the alphabet.

Consider two languages A, B on the same alphabet,

- Union: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
- Kleene star: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0, x_1, x_2, \ldots, x_k \in A\}$

Operations on Languages

For instance, $A = \{\varepsilon, \text{good}, \text{bad}\}$, $B = \{\text{cat}, \text{dog}\}$

empty string

Operations on Languages

For instance, $A = \{\varepsilon, \text{good}, \text{bad}\}$, $B = \{\text{cat}, \text{dog}\}$

- Union: $A \cup B = \{\varepsilon, \text{good}, \text{bad}, \text{cat}, \text{dog}\}$

Operations on Languages

For instance, $A = \{\varepsilon, \text{good}, \text{bad}\}$, $B = \{\text{cat}, \text{dog}\}$

- Union: $A \cup B = \{\varepsilon, \text{good}, \text{bad}, \text{cat}, \text{dog}\}$
- Concatenation: $A \circ B = \{\text{cat}, \text{dog}, \text{goodcat}, \text{gooddog}, \text{badcat}, \text{baddog}\}$

Operations on Languages

For instance, $A = \{\varepsilon, \text{good}, \text{bad}\}$, $B = \{\text{cat}, \text{dog}\}$

- Union: $A \cup B = \{\varepsilon, \text{good}, \text{bad}, \text{cat}, \text{dog}\}$
- Concatenation: $A \circ B = \{\text{cat}, \text{dog}, \text{goodcat}, \text{gooddog}, \text{badcat}, \text{baddog}\}$
- Kleene star: $A^* = \{\varepsilon, \text{good}, \text{bad}, \text{goodgood}, \text{goodbad}, \text{badgood}, \text{badbad}, \text{goodgoodgood}, \text{goodgoodbad}, \dots\}$

Regular Expressions (Regex)

A regular expression is a combination of operations and basic languages.

Formally, a regular expression is one of followings:

- $\emptyset$ (empty set)
- ε (empty string)
- Any symbol from the alphabet Σ
- $R_1 \cup R_2$, where both R_1 and R_2 are regular expressions
- $R_1 \circ R_2$ (shorthanded as $R_1 R_2$), where both R_1 and R_2 are regular expressions
- R^*, where R is a regular expression

Regular Expressions (Regex)

Every regular expression R naturally corresponds to a language.

For instance, on alphabet $\Sigma = \{a, b, c\}$:

- $\emptyset \cup \varepsilon = \{\varepsilon\}$
- $\emptyset ab = \emptyset$
- $(a \cup b \cup c)^* = \Sigma^* = \{\text{all strings on the alphabet } \Sigma\}$

Regular Expressions (Regex)

Every regular expression R naturally corresponds to a language.

For instance, on alphabet $\Sigma = \{a, b, c\}$:

- $\emptyset \cup \varepsilon = \{\varepsilon\}$
- $\emptyset ab = \emptyset$
- $(a \cup b \cup c)^* = \Sigma^* = \{\text{all strings on the alphabet } \Sigma\}$
- $\Sigma^* aba \Sigma^*$

Regular Expressions (Regex)

Every regular expression R naturally corresponds to a language.

For instance, on alphabet $\Sigma = \{a, b, c\}$:

- $\emptyset \cup \varepsilon = \{\varepsilon\}$
- $\emptyset ab = \emptyset$
- $(a \cup b \cup c)^* = \Sigma^* = \{\text{all strings on the alphabet } \Sigma\}$
- $\Sigma^* aba \Sigma^* = \{w \mid w \text{ contains the consecutive substring } aba\}$

Regular Expressions (Regex)

Every regular expression R naturally corresponds to a language.

For instance, on alphabet $\Sigma = \{a, b, c\}$:

- $\emptyset \cup \varepsilon = \{\varepsilon\}$
- $\emptyset ab = \emptyset$
- $(a \cup b \cup c)^* = \Sigma^* = \{\text{all strings on the alphabet } \Sigma\}$
- $\Sigma^* aba \Sigma^* = \{w \mid w \text{ contains the consecutive substring } aba\}$
- $\Sigma^* a \Sigma^* b \Sigma^* a \Sigma^*$

Regular Expressions (Regex)

Every regular expression R naturally corresponds to a language.

For instance, on alphabet $\Sigma = \{a, b, c\}$:

- $\emptyset \cup \varepsilon = \{\varepsilon\}$
- $\emptyset ab = \emptyset$
- $(a \cup b \cup c)^* = \Sigma^* = \{\text{all strings on the alphabet } \Sigma\}$
- $\Sigma^* aba \Sigma^* = \{w \mid w \text{ contains the consecutive substring } aba\}$
- $\Sigma^* a \Sigma^* b \Sigma^* a \Sigma^* = \{w \mid w \text{ contains the substring } aba\}$

Regular Expressions (Regex)

Every regular expression R naturally corresponds to a language.

For instance, on alphabet $\Sigma = \{a, b, c\}$:

- $\emptyset \cup \varepsilon = \{\varepsilon\}$
- $\emptyset ab = \emptyset$
- $(a \cup b \cup c)^* = \Sigma^* = \{\text{all strings on the alphabet } \Sigma\}$
- $\Sigma^* aba \Sigma^* = \{w \mid w \text{ contains the consecutive substring } aba\}$
- $\Sigma^* a \Sigma^* b \Sigma^* a \Sigma^* = \{w \mid w \text{ contains the substring } aba\}$
- $(b \cup c)^* a (b \cup c)^* a (b \cup c)^*$

Regular Expressions (Regex)

Every regular expression R naturally corresponds to a language.

For instance, on alphabet $\Sigma = \{a, b, c\}$:

- $\emptyset \cup \varepsilon = \{\varepsilon\}$
- $\emptyset ab = \emptyset$
- $(a \cup b \cup c)^* = \Sigma^* = \{\text{all strings on the alphabet } \Sigma\}$
- $\Sigma^* aba \Sigma^* = \{w \mid w \text{ contains the consecutive substring } aba\}$
- $\Sigma^* a \Sigma^* b \Sigma^* a \Sigma^* = \{w \mid w \text{ contains the substring } aba\}$
- $(b \cup c)^* a (b \cup c)^* a (b \cup c)^* = \{w \mid w \text{ contains exactly two } a\}$

Regular Expressions (Regex)

Every regular expression R naturally corresponds to a language.

For instance, on alphabet $\Sigma = \{a, b, c\}$:

- $\emptyset \cup \varepsilon = \{\varepsilon\}$
- $\emptyset ab = \emptyset$
- $(a \cup b \cup c)^* = \Sigma^* = \{\text{all strings on the alphabet } \Sigma\}$
- $\Sigma^* aba \Sigma^* = \{w \mid w \text{ contains the consecutive substring } aba\}$
- $\Sigma^* a \Sigma^* b \Sigma^* a \Sigma^* = \{w \mid w \text{ contains the substring } aba\}$
- $(b \cup c)^* a (b \cup c)^* a (b \cup c)^* = \{w \mid w \text{ contains exactly two } a\}$
- $(b \cup c)^* (a (b \cup c)^* a (b \cup c)^*)^*$

Regular Expressions (Regex)

Every regular expression R naturally corresponds to a language.

For instance, on alphabet $\Sigma = \{a, b, c\}$:

- $\emptyset \cup \varepsilon = \{\varepsilon\}$
- $\emptyset ab = \emptyset$
- $(a \cup b \cup c)^* = \Sigma^* = \{\text{all strings on the alphabet } \Sigma\}$
- $\Sigma^* aba \Sigma^* = \{w \mid w \text{ contains the consecutive substring } aba\}$
- $\Sigma^* a \Sigma^* b \Sigma^* a \Sigma^* = \{w \mid w \text{ contains the substring } aba\}$
- $(b \cup c)^* a (b \cup c)^* a (b \cup c)^* = \{w \mid w \text{ contains exactly two } a\}$
- $(b \cup c)^* (a (b \cup c)^* a (b \cup c)^*)^* = \{w \mid w \text{ contains an even number of } a\}$

Regular Expressions (Regex)

The standard (POSIX) Regexes are essentially the same as the regular expressions we defined here.

For instance, the regex for all integers:

```
^-?[0-9]+$
```

can be equivalently expressed as

$$(-\cup \varepsilon)(0 \cup \cdots \cup 9)(0 \cup \cdots \cup 9)^*$$

Equivalence of Regex and DFA

Theorem. A language can be expressed as a regex **if and only if** it is regular (i.e., it can be computed by a DFA).

Equivalence of Regex and DFA

Theorem. A language can be expressed as a regex **if and only if** it is regular (i.e., it can be computed by a DFA).

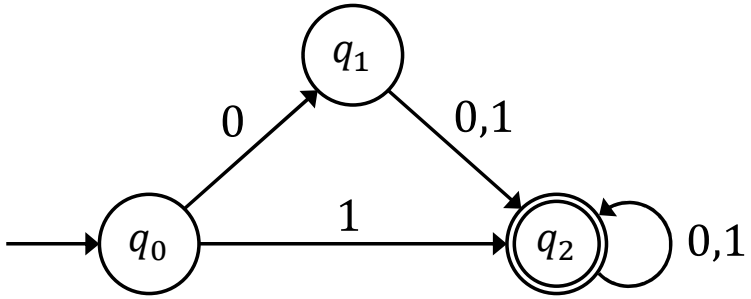
We need to prove both directions:

- For every regular language L, we can express it as a regex;
- For every regex, we can construct a DFA that computes it.

From DFA to Regex

Task: Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, construct a regex for $L(M)$.

Idea: $L(M) = \{\text{strings that corresponds to paths from } q_0 \text{ to } q \in F\}$



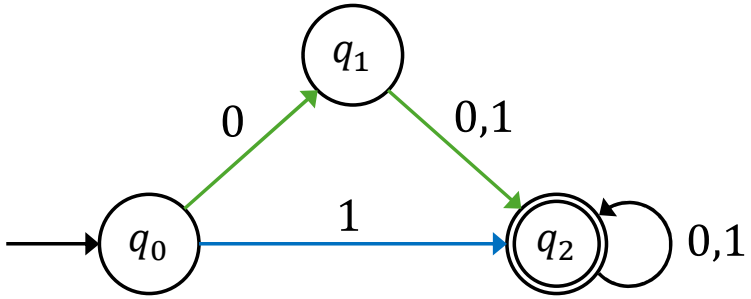
From DFA to Regex

Task: Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, construct a regex for $L(M)$.

Idea: $L(M) = \{\text{strings that corresponds to paths from } q_0 \text{ to } q \in F\}$

Paths from q_0 to q_2 are either
$q_0, q_2, (q_2, \dots)$ or $q_0, q_1, q_2, (q_2, \dots)$,

so $L(M) = 1\Sigma^* \cup 0\Sigma\Sigma^*$

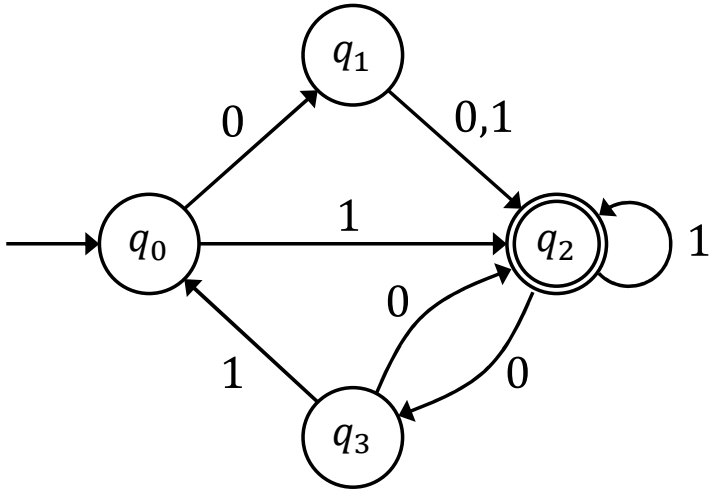


From DFA to Regex

Task: Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, construct a regex for $L(M)$.

Idea: $L(M) = \{\text{strings that corresponds to paths from } q_0 \text{ to } q \in F\}$

How do we systematically describe all the paths when M can be arbitrary?



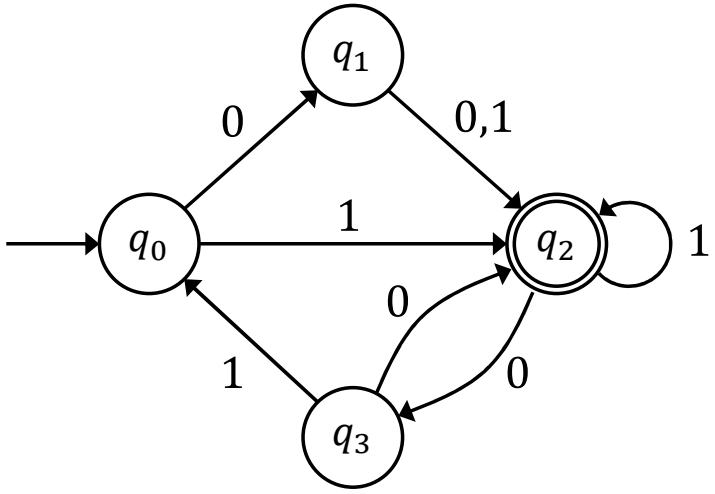
From DFA to Regex

Task: Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, construct a regex for $L(M)$.

Idea: $L(M) = \{\text{strings that corresponds to paths from } q_0 \text{ to } q \in F\}$

How do we systematically describe all the paths when M can be arbitrary?

Dynamic programming!

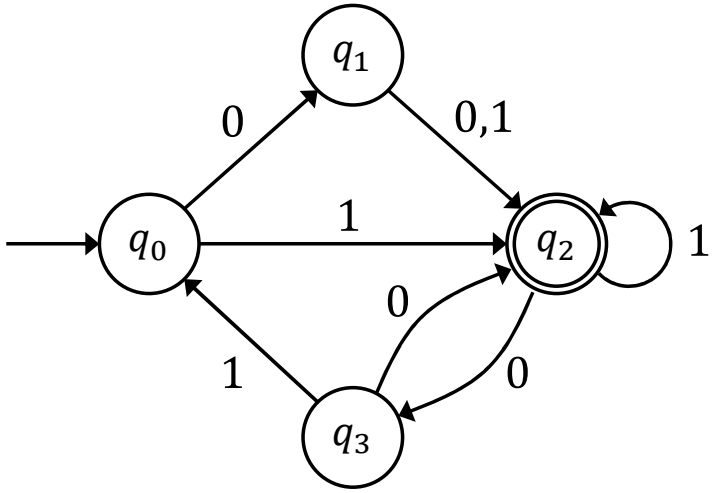


From DFA to Regex

Define $E_{i,j} = \{x \in \Sigma \mid \delta(q_i, x) = q_j\}$.

Each $E_{i,j}$ is a regex which is a union of symbols,

e.g. $E_{0,1} = 0$, $E_{2,2} = 1$, $E_{1,2} = 0 \cup 1 = \Sigma$, $E_{2,0} = \emptyset$

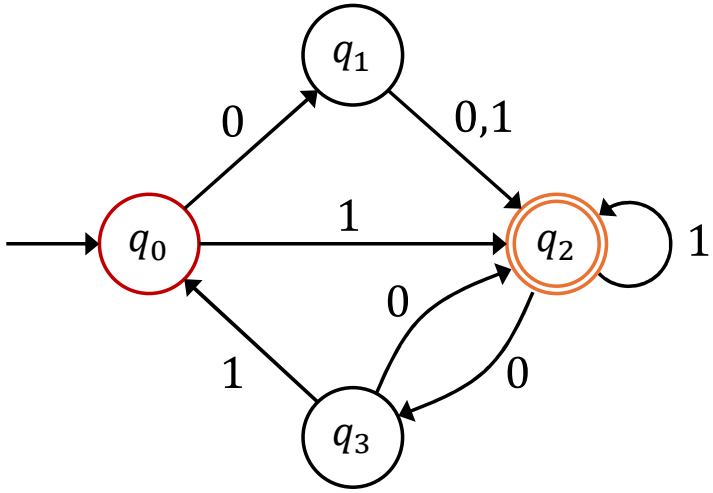


From DFA to Regex

Define $E_{i,j} = \{x \in \Sigma \mid \delta(q_i, x) = q_j\}$.

Define $R_{i,j,k}$ as the regex that corresponds to paths from q_i to q_j that goes through only states in $q_0, q_1, \ldots, q_k$, then:

- $R_{i,j,-1} = E_{i,j}$,

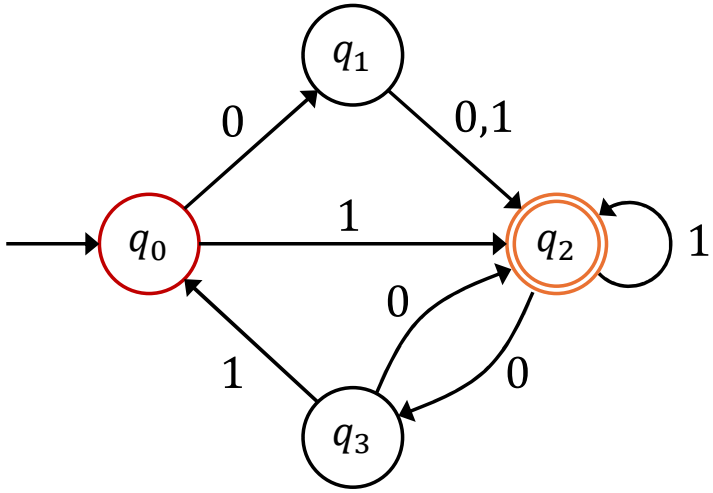


From DFA to Regex

Define $E_{i,j} = \{x \in \Sigma \mid \delta(q_i, x) = q_j\}$.

Define $R_{i,j,k}$ as the regex that corresponds to paths from q_i to q_j that goes through only states in $q_0, q_1, \ldots, q_k$, then:

- $R_{i,j,-1} = E_{i,j}$,
- $R_{i,j,k} = R_{i,j,k-1} \cup R_{i,k,k-1} R^*_{k,k,k-1} R_{k,j,k-1}$



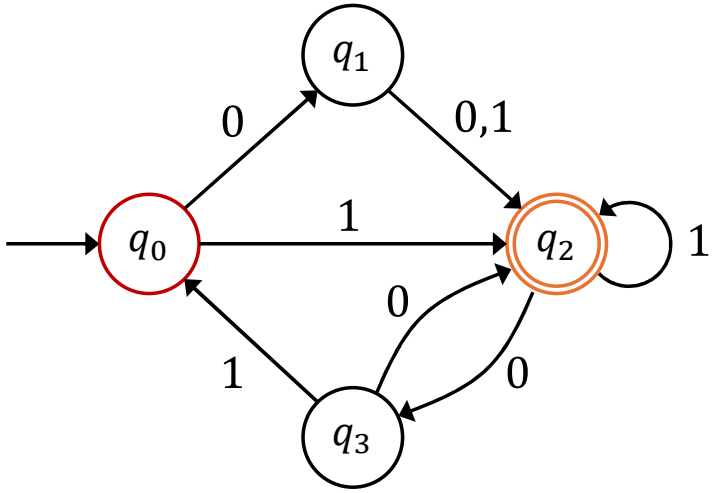
From DFA to Regex

Define $E_{i,j} = \{x \in \Sigma \mid \delta(q_i, x) = q_j\}$.

Define $R_{i,j,k}$ as the regex that corresponds to paths from q_i to q_j that goes through only states in $q_0, q_1, \ldots, q_k$, then:

- $R_{i,j,-1} = E_{i,j}$,
- $R_{i,j,k} = R_{i,j,k-1} \cup R_{i,k,k-1} R_{k,k,k-1}^{*} R_{k,j,k-1}$

if the path does not go through q_k



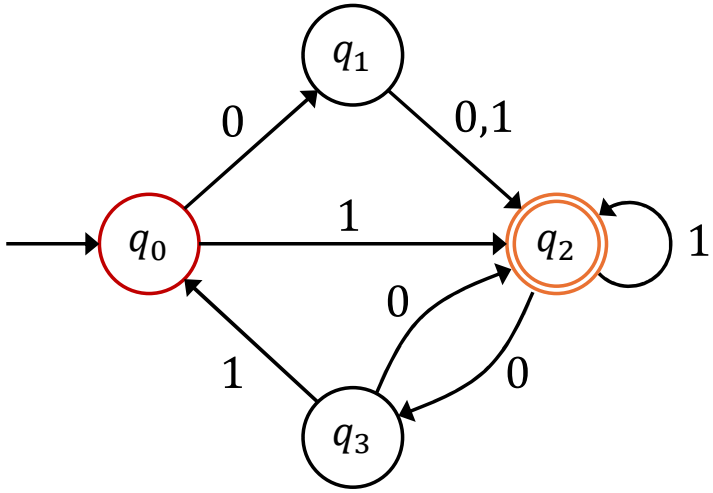
From DFA to Regex

Define $E_{i,j} = \{x \in \Sigma \mid \delta(q_i, x) = q_j\}$.

Define $R_{i,j,k}$ as the regex that corresponds to paths from q_i to q_j that goes through only states in $q_0, q_1, \ldots, q_k$, then:

- $R_{i,j,-1} = E_{i,j}$,
- $R_{i,j,k} = R_{i,j,k-1} \cup R_{i,k,k-1} R^*_{k,k,k-1} R_{k,j,k-1}$

if the path goes through q_k, so it must be $q_i, \ldots\ldots, q_k(\ldots\ldots, q_k, \ldots\ldots q_k), \ldots\ldots q_j$

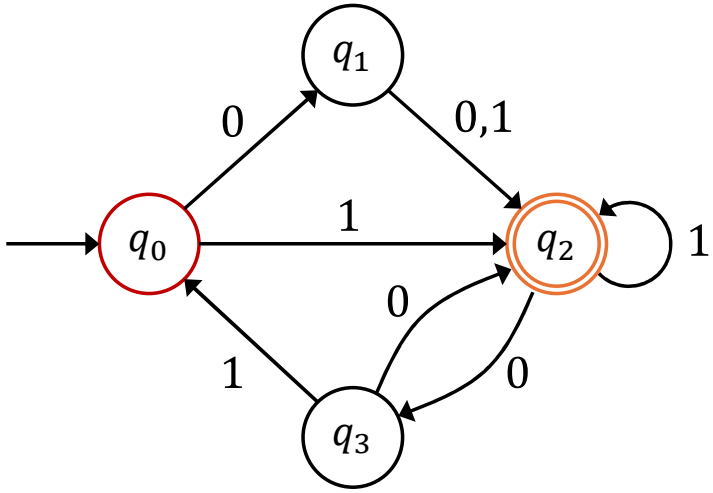


From DFA to Regex

Define $E_{i,j} = \{x \in \Sigma \mid \delta(q_i, x) = q_j\}$.

Define $R_{i,j,k}$ as the regex that corresponds to paths from q_i to q_j that goes through only states in $q_0, q_1, \ldots, q_k$, then:

- $R_{i,j,-1} = E_{i,j}$,
- $R_{i,j,k} = R_{i,j,k-1} \cup R_{i,k,k-1} R^*_{k,k,k-1} R_{k,j,k-1}$
- $L(M)$ is the union of $R_{0,j,|Q|-1}$ for $q_j \in F$



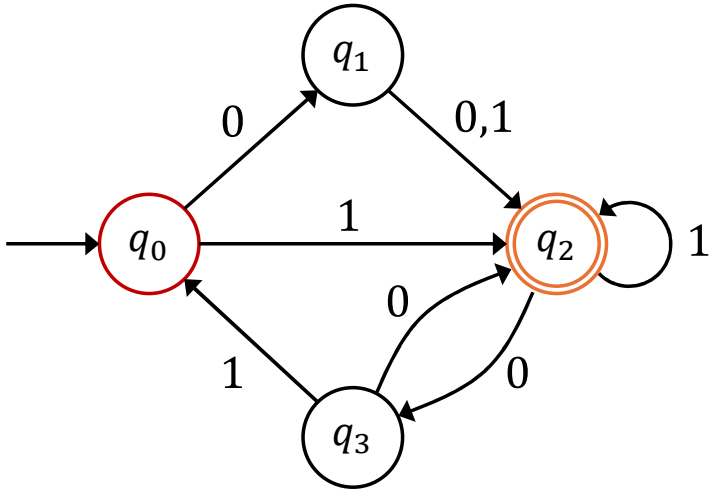
From DFA to Regex

Define $E_{i,j} = \{x \in \Sigma \mid \delta(q_i, x) = q_j\}$.

Define $R_{i,j,k}$ as the regex that corresponds to paths from q_i to q_j that goes through only states in $q_0, q_1, \ldots, q_k$, then:

- $R_{i,j,-1} = E_{i,j}$,
- $R_{i,j,k} = R_{i,j,k-1} \cup R_{i,k,k-1} R^*_{k,k,k-1} R_{k,j,k-1}$
- $L(M)$ is the union of $R_{0,j,|Q|-1}$ for $q_j \in F$

The idea is the same as Floyd-Warshall.



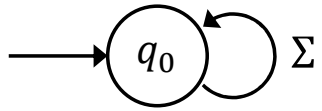
From Regex to DFA

Task: Given a regex R, construct a DFA M such that $L(M) = R$.

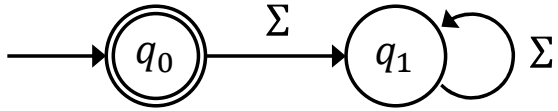
Idea: Since regexes are recursively defined, we should use induction (for example, on the length of R).

From Regex to DFA: Base Cases

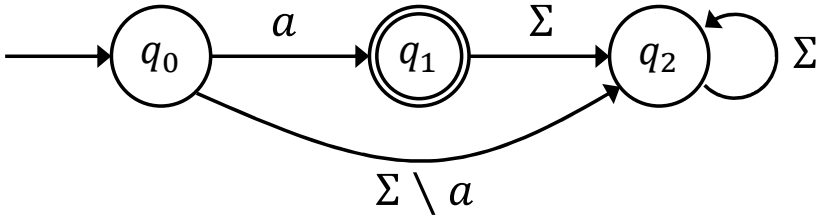
- If $R = \emptyset$:



- If $R = \varepsilon$:



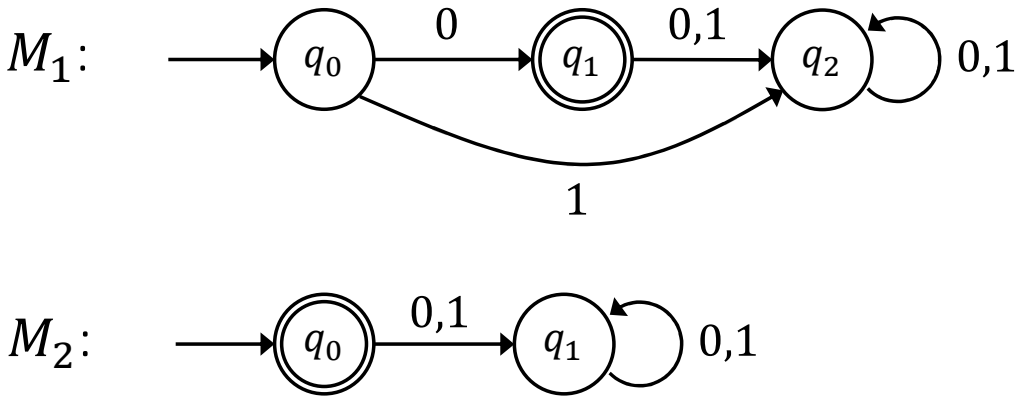
- If $R = a$ for some $a \in \Sigma$:



From Regex to DFA: Union

Suppose that R_1 is computed by M_1 and R_2 is computed by M_2.

How to construct a DFA M that computes $R_1 \cup R_2 = L(M_1) \cup L(M_2)$?

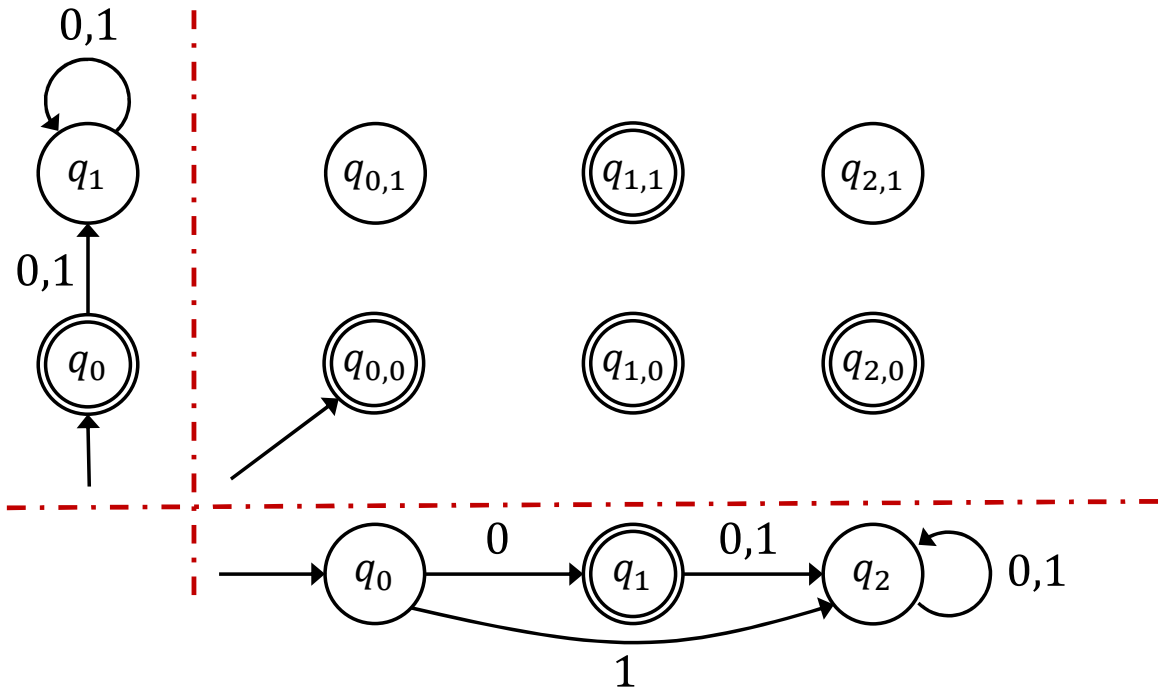


From Regex to DFA: Union

Suppose that R_1 is computed by M_1 and R_2 is computed by M_2.

How to construct a DFA M that computes $R_1 \cup R_2 = L(M_1) \cup L(M_2)$?

M simulates M_1 and M_2 in parallel.

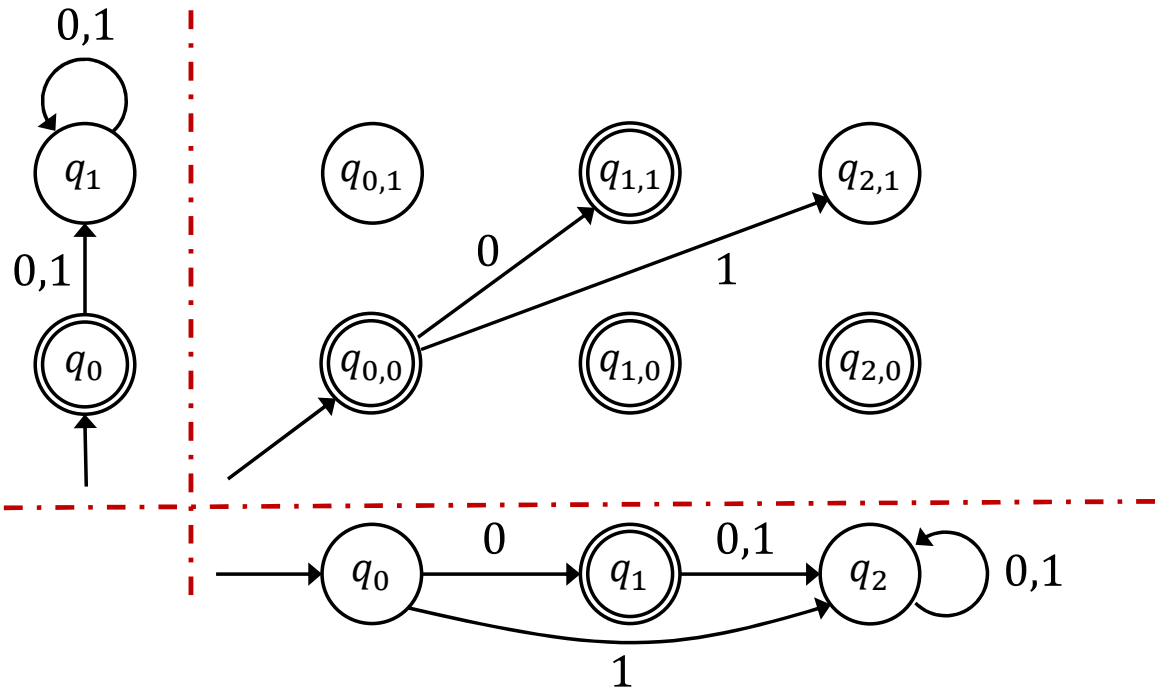


From Regex to DFA: Union

Suppose that R_1 is computed by M_1 and R_2 is computed by M_2.

How to construct a DFA M that computes $R_1 \cup R_2 = L(M_1) \cup L(M_2)$?

M simulates M_1 and M_2 in parallel.

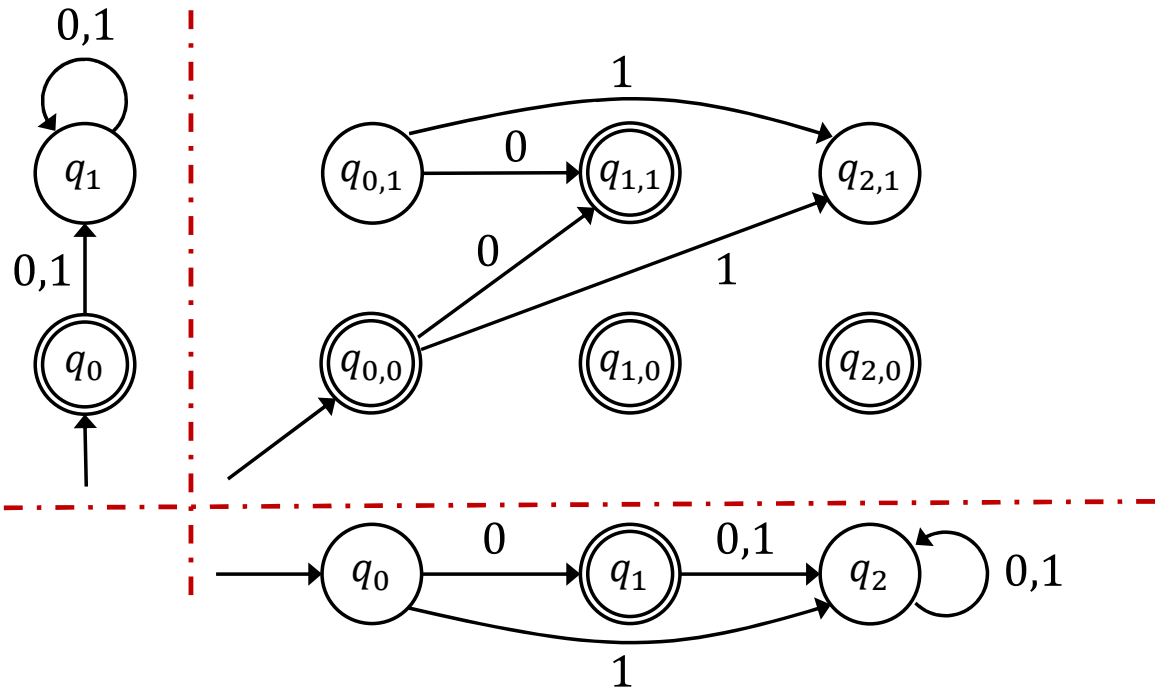


From Regex to DFA: Union

Suppose that R_1 is computed by M_1 and R_2 is computed by M_2.

How to construct a DFA M that computes $R_1 \cup R_2 = L(M_1) \cup L(M_2)$?

M simulates M_1 and M_2 in parallel.

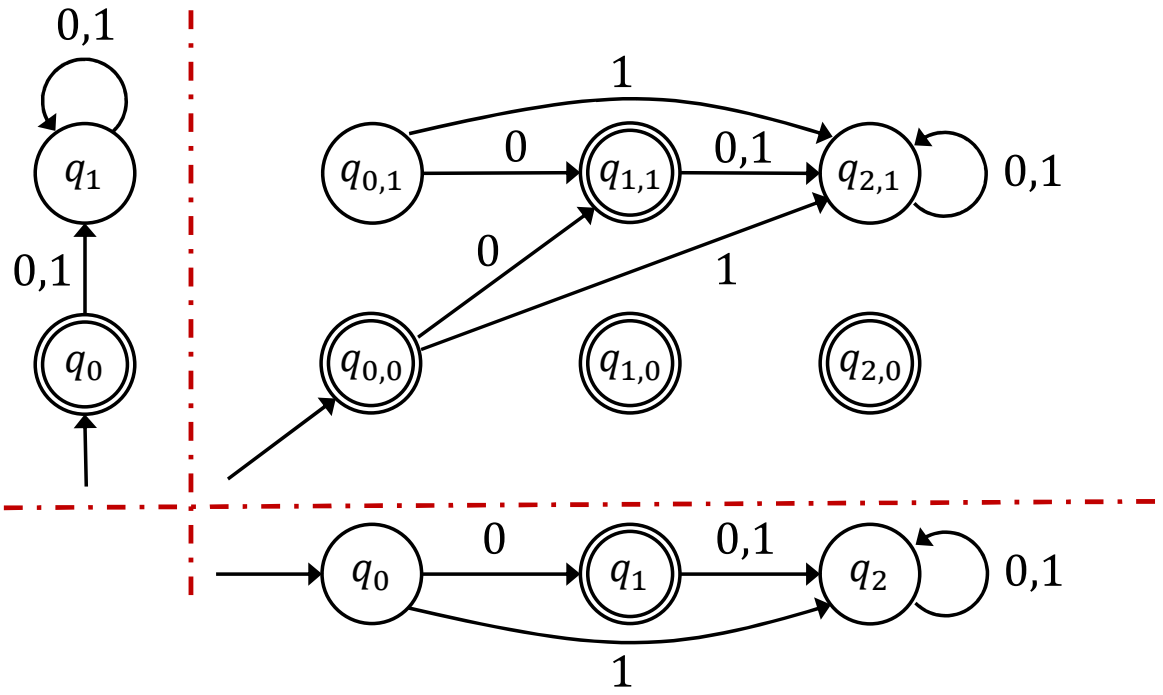


From Regex to DFA: Union

Suppose that R_1 is computed by M_1 and R_2 is computed by M_2.

How to construct a DFA M that computes $R_1 \cup R_2 = L(M_1) \cup L(M_2)$?

M simulates M_1 and M_2 in parallel.



From Regex to DFA: Union

Formality, let $M_1 = (Q_1, \Sigma, \delta_1, q_0^{(1)}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_0^{(2)}, F_2)$.

We define $M = (Q, \Sigma, \delta, q_0, F)$ where:

- $Q = Q_1 \times Q_2$
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
- $q_0 = (q_0^{(1)}, q_0^{(2)})$
- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

From Regex to DFA: Union

Proof for $L(M) = L(M_1) \cup L(M_2)$:

For input $w = w_1 w_2 \ldots w_n$, suppose the execution sequences of M_1 and M_2 on w are the states:

$$r_0 = q_0^{(1)}, r_{i+1} = \delta_1(r_i, w_{i+1})$$
$$r_0' = q_0^{(2)}, r_{i+1}' = \delta_2(r_i', w_{i+1})$$

Then the execution sequence of M on w is exactly (r_i, r_i') (by the definition of M).

From Regex to DFA: Union

Proof for $L(M) = L(M_1) \cup L(M_2)$:

For input $w = w_1 w_2 \ldots w_n$, suppose the execution sequences of M_1 and M_2 on w are the states:

$$r_0 = q_0^{(1)}, r_{i+1} = \delta_1(r_i, w_{i+1})$$
$$r_0' = q_0^{(2)}, r_{i+1}' = \delta_2(r_i', w_{i+1})$$

Then the execution sequence of M on w is exactly (r_i, r_i') (by the definition of M). So

$$w \in L(M) \Leftrightarrow (r_n, r_n') \in F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$$

From Regex to DFA: Union

Proof for $L(M) = L(M_1) \cup L(M_2)$:

For input $w = w_1 w_2 \ldots w_n$, suppose the execution sequences of M_1 and M_2 on w are the states:

$$r_0 = q_0^{(1)}, r_{i+1} = \delta_1(r_i, w_{i+1})$$
$$r_0' = q_0^{(2)}, r_{i+1}' = \delta_2(r_i', w_{i+1})$$

Then the execution sequence of M on w is exactly (r_i, r_i') (by the definition of M). So

$$w \in L(M) \Leftrightarrow (r_n, r_n') \in F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$$
$$\Leftrightarrow r_n \in F_1 \; or \; r_n' \in F_2 \Leftrightarrow w \in L(M_1) \cup L(M_2).$$