Function

Zeyu Mi

2023-12-8

Adapted From:

《数理逻辑与集合论》11.1,11.2
IALC 1.1,1.5,2.2
Stanford CS103 Turing Machine

Binary Relation: Recap

- **Binary Relation Basics**
- **Equivalence Relation and Partitions**
  - Reflexivity, Symmetry and Transitivity.
- **Compatible Relation and Covers**
  - Reflexivity and Symmetry
- **Partial Order Relation**
  - Reflexivity, Anti-symmetry and Transitivity
- **Closure**

Recap: Closure

- For a relation $\boldsymbol{R}$ on non-empty set $\boldsymbol{A}$
  - $\boldsymbol{R}$ is reflexive $\Leftrightarrow \boldsymbol{r}(\boldsymbol{R}) = \boldsymbol{R}$
  - $\boldsymbol{R}$ is symmetric $\Leftrightarrow \boldsymbol{s}(\boldsymbol{R}) = \boldsymbol{R}$
  - $\boldsymbol{R}$ is transitive $\Leftrightarrow \boldsymbol{t}(\boldsymbol{R}) = \boldsymbol{R}$

Constructing Closure

For a relation $\boldsymbol{R}$ on non-empty set $\boldsymbol{A}$, $|\boldsymbol{A}| = n$,

$$\boldsymbol{t(R) = R \cup R^2 \cdots \cup R^n}$$

Lemma

Let $\boldsymbol{R}$ be a relation on a set $\boldsymbol{A}$. There is a path of length $\boldsymbol{n}(\boldsymbol{n} \in \mathbb{N}^+)$ from $\boldsymbol{a}$ to $\boldsymbol{b}$ if and only if $\langle \boldsymbol{a}, \boldsymbol{b} \rangle \in \boldsymbol{R^n}$

Proof:

By definition, there is a path from $\boldsymbol{a}$ to $\boldsymbol{b}$ of length 1 if and only if $\langle \boldsymbol{a}, \boldsymbol{b} \rangle \in \boldsymbol{R}$, so the lemma is true when $n = 1$.

Assume that the lemma is true when $n = k(k \in \mathbb{N}^+)$. For $n = k + 1$

There is a path of length $k + 1$ from $\boldsymbol{a}$ to $\boldsymbol{b}$

$\Leftrightarrow$ There exists $\boldsymbol{c}$ such that there's a path of length 1 from $\boldsymbol{a}$ to $\boldsymbol{c}$ and a path of length k from $\boldsymbol{c}$ to $\boldsymbol{b}$

$\Leftrightarrow (\exists \boldsymbol{c})\big(\langle \boldsymbol{a}, \boldsymbol{c} \rangle \in \boldsymbol{R} \wedge \langle \boldsymbol{c}, \boldsymbol{b} \rangle \in \boldsymbol{R^k}\big) \Leftrightarrow \langle \boldsymbol{a}, \boldsymbol{b} \rangle \in \boldsymbol{R^{k+1}}$

So the lemma is true when $n = k + 1$. Q.E.D

Connectivity Relation(连接关系)

- Let $\boldsymbol{R}$ be a relation on a set $\boldsymbol{A}$. The **connectivity relation** $\boldsymbol{R^+}$ consists of the pairs $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ such that there is a path of length at least one from $\boldsymbol{a}$ to $\boldsymbol{b}$ in $\boldsymbol{R}$.
- By definition:

$$\boldsymbol{R^+ = R \cup R^2 \cup R^3 \cup \cdots}$$

- But we only need to consider the path of length as most $|\boldsymbol{A}|$, assume that $|\boldsymbol{A}| = \boldsymbol{n}$

$$\boldsymbol{R^+ = R \cup R^2 \cup R^3 \cup \cdots \cup R^n}$$

Constructing Closure

For a relation $\boldsymbol{R}$ on non-empty set $\boldsymbol{A}$, $|\boldsymbol{A}| = n$,
$\boldsymbol{t(R) = R^+ = R \cup R^2 \cdots \cup R^n}$

Proof:

- $R \subseteq R^+$ by definition.
- $\boldsymbol{R^+}$ **is transitive**: For $\langle \boldsymbol{x, y} \rangle \in \boldsymbol{R^+}$ and $\langle \boldsymbol{y, z} \rangle \in \boldsymbol{R^+}$, it means there are paths from $\boldsymbol{x}$ to $\boldsymbol{y}$ and $\boldsymbol{y}$ to $\boldsymbol{z}$, so there's a path from $\boldsymbol{x}$ to $\boldsymbol{z}$. Then $\langle \boldsymbol{x, z} \rangle \in \boldsymbol{R^+}$.
- Assume $\boldsymbol{R''}$ is transitive and $\boldsymbol{R \subseteq R''}$.
  - For any $\langle \boldsymbol{x, y} \rangle \in \boldsymbol{R^+}$, it means that there's a path of length $k (k \in \mathbb{N}^+)$ from $\boldsymbol{x}$ to $\boldsymbol{y}$,
  - Let the path be $\boldsymbol{x, p_1, .. p_{k-1}, y}$, so $\langle \boldsymbol{x, p_1} \rangle, \langle \boldsymbol{p_1, p_2} \rangle \ldots \langle \boldsymbol{p_{k-1}, y} \rangle \in \boldsymbol{R}$.
  - Because $\boldsymbol{R \subseteq R''}$, $\langle \boldsymbol{x, p_1} \rangle, \langle \boldsymbol{p_1, p_2} \rangle \ldots \langle \boldsymbol{p_{k-1}, y} \rangle \in \boldsymbol{R''}$.
  - According to the transitivity of $\boldsymbol{R''}$, we have $\langle \boldsymbol{x, y} \rangle \in \boldsymbol{R''}$.
  - So $R^+ \subseteq R''$.

Exercise

- $\boldsymbol{A = \{a, b, c\}}$, $\boldsymbol{R = \{\langle a, b\rangle, \langle b, c\rangle, \langle c, a\rangle\}}$, find $\boldsymbol{r(R)}$, $\boldsymbol{s(R)}$ and $\boldsymbol{t(R)}$.

Exercise

- $A = \{a, b, c\}$, $R = \{\langle a, b\rangle, \langle b, c\rangle, \langle c, a\rangle\}$, find $r(R), s(R)$ and $t(R)$.
- **Answer:**
  - $r(R) = R \cup R^0 = \{\langle a, a\rangle, \langle a, b\rangle, \langle b, b\rangle, \langle b, c\rangle, \langle c, a\rangle, \langle c, c\rangle\}$
  - $s(R) = R \cup R^{-1} = \{\langle a, b\rangle, \langle a, c\rangle, \langle b, a\rangle, \langle b, c\rangle, \langle c, a\rangle, \langle c, b\rangle\}$
  - $R^2 = \{\langle a, c\rangle, \langle b, a\rangle, \langle c, b\rangle\}$,
  - $R^3 = \{\langle a, a\rangle, \langle b, b\rangle, \langle c, c\rangle\}$
  - So $t(R) = R \cup R^2 \cup R^3 =$ $\{\langle a, a\rangle, \langle a, b\rangle, \langle a, c\rangle, \langle b, a\rangle, \langle b, b\rangle, \langle b, c\rangle, \langle c, a\rangle, \langle c, b\rangle, \langle c, c\rangle\}$

Transitive Closure

- $t(R) = R \cup R^2 \cup R^3 \cup \cdots \cup R^n$

It's hard to compute all of those when R is large!

Transitive Closure: Warshall Algorithm

- Warshall Algorithm is an efficient algorithm to calculate transitive closure

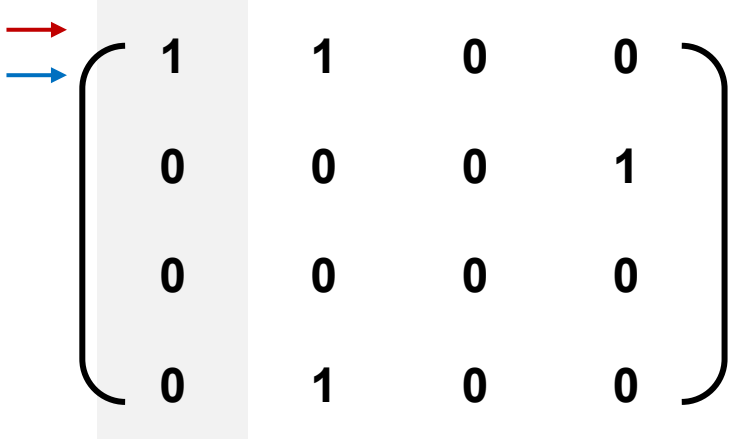
```
Warshall_Algorithm(M[1…n,1…n]):
  M^0 ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if M^(i-1)[j,i]==1:
        for k ← 1 to n:
          M^i[j,k] = M^(i-1)[j,k]V M^(i-1)[i,k]
  return M^n
```

Transitive Closure: Warshall Algorithm

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

```
Warshall_Algorithm(M[1…n,1…n]):
  M^0 ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if M^(i-1)[j,i]==1:
        for k ← 1 to n:
          M^i[j,k] = M^(i-1)[j,k]∨M^(i-1)[i,k]
  return M^n
```

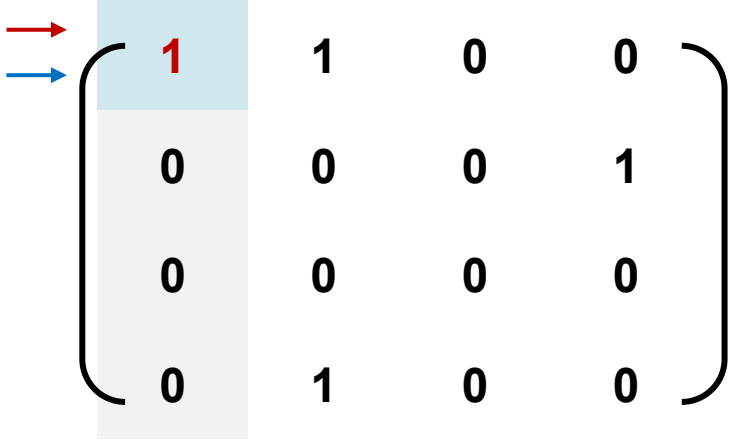
Transitive Closure: Warshall Algorithm




```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

i=1,j=1

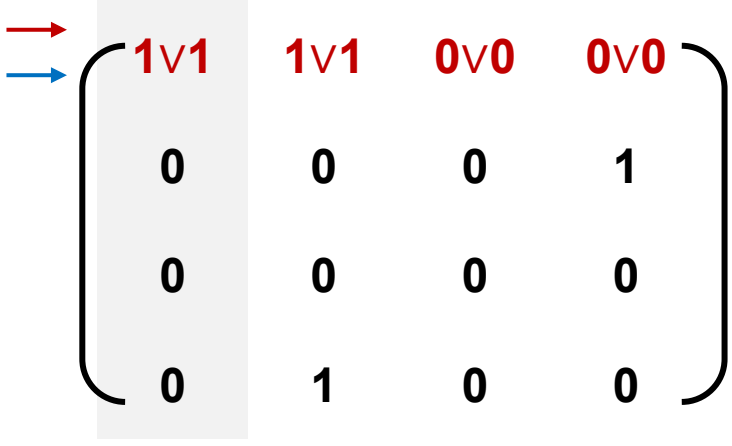
Transitive Closure: Warshall Algorithm




```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

i=1,j=1

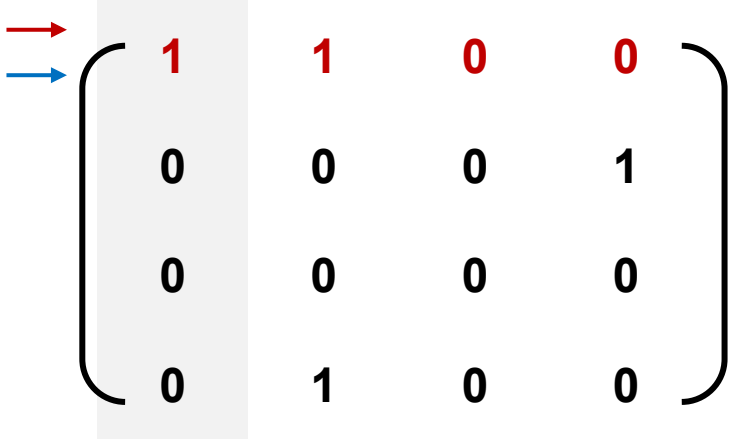
Transitive Closure: Warshall Algorithm



i=1,j=1

```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]∨Mⁱ⁻¹[i,k]
  return Mⁿ
```

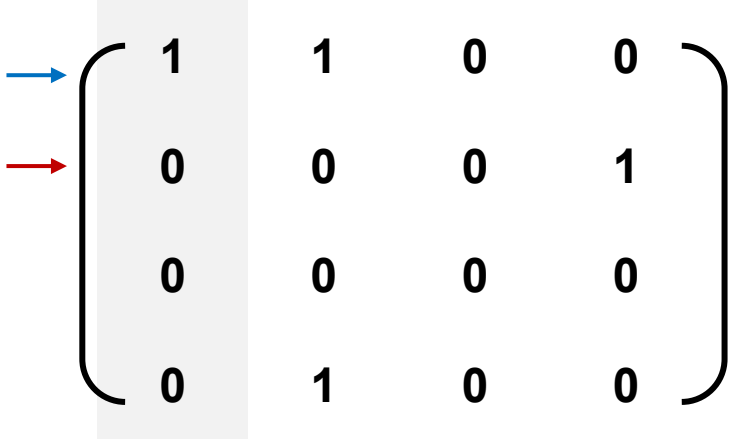
Transitive Closure: Warshall Algorithm




```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

i=1,j=1

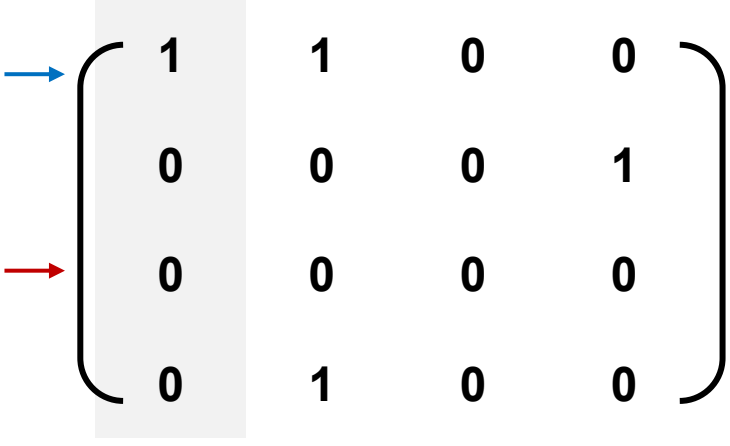
Transitive Closure: Warshall Algorithm




```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

i=1,j=2

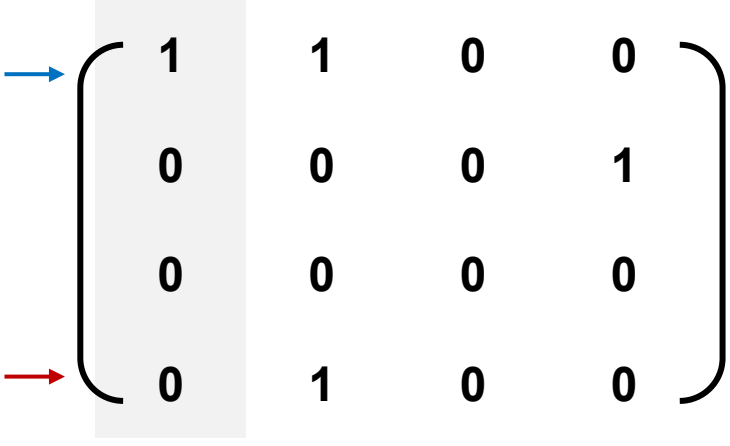
Transitive Closure: Warshall Algorithm




```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]∨Mⁱ⁻¹[i,k]
  return Mⁿ
```

i=1,j=3

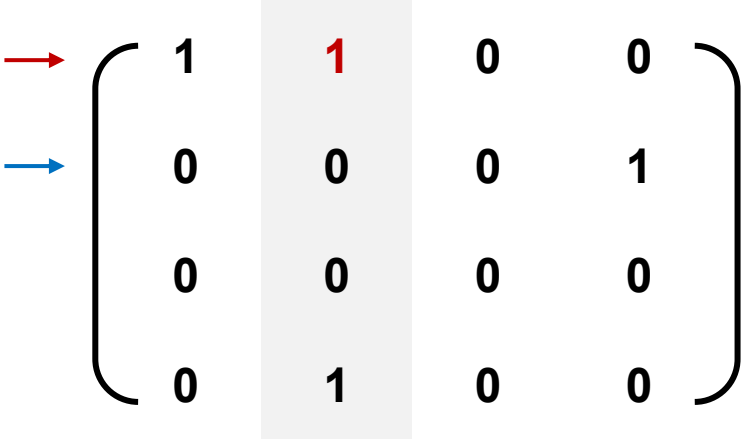
Transitive Closure: Warshall Algorithm




```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

i=1,j=4

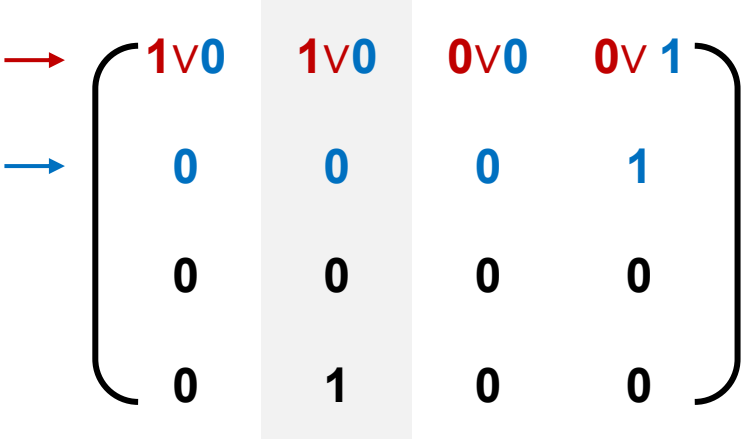
Transitive Closure: Warshall Algorithm



i=2,j=1

```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

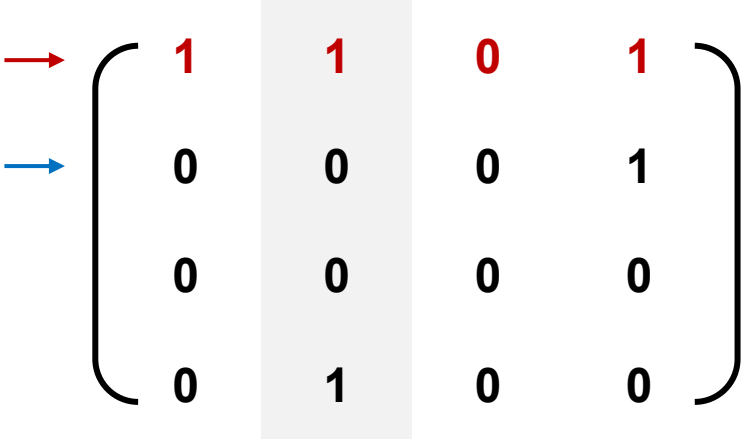
Transitive Closure: Warshall Algorithm



i=2,j=1

```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]∨Mⁱ⁻¹[i,k]
  return Mⁿ
```

Transitive Closure: Warshall Algorithm

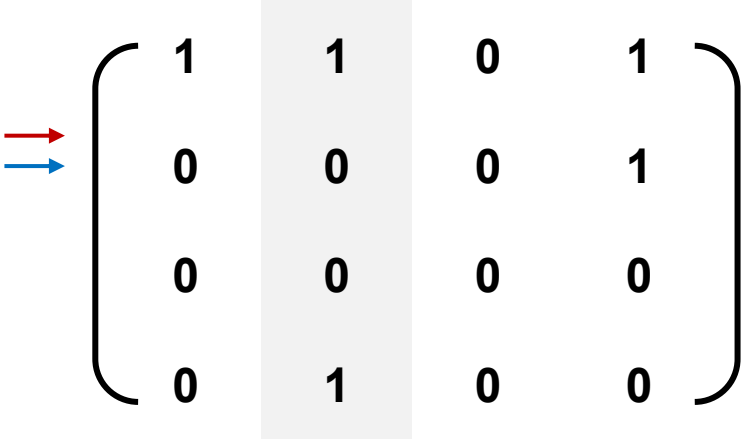


$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

i=2,j=1

```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

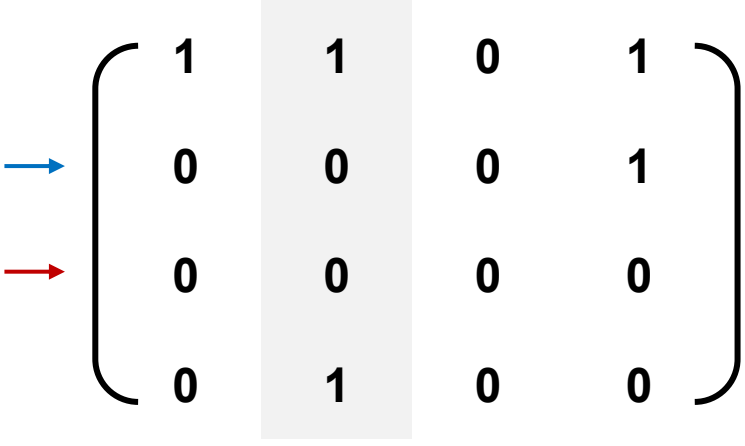
Transitive Closure: Warshall Algorithm




```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

i=2,j=2

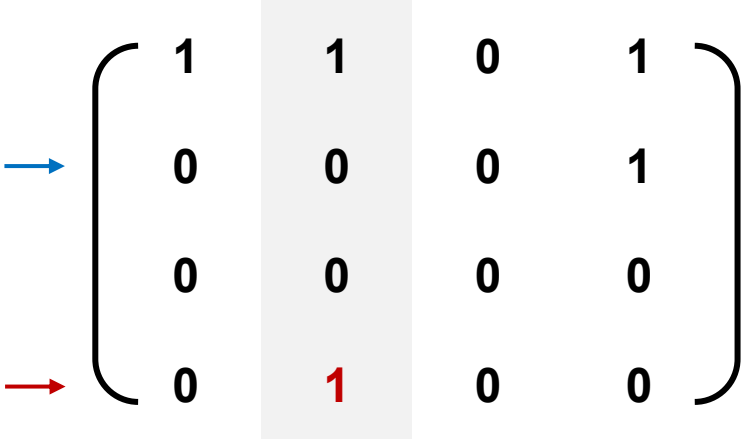
Transitive Closure: Warshall Algorithm



i=2,j=3

```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

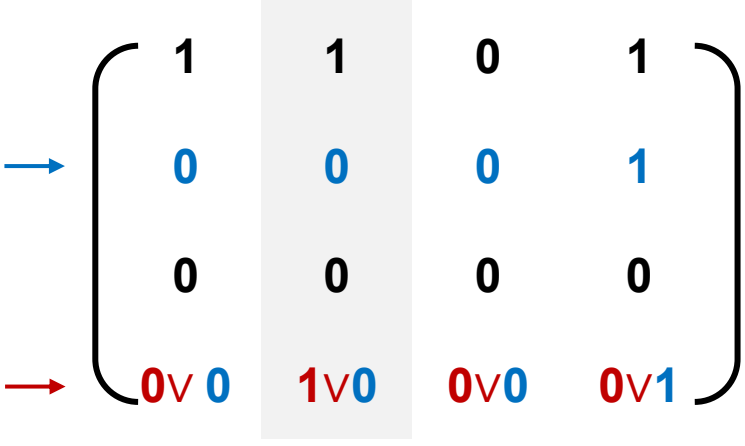
Transitive Closure: Warshall Algorithm



i=2,j=4

```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

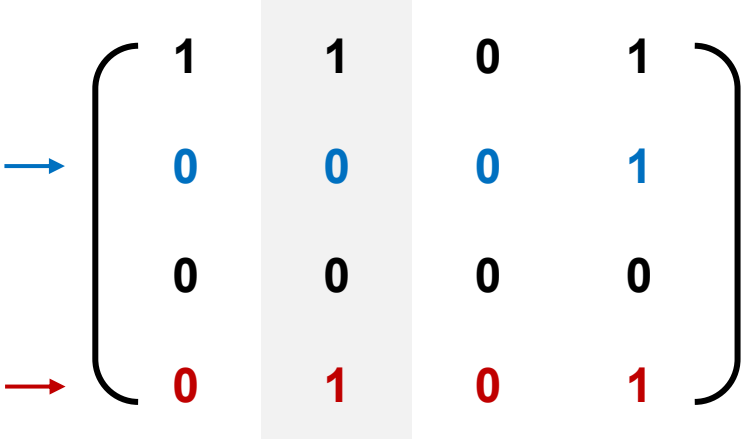
Transitive Closure: Warshall Algorithm



i=2,j=4

```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]∨Mⁱ⁻¹[i,k]
  return Mⁿ
```

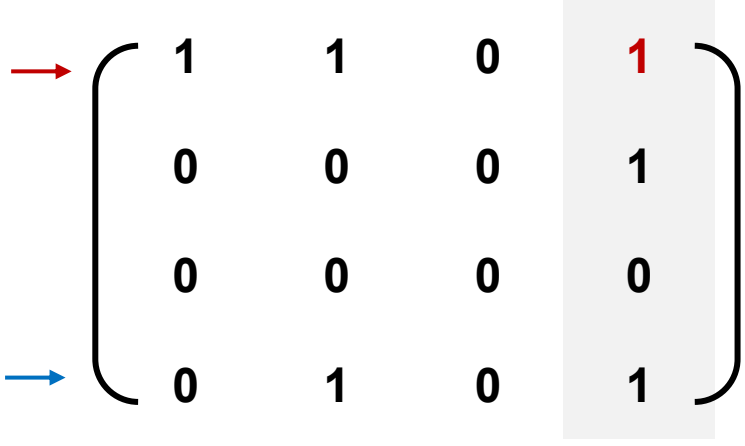
Transitive Closure: Warshall Algorithm



i=2,j=4

```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

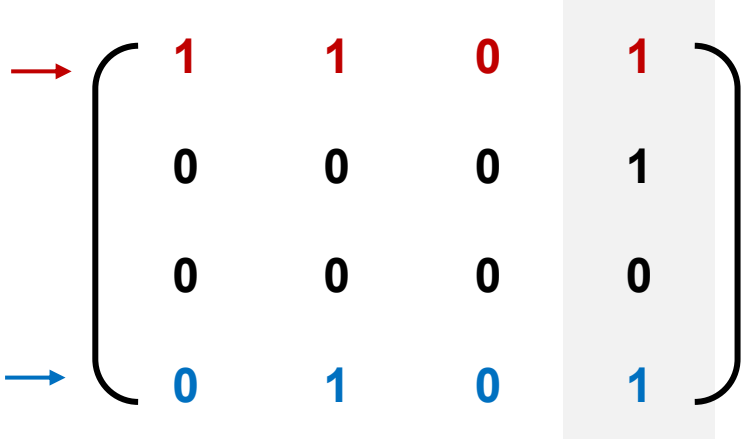
Transitive Closure: Warshall Algorithm



i=4,j=1

```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

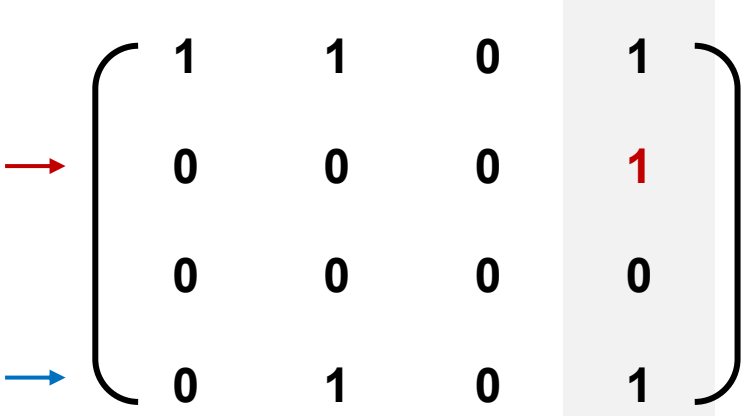
Transitive Closure: Warshall Algorithm



i=4,j=1

```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

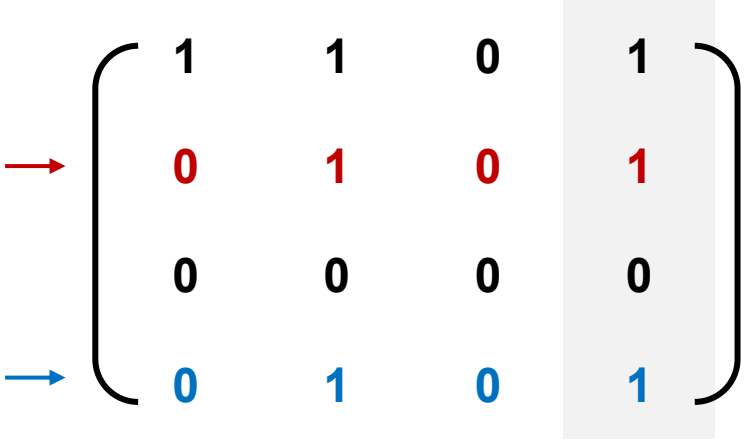
Transitive Closure: Warshall Algorithm



i=4,j=1

```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

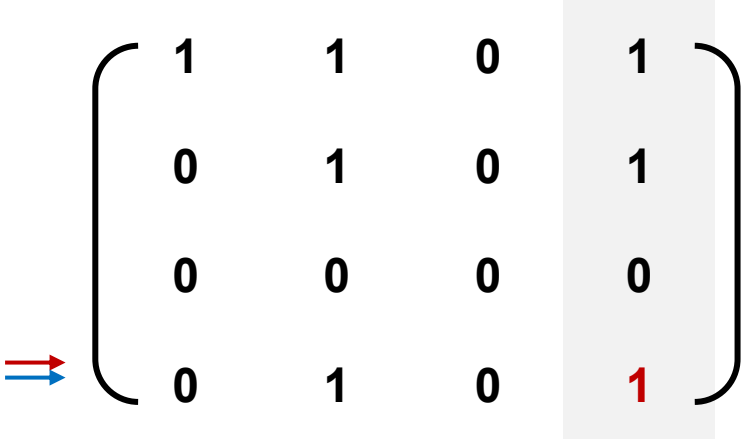
Transitive Closure: Warshall Algorithm



i=4,j=1

```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

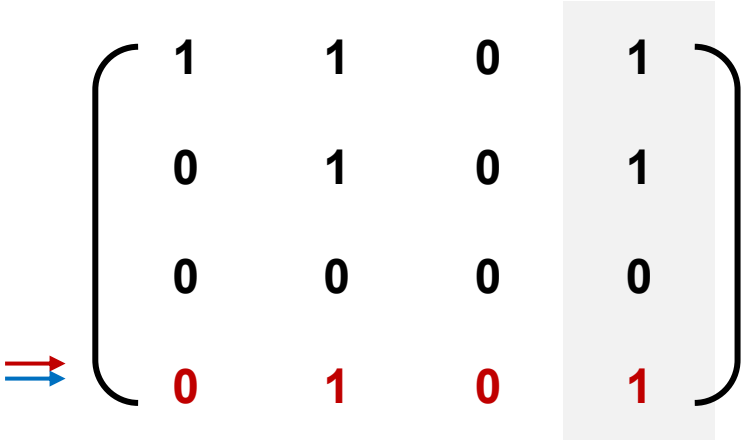
Transitive Closure: Warshall Algorithm




```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

i=4,j=4

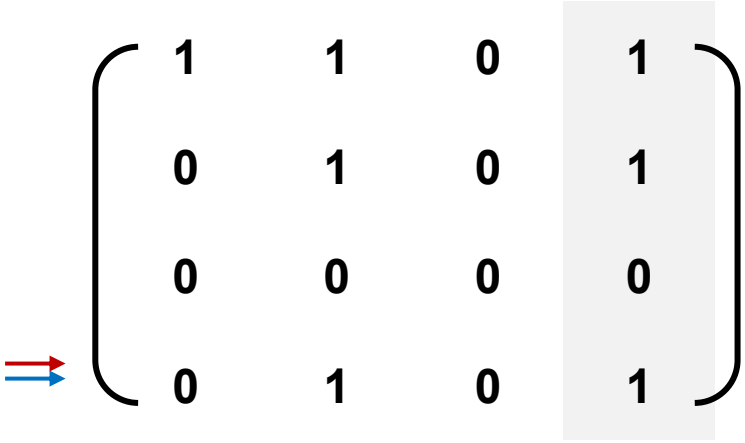
Transitive Closure: Warshall Algorithm




```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

i=4,j=4

Transitive Closure: Warshall Algorithm




```
Warshall_Algorithm(M[1…n,1…n]):
  M⁰ ← M
  for i ← 1 to n:
    for j ← 1 to n:
      if Mⁱ⁻¹[j,i]==1:
        for k ← 1 to n:
          Mⁱ[j,k] = Mⁱ⁻¹[j,k]VMⁱ⁻¹[i,k]
  return Mⁿ
```

i=4,j=4

Closures

- For a relation $\boldsymbol{R}$ on a non-empty set $\boldsymbol{A}$
  - If $\boldsymbol{R}$ is reflexive, then $\boldsymbol{s}(\boldsymbol{R})$ and $\boldsymbol{t}(\boldsymbol{R})$ are reflexive
  - If $\boldsymbol{R}$ is symmetric, then $\boldsymbol{r}(\boldsymbol{R})$ and $\boldsymbol{t}(\boldsymbol{R})$ are symmetric
  - If $\boldsymbol{R}$ is transitive, then $\boldsymbol{r}(\boldsymbol{R})$ are transitive.

s(R) may not be transitive!!

Closures

- For a relation $\boldsymbol{R}$ on non-empty set $\boldsymbol{A}$, $rs(R)$ means $r(s(R))$, the similar definition for other closure symbols, so we have:
  - $rs(R) = sr(R)$
  - $rt(R) = tr(R)$

  r is commutative with **t** and **s**

  - $st(R) \subseteq ts(R)$

  s and **t** are not commutative

Binary Relation: Recap

- **Binary Relation Basics**
- **Equivalence Relation and Partitions**
  - Reflexivity, Symmetry and Transitivity.
- **Compatible Relation and Covers**
  - Reflexivity and Symmetry
- **Partial Order Relation**
  - Reflexivity, Anti-symmetry and Transitivity
- **Closure**

How we investigate computability?

2.1 Discussion on Problems

Part1. Intro & Set theory(I) : Basics & Formal Language.

Part2. Set Theory(II): Axiom system & Cardinality.

Part3. Capture Structures: Binary Relation & Function

2.2 Discussion on Computation

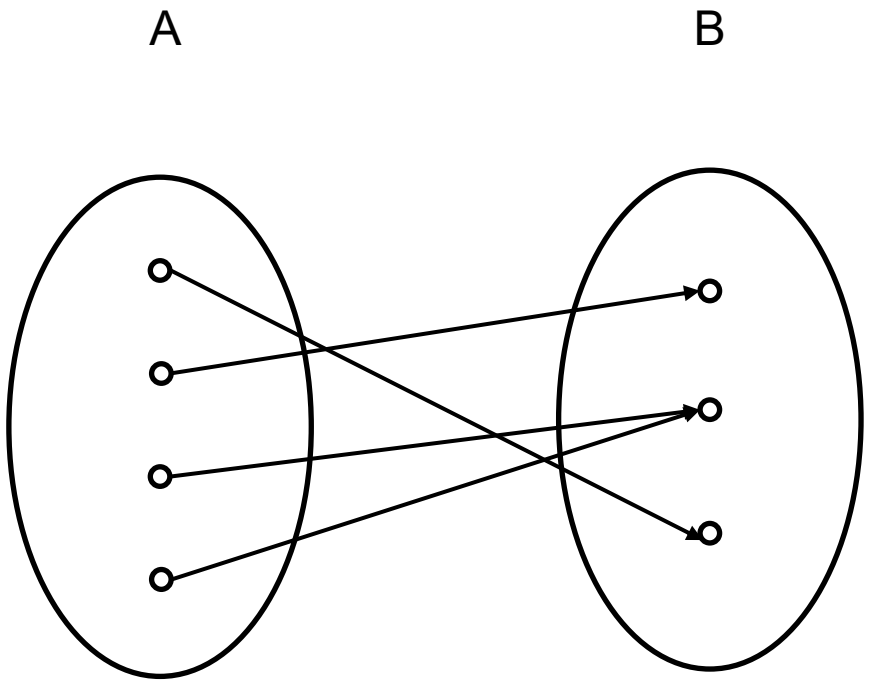
Part1. Turing Machine Basics.

Part2. Variants of Turing Machine. Church-Turing Thesis.

2.3 Discussion on computability

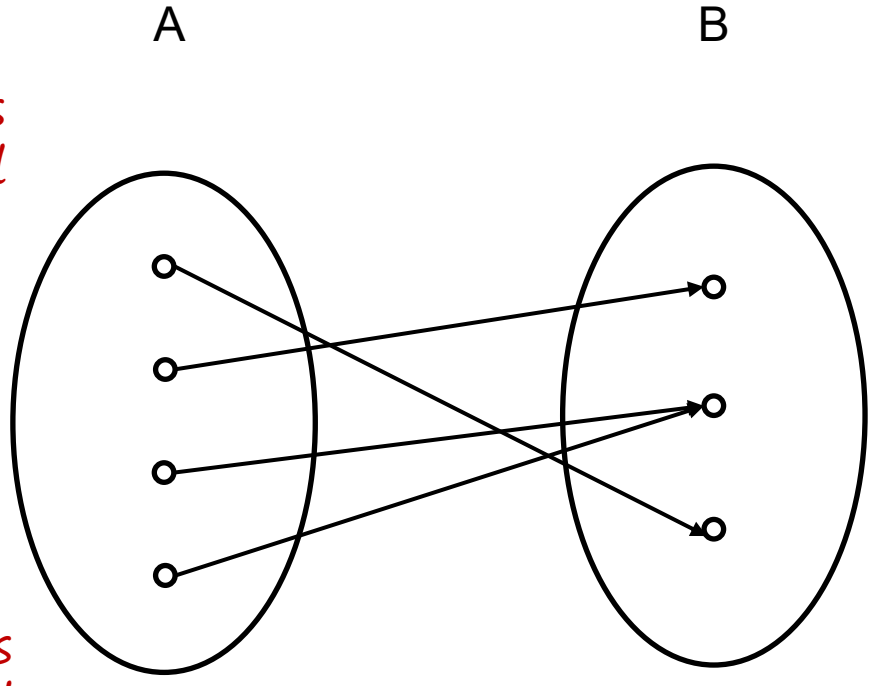
Part1. The Language of Turing Machine. R & RE

Part2. Undecidability.



A special relation from A to B

Every elements in A are paired



Every elements in A are paired only once

A special relation from A to B

Function

- A relation f from set $\boldsymbol{A}$ to set $\boldsymbol{B}$ is called a function if:
  - For any $\boldsymbol{x} \in dom(f)$, there exists ***unique*** $\boldsymbol{y}$ such that $\boldsymbol{xfy}$

$$(\forall \boldsymbol{x})(\forall \boldsymbol{y_1})(\forall \boldsymbol{y_2})((\boldsymbol{xfy_1} \wedge \boldsymbol{xfy_2}) \rightarrow \boldsymbol{y_1} = \boldsymbol{y_2})$$

  - $dom(f) = \boldsymbol{A}$

$$(\forall \boldsymbol{x})(\boldsymbol{x} \in \boldsymbol{A} \rightarrow (\exists \boldsymbol{y})(\boldsymbol{y} \in \boldsymbol{B} \wedge \boldsymbol{xfy}))$$

- A function f from set $\boldsymbol{A}$ to set $\boldsymbol{B}$ is donated as $f: \boldsymbol{A} \rightarrow \boldsymbol{B}$.
- $\boldsymbol{xfy}$ written as $f: \boldsymbol{x}| \rightarrow \boldsymbol{y}$ or $f(\boldsymbol{x}) = \boldsymbol{y}$

Function

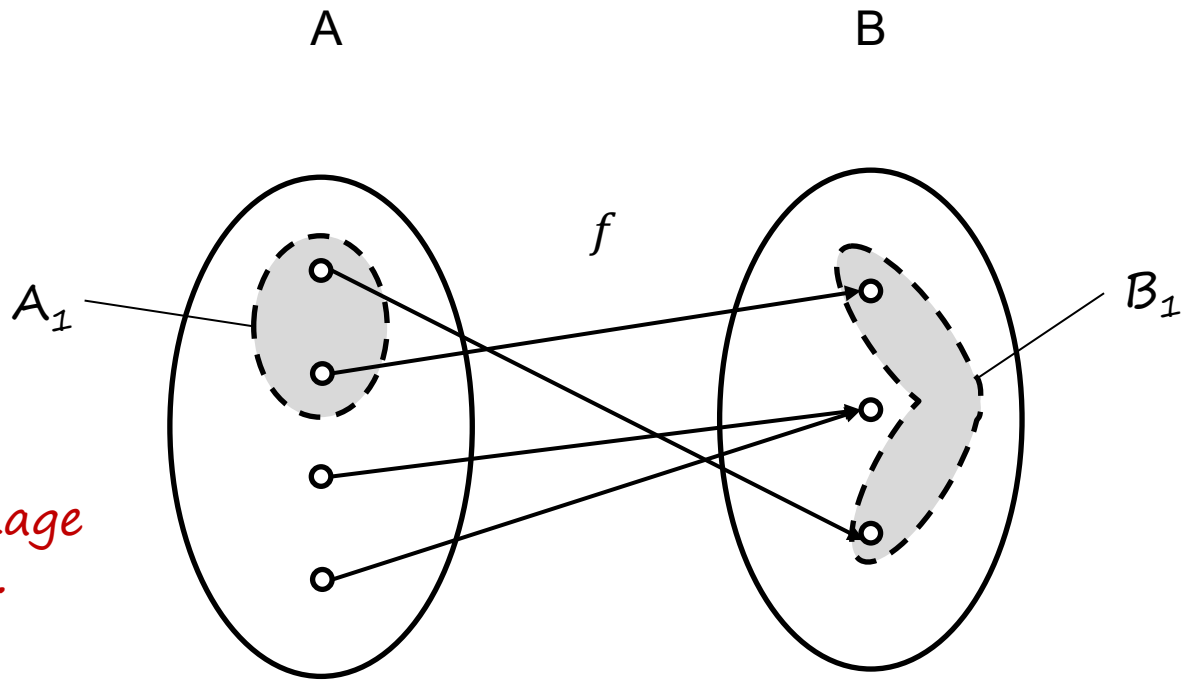
- All the function from set $\boldsymbol{A}$ to set $\boldsymbol{B}$ is $\boldsymbol{A_B}$

$$\boldsymbol{A_B} = \{f \,|\, f : \boldsymbol{A} \rightarrow \boldsymbol{B}\}$$

- If $|A| = n$, $|B| = m$
  - then $|A_B| = m^n$

Function

- For relation $\boldsymbol{R}$ on $\boldsymbol{A}$, for a set $\boldsymbol{B}$
  - the **restriction**(限制) of $\boldsymbol{R}$ on $\boldsymbol{B}$, donate as $\boldsymbol{R} \upharpoonright \boldsymbol{B}$ is:
    $$R \upharpoonright B = \{\langle x, y\rangle | \langle x, y\rangle \in R \wedge x \in B\}$$
  - The **image** (象) of R on B , donate as $R[B]$ is:
    $$R[B] = \{y | \exists x(\langle x, y\rangle \in R \wedge x \in B)\}$$
- Similarly for a function $f: \boldsymbol{A} \to \boldsymbol{B}$, $\forall \boldsymbol{A_1} \subseteq \boldsymbol{A}$ we can define $f[A_1]$ to be the **image**(象) of A_1 on function f.
- $\forall B_1 \subseteq B$, $f^{-1}[B_1]$ is the **inverse image or preimage**(原象) of B_1 on f.



A_1 is the inverse image of B_1 on f.

B_1 is the image of A_1 on f.

$f: A \rightarrow B$

Common Functions

- $f: \boldsymbol{A} \to \boldsymbol{B}$, then $\boldsymbol{f}$ is called **constant function** (常函数) if:
$$(\exists y)(\forall x)(x \in A \to f(x) = y)$$
- $f: \boldsymbol{A} \to \boldsymbol{A}$, then $\boldsymbol{f}$ is called **identity function** (恒等函数) if:
$$(\forall x)(x \in A \to f(x) = x)\ or\ f = I_A$$
- $f: \boldsymbol{A^n} \to \boldsymbol{A}(\boldsymbol{n} \in \mathbb{N})$, then $\boldsymbol{f}$ is called a **n-ary operator**(n元运算)
- $f: \mathbb{R} \to \mathbb{R}$
  - If $(x \leq y) \to (f(x) \leq f(y))$, we call f is **monotonically increasing**(单调递增)
  - If $(x < y) \to (f(x) < f(y))$, we call f is **strictly monotonically increasing**(严格单调递增)
  - If $(x \leq y) \to (f(x) \geq f(y))$, we call f is **monotonically decreasing**(单调递减)
  - If $(x < y) \to (f(x) > f(y))$, we call f is **strictly monotonically decreasing**(严格单调递减)

Common Functions

- $\boldsymbol{F: A \to B_C}$ is called a **functional**(noun, 泛函)
  - We define that $F(y) = (f(x) = x + y)$, where $f(x)$ is a certain function.
  - Then $F(1)$ is the function $f(x) = x + 1$
- Let E be the universal set, for any $A \subseteq E$, the **indicator function, or characteristic function**(特征函数) $\boldsymbol{\chi_A: E \to \{0, 1\}}$ is defined by:

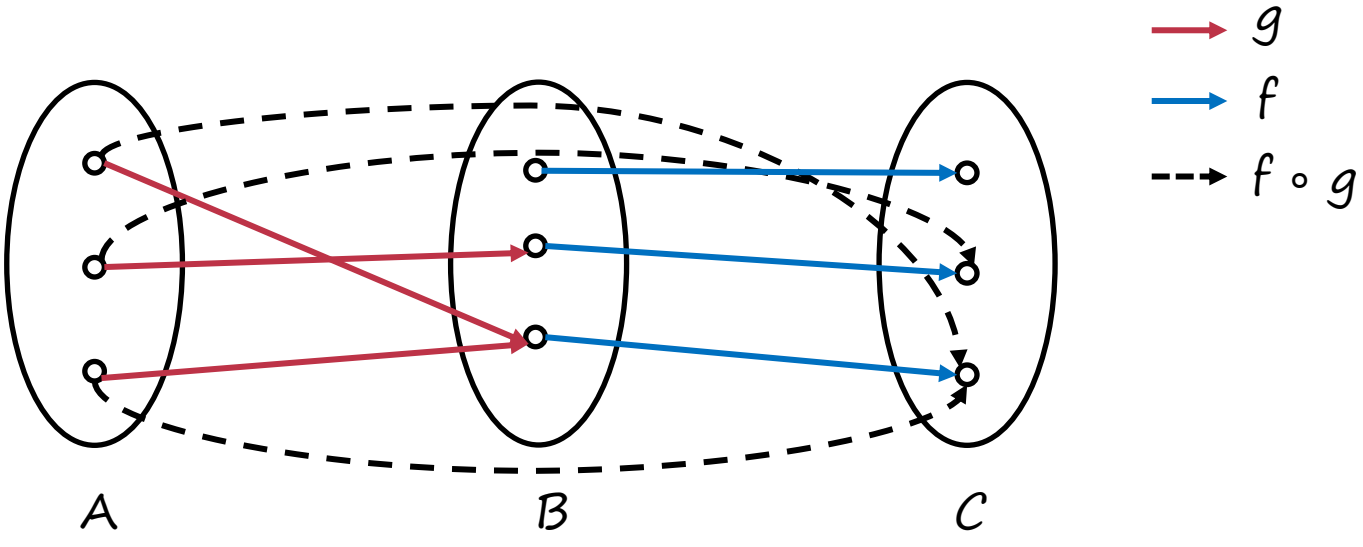
$$\chi_A(x) = \begin{cases} 1, x \in A \\ 0, x \notin A \end{cases}$$

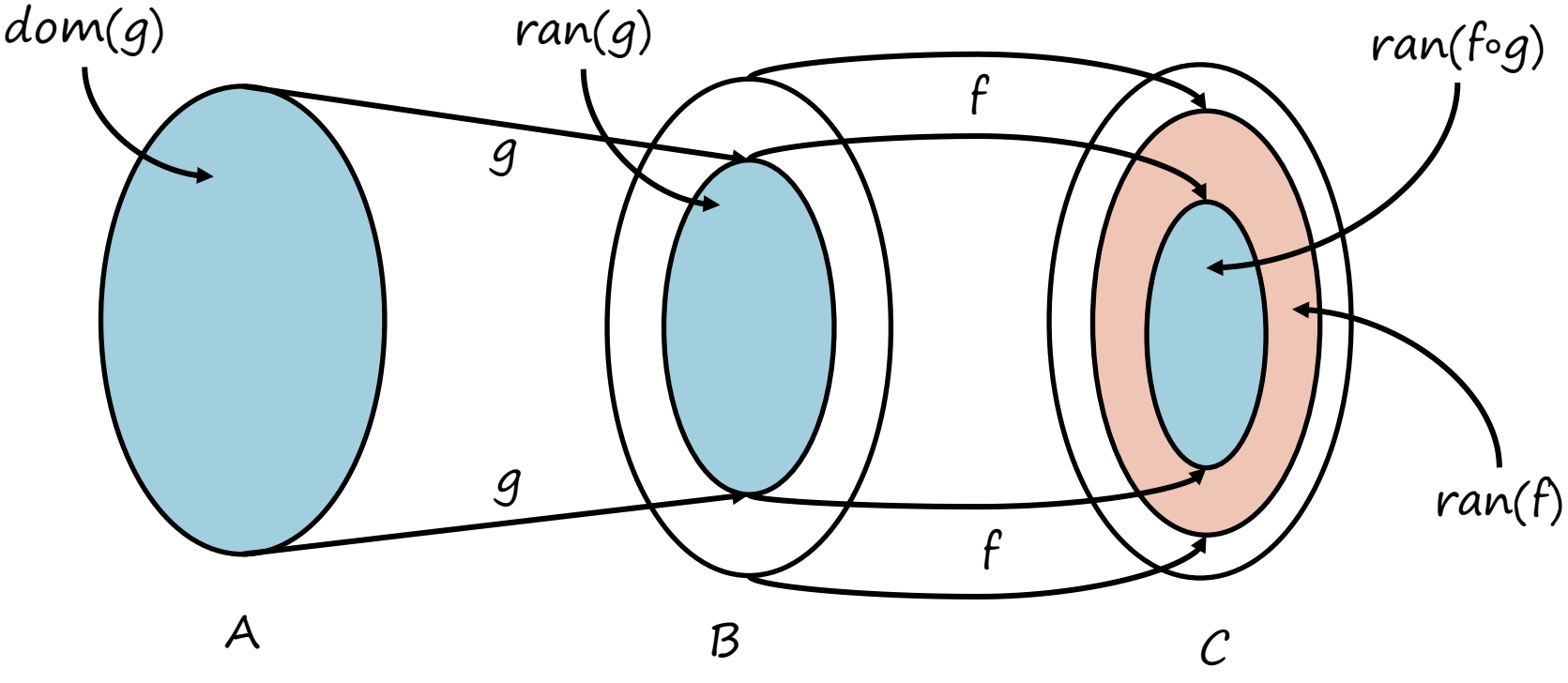
- Let $\boldsymbol{R}$ be an equivalence relation on $\boldsymbol{A}$, $\boldsymbol{g}$ is a **canonical map or natural map**(典型映射或自然映射) from $\boldsymbol{A}$ to quotient set $\boldsymbol{A/R}$ if:

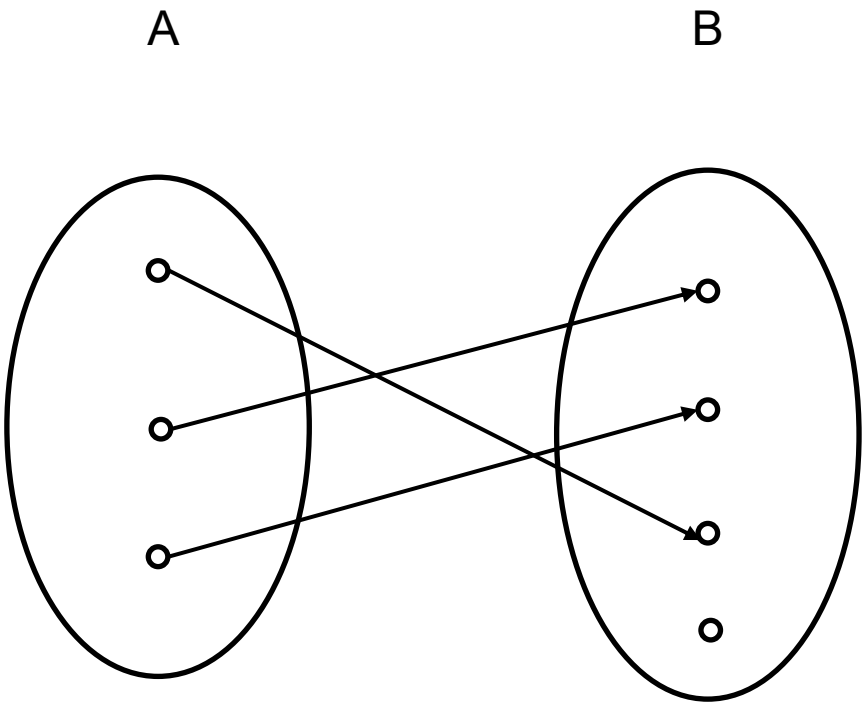
$$\boldsymbol{g: A \to A/R, g(x) = [x]_R}$$

Compose of Function

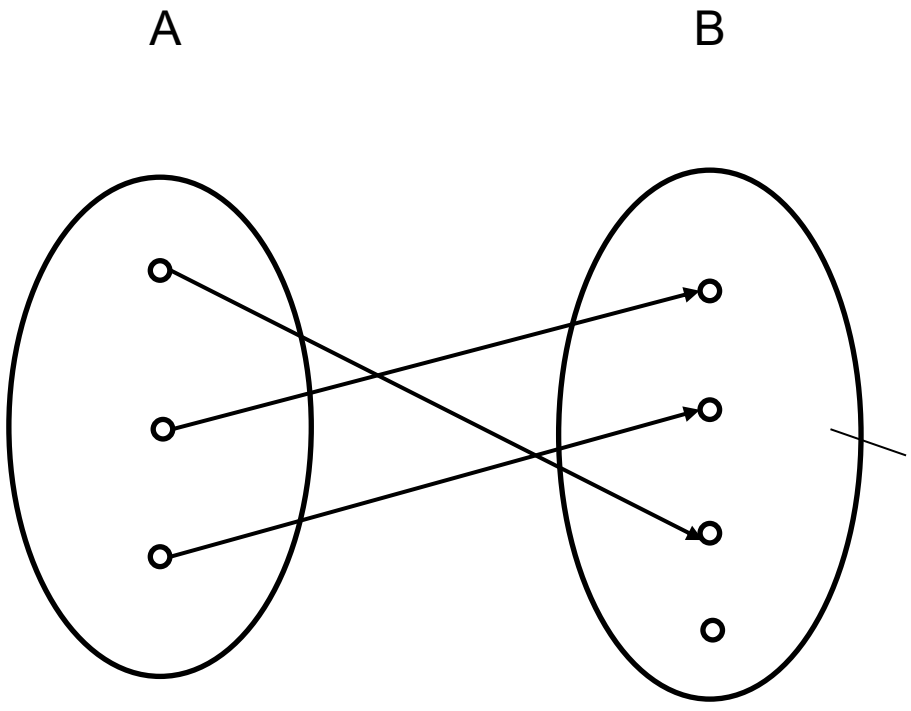
- Let $g: \boldsymbol{A} \to \boldsymbol{B}$, $f: \boldsymbol{B} \to \boldsymbol{C}$, then:
  - $f \circ g$ is a function $f \circ g: \boldsymbol{A} \to \boldsymbol{C}$
  - For any $x \in A$, $(f \circ g)(x) = f(g(x))$




dom(g)
ran(g)
ran(f∘g)
g
f
g
f
ran(f)
A
B
C



A special function



No elements in B are paired twice

A special function

Injection(单射)

- $f: A \to B$ is an **injection** or **injective** if for any $x_1, x_2 \in A, x_1 \neq x_2$, then $f(x_1) \neq f(x_2)$.

$$(\forall x_1)(\forall x_2)(x_1 \in A \wedge x_2 \in A \wedge x_1 \neq x_2 \to f(x_1) \neq f(x_2))$$

Different inputs has different outputs

- Or if $f(x_1) = f(x_2)$, then $x_1 = x_2$

$$(\forall x_1)(\forall x_2)(x_1 \in A \wedge x_2 \in A \wedge f(x_1) = f(x_2) \to x_1 = x_2)$$

Same outputs have the same input

Injection

- $f(x) = x$ $(\mathbb{R} \to \mathbb{R})$
  - injective
  - For any $x_1 \neq x_2$, $f(x_1) = x_1 \neq x_2 = f(x_2)$
- $f(x) = x^2$ $(\mathbb{R} \to \mathbb{R})$
  - not injective.
  - $f(-1) = f(1) = 1$

Injection and Composition

Let $g: \boldsymbol{A} \to \boldsymbol{B}$, $f: \boldsymbol{B} \to \boldsymbol{C}$, if f, g are injective, then $f \circ g$ is also injective.

Proof:

For any $x_1, x_2 \in A, x_1 \neq x_2$, we have $g(x_1) \neq g(x_2)$ since g is injective.

Now $g(x_1), g(x_2) \in B$, and $g(x_1) \neq g(x_2)$, then we have $f\big(g(x_1)\big) \neq f(g(x_2))$ since f is injective.

$(f \circ g)(x) = f(g(x))$ by definition, so $(f \circ g)(x_1) \neq (f \circ g)(x_2)$

Q.E.D

Injection and Composition

Let $g: \boldsymbol{A} \to \boldsymbol{B}$, $f: \boldsymbol{B} \to \boldsymbol{C}$, if $f \circ g$ is injective, then g is also injective.

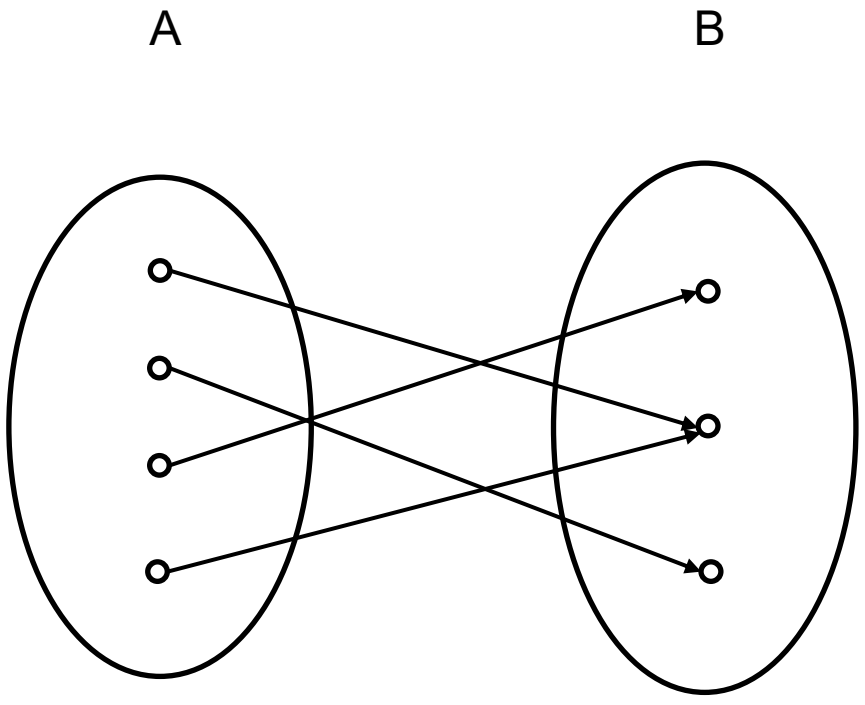
Proof:

Assume that g is not injective, then there exists $x_1, x_2 \in A$, $x_1 \neq x_2$ and $g(x_1) = g(x_2)$.

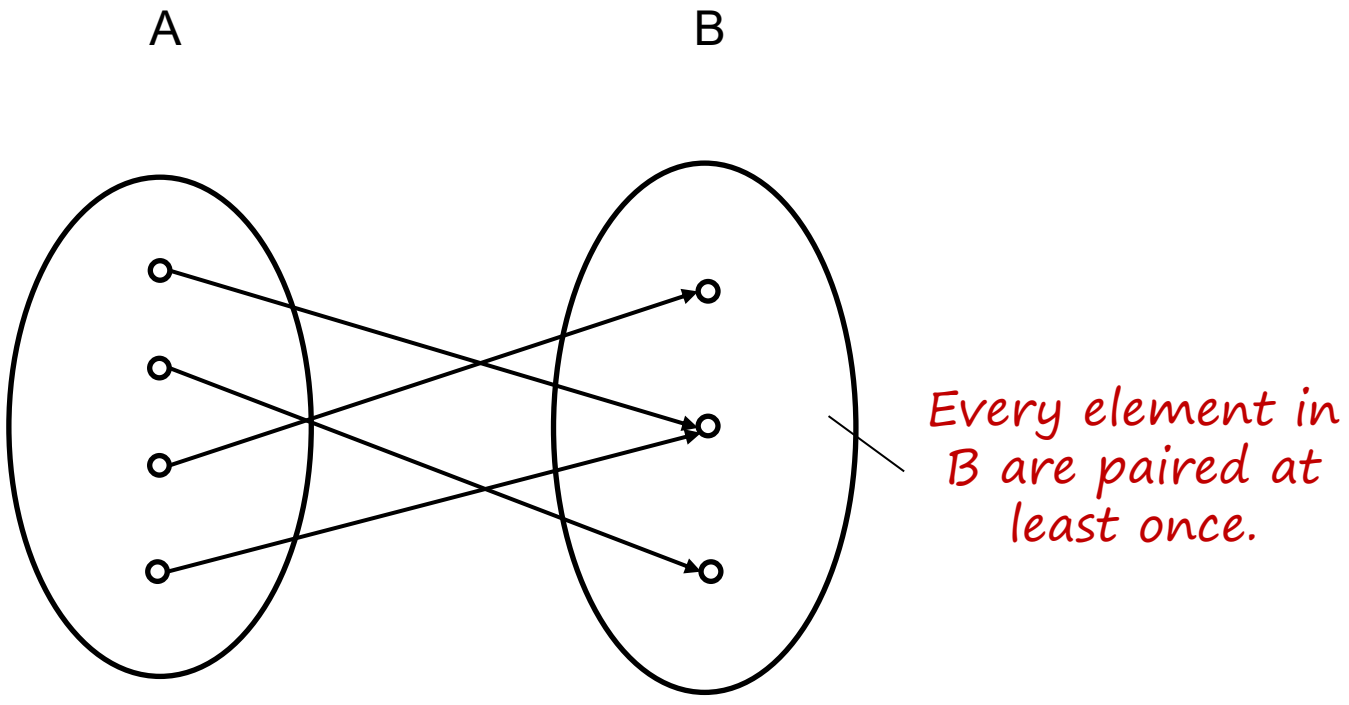
Now $g(x_1), g(x_2) \in B$, and $g(x_1) = g(x_2)$, then we have $f\big(g(x_1)\big) = f\big(g(x_2)\big)$.

But $f \circ g$ is injective, it means $f\big(g(x_1)\big) \neq f\big(g(x_2)\big)$. Contradiction

Q.E.D



A special function



A special function

Surjection(滿射)

- $f: A \to B$ is a **surjection** or **surjective** if $ran(f) = B$, or for any $y \in B$, there exists $x \in A$ such that $f(x) = B$

$$(\forall y)(\exists x)\big(y \in B \to (x \in A \wedge f(x) = y)\big)$$

Every possible output has an input

Surjection

- $f(x) = x + 5\ (\mathbb{R} \to \mathbb{R})$
  - surjective.
  - For any $x_0 \in \mathbb{R}$, $f(x_0 - 5) = x_0$
- $f(x) = x^2\ (\mathbb{R} \to \mathbb{R})$
  - not surjective.
  - There's no element $x_0 \in \mathbb{R}$ such that $f(x_0) = x_0^2 = -1$

Surjection and Composition

Let $g: \boldsymbol{A} \to \boldsymbol{B}$, $f: \boldsymbol{B} \to \boldsymbol{C}$, if f, g are surjective, then $f \circ g$ is also surjective.

Proof:

For any $y_1 \in C$, we have $x_1 \in B$ such that $f(x_1) = y_1$ since f is surjective.

Now $x_1 \in B$, so we have $x_0 \in A$ such that $g(x_0) = x_1$ since g is surjective.

So for any $y_1 \in C$, we have $x_0 \in A$ such that $f\big(g(x_0)\big) = y_1$, or $(f \circ g)(x_0) = y_1$, it means that $f \circ g$ is surjective

Q.E.D

Surjection and Composition

Let $g: \boldsymbol{A} \to \boldsymbol{B}$, $f: \boldsymbol{B} \to \boldsymbol{C}$, if $f \circ g$ is surjective, then f is also surjective.

Proof:

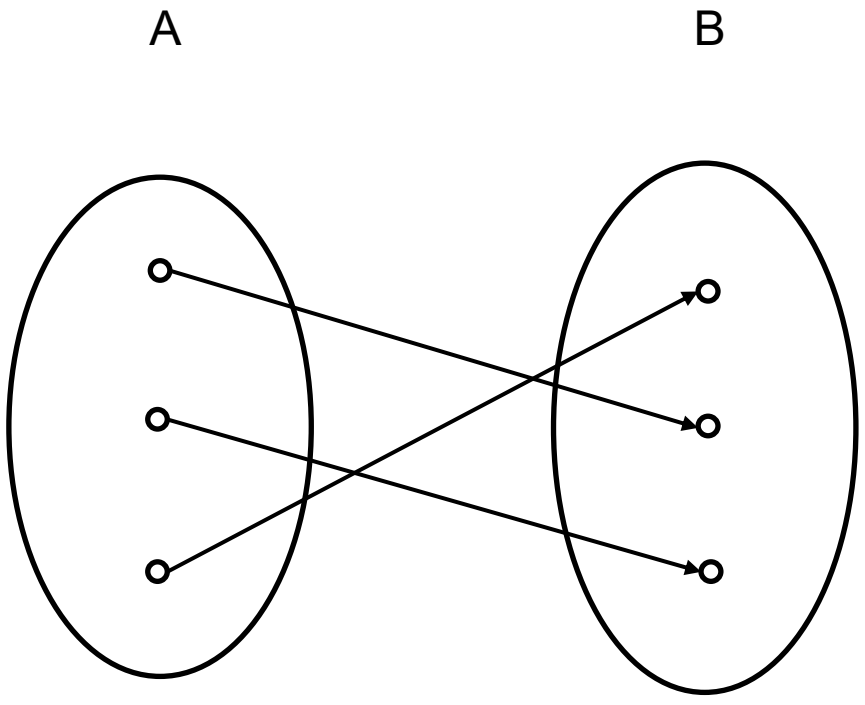
For any $y \in C$, we have $x \in A$ such that $(f \circ g)(x) = y$ as $f \circ g$ is surjective.

Let $x_0 = g(x) \in B$, so we have $(f \circ g)(x) = f\big(g(x)\big) = f(x_0) = y$.

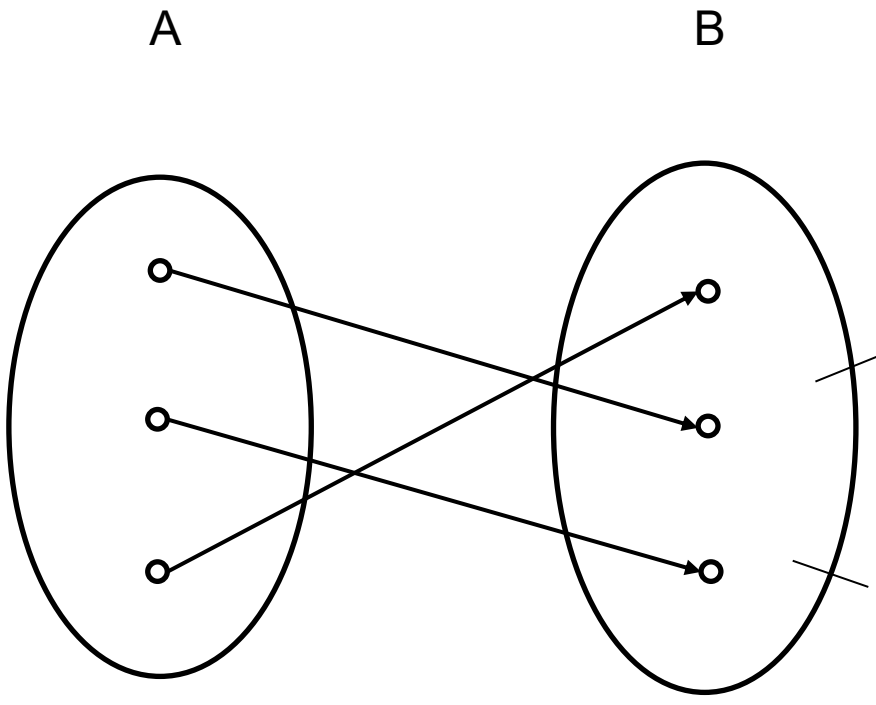
So for any $y \in C$, we have $x_0 \in B$ such that $f(x_0) = y$.

f is surjective.

Q.E.D



A special function



Every element in B are paired at least once.

No elements in B are paired twice

A special function

Bijection(双射)

- $f: A \to B$ is a **bijection** or **bijective** if f is **both** injective and surjective.
- Bijections are sometimes called **one-to-one correspondences(一一对应)**.

Bijection

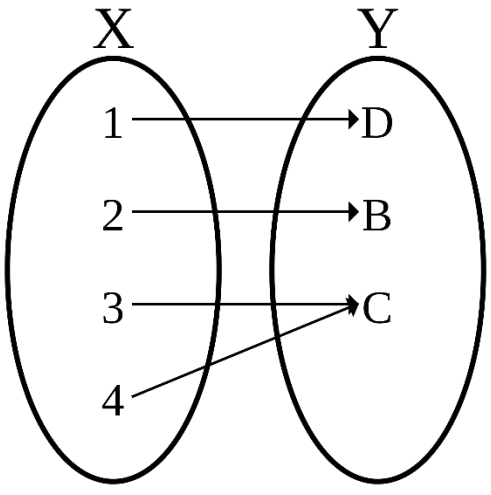
- $f(x) = x + 5$ $(\mathbb{R} \to \mathbb{R})$ is bijective.
- $f(x) = e^x$ $(\mathbb{R} \to \mathbb{R})$ is not bijective since it's not surjective.
- $f(x) = x^3 - x$ $(\mathbb{R} \to \mathbb{R})$ is not bijective since it's not injective.

Bijection and Composition

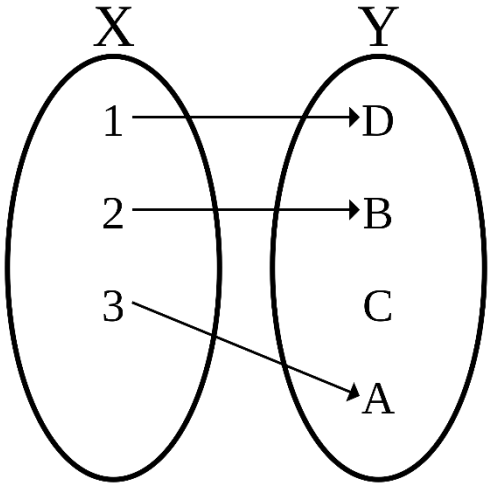
Let $g: A \to B,\ f: B \to C$, if f, g is bijective, then $f \circ g$ is also bijective.

Let $g: A \to B,\ f: B \to C$, if $f \circ g$ is bijective, then g is injective and f is surjective.

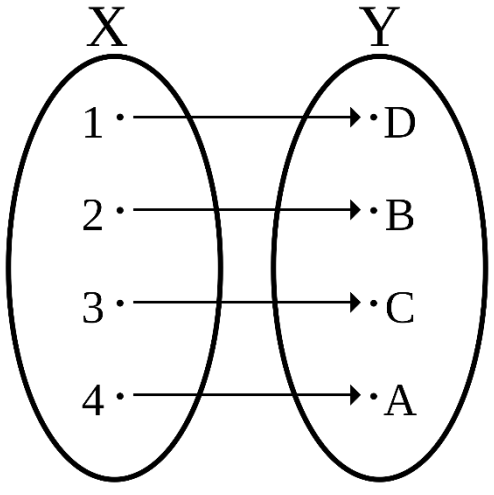
Bijection, injection and surjection



Surjection



Injection



Bijection

Comparing Cardinalities

For set A and B, A and B are **equianumerous (等势)** if there is a way to pair their elements off without leaving any elements uncovered, marked as $A \approx B$. Otherwise marked as $\neg A \approx B$.

Comparing Cardinalities

~~For set A and B, A and B are **equianumerous (等势)** if there is a way to pair their elements off without leaving any elements uncovered, marked as $A \approx B$. Otherwise marked as $\neg A \approx B$.~~

For set A and B, A and B are **equianumerous (等势)** if there is a bijection from $\boldsymbol{A}$ to $\boldsymbol{B}$, marked as $A \approx B$. Otherwise marked as $\neg A \approx B$.

Comparing Cardinalities

For set A and B, $card(A) \leq card(B)$ if there is a way to pair their elements off without leaving **any elements of A** uncovered.

Comparing Cardinalities

~~For set A and B, $card(A) \leq card(B)$ if there is a way to pair their elements off without leaving **any elements of A** uncovered.~~

For set A and B, $card(A) \leq card(B)$ if there **is an injection from $\boldsymbol{A}$ to $\boldsymbol{B}$**.

Comparing Cardinalities

- $f: A \to B$
  - If f is injective, then $|A| \leq |B|$
  - If f is surjective, then $|A| \geq |B|$
  - If f is bijective, then $|A| = |B|$

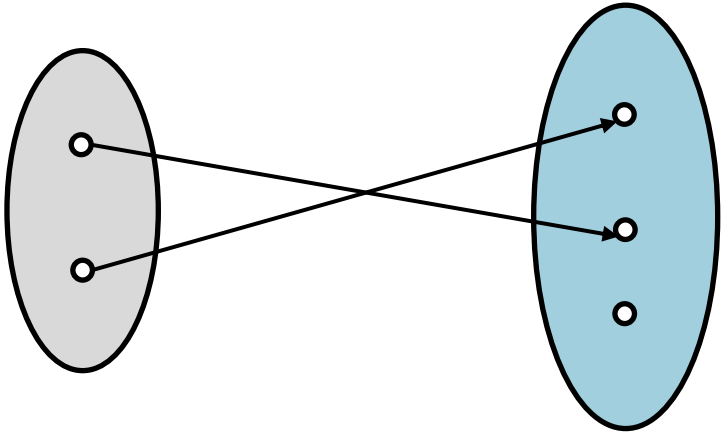
Axiom of Choice

ZF.10 Axiom of choice(选择公理)

For any relation R, there exists a function F, such that F is a subset of R and the domain of F and R are equal.

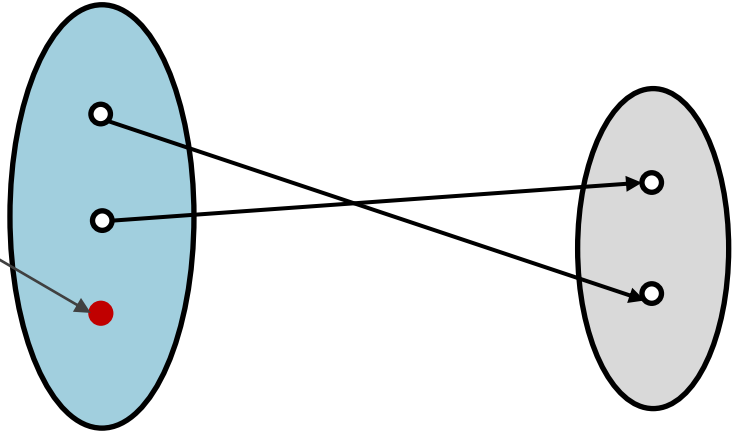
$$(\forall \text{ Relation } R)(\exists \text{ Function } F)(F \subseteq R \wedge dom(R) = dom(F))$$

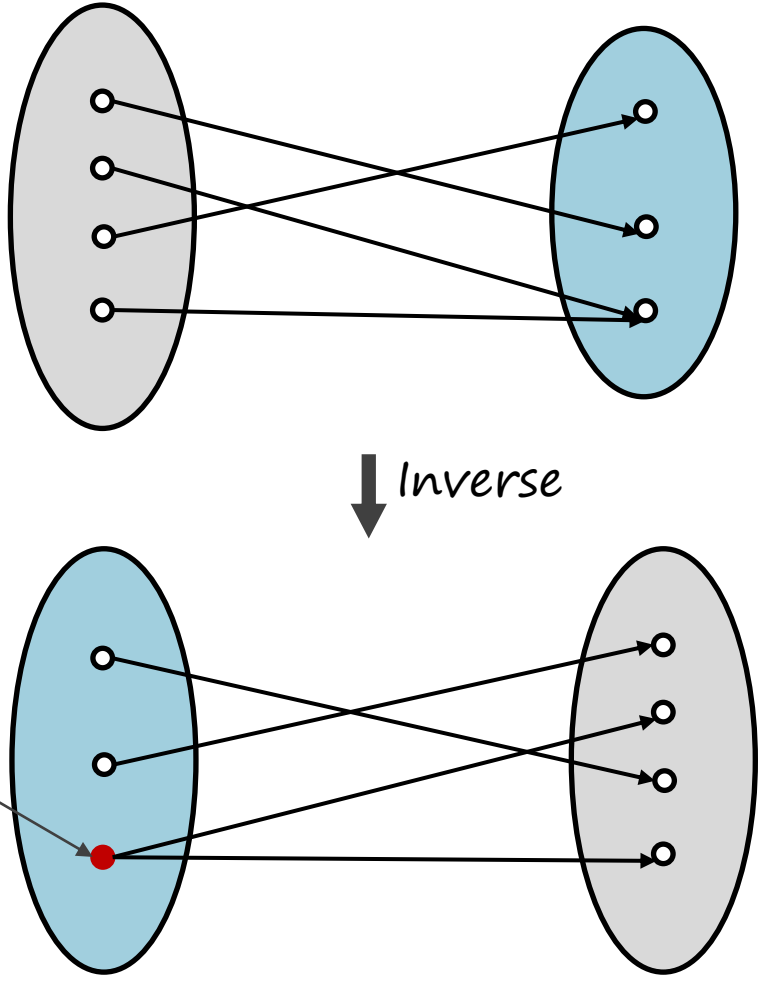
- $\boldsymbol{R = \{\langle 1, a\rangle, \langle 1, b\rangle, \langle 2, a\rangle\}}$
- $f_1 = \{\langle 1, a\rangle, \langle 2, a\rangle\}$
- $f_2 = \{\langle 1, b\rangle, \langle 2, a\rangle\}$



Inverse

Some element are not paired




Inverse
Some element are paired twice

Inverse of Function

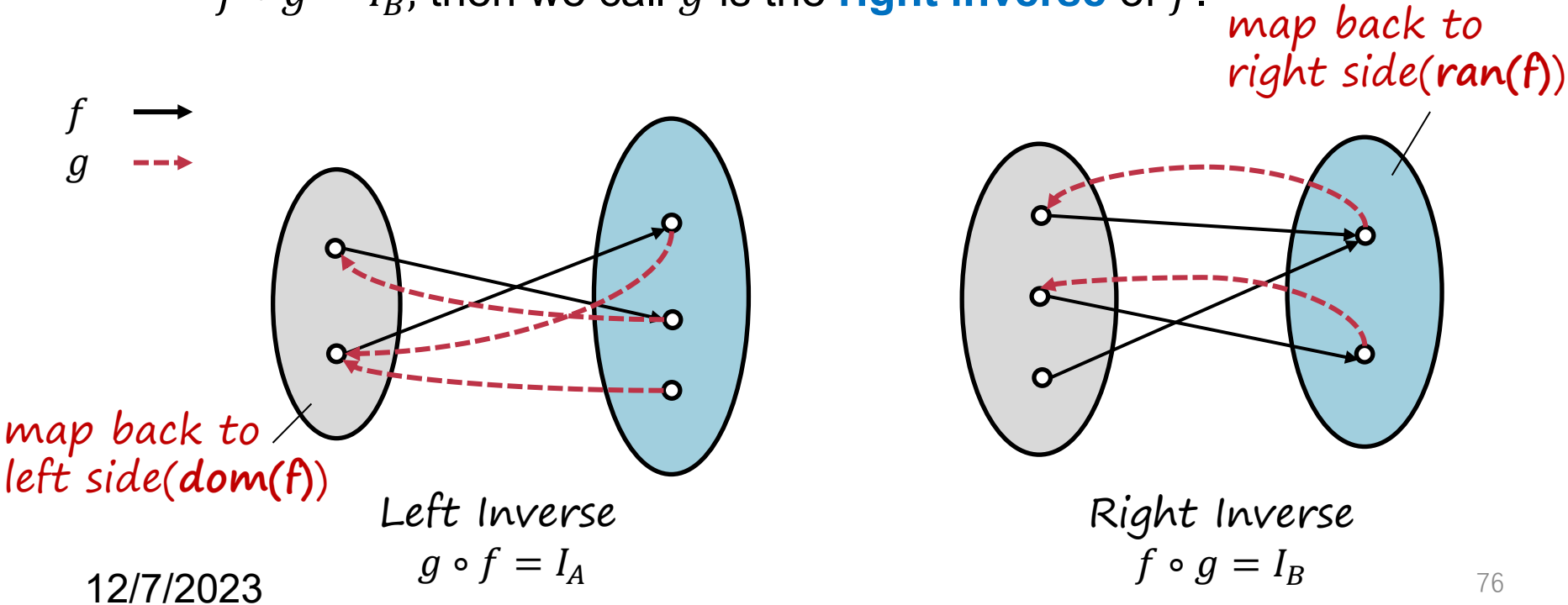
- Let $f: \boldsymbol{A} \to \boldsymbol{B}$
  - The inverse (defined by relation) of f may not be a function
  - If the inverse of f is a function, we call f is **inversible**（可逆的）

$f: A \to B$, if there exists $g: B \to A$, such that $g \circ f = I_A$ and $f \circ g = I_B$, we call that f is **inversible**, and g is the **inverse function(反函数)** of f.

Some times $g \circ f = I_A$ and $f \circ g = I_B$ may not hold simultaneously.

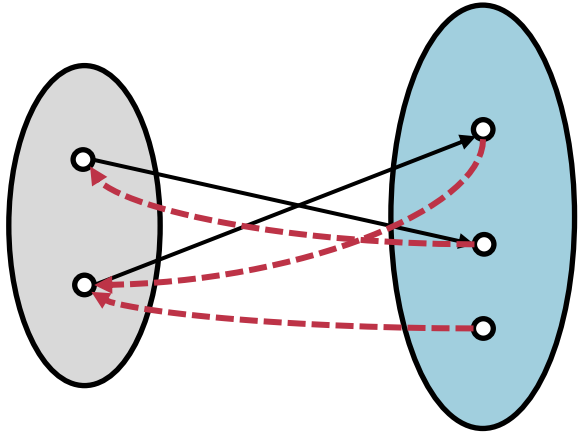
Left Inverse(左逆) and Right Inverse(右逆)

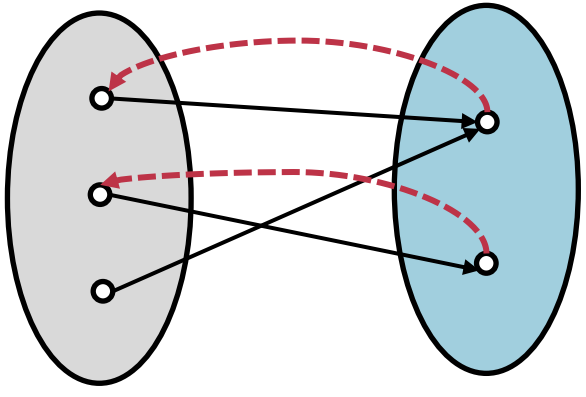
- Let $f: \boldsymbol{A} \to \boldsymbol{B}$, if there exists $g: \boldsymbol{B} \to \boldsymbol{A}$, such that
  - $g \circ f = I_A$, then we call g is the **left inverse** of f.
  - $f \circ g = I_B$, then we call g is the **right inverse** of f.



Left Inverse and Right Inverse

- Let $f: \boldsymbol{A} \rightarrow \boldsymbol{B}$, $A \neq \emptyset$
  - f has left inverse if and only if f is injective.
  - f has right inverse if and only if f is surjective.
  - f has both left and right inverse if and only if f is bijective.
  - f is bijective, then the left and right inverse of f are equal.





Left Inverse and Right Inverse

- Let $f: \boldsymbol{A} \to \boldsymbol{B}$, $A \neq \emptyset$, assume that g is the left inverse of f, and h is the right inverse of f.
  - We have $g \circ f = I_A$, $f \circ h = I_B$
  - Then $g = g \circ I_B = g \circ (f \circ h) = (g \circ f) \circ h = I_A \circ h = h$
  - So if f has both left and right inverse, then they are equal.

Inverse of Function

- Let $f: \boldsymbol{A} \to \boldsymbol{B}$
  - The inverse (defined by relation) of f may not be a function
  - If the inverse of f is a function, we call f is **inversible**（可逆的）

$f: A \to B$, if there exists $g: B \to A$, such that $g \circ f = I_A$ and $f \circ g = I_B$, we call that f is **inversible**, and g is the **inverse function(反函数)** of f.

- f has both left and right inverse if and only if f is bijective.

f is inversible if and only if f is bijective.

Inverse of Function

- Let $f: \boldsymbol{A} \to \boldsymbol{B}$ be a bijection
  - f^{-1} is a function and is the inverse function of f.
  - f^{-1} is bijective too.
  - $\forall x \in A, f^{-1}\big(f(x)\big) = x.\ \forall y \in B, f\big(f^{-1}(y)\big) = y$

How we investigate computability?

2.1 Discussion on Problems

Part1. Intro & Set theory(I) : Basics & Formal Language.

Part2. Set Theory(II): Axiom system & Cardinality.

Part3. Capture Structures: Binary Relation & Function

2.2 Discussion on Computation

Part1. Turing Machine Basics.

Part2. Variants of Turing Machine. Church-Turing Thesis.

2.3 Discussion on computability

Part1. The Language of Turing Machine. R & RE

Part2. Undecidability.

Ultimate Problem:

What problems can a computer solve?

What We Have Done

- **Computational Problem(计算问题)**
  - Well-defined input
  - Constraints that the output must satisfy.
- **A computational problem is computable(可计算) if there is a procedure (or algorithm) for it which:**
  - Rigorously follows some instructions, requires no ingenuity.
  - Always finishes (terminates) after a finite number of steps.

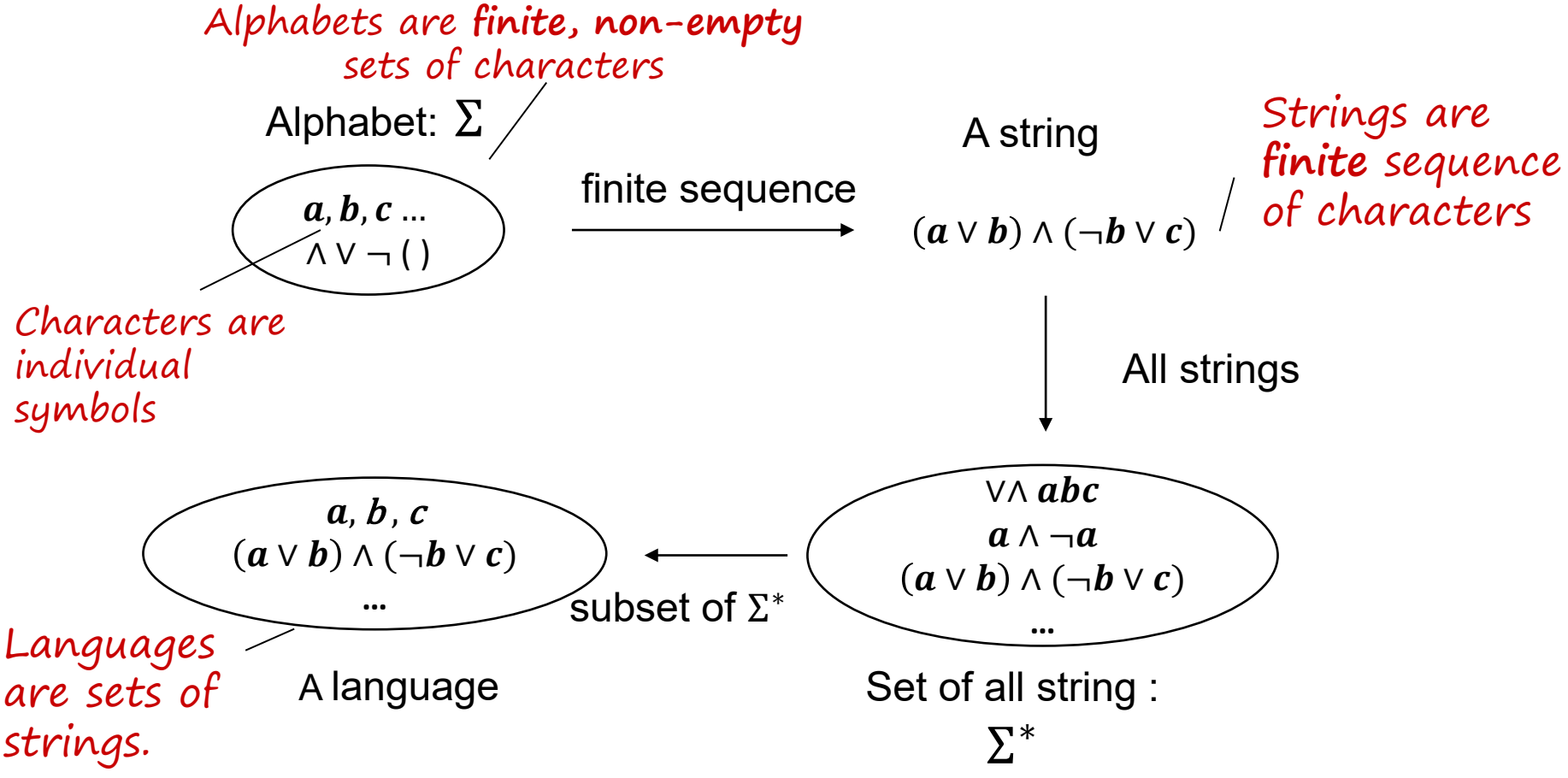
What We Have Done

- **A computational problem can be converted into language recognizing problem.**
  - Set basics & formal language theory
  - A language is naturally a set of string.
  - Given a language L and take w as input, determine whether $w \in L$.

~~What problems can a computer solve?~~

What languages can a computer recognize?

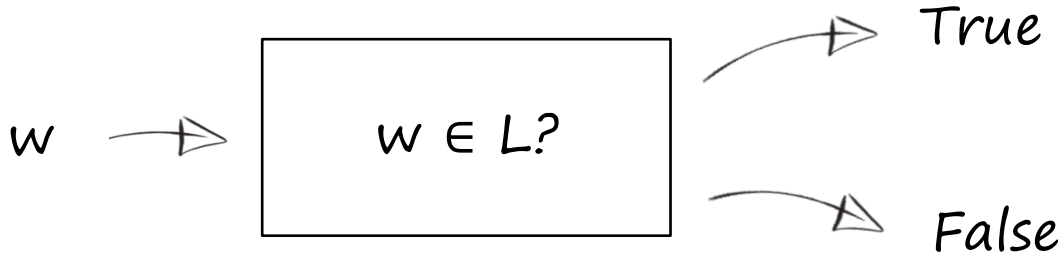
Recap: Formal Languages



Recap: Language Recognizing

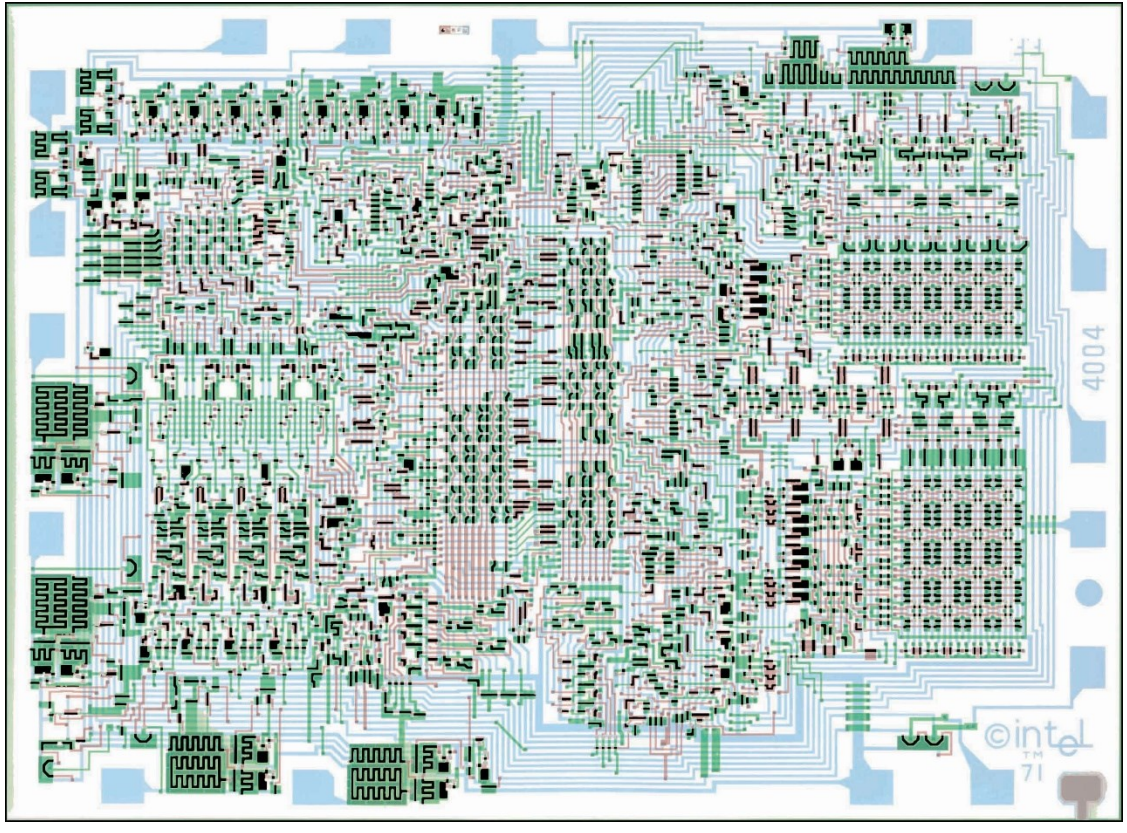
Throughout the whole course of computability, we'll focus on the membership problem of a language, or language recognizing, that is:

Given a language L and take w as input, determine whether $w \in L$.



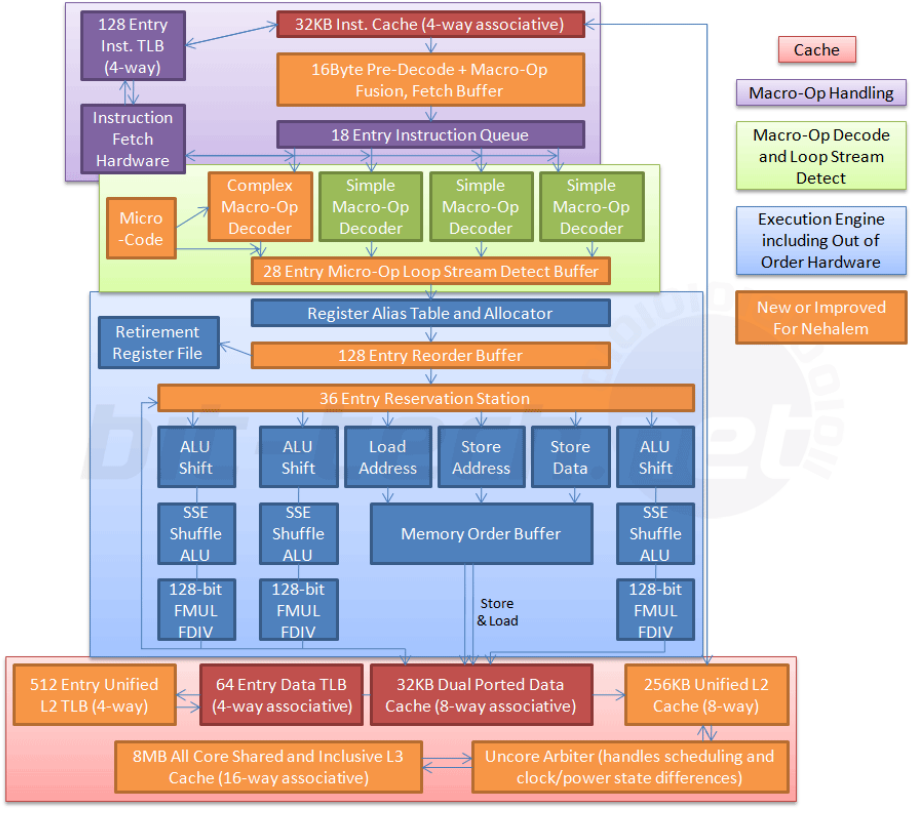
What languages can a **computer** recognize?

Computers are complicated.



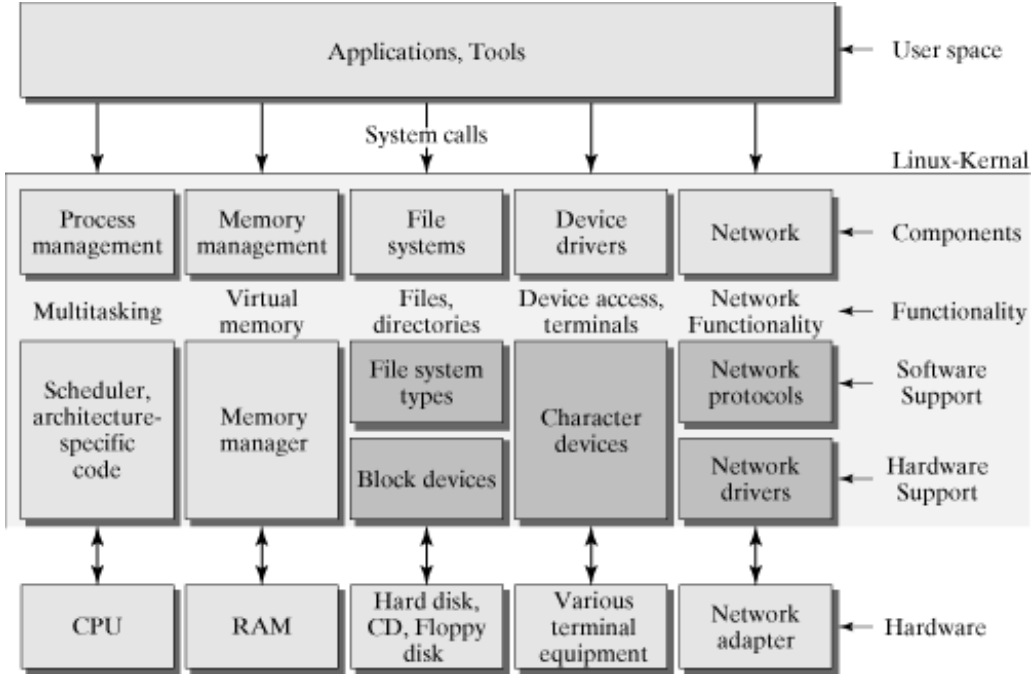
Intel 4004, first commercial CPU

Computers are complicated.



Core i7 Architecture.

Computers are complicated.



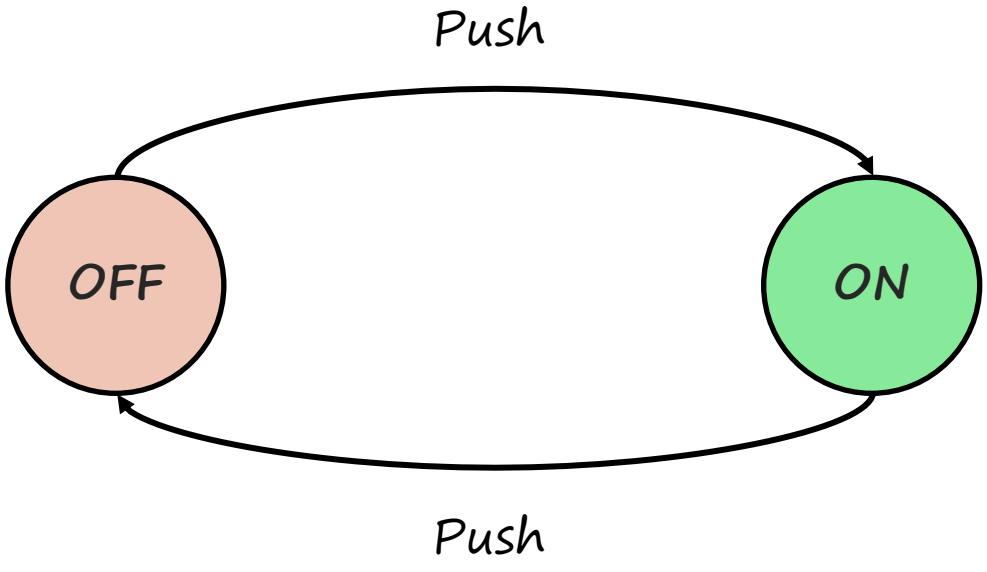
Architecture of Operating System

We need to model computer in a simple and clean manner!!

Automata(自动机) Theory

- An automaton is a mathematical model of a computing device
  - It is significantly easier to manipulate our abstract models of computers than it is to manipulate actual computers
  - If we pick our models correctly, we can make broad, sweeping claims about huge classes of real computers by arguing that they're just special cases of our more general models.

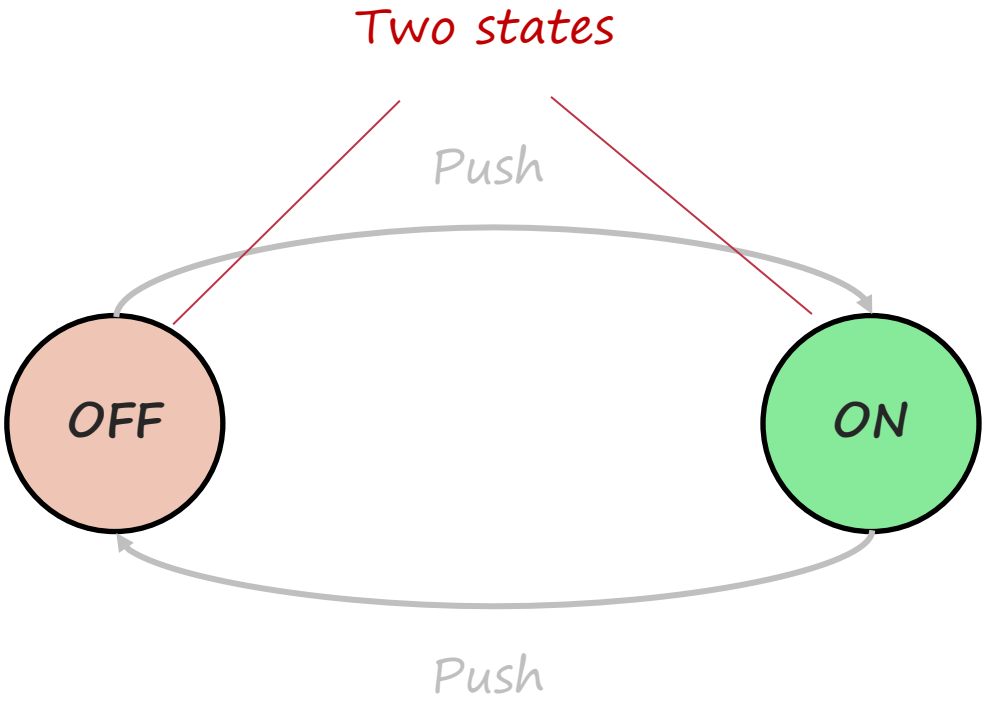
What is an Automaton



An "automata" for an on-off switch

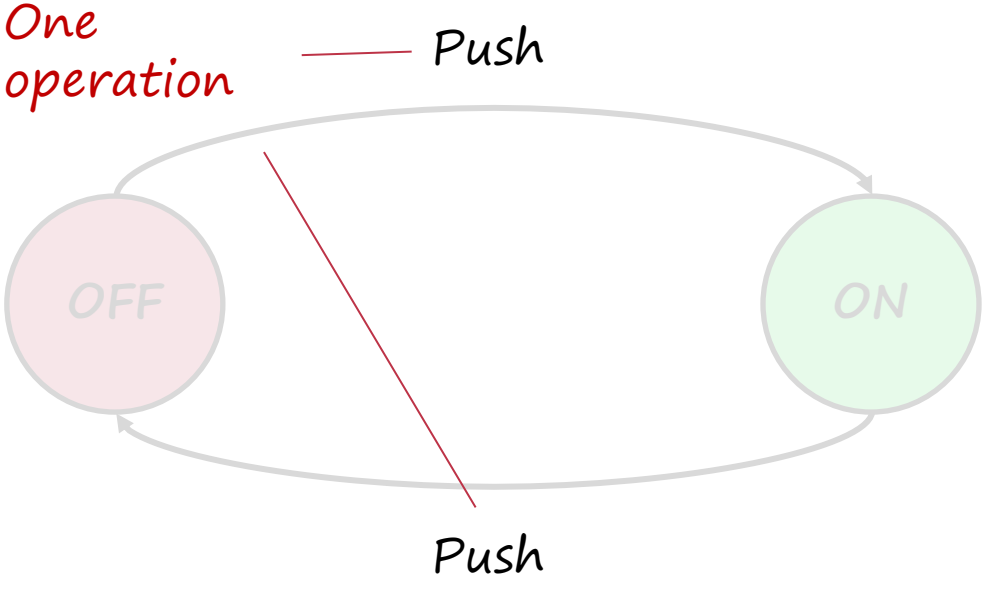
What is an Automaton

- Q: {“off”, “on”}



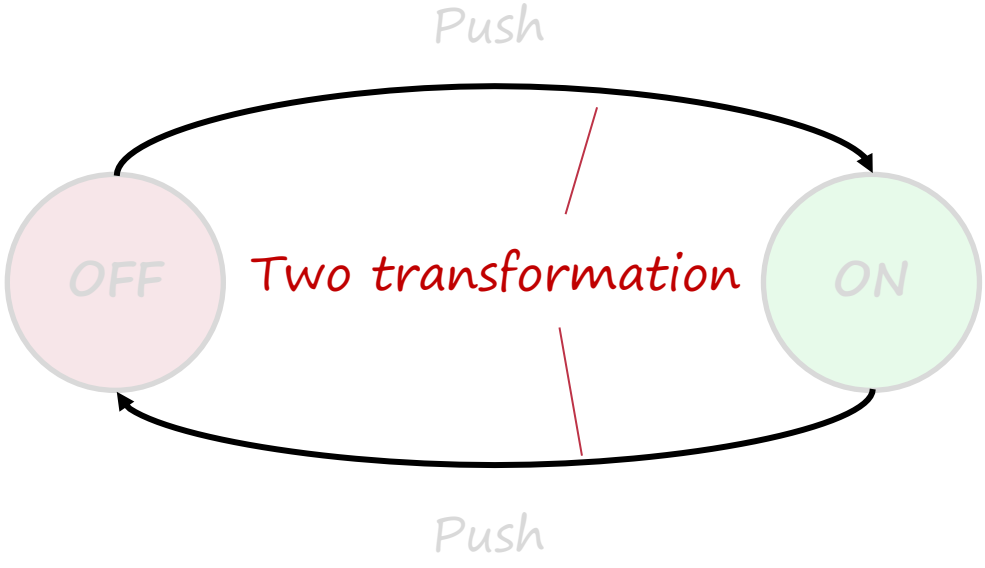
What is an Automaton

- Q: {“off”, “on”}
- Σ : {“push”}

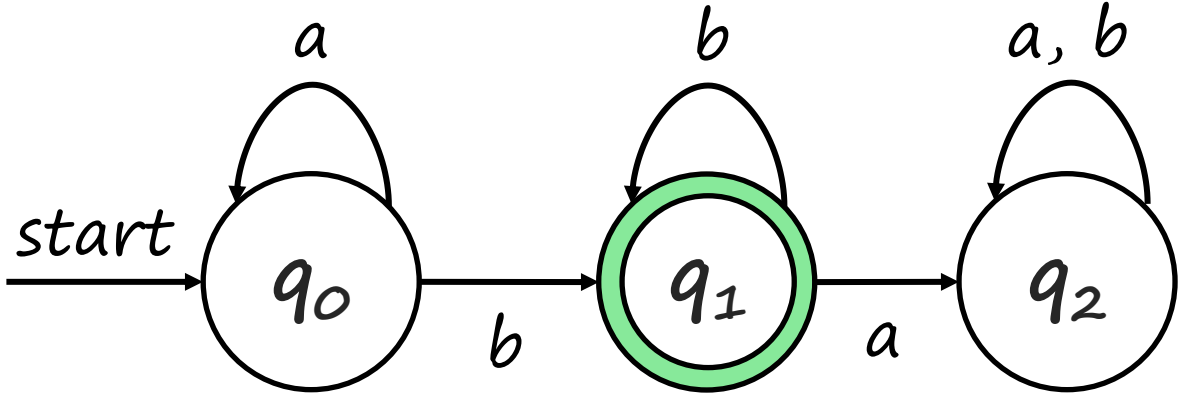


What is an Automaton

- Q: {“off”, “on”}
- Σ : {“push”}
- δ: {(“on”, “push”, “off”), (“off”, “push”, “on”)}



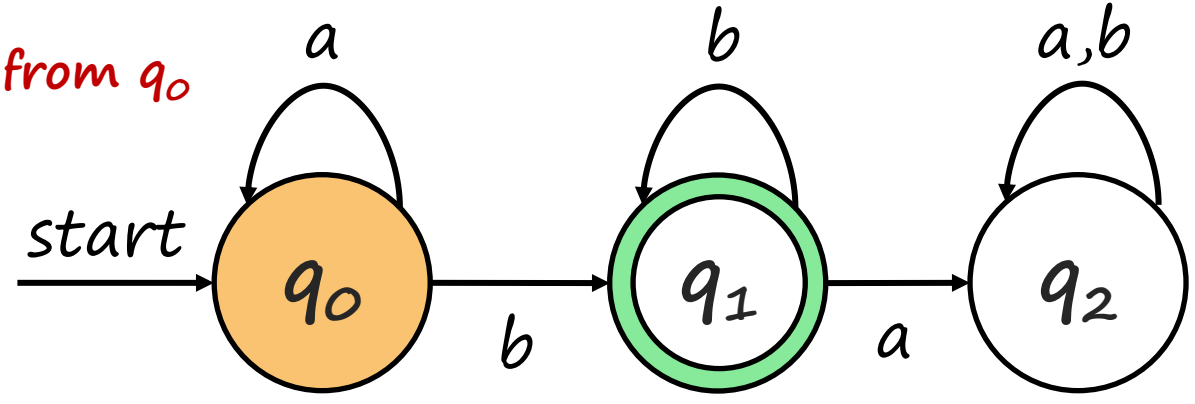
What is an Automaton



A automata for "aa...bb..."

What is an Automaton

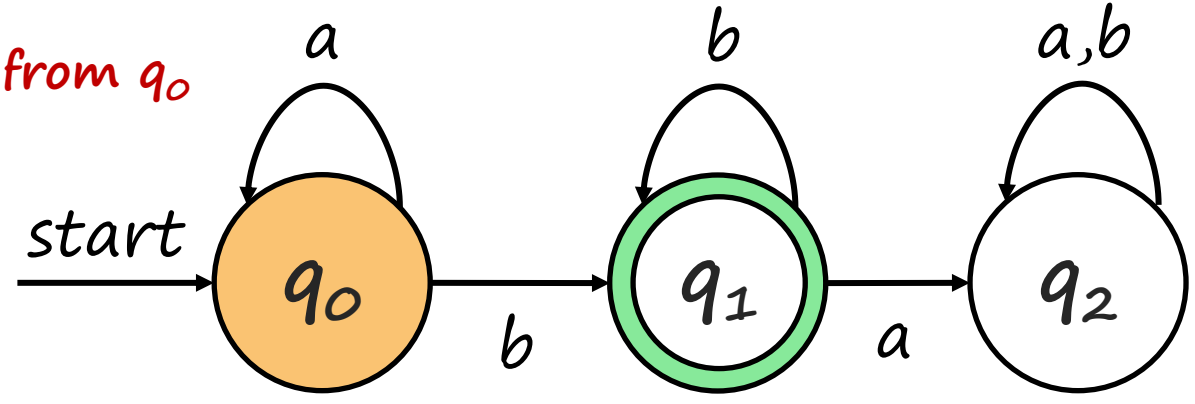
We'll start from q_0



Input: a a b b

What is an Automaton

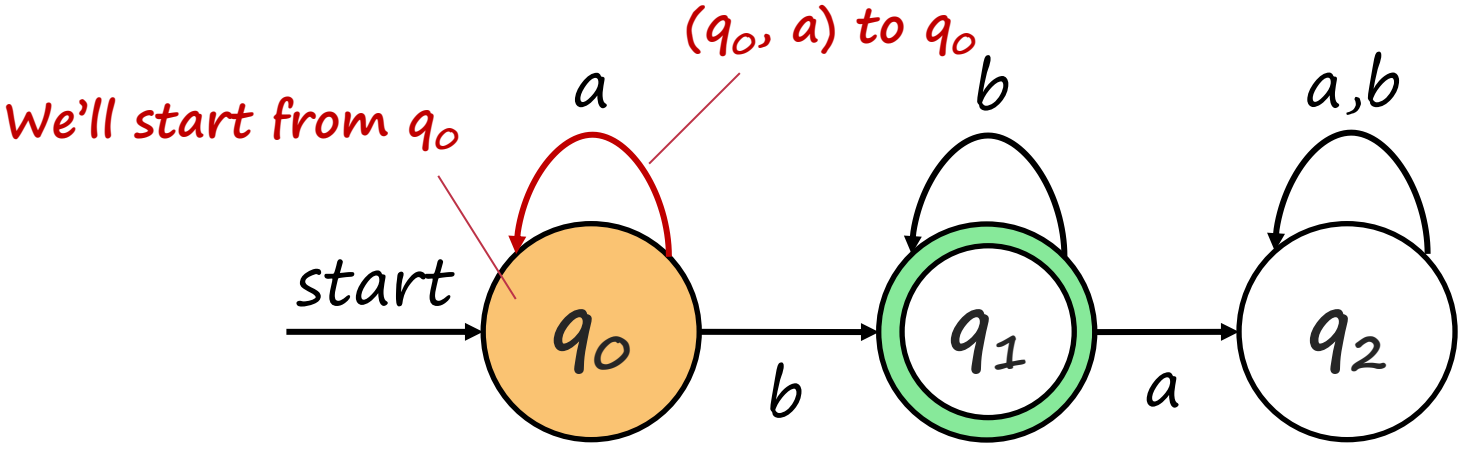
We'll start from q_0



Input: a a b b

next input symbol

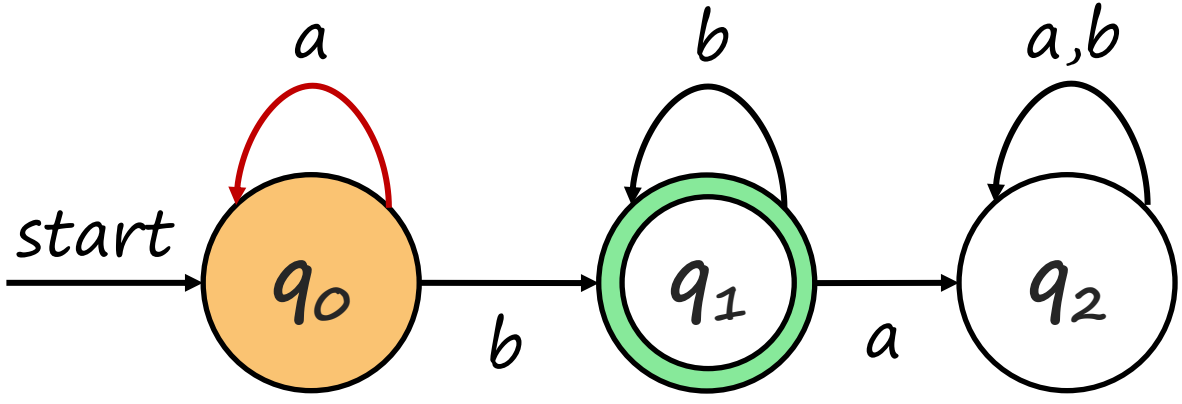
What is an Automaton



Input: a a b b

next input symbol

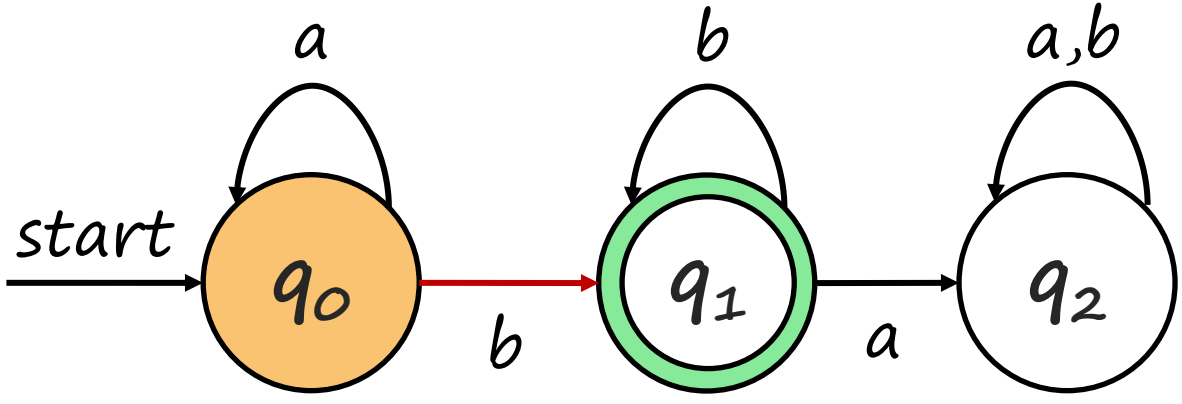
What is an Automaton



Input: a a b b

next input symbol

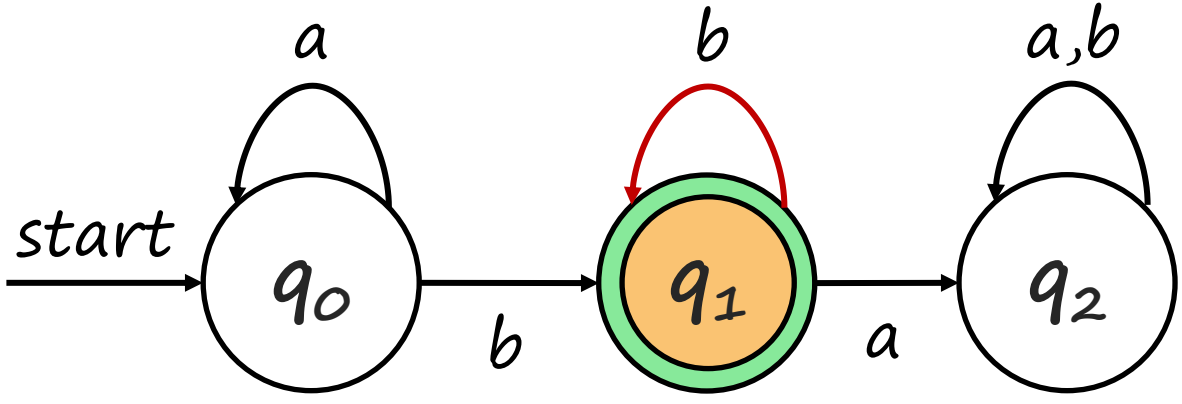
What is an Automaton



Input: a a b b

next input symbol

What is an Automaton

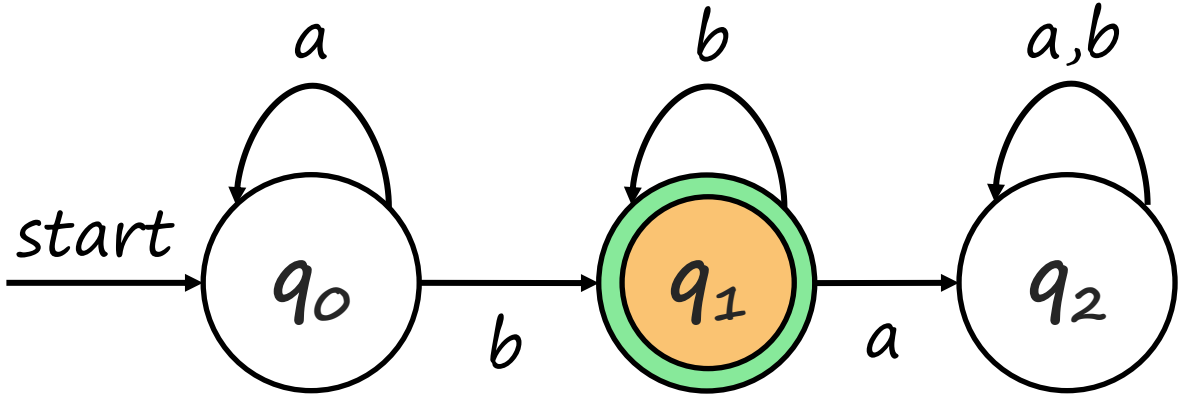


Input: a a b b

next input symbol

What is an Automaton

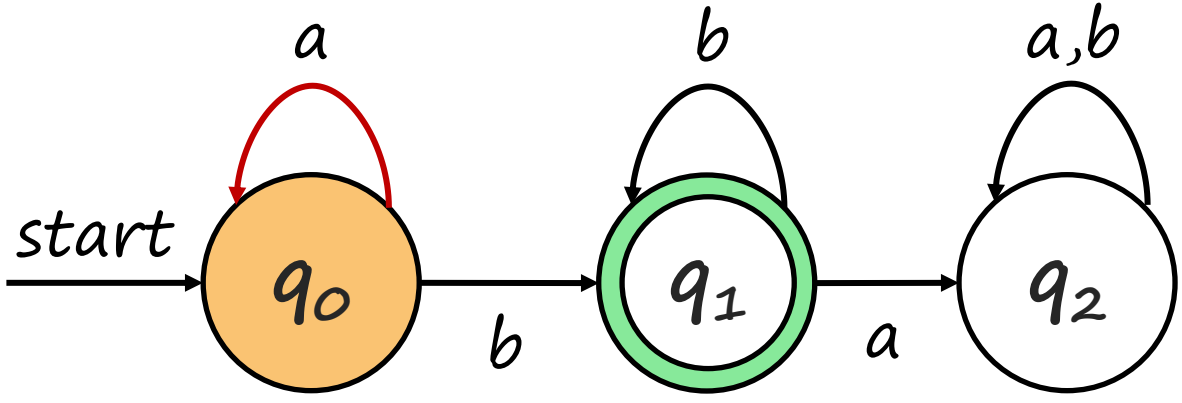
q_1 is the final state.



Input: a a b b

next input symbol

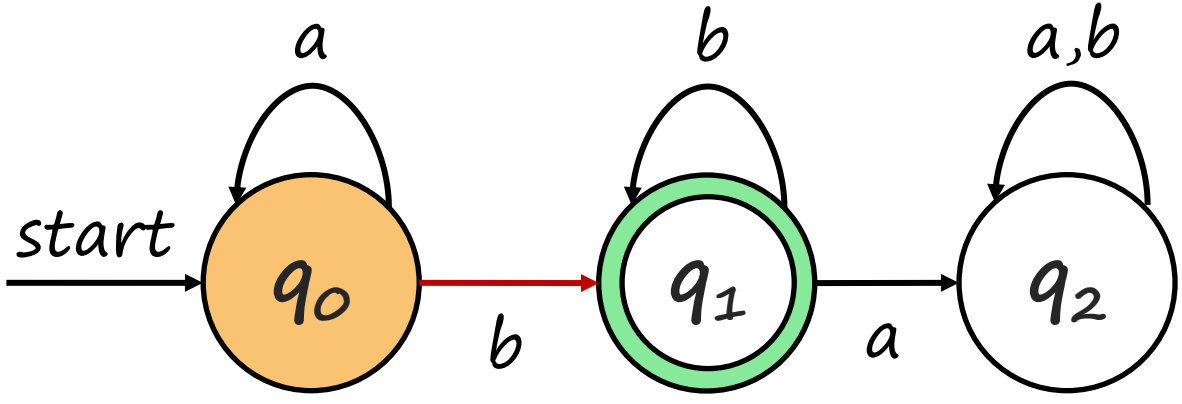
What is an Automaton



Input: a b b a

next input symbol

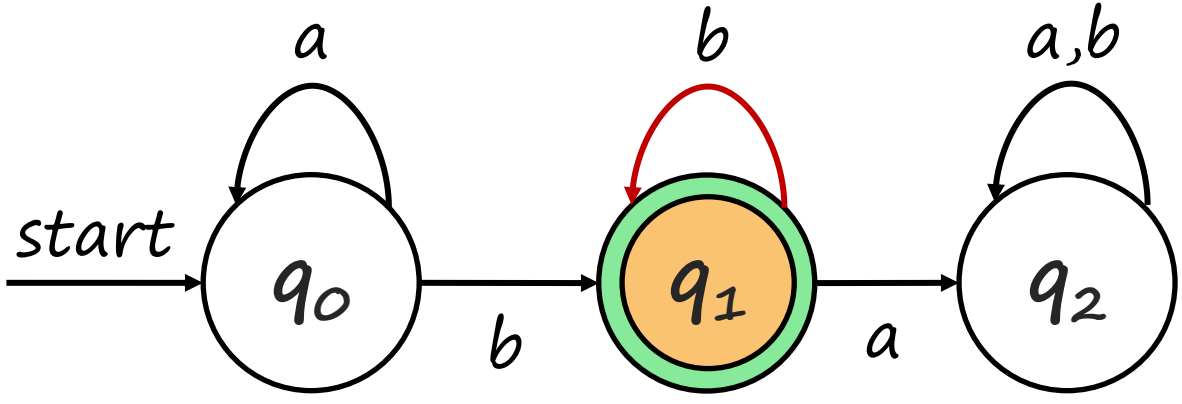
What is an Automaton



Input: a b b a

next input symbol

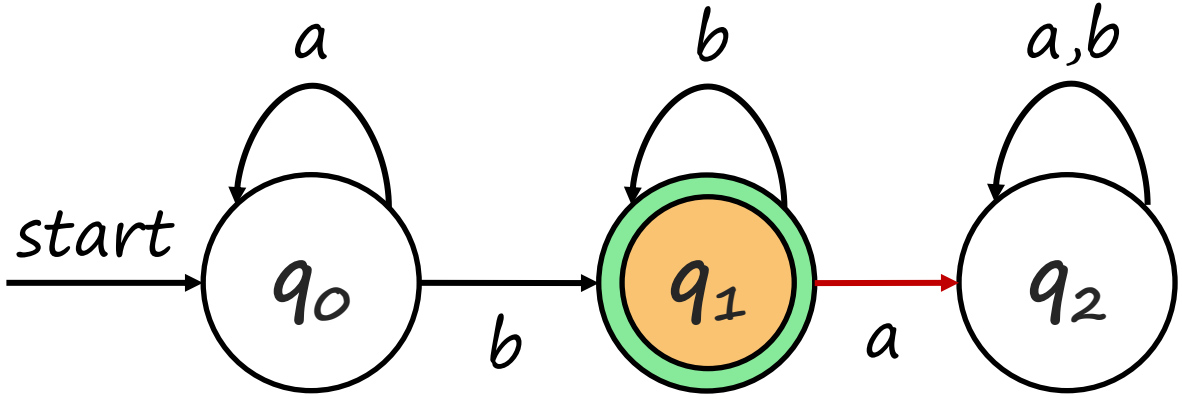
What is an Automaton



Input: a b b a

next input symbol

What is an Automaton

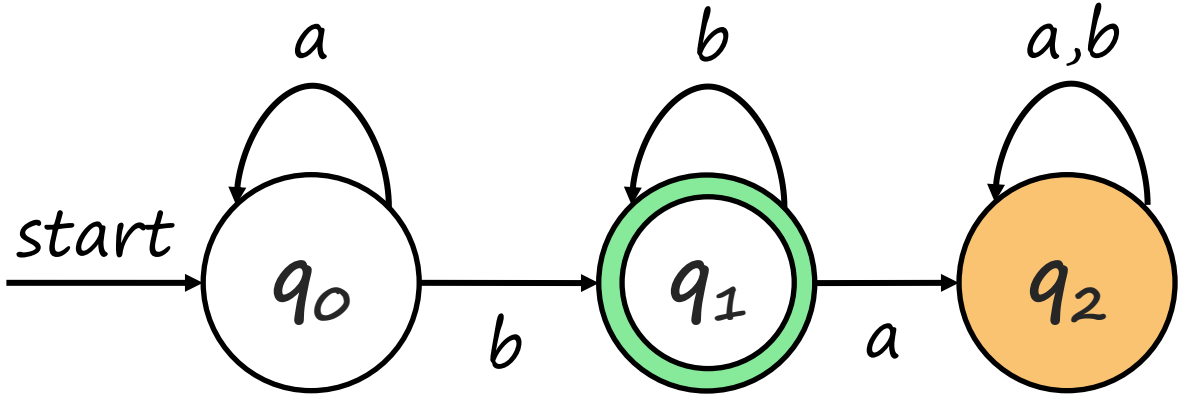


Input: a b b a

next input symbol

What is an Automaton

q_2 is not the final state.

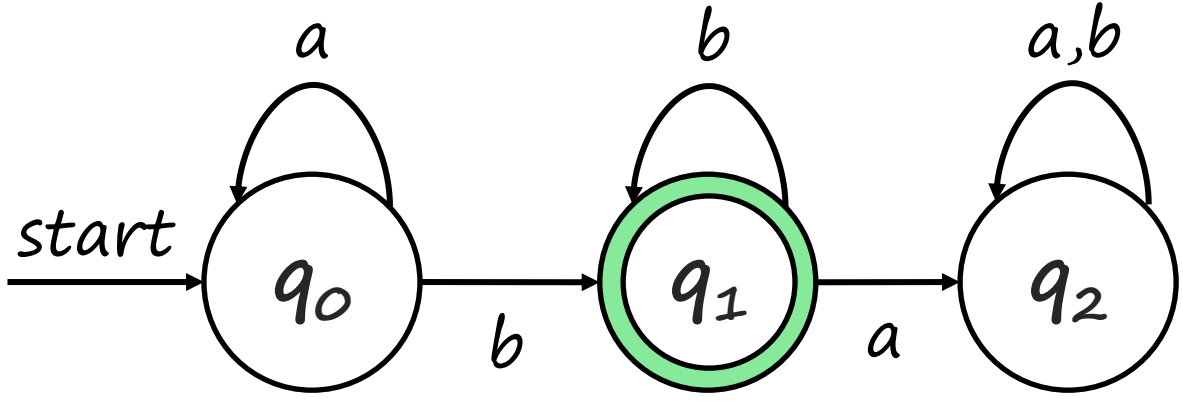


Input: a b b a

next input symbol

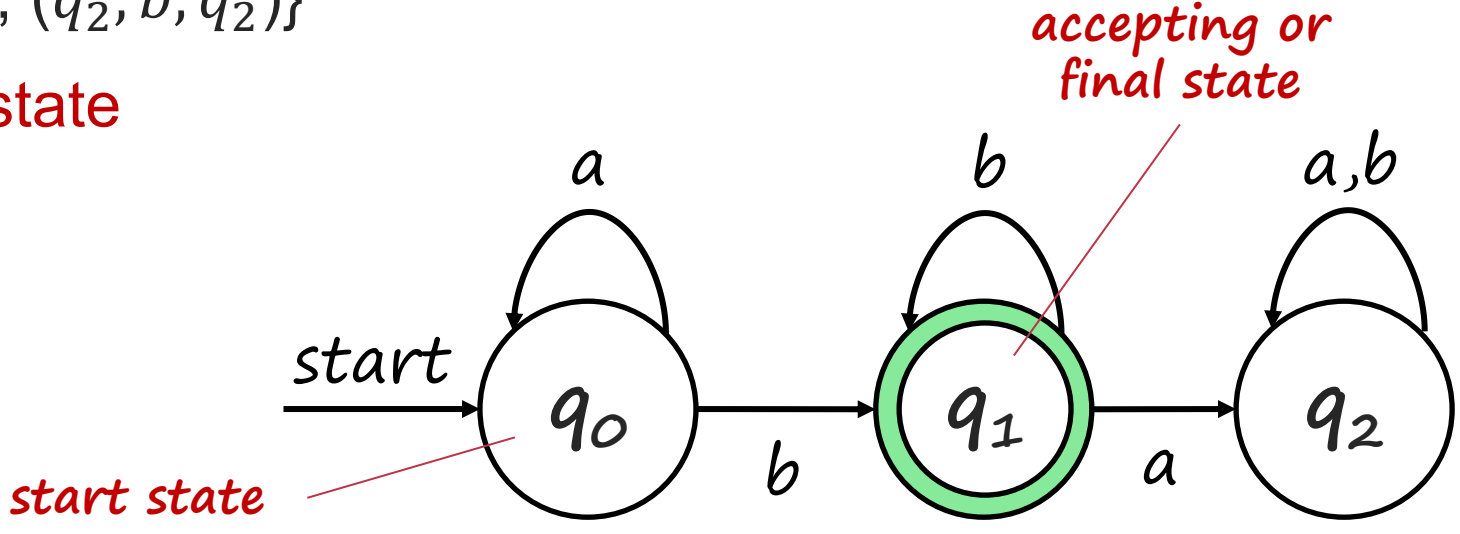
What is an Automaton

- Q: $\{q_0, q_1, q_2\}$
- Σ : $\{a, b\}$
- δ: $\{(q_0, a, q_0), (q_0, b, q_1), (q_1, b, q_1), (q_1, a, q_2), (q_2, a, q_2), (q_2, b, q_2)\}$



What is an Automaton

- Q: $\{q_0, q_1, q_2\}$
- Σ : $\{a, b\}$
- δ: $\{(q_0, a, q_0), (q_0, b, q_1), (q_1, b, q_1), (q_1, a, q_2), (q_2, a, q_2), (q_2, b, q_2)\}$
- q_0: start state
- F: $\{q_1\}$



Deterministic Finite Automaton(DFA, 确定有限自动机)

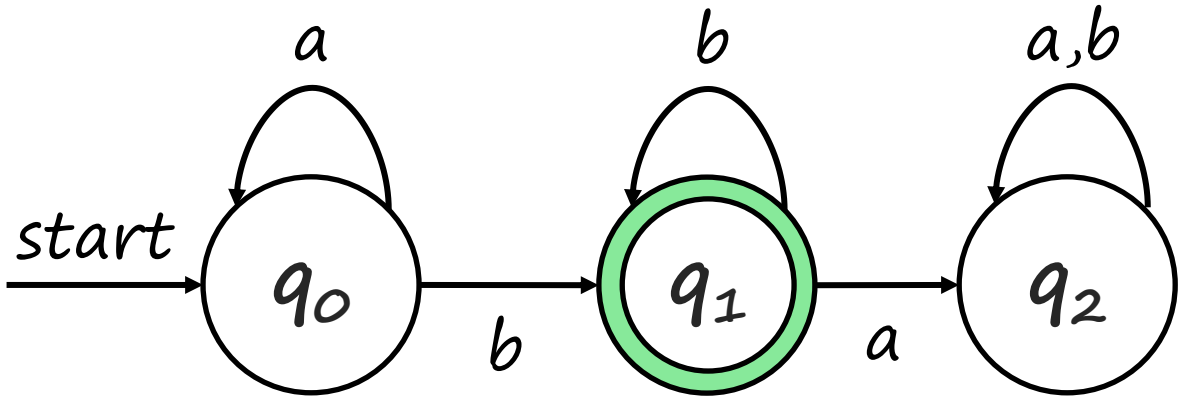
- A **deterministic finite automaton(DFA)** consists of:
  - Q:A finite set of states
  - Σ: A finite set of input symbols
  - δ: A transition function. $Q \times \Sigma \to Q$
  - q_0: The start state
  - F: A set of final or accepting states. $F \subseteq Q$
- We can donate a DFA as a “five tuple”

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

Deterministic Finite Automaton(DFA)

- A DFA takes a finite string(sequence of input symbols) as input, and process the input symbols of the string one by one, and change its state according to the transition function δ.
- DFA will always stop, if the state is in the set of accepting states(F) when a DFA stops, we say that this DFA **accepts** the input. Otherwise it **rejects** the input.

Simpler Notations for DFA



Transition Diagram

start state

	a	b
→ q_0	q_0	q_1
* q_1	q_2	q_1
q_2	q_2	q_2

final state

Transition Table

Extending the Transition Function

For a DFA $(Q, \Sigma, \delta, q_0, F)$, we can define an ***extended transition function***

$$\hat{\delta}: Q \times \Sigma^* \rightarrow Q$$

$\hat{\delta}$ takes a state q and a string w as input, and returns the state after processing string w from state q.

- $\hat{\delta}(q, \epsilon) = q$
- If $w = xa$, where $a \in \Sigma$. Then $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$

Extending the Transition Function

- $\hat{\delta}(q_0, \epsilon) = q_0$
- $\hat{\delta}(q_0, 1) = \delta\big(\hat{\delta}(q_0, \epsilon), 1\big) = \delta(q_0, 1) = q_1$
- $\hat{\delta}(q_0, 11) = \delta\big(\hat{\delta}(q_0, 1), 1\big) = \delta(q_1, 1) = q_0$
- $\hat{\delta}(q_0, 110) = \delta\big(\hat{\delta}(q_0, 11), 0\big) = \delta(q_0, 0) = q_2$
- $\hat{\delta}(q_0, 1101) = \delta\big(\hat{\delta}(q_0, 110), 1\big) = \delta(q_2, 1) = q_3$
- $\hat{\delta}(q_0, 11010) = \delta\big(\hat{\delta}(q_0, 1101), 0\big) = \delta(q_3, 0) = q_1$
- $\hat{\delta}(q_0, 110101) = \delta\big(\hat{\delta}(q_0, 11010), 1\big) = \delta(q_1, 1) = q_0$

	0	1
$* \rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

The Language of DFA

- For a DFA $\boldsymbol{A} = (Q, \Sigma, \delta, q_0, F)$, the language of $\boldsymbol{A}$, donated as $L(\boldsymbol{A})$, is the set of all string that A accepts.

$$L(A) = \{w | \hat{\delta}(q_0, w) \in F\}$$

- For a certain language L, if there exists a DFA $\boldsymbol{A}$ such that $L = L(\boldsymbol{A})$, then we call L is a **regular language(正则语言)**.

We'll not take much time in DFA and regular language. Feel free to read chapter 2,3,4 in IALC if you are interested in it.

Next

- **Turing Machine Basics.**