Lecture 34

Reductions, NP-complete

NP-Completeness

NP-Completeness

Diversity in **NP**:

NP-Completeness

Diversity in **NP**:

- *Primes* is solvable in polynomial-time.

NP-Completeness

Diversity in **NP**:

- *Primes* is solvable in polynomial-time.
- *GI* is solvable in quasi-polynomial time ($O(n^{\log^c n})$).

NP-Completeness

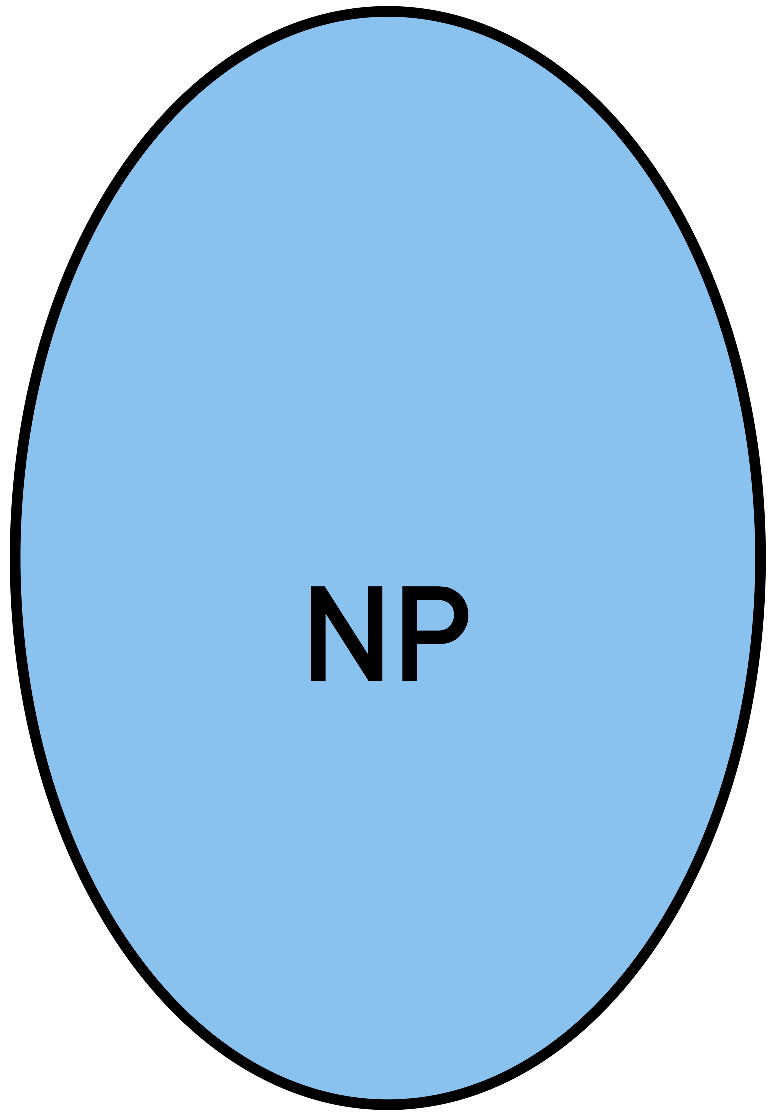
Diversity in **NP**:

- *Primes* is solvable in polynomial-time.
- *GI* is solvable in quasi-polynomial time ($O(n^{\log^c n})$).
- *IndSet* is solvable in $O(1.1996^n)$.

NP-Completeness

Diversity in **NP**:

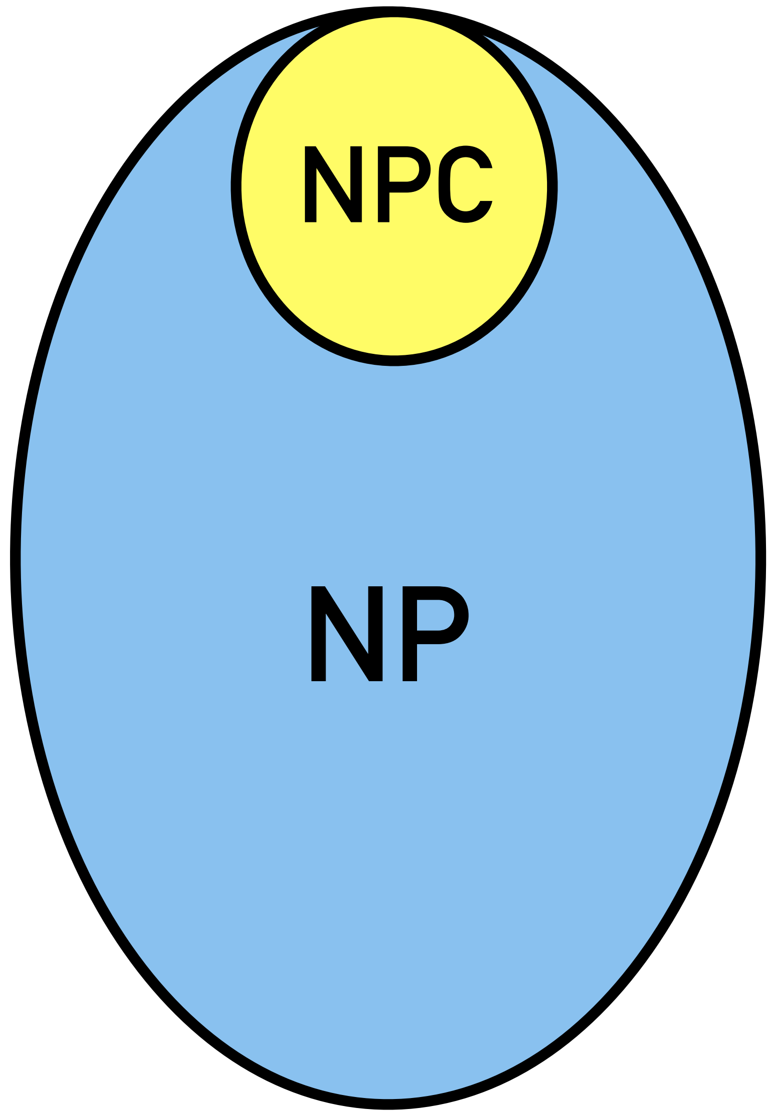
- *Primes* is solvable in polynomial-time.
- *GI* is solvable in quasi-polynomial time ($O(n^{\log^c n})$).
- *IndSet* is solvable in $O(1.1996^n)$.



NP-Completeness

Diversity in **NP**:

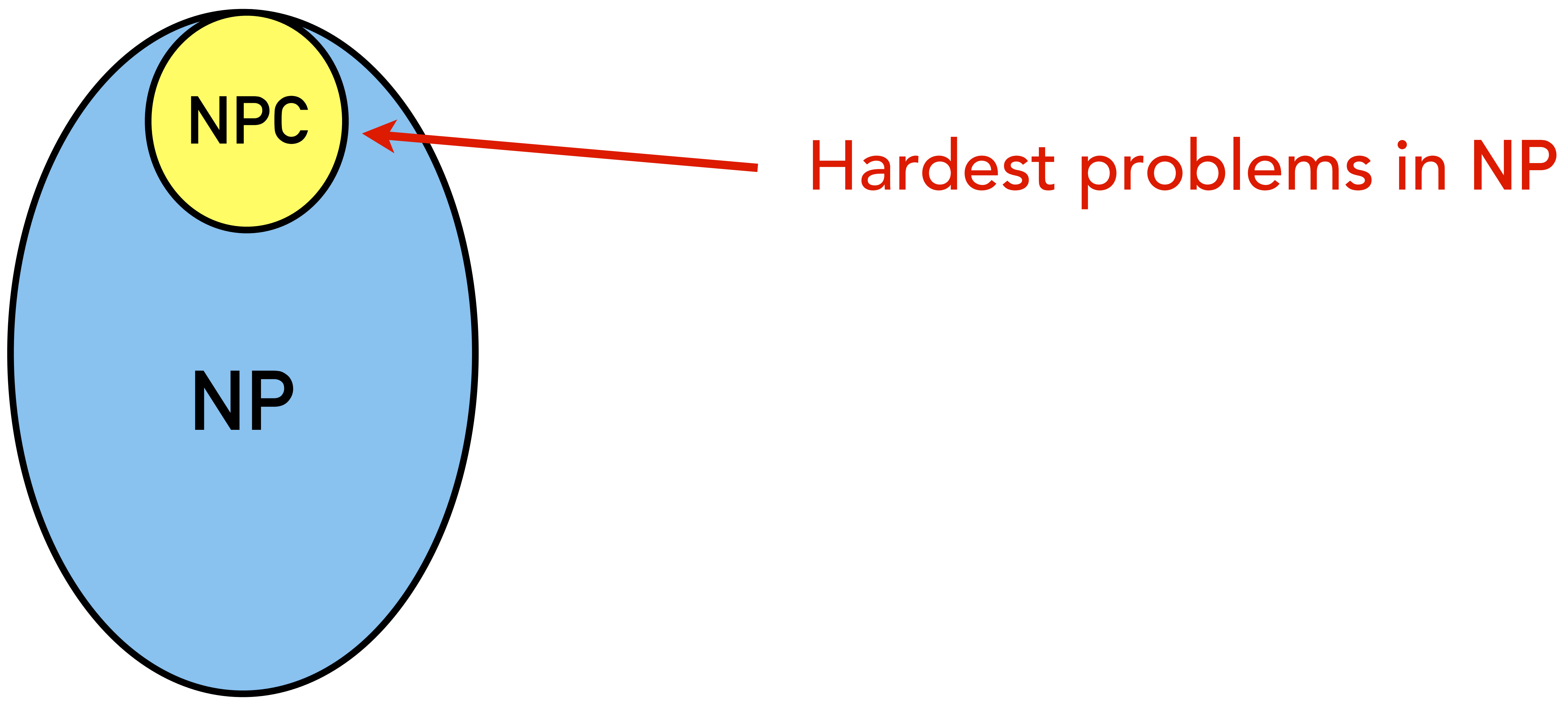
- *Primes* is solvable in polynomial-time.
- *GI* is solvable in quasi-polynomial time ($O(n^{\log^c n})$).
- *IndSet* is solvable in $O(1.1996^n)$.



NP-Completeness

Diversity in **NP**:

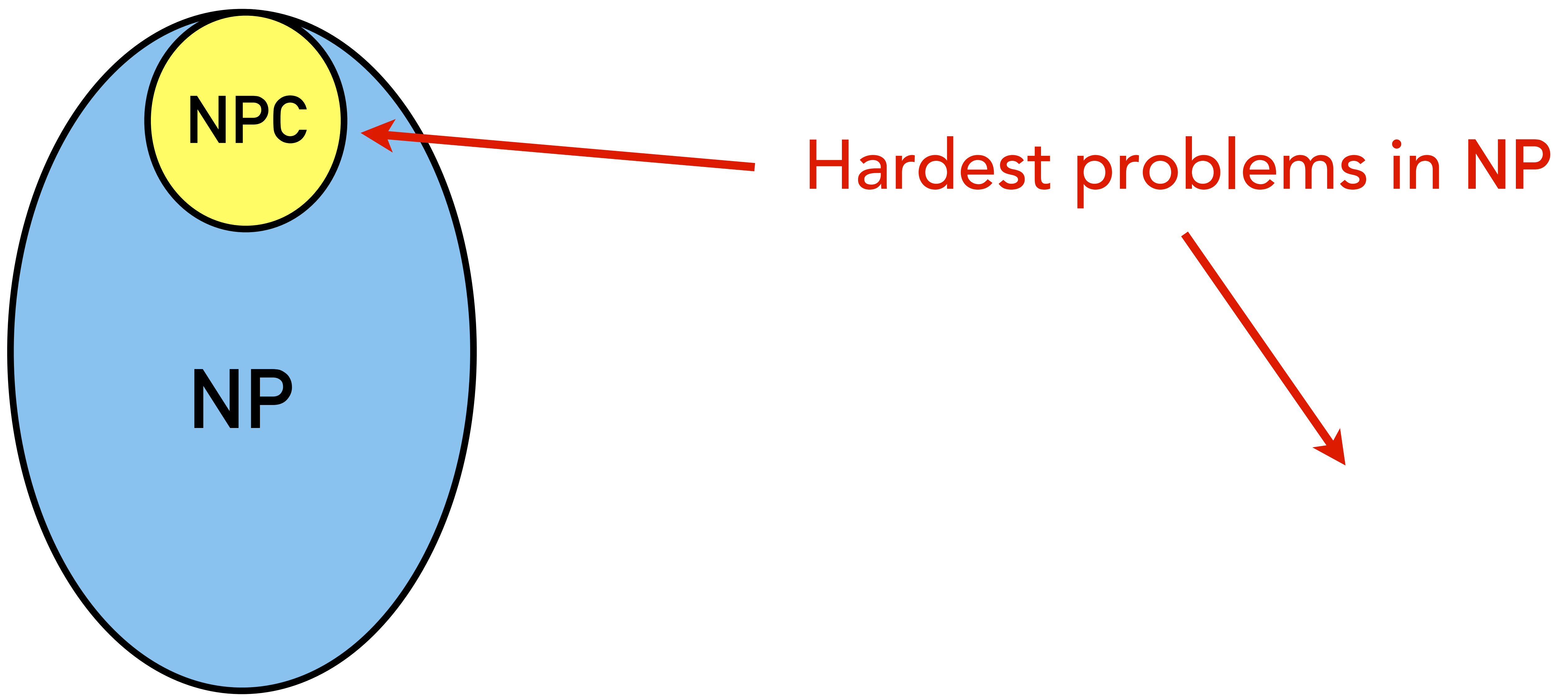
- *Primes* is solvable in polynomial-time.
- *GI* is solvable in quasi-polynomial time ($O(n^{\log^c n})$).
- *IndSet* is solvable in $O(1.1996^n)$.



NP-Completeness

Diversity in **NP**:

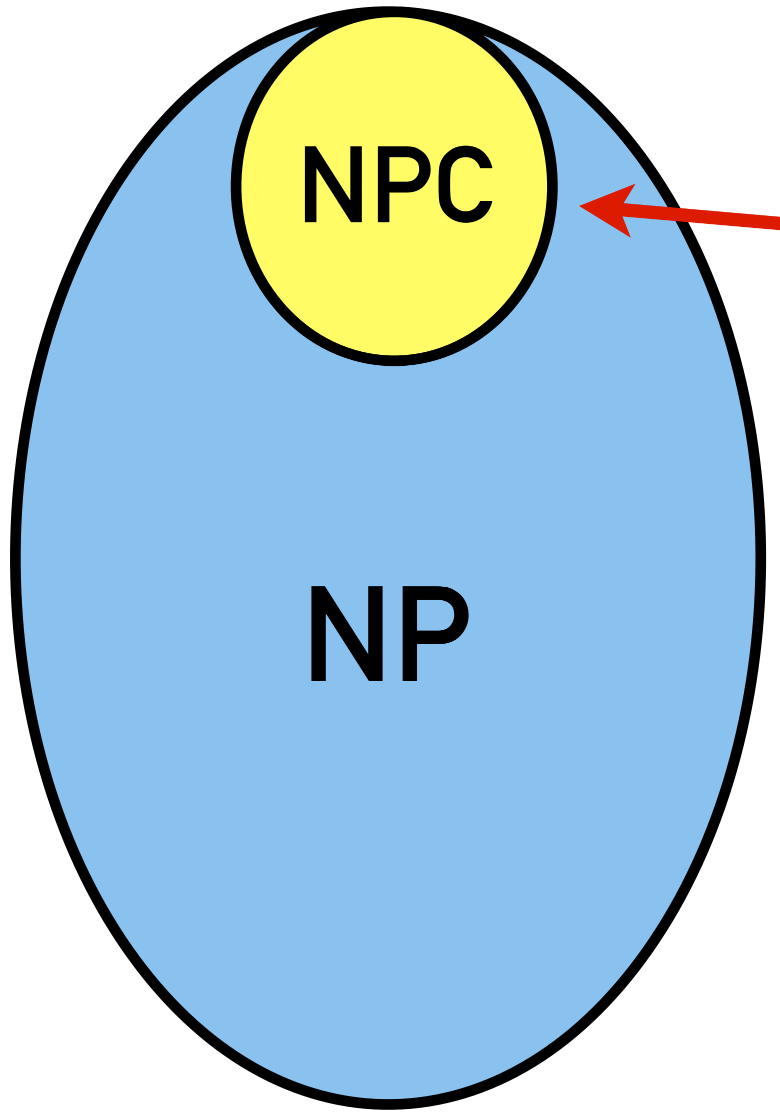
- *Primes* is solvable in polynomial-time.
- *GI* is solvable in quasi-polynomial time ($O(n^{\log^c n})$).
- *IndSet* is solvable in $O(1.1996^n)$.



NP-Completeness

Diversity in **NP**:

- *Primes* is solvable in polynomial-time.
- *GI* is solvable in quasi-polynomial time ($O(n^{\log^c n})$).
- *IndSet* is solvable in $O(1.1996^n)$.



Hardest problems in NP

NPC problem is in **P** $\Longrightarrow$ Every problem in **NP** is in **P**.

NP-Completeness

To understand NP-completeness we need to first learn about Reductions.

Reductions

Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2,

Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$,

Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if ∃ a polytime algorithm A,

Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if ∃ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if ∃ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

$$x \in L_1$$

Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

$$x \in L_1 \implies A(x) \in L_2$$

Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

$$x \in L_1 \implies A(x) \in L_2$$

$$x \notin L_1$$

Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if ∃ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

$$x \in L_1 \implies A(x) \in L_2$$

$$x \notin L_1 \implies A(x) \notin L_2$$

Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

$$x \in L_1 \implies A(x) \in L_2$$

$$x \notin L_1 \implies A(x) \notin L_2$$

Output of A on input x

Reductions

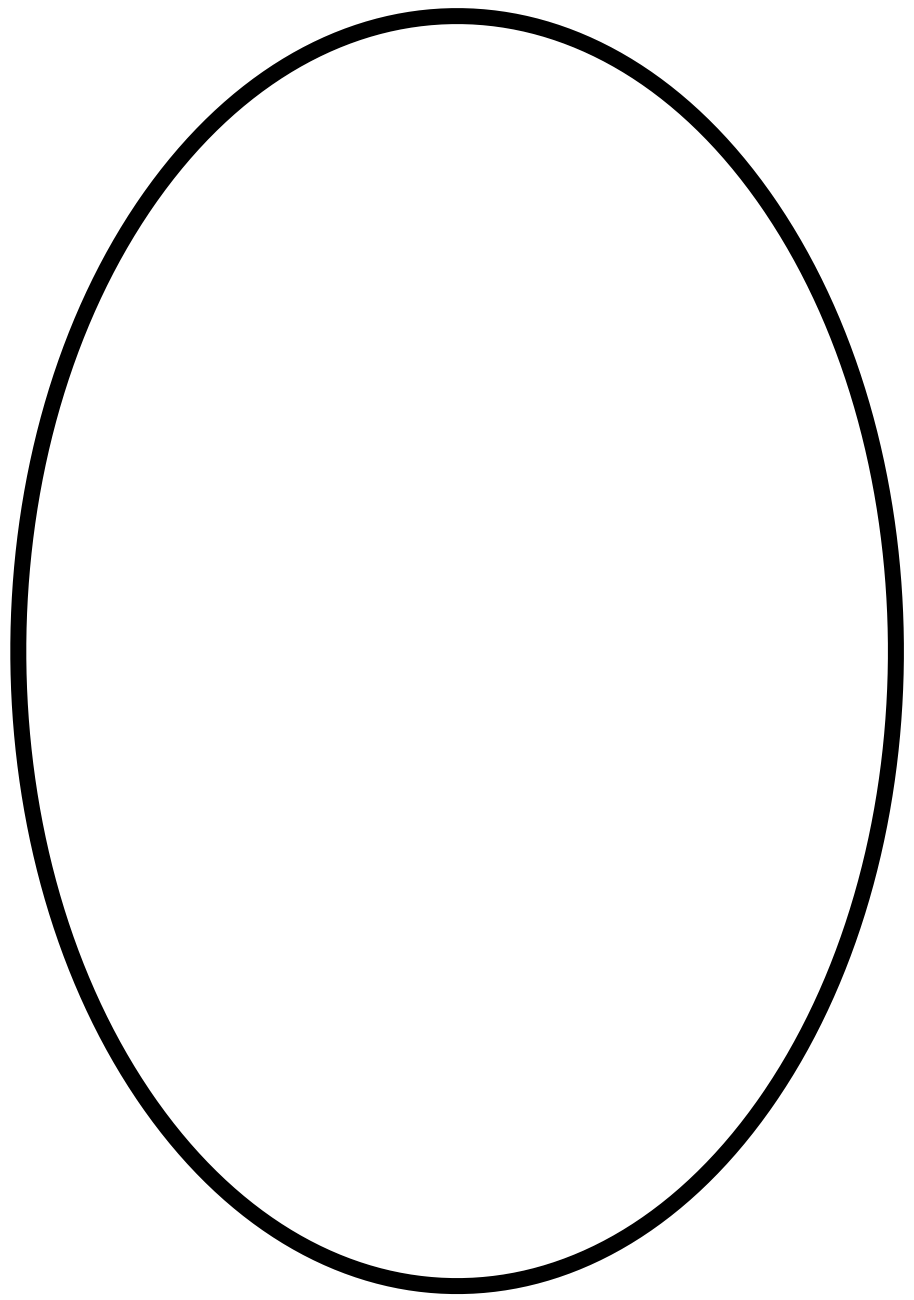
A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if ∃ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

$$x \in L_1 \iff A(x) \in L_2$$

Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

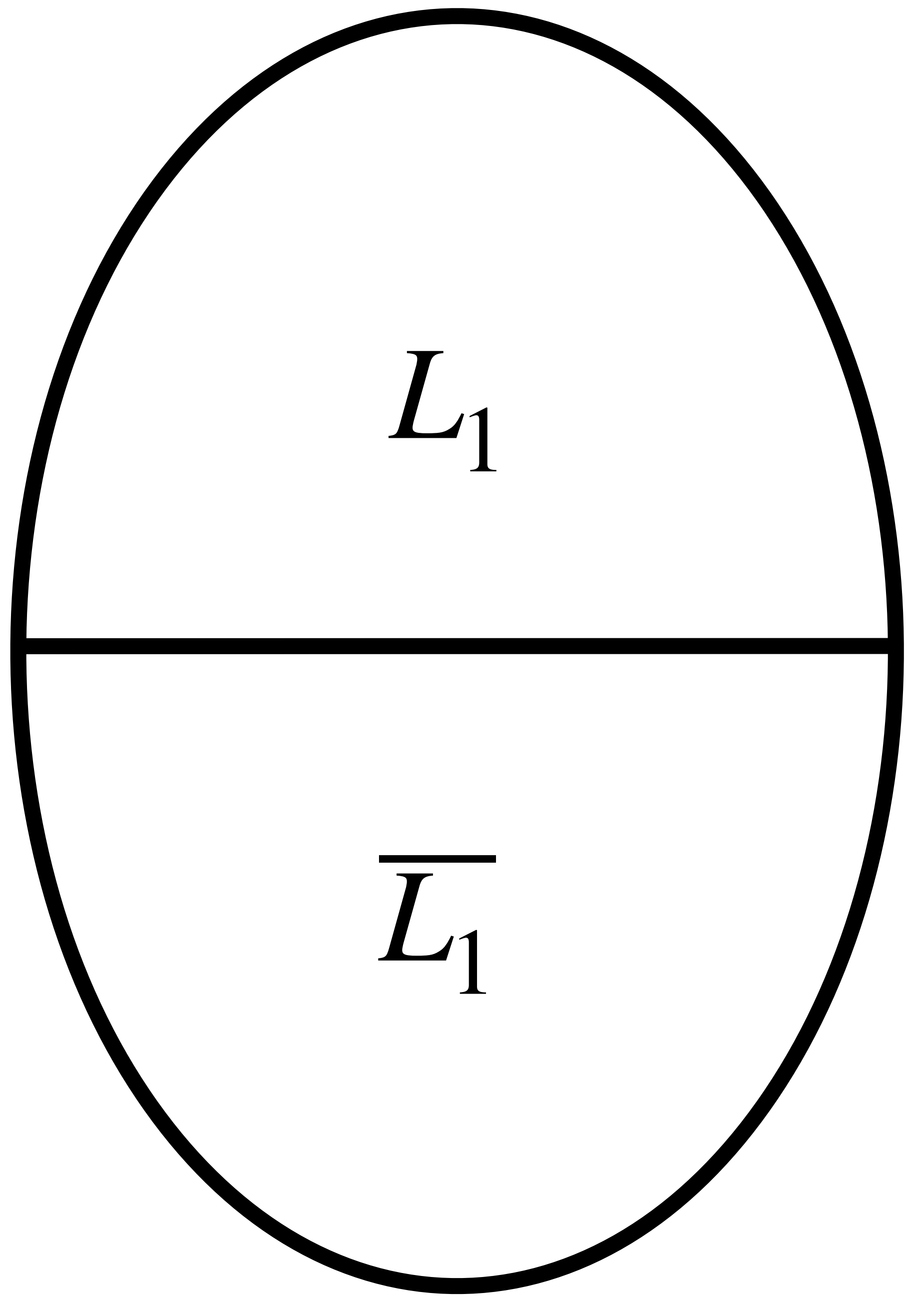
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

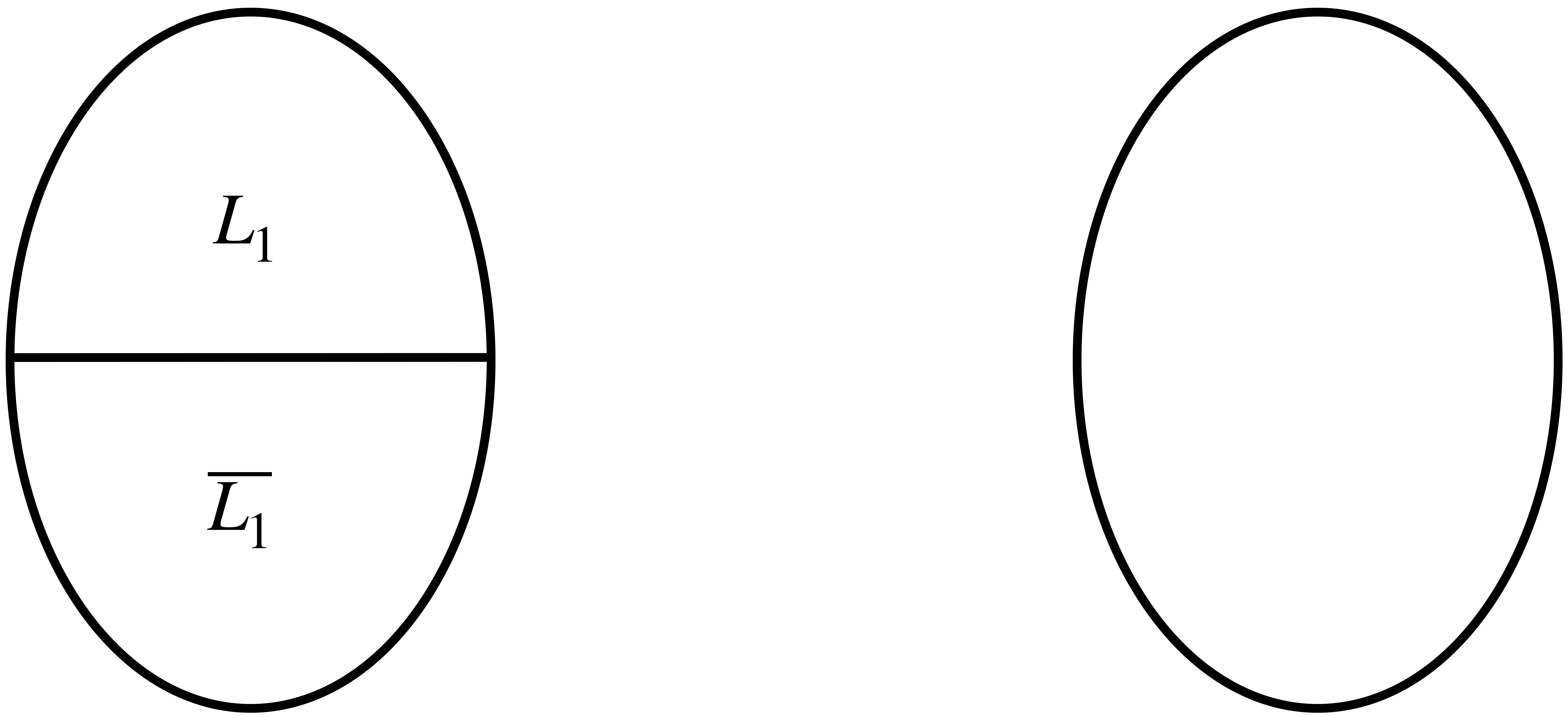
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if ∃ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

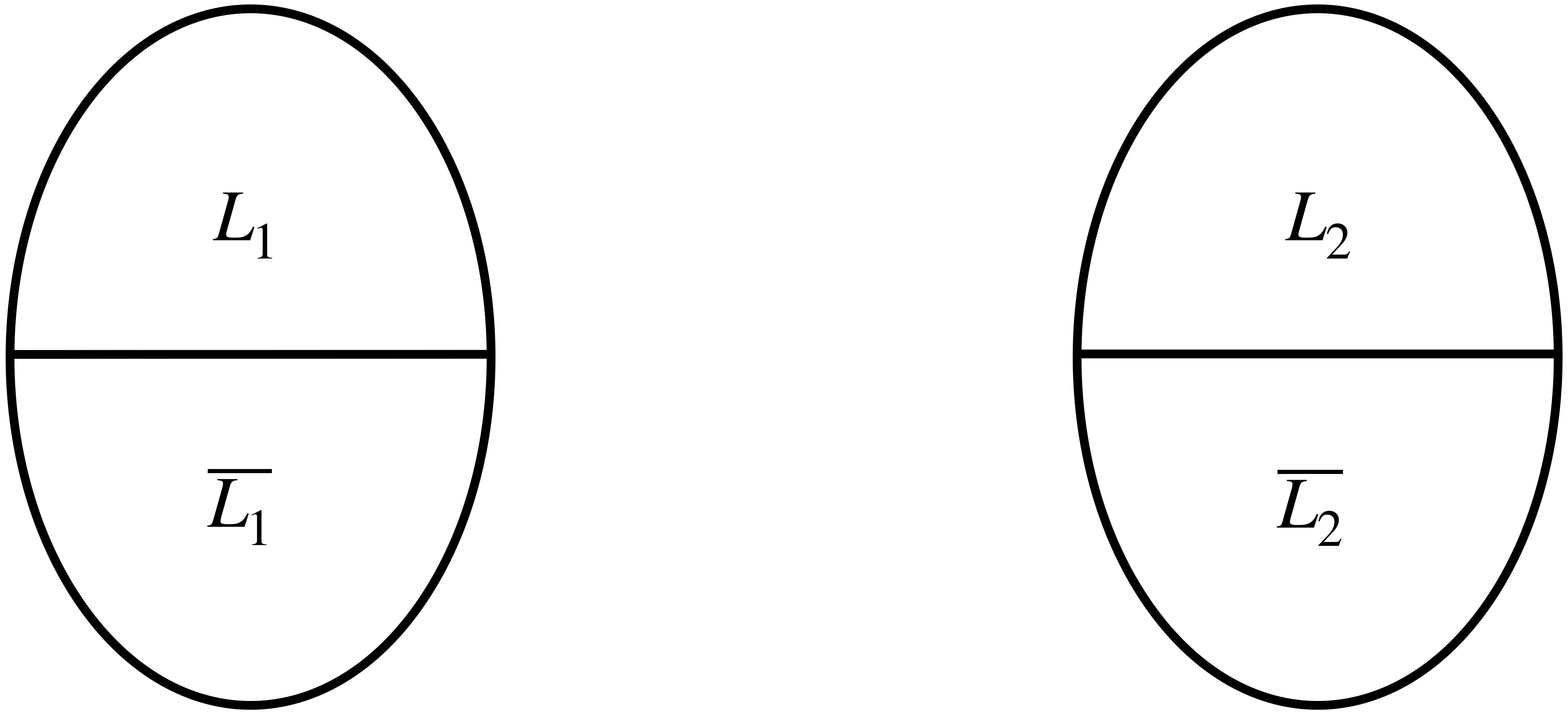
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

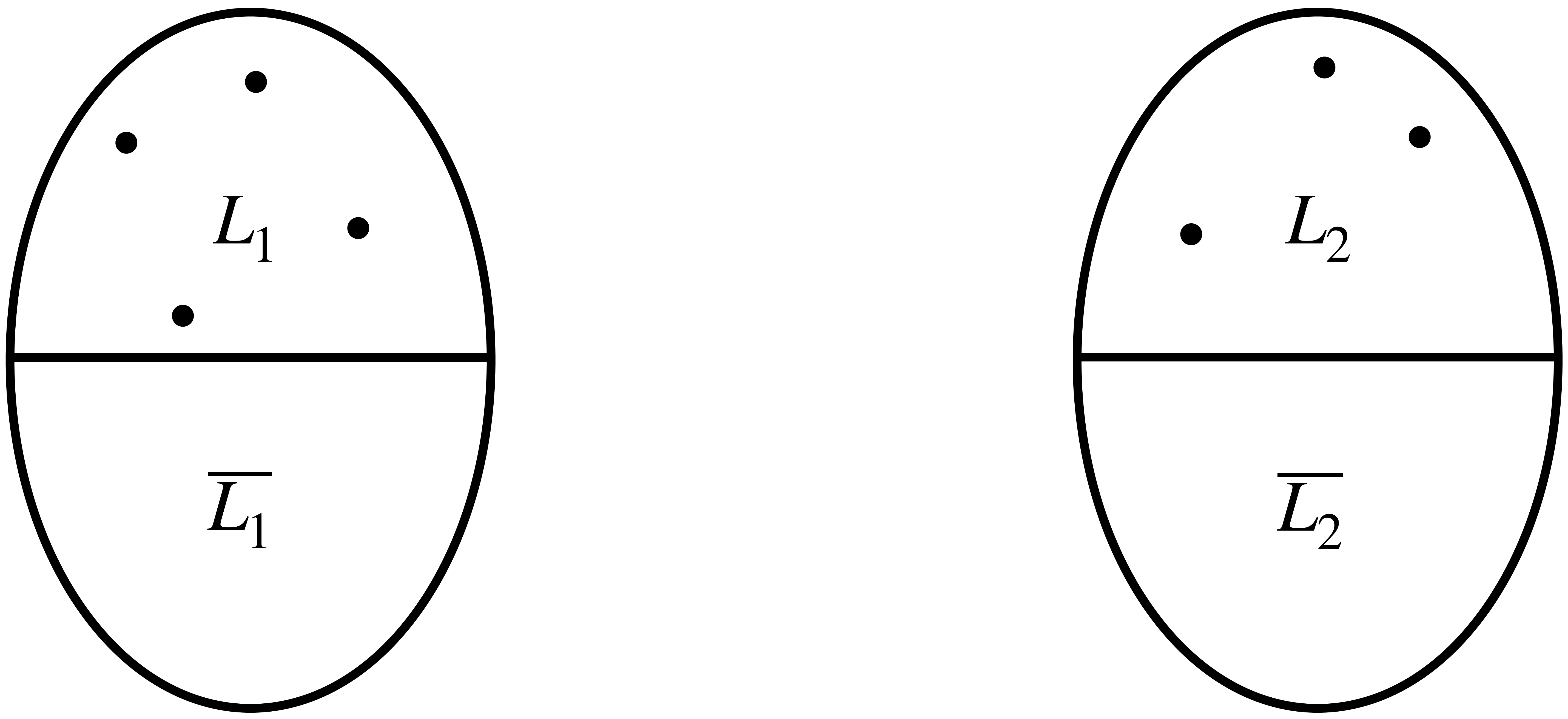
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

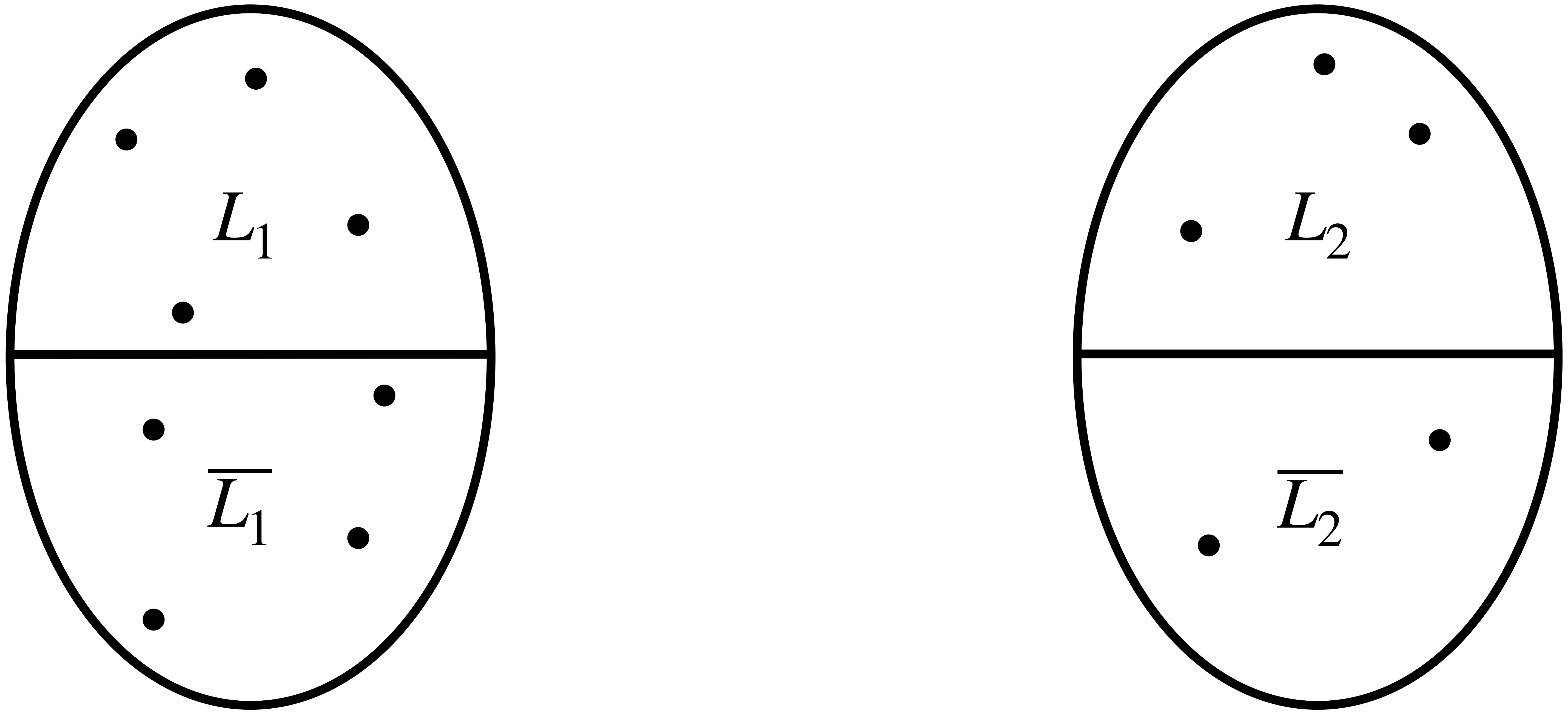
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

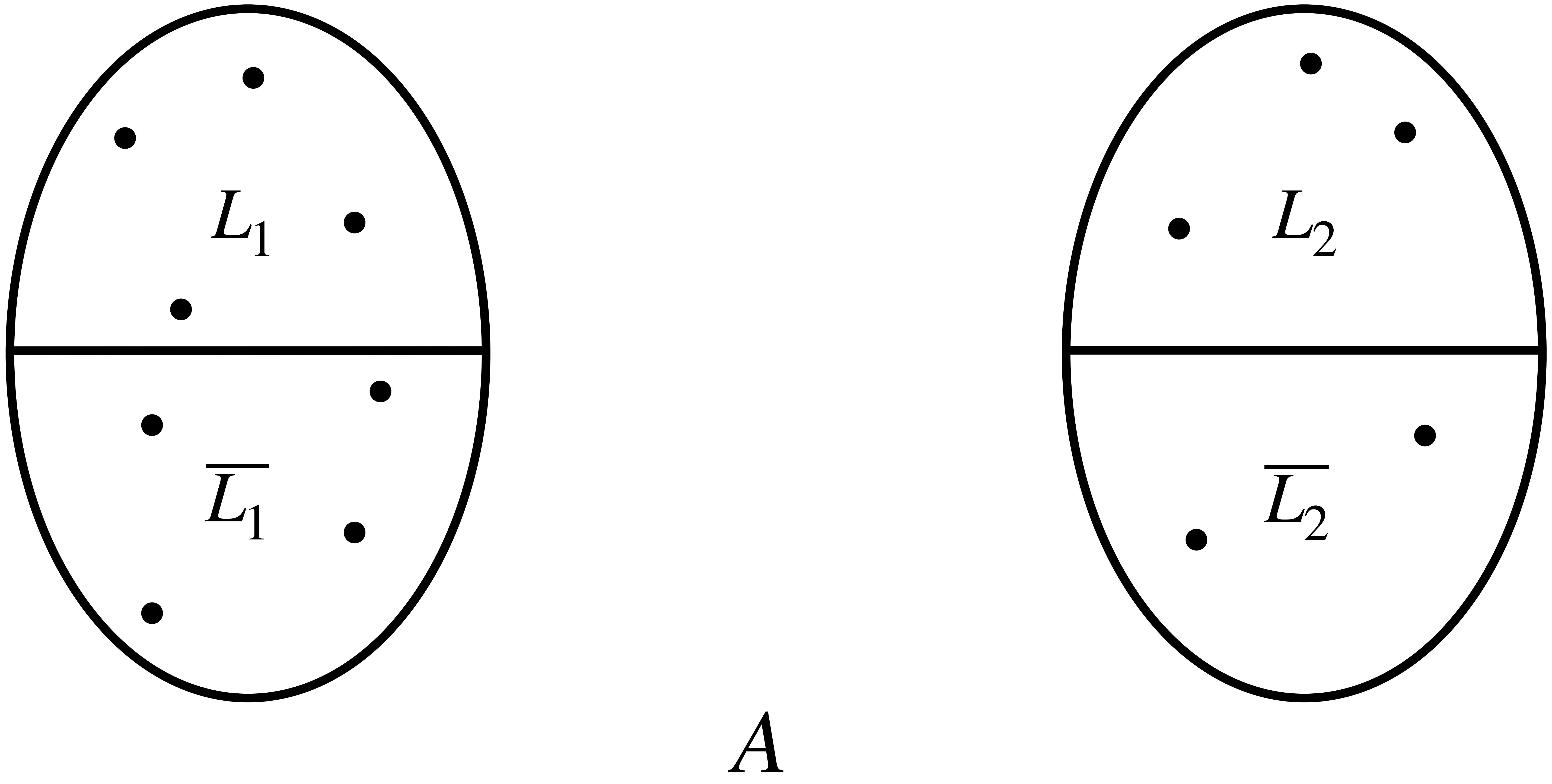
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

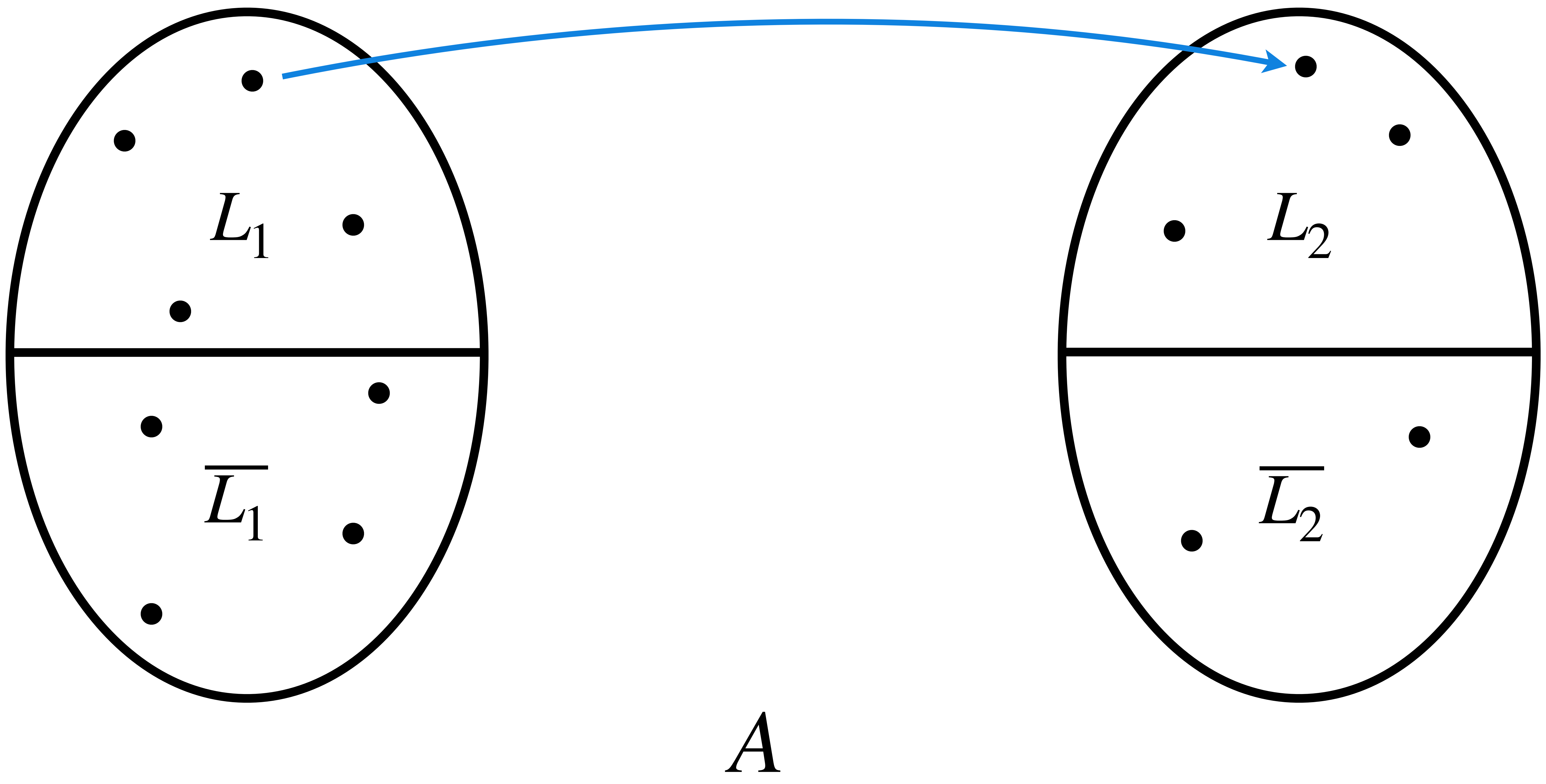
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

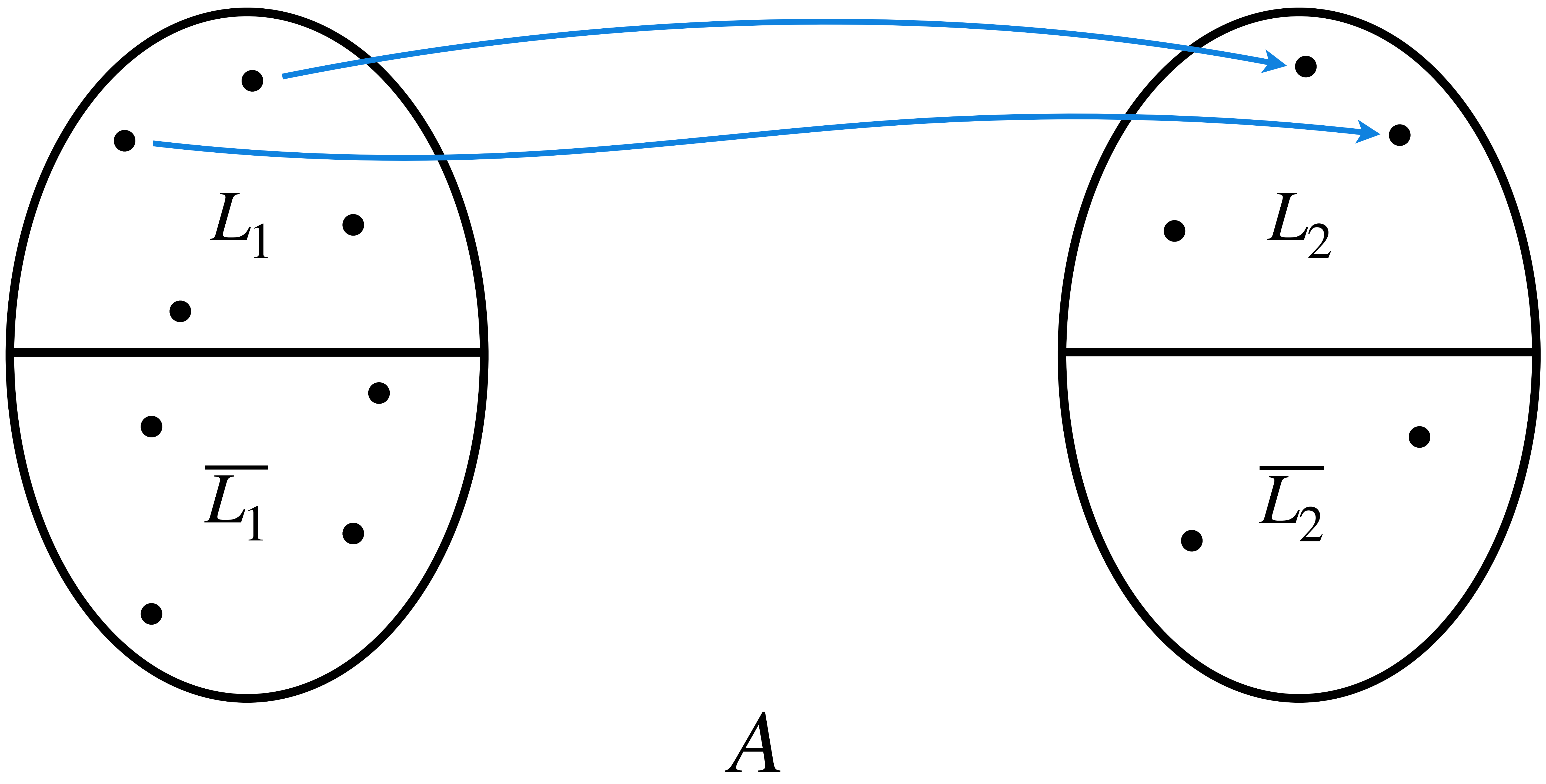
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

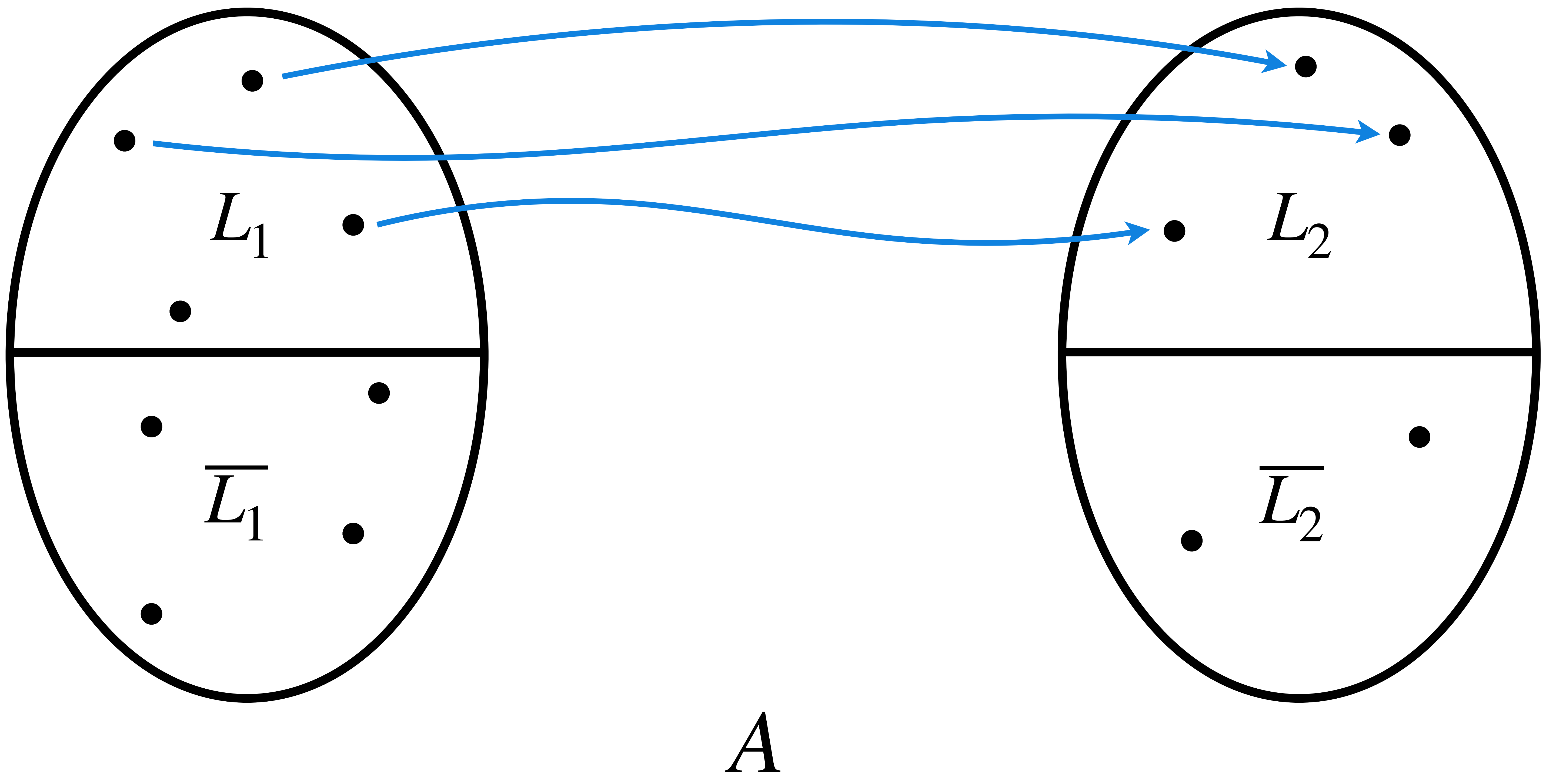
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

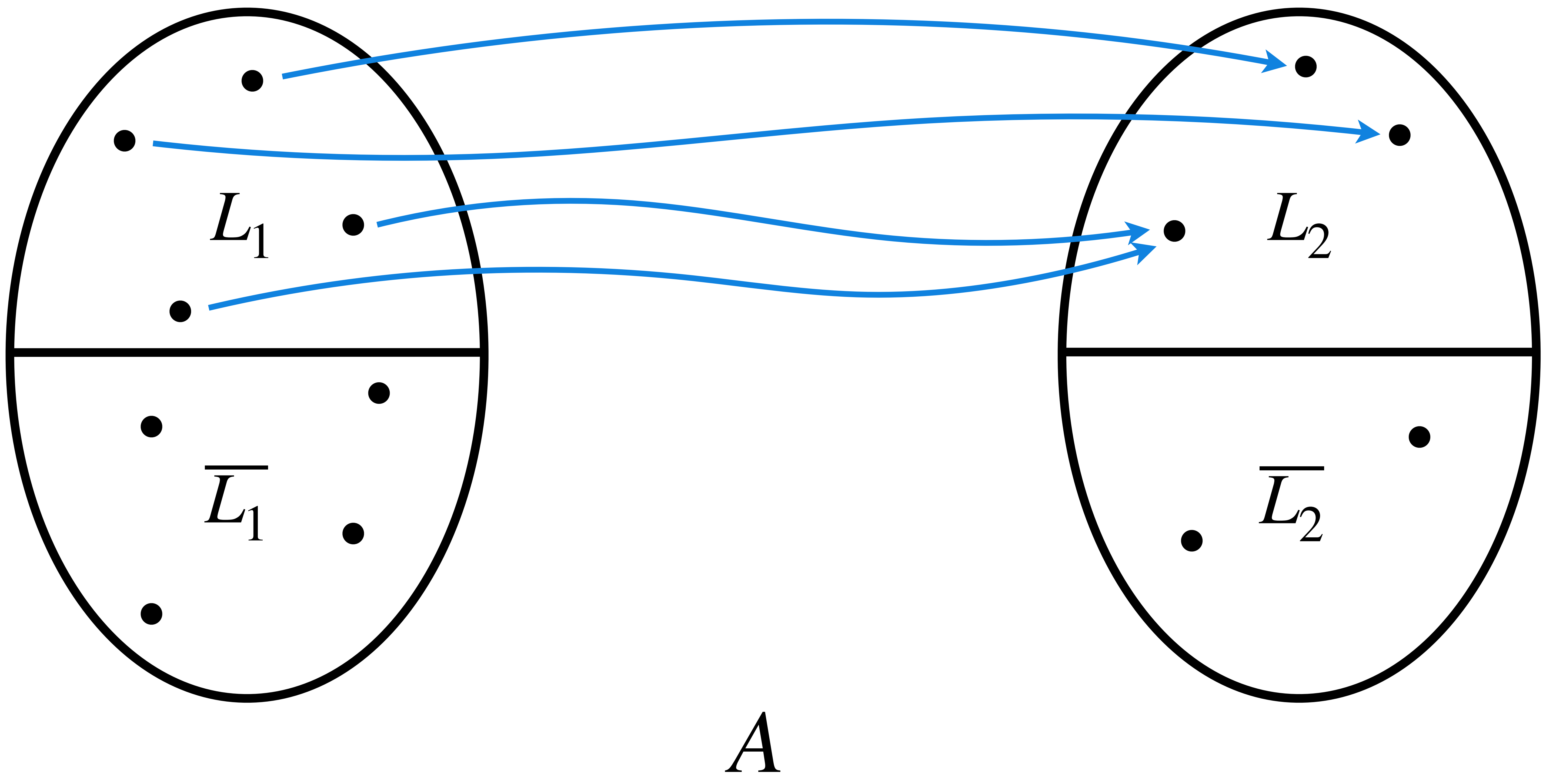
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

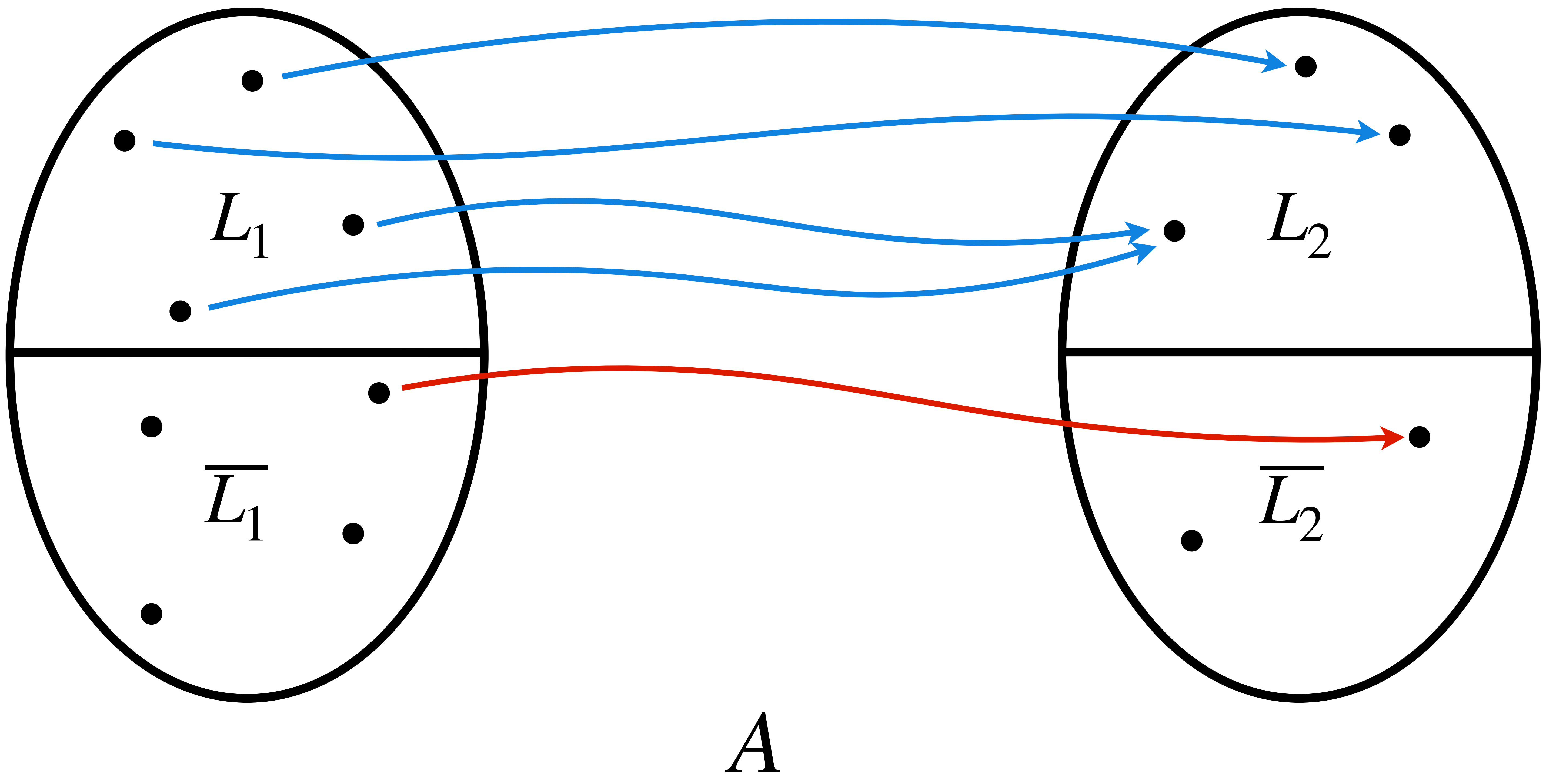
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if ∃ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

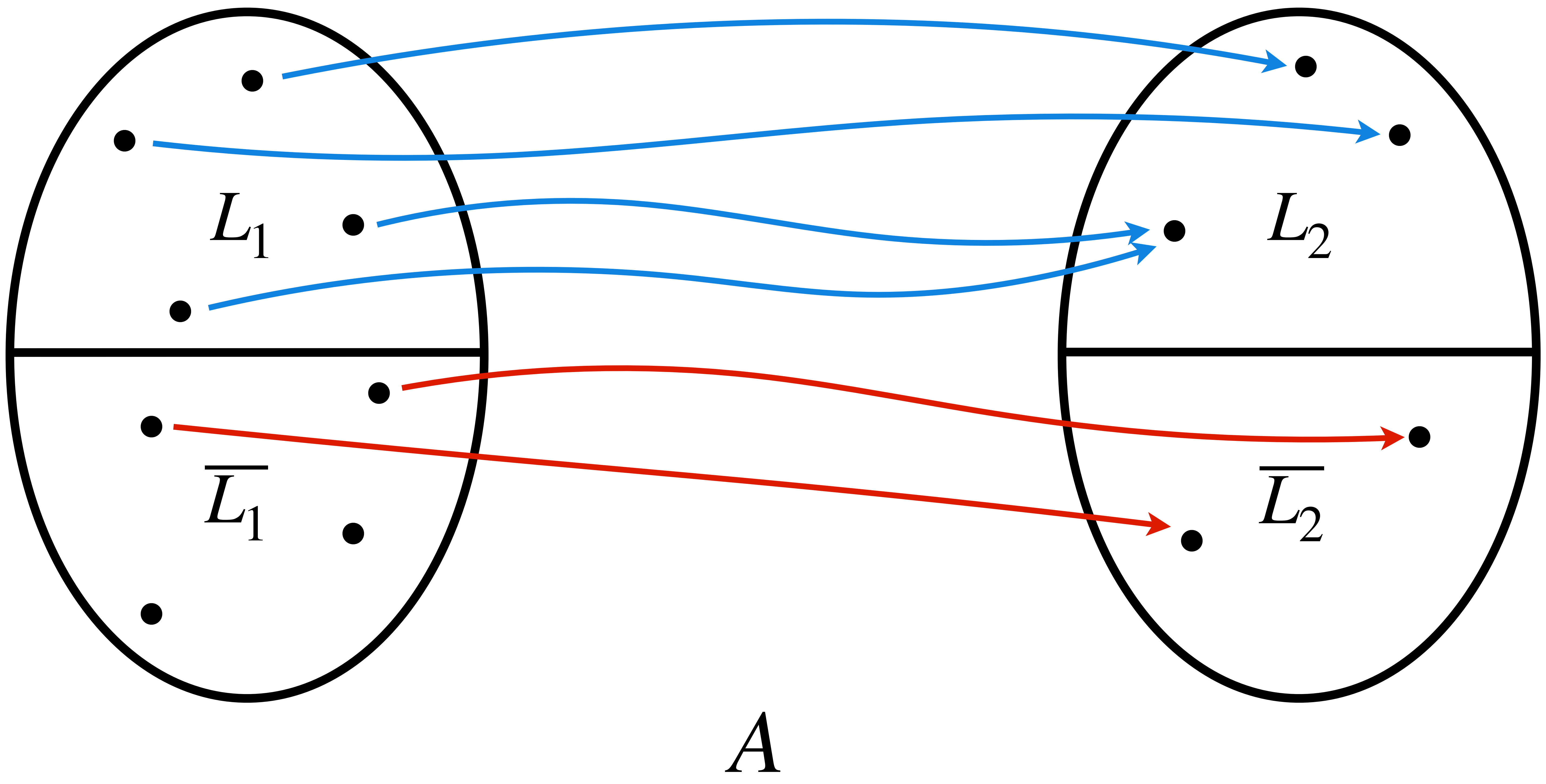
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

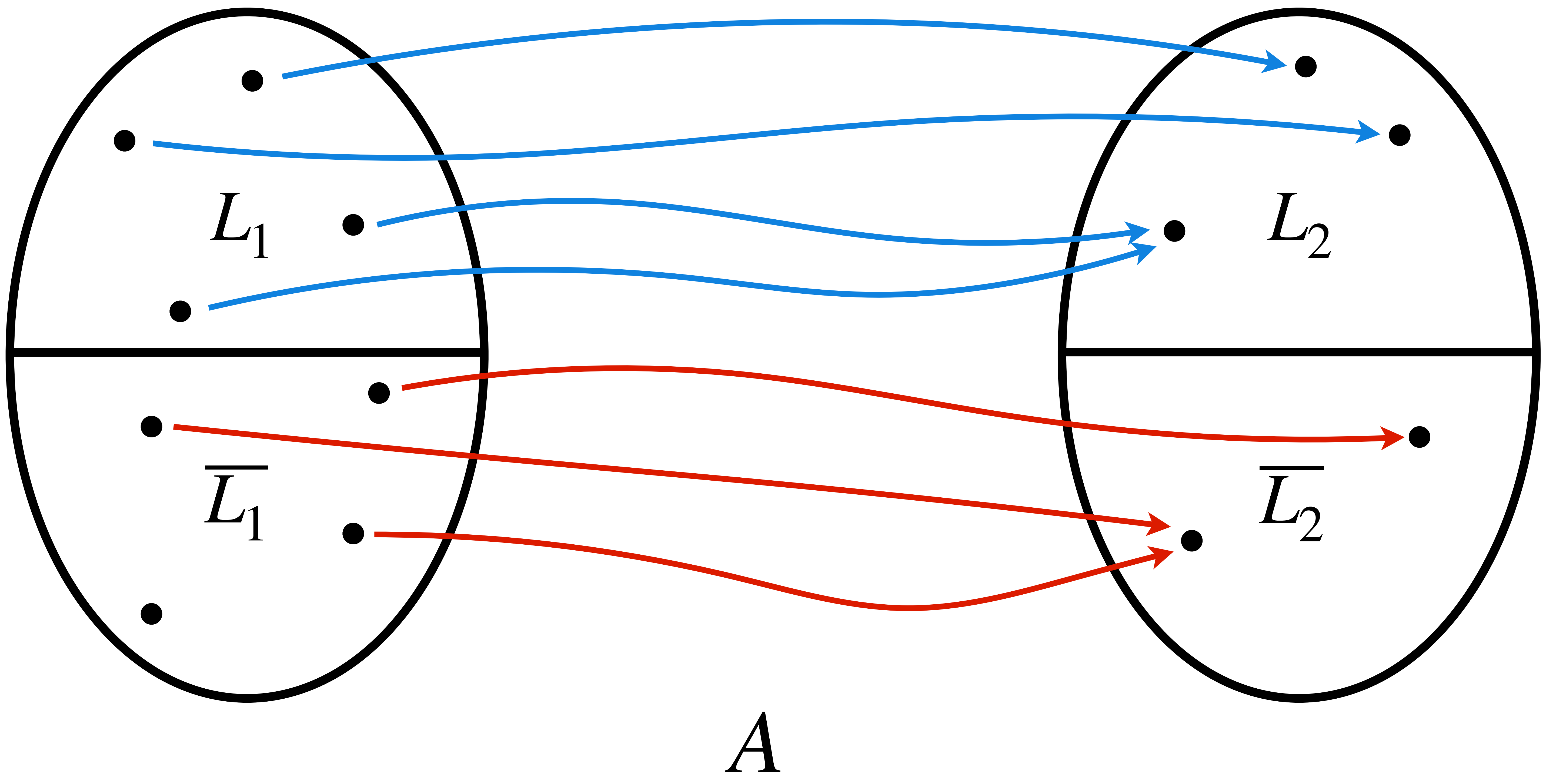
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

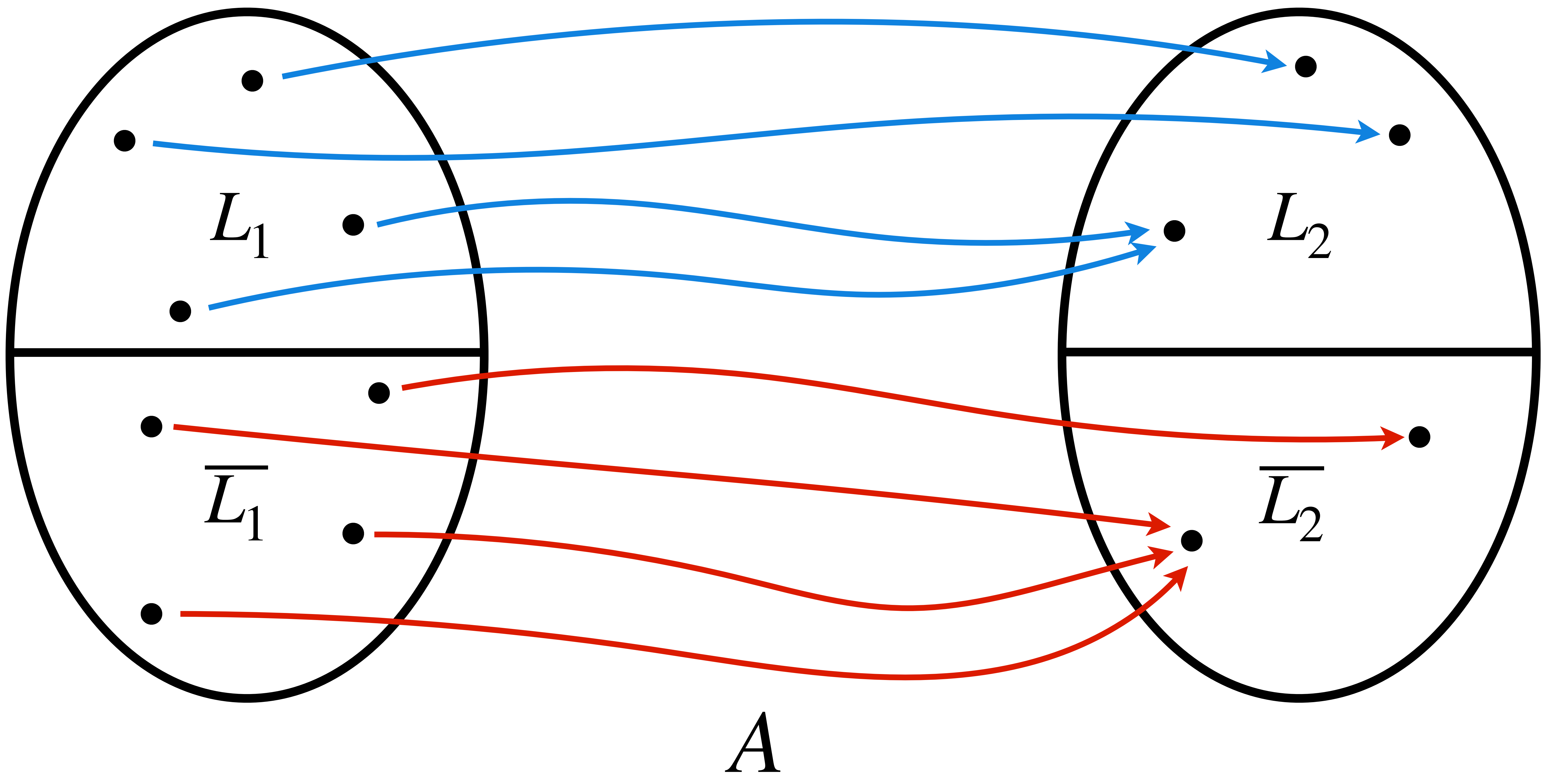
$$x \in L_1 \iff A(x) \in L_2$$



Reductions

A decision problem L_1 is polytime reducible to a decision problem L_2, denoted by $L_1 \leq_p L_2$, if $\exists$ a polytime algorithm A, such that $\forall x \in \{0,1\}^*$,

$$x \in L_1 \iff A(x) \in L_2$$



Reductions

Reductions

We can decide L_1 in polytime,

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.

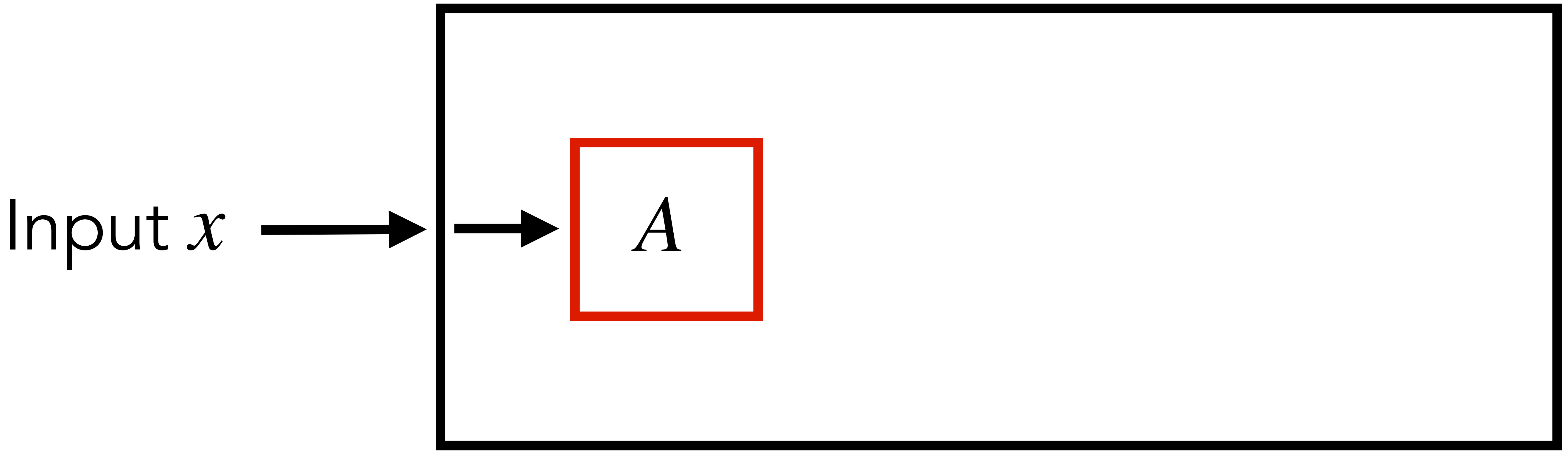
Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.

Polytime Algorithm A_1 for L_1

Reductions

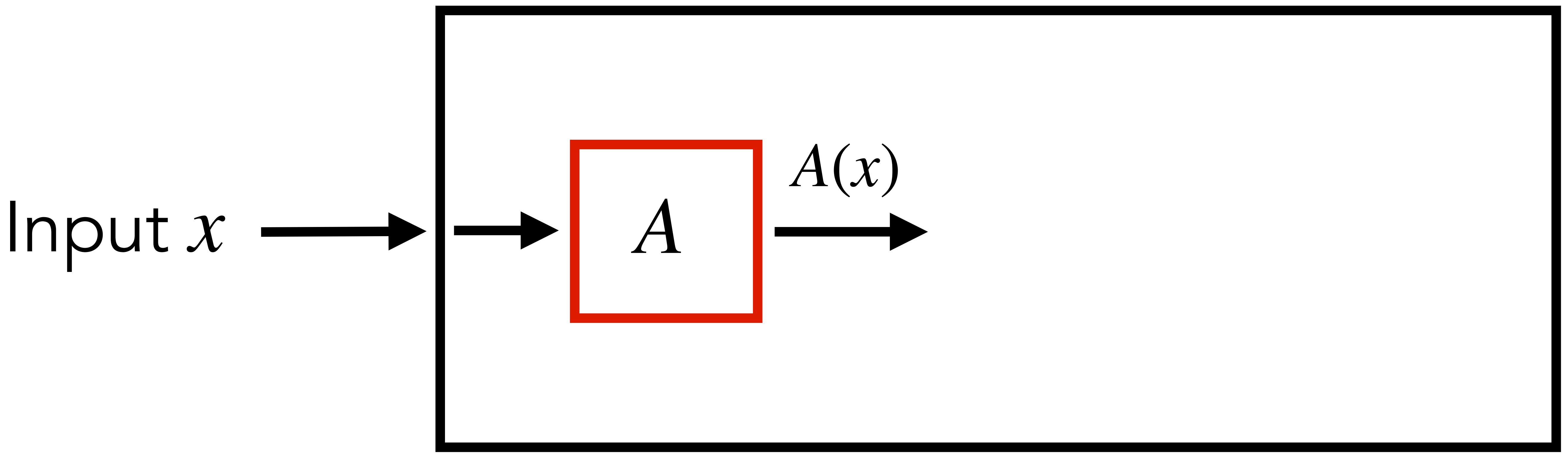
We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.



Polytime Algorithm A_1 for L_1

Reductions

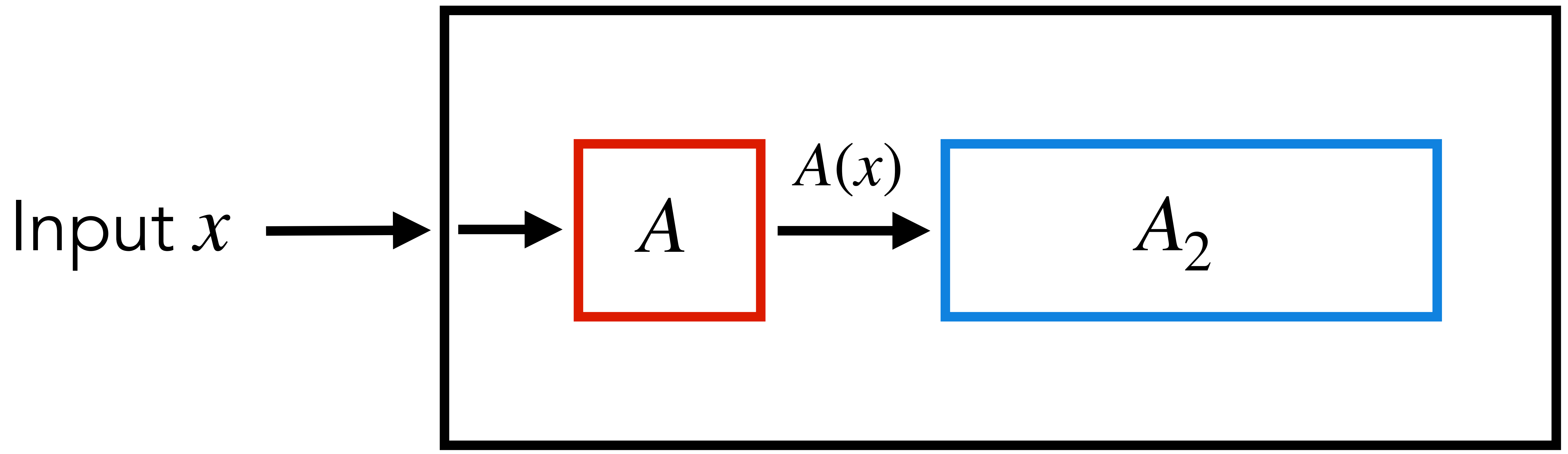
We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.



Polytime Algorithm A_1 for L_1

Reductions

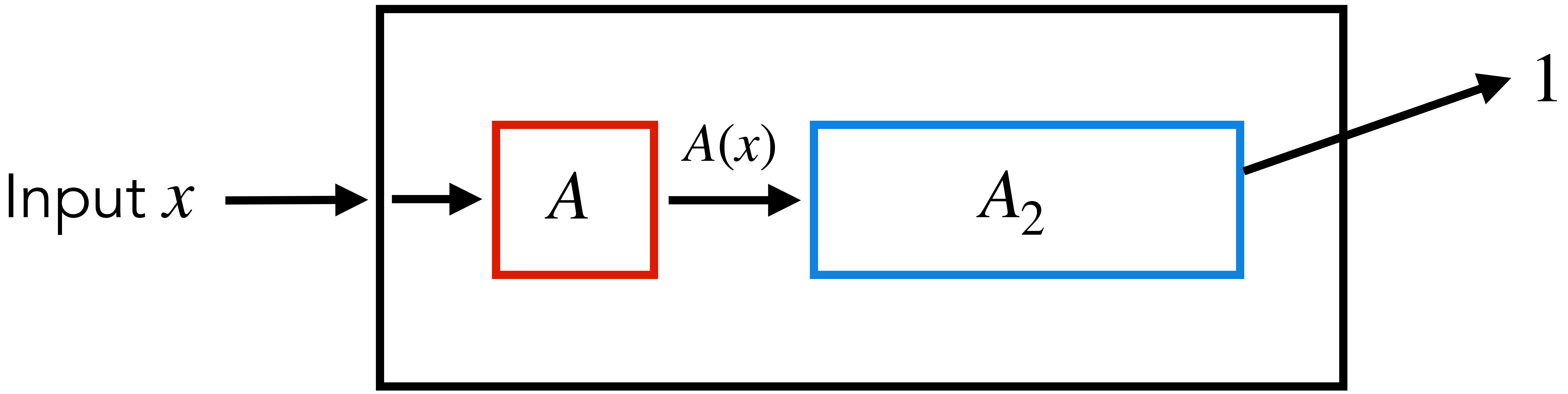
We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.



Polytime Algorithm A_1 for L_1

Reductions

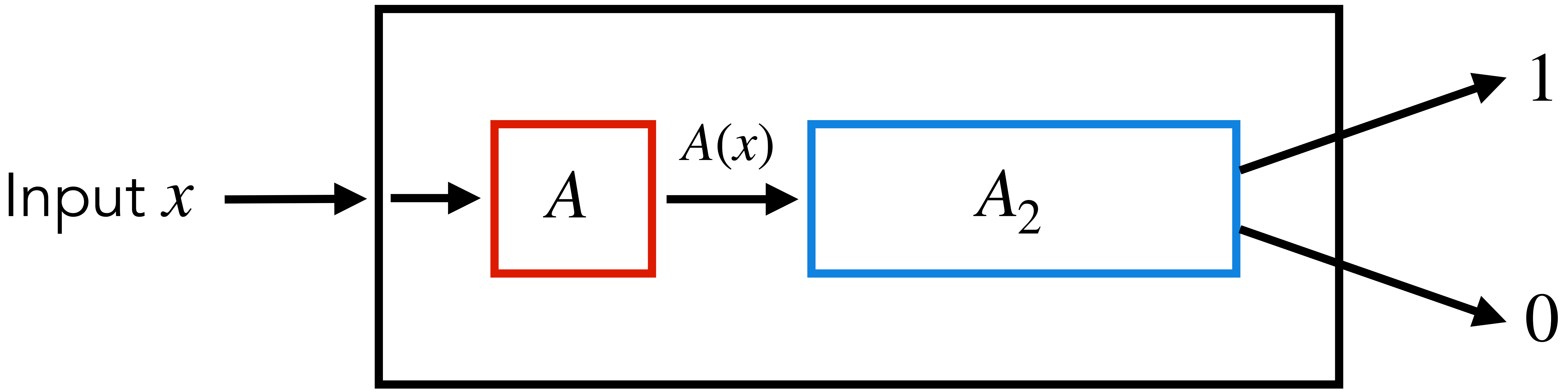
We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.



Polytime Algorithm A_1 for L_1

Reductions

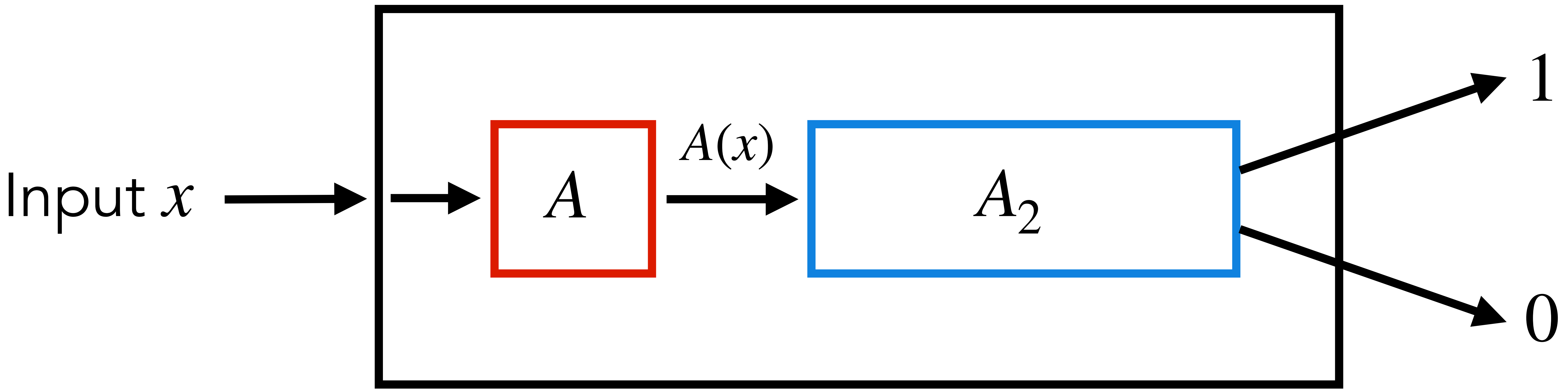
We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.



Polytime Algorithm A_1 for L_1

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.

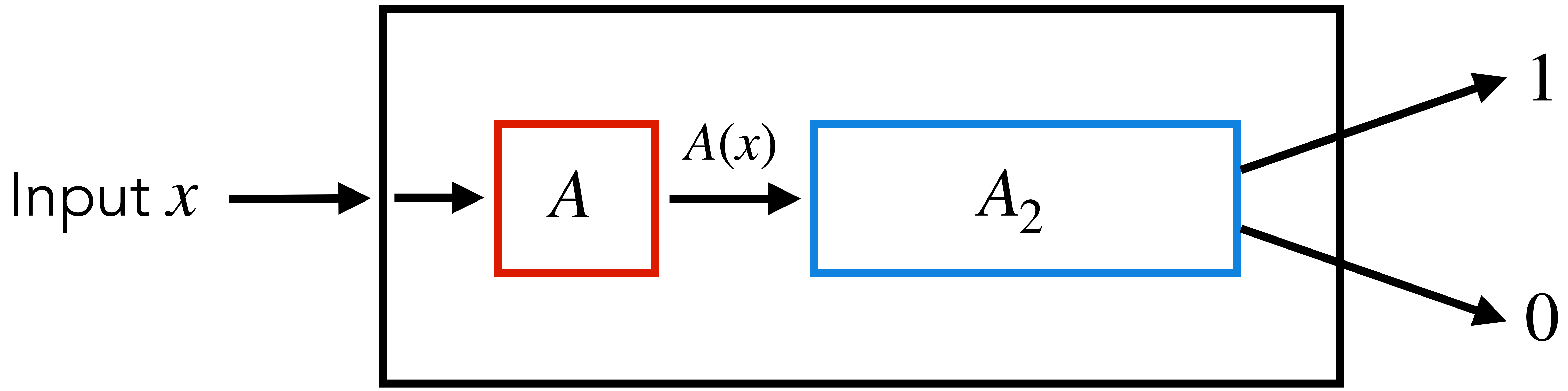


Polytime Algorithm A_1 for L_1

$x \in L_1$

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.

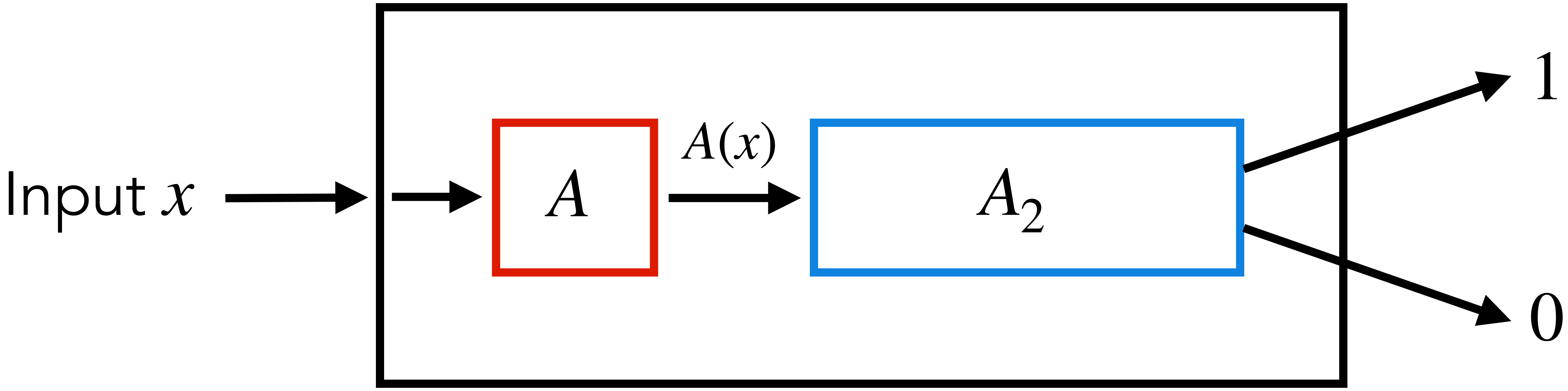


Polytime Algorithm A_1 for L_1

$$x \in L_1 \implies A(x) \in L_2$$

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.

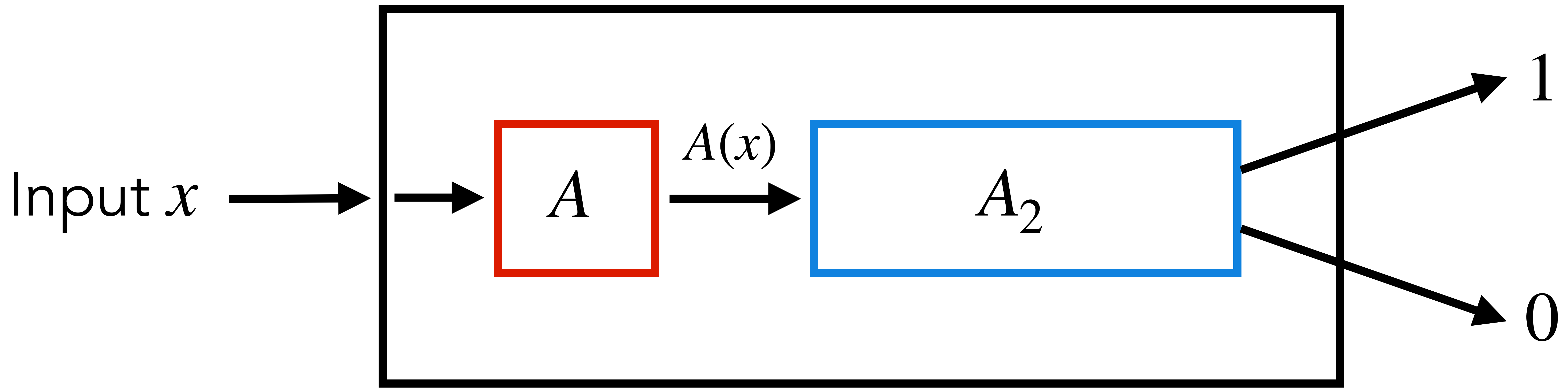


Polytime Algorithm A_1 for L_1

$$x \in L_1 \implies A(x) \in L_2 \implies A_2 \text{ outputs } 1 \text{ on } A(x)$$

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.

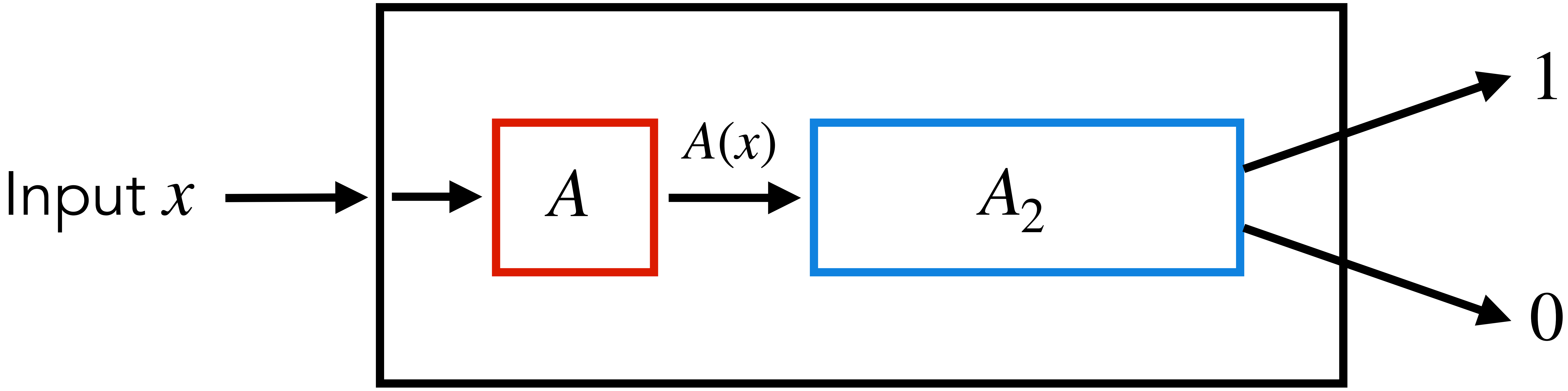


Polytime Algorithm A_1 for L_1

$$x \in L_1 \implies A(x) \in L_2 \implies A_2 \text{ outputs } 1 \text{ on } A(x) \implies A_1 \text{ outputs } 1$$

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.



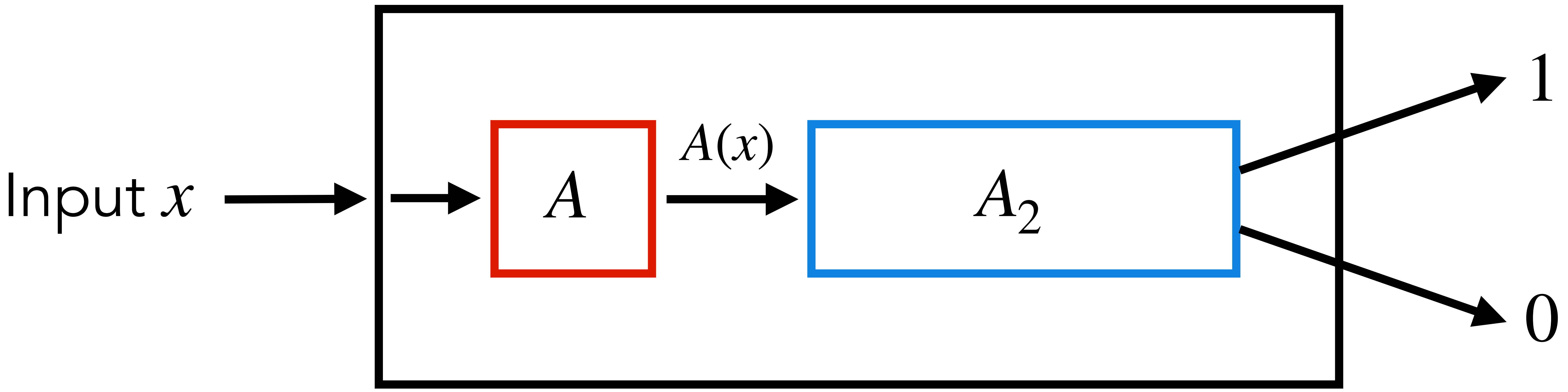
Polytime Algorithm A_1 for L_1

$x \in L_1 \implies A(x) \in L_2 \implies A_2$ outputs 1 on $A(x) \implies A_1$ outputs 1

$x \notin L_1$

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.



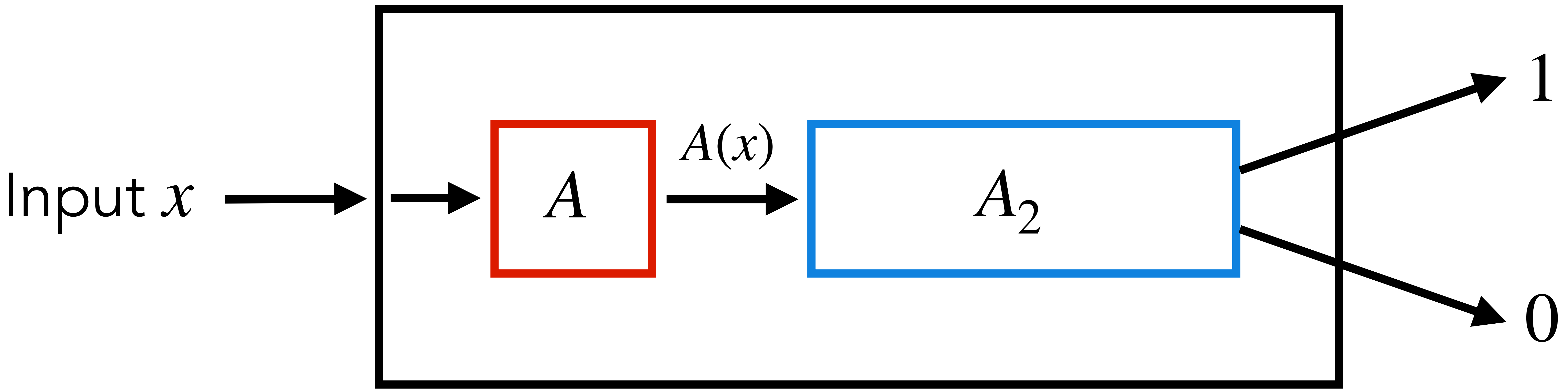
Polytime Algorithm A_1 for L_1

$$x \in L_1 \implies A(x) \in L_2 \implies A_2 \text{ outputs } 1 \text{ on } A(x) \implies A_1 \text{ outputs } 1$$

$$x \notin L_1 \implies A(x) \notin L_2$$

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.



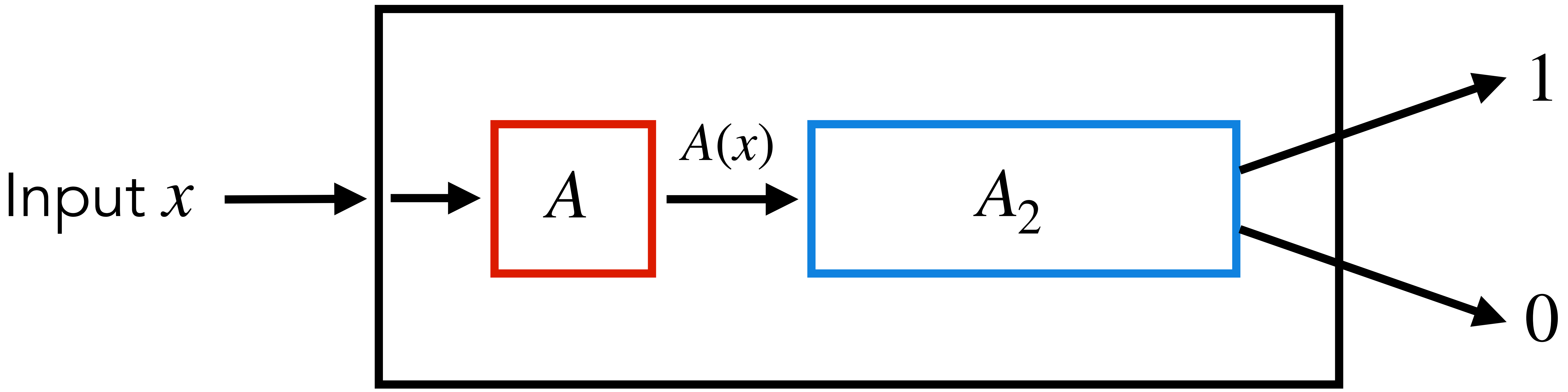
Polytime Algorithm A_1 for L_1

$$x \in L_1 \implies A(x) \in L_2 \implies A_2 \text{ outputs } 1 \text{ on } A(x) \implies A_1 \text{ outputs } 1$$

$$x \notin L_1 \implies A(x) \notin L_2 \implies A_2 \text{ outputs } 0 \text{ on } A(x)$$

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.



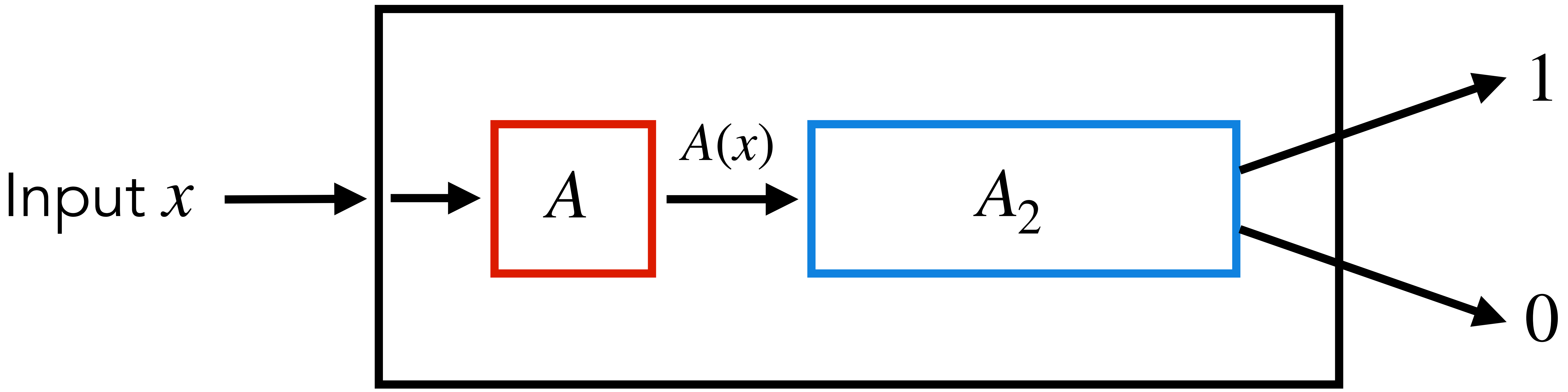
Polytime Algorithm A_1 for L_1

$$x \in L_1 \implies A(x) \in L_2 \implies A_2 \text{ outputs } 1 \text{ on } A(x) \implies A_1 \text{ outputs } 1$$

$$x \notin L_1 \implies A(x) \notin L_2 \implies A_2 \text{ outputs } 0 \text{ on } A(x) \implies A_1 \text{ outputs } 0$$

Reductions

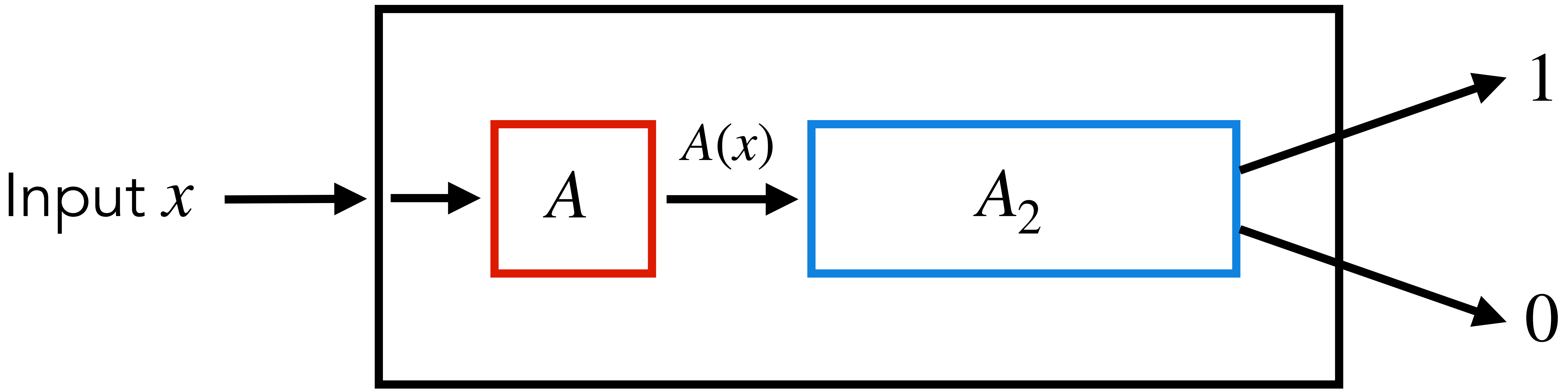
We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.



Polytime Algorithm A_1 for L_1

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.

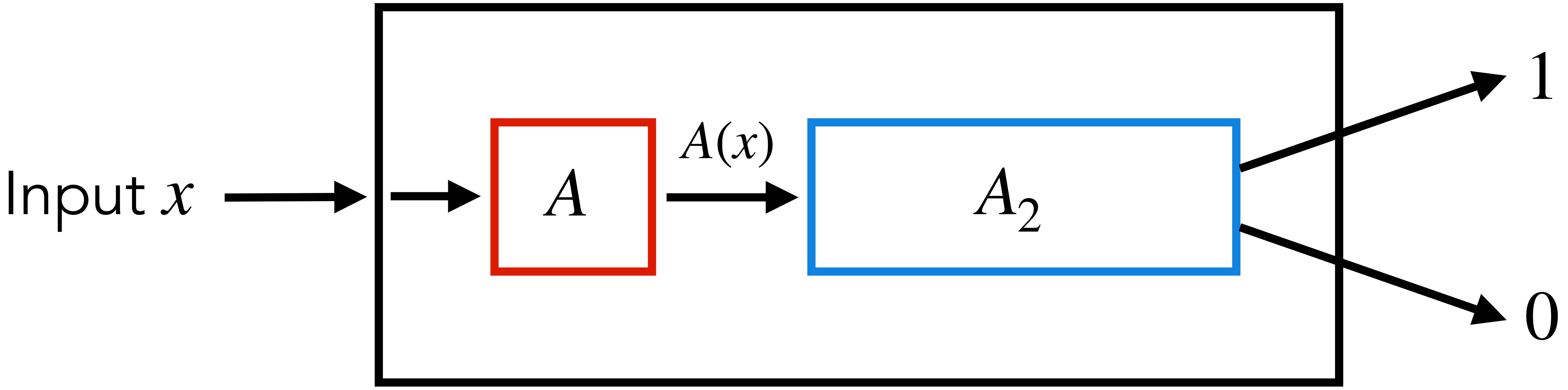


Polytime Algorithm A_1 for L_1

Fact: Suppose $L_1 \leq_p L_2$.

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.

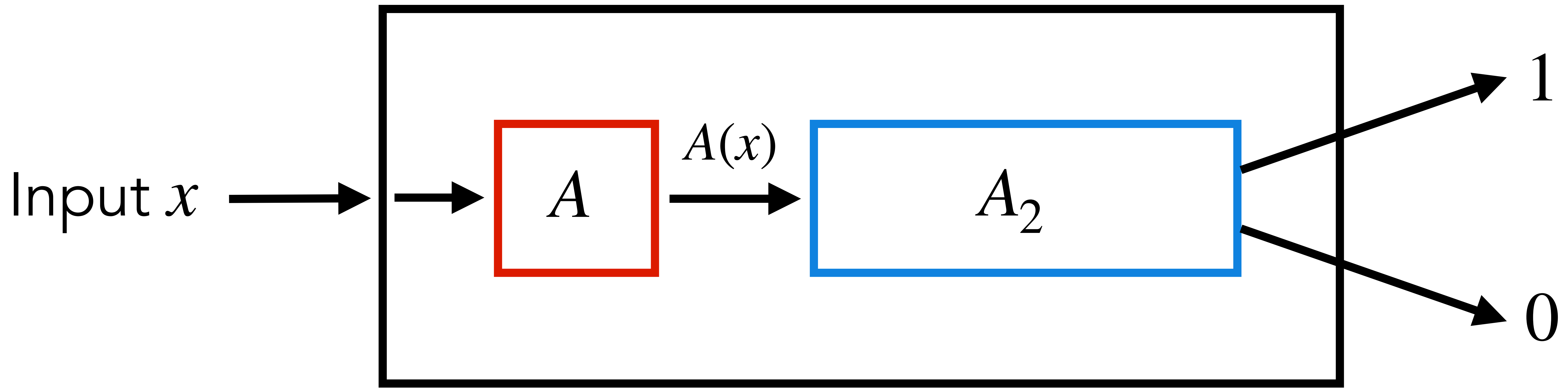


Polytime Algorithm A_1 for L_1

Fact: Suppose $L_1 \leq_p L_2$. If L_2 is decidable in polytime, then so is L_1.

Reductions

We can decide L_1 in polytime, if we know that $L_1 \leq_p L_2$ via A and L_2 is decidable by a polytime algorithm A_2.



Polytime Algorithm A_1 for L_1

Fact: Suppose $L_1 \leq_p L_2$. If L_1 is not polytime decidable, then so is L_2.

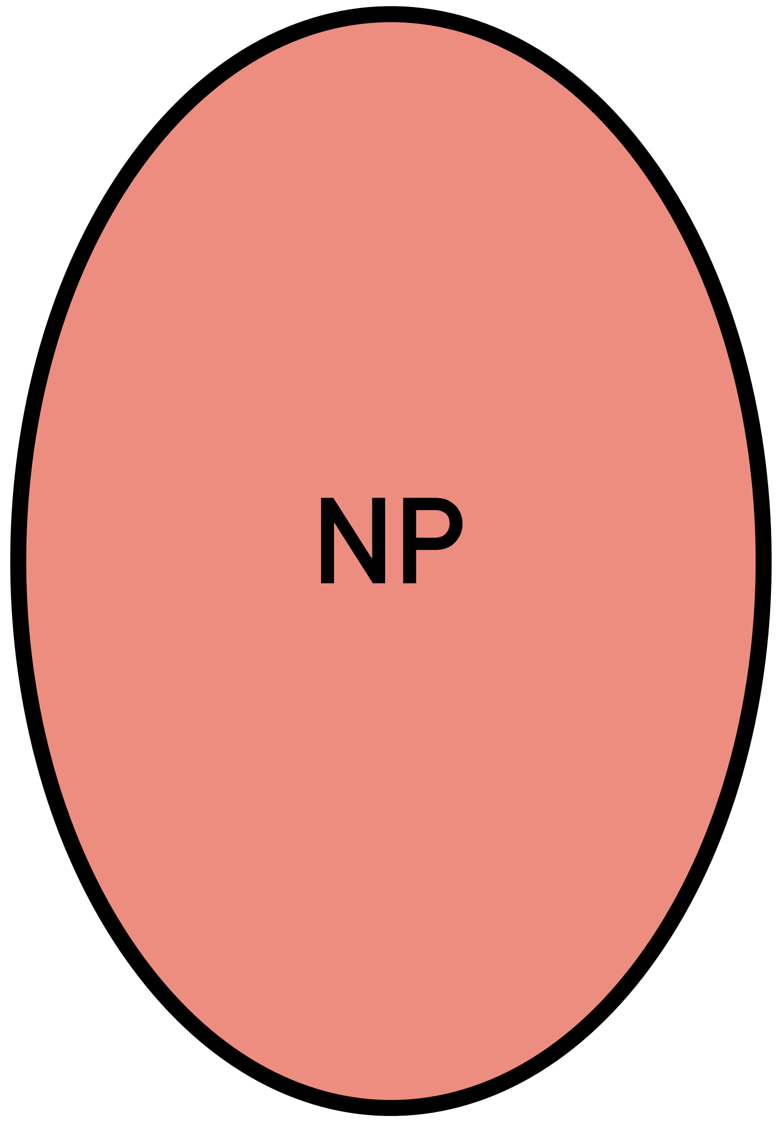
NP-Completeness and NP-Hardness

NP-Completeness and NP-Hardness

Defn: (1) A decision problem L is called NP-hard if $L' \leq_p L$, for every $L' \in$ NP.

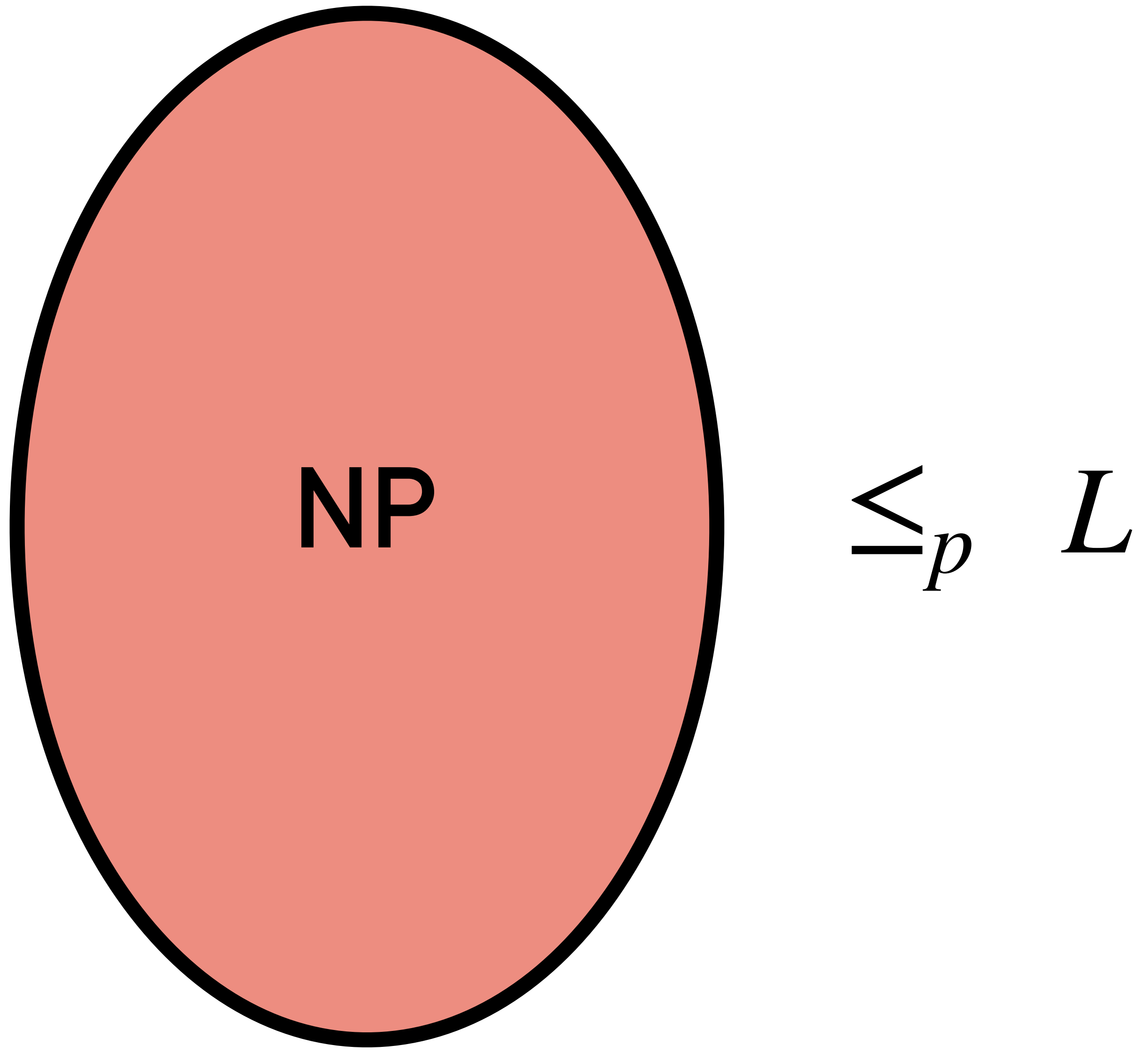
NP-Completeness and NP-Hardness

Defn: (1) A decision problem L is called NP-hard if $L' \leq_p L$, for every $L' \in$ NP.



NP-Completeness and NP-Hardness

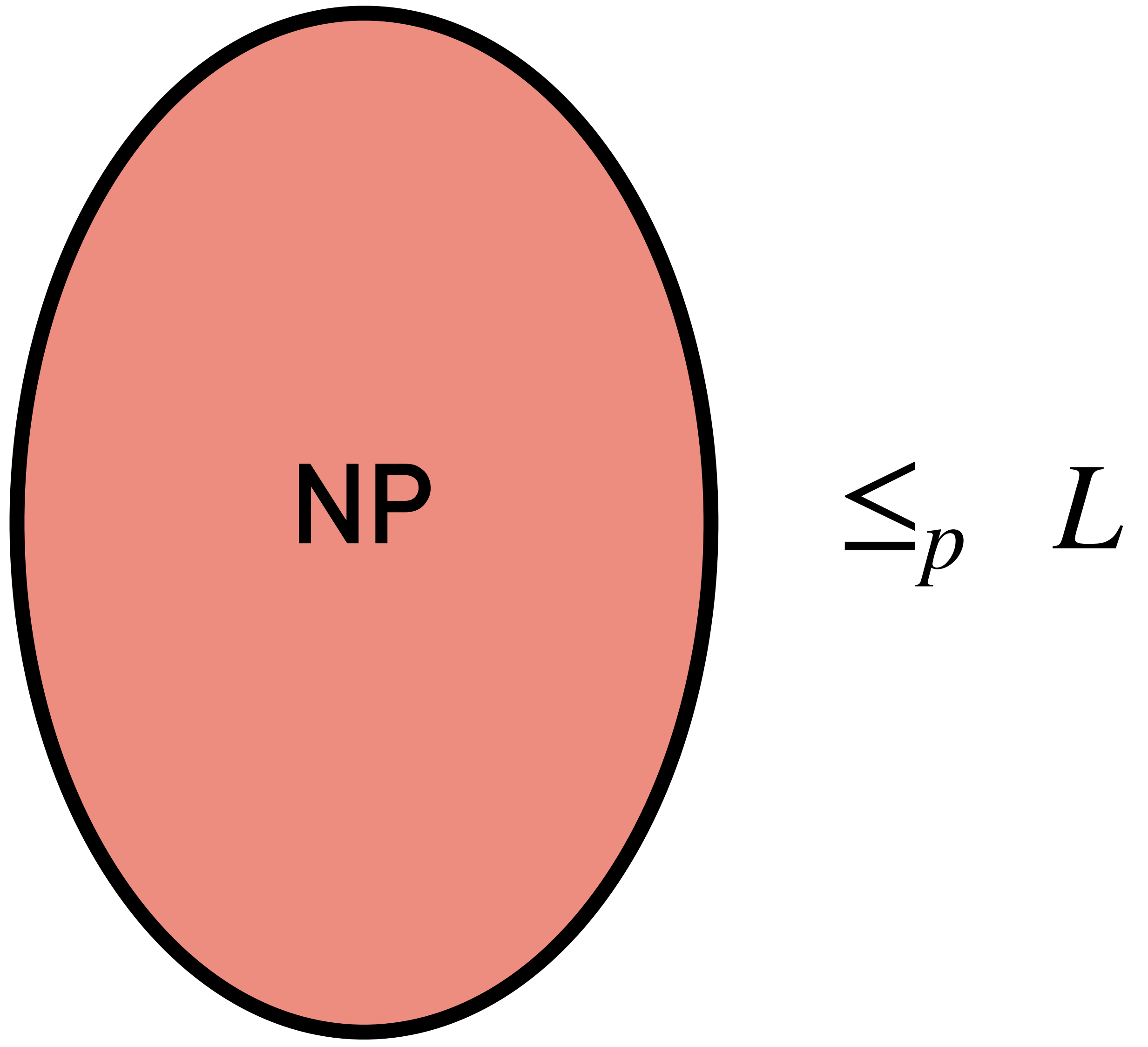
Defn: (1) A decision problem L is called NP-hard if $L' \leq_p L$, for every $L' \in$ NP.



NP-Completeness and NP-Hardness

Defn: (1) A decision problem L is called NP-hard if $L' \leq_p L$, for every $L' \in$ NP.

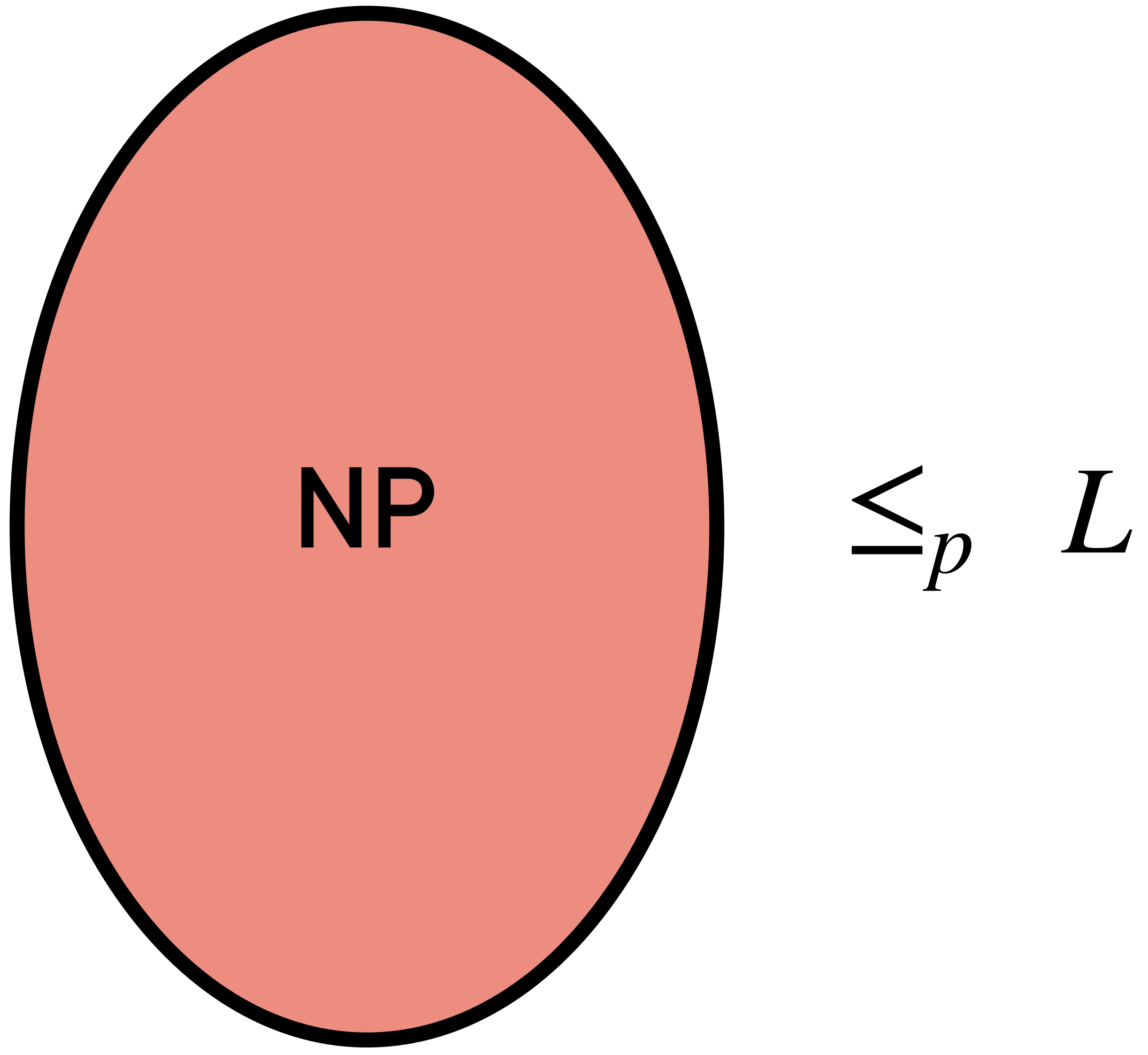
(2) An **NP-hard** decision problem L is called NP-complete if $L \in$ NP.



NP-Completeness and NP-Hardness

Defn: (1) A decision problem L is called **NP-hard** if $L' \leq_p L$, for every $L' \in$ **NP**.

(2) An **NP-hard** decision problem L is called **NP-complete** if $L \in$ **NP**.



Observation: If L is polytime solvable, then so is every problem in **NP**.

NP-Completeness and NP-Hardness

NP-Completeness and NP-Hardness

Claim: If a decision problem L is NP-complete, then $L \in \mathsf{P}$ if and only if $\mathsf{P} = \mathsf{NP}$.

NP-Completeness and NP-Hardness

Claim: If a decision problem L is NP-complete, then $L \in$ P if and only if P $=$ NP.

Proof:

NP-Completeness and NP-Hardness

Claim: If a decision problem L is **NP-complete**, then $L \in \mathsf{P}$ if and only if $\mathsf{P} = \mathsf{NP}$.

Proof: $L \in \mathsf{P} \implies \mathsf{NP} \subseteq \mathsf{P}$:

NP-Completeness and NP-Hardness

Claim: If a decision problem L is **NP-complete**, then $L \in \mathsf{P}$ if and only if $\mathsf{P} = \mathsf{NP}$.

Proof: $L \in \mathsf{P} \implies \mathsf{NP} \subseteq \mathsf{P}$:

Let $L' \in \mathsf{NP}$.

NP-Completeness and NP-Hardness

Claim: If a decision problem L is **NP-complete**, then $L \in \mathsf{P}$ if and only if $\mathsf{P} = \mathsf{NP}$.

Proof: $L \in \mathsf{P} \implies \mathsf{NP} \subseteq \mathsf{P}$:

Let $L' \in \mathsf{NP}$. Then $L' \leq_p L$ because L is **NP-complete**.

NP-Completeness and NP-Hardness

Claim: If a decision problem L is **NP-complete**, then $L \in \mathsf{P}$ if and only if $\mathsf{P} = \mathsf{NP}$.

Proof: $L \in \mathsf{P} \implies \mathsf{NP} \subseteq \mathsf{P}$:

Let $L' \in \mathsf{NP}$. Then $L' \leq_p L$ because L is **NP-complete**. Hence, $L' \in \mathsf{P}$.

NP-Completeness and NP-Hardness

Claim: If a decision problem L is **NP-complete**, then $L \in \mathsf{P}$ if and only if $\mathsf{P} = \mathsf{NP}$.

Proof: $L \in \mathsf{P} \implies \mathsf{NP} \subseteq \mathsf{P}$:

Let $L' \in \mathsf{NP}$. Then $L' \leq_p L$ because L is **NP-complete**. Hence, $L' \in \mathsf{P}$.

$\mathsf{P} = \mathsf{NP} \implies L \in \mathsf{P}$:

NP-Completeness and NP-Hardness

Claim: If a decision problem L is **NP-complete**, then $L \in \mathsf{P}$ if and only if $\mathsf{P} = \mathsf{NP}$.

Proof: $L \in \mathsf{P} \implies \mathsf{NP} \subseteq \mathsf{P}$:

Let $L' \in \mathsf{NP}$. Then $L' \leq_p L$ because L is **NP-complete**. Hence, $L' \in \mathsf{P}$.

$\mathsf{P} = \mathsf{NP} \implies L \in \mathsf{P}$:

$L \in \mathsf{NP}$ because it is **NP-complete**.

NP-Completeness and NP-Hardness

Claim: If a decision problem L is **NP-complete**, then $L \in$ P if and only if P = NP.

Proof: $L \in \mathsf{P} \implies \mathsf{NP} \subseteq \mathsf{P}$:

Let $L' \in$ NP. Then $L' \leq_p L$ because L is **NP-complete**. Hence, $L' \in$ P.

$\mathsf{P} = \mathsf{NP} \implies L \in \mathsf{P}$:

$L \in$ NP because it is **NP-complete**. Therefore, $L \in$ P.

NP-Completeness and NP-Hardness

Claim: If a decision problem L is **NP-complete**, then $L \in \mathsf{P}$ if and only if $\mathsf{P} = \mathsf{NP}$.

Proof: $L \in \mathsf{P} \implies \mathsf{NP} \subseteq \mathsf{P}$:

Let $L' \in \mathsf{NP}$. Then $L' \leq_p L$ because L is **NP-complete**. Hence, $L' \in \mathsf{P}$.

$\mathsf{P} = \mathsf{NP} \implies L \in \mathsf{P}$:

$L \in \mathsf{NP}$ because it is **NP-complete**. Therefore, $L \in \mathsf{P}$. ∎

Cook-Levin Theorem

Cook-Levin Theorem

Cook-Levin Theorem [Coo71, Lev73]:

Cook-Levin Theorem

Cook-Levin Theorem [Coo71, Lev73]:

1) *SAT* is NP-complete.

Cook-Levin Theorem

Cook-Levin Theorem [Coo71, Lev73]:

1) *SAT* is NP-complete.

2) *3SAT* is NP-complete.

Cook-Levin Theorem

Cook-Levin Theorem [Coo71, Lev73]:

1) *SAT* is NP-complete.

2) *3SAT* is NP-complete.

$SAT = \{\phi \mid \phi \text{ is a satisfiable CNF formula}\}$

Cook-Levin Theorem

Cook-Levin Theorem [Coo71, Lev73]:

1) *SAT* is NP-complete.

2) *3SAT* is NP-complete.

$SAT = \{\phi \mid \phi \text{ is a satisfiable CNF formula}\}$

$3SAT = \{\phi \mid \phi \text{ is a satisfiable 3CNF formula}\}$

Boolean Formulas

Boolean Formulas

A boolean formula consists of:

Boolean Formulas

A boolean formula consists of:

- Variables: $u_1, u_2, \ldots, u_k$, where $u_i \in \{0,1\}$.

Boolean Formulas

A boolean formula consists of:

- Variables: $u_1, u_2, \ldots, u_k$, where $u_i \in \{0,1\}$.
- Operators: AND ($\land$), OR ($\lor$), NOT ($\neg$)

Boolean Formulas

A boolean formula consists of:

- Variables: $u_1, u_2, \ldots, u_k$, where $u_i \in \{0,1\}$.
- Operators: AND ($\wedge$), OR ($\vee$), NOT ($\neg$)

Examples: $\phi_1 = u_1 \wedge \neg u_1$

Boolean Formulas

A boolean formula consists of:

- Variables: $u_1, u_2, \ldots, u_k$, where $u_i \in \{0,1\}$.
- Operators: AND ($\wedge$), OR ($\vee$), NOT ($\neg$)

Examples: $\phi_1 = u_1 \wedge \neg u_1$, $\phi_2 = (u_1 \vee u_2) \wedge (u_3 \wedge \neg u_4)$

Boolean Formulas

A boolean formula consists of:

- Variables: $u_1, u_2, \ldots, u_k$, where $u_i \in \{0,1\}$.
- Operators: AND ($\wedge$), OR ($\vee$), NOT ($\neg$)

Examples: $\phi_1 = u_1 \wedge \neg u_1$, $\phi_2 = (u_1 \vee u_2) \wedge (u_3 \wedge \neg u_4)$

Let ϕ be a boolean formula over $u_1, u_2, \ldots, u_n$ and $z \in \{0,1\}^n$:

Boolean Formulas

A boolean formula consists of:

- Variables: $u_1, u_2, \ldots, u_k$, where $u_i \in \{0,1\}$.
- Operators: AND ($\wedge$), OR ($\vee$), NOT ($\neg$)

Examples: $\phi_1 = u_1 \wedge \neg u_1$, $\phi_2 = (u_1 \vee u_2) \wedge (u_3 \wedge \neg u_4)$

Let ϕ be a boolean formula over $u_1, u_2, \ldots, u_n$ and $z \in \{0,1\}^n$:

- $\phi(z)$ denotes the value of ϕ when $u_i = z_i$.

Boolean Formulas

A boolean formula consists of:

- Variables: $u_1, u_2, \ldots, u_k$, where $u_i \in \{0,1\}$.
- Operators: AND ($\wedge$), OR ($\vee$), NOT ($\neg$)

Examples: $\phi_1 = u_1 \wedge \neg u_1$, $\phi_2 = (u_1 \vee u_2) \wedge (u_3 \wedge \neg u_4)$

Let ϕ be a boolean formula over $u_1, u_2, \ldots, u_n$ and $z \in \{0,1\}^n$:

- $\phi(z)$ denotes the value of ϕ when $u_i = z_i$.
- ϕ is satisfiable if $\exists z$ such that $\phi(z) = 1$. Otherwise, ϕ is unsatisfiable.

Boolean Formulas

A boolean formula consists of:

- Variables: $u_1, u_2, \ldots, u_k$, where $u_i \in \{0,1\}$.
- Operators: AND ($\wedge$), OR ($\vee$), NOT ($\neg$)

Examples: $\phi_1 = u_1 \wedge \neg u_1$, $\phi_2 = (u_1 \vee u_2) \wedge (u_3 \wedge \neg u_4)$

Let ϕ be a boolean formula over $u_1, u_2, \ldots, u_n$ and $z \in \{0,1\}^n$:

- $\phi(z)$ denotes the value of ϕ when $u_i = z_i$.
- ϕ is satisfiable if $\exists z$ such that $\phi(z) = 1$. Otherwise, ϕ is unsatisfiable.

Examples: ϕ_1 is unsatisfiable,

Boolean Formulas

A boolean formula consists of:

- Variables: $u_1, u_2, \ldots, u_k$, where $u_i \in \{0,1\}$.
- Operators: AND ($\wedge$), OR ($\vee$), NOT ($\neg$)

Examples: $\phi_1 = u_1 \wedge \neg u_1$, $\phi_2 = (u_1 \vee u_2) \wedge (u_3 \wedge \neg u_4)$

Let ϕ be a boolean formula over $u_1, u_2, \ldots, u_n$ and $z \in \{0,1\}^n$:

- $\phi(z)$ denotes the value of ϕ when $u_i = z_i$.
- ϕ is satisfiable if $\exists z$ such that $\phi(z) = 1$. Otherwise, ϕ is unsatisfiable.

Examples: ϕ_1 is unsatisfiable, ϕ_2 is satisfiable as $\phi_2(z) = 1$, for $z = 1010$.

SAT **and** *3SAT*

SAT and *3SAT*

Defn: A CNF formula is a boolean formula in the form of AND of ORs of variables or negation

SAT and *3SAT*

Defn: A CNF formula is a boolean formula in the form of AND of ORs of variables or negation of variables.

SAT and *3SAT*

Defn: A CNF formula is a boolean formula in the form of AND of ORs of variables or negation of variables.

Example: $(u_1 \vee \neg u_2 \vee u_3) \wedge (u_3 \vee u_2 \vee \neg u_1) \wedge (u_2 \vee \neg u_3 \vee u_4)$

SAT and *3SAT*

Defn: A CNF formula is a boolean formula in the form of AND of ORs of variables or negation of variables.

Example: $(u_1 \vee \neg u_2 \vee u_3) \wedge (u_3 \vee u_2 \vee \neg u_1) \wedge (u_2 \vee \neg u_3 \vee u_4)$

Literals

SAT and *3SAT*

Defn: A CNF formula is a boolean formula in the form of AND of ORs of variables or negation of variables.

Example: $(u_1 \vee \neg u_2 \vee u_3) \wedge (u_3 \vee u_2 \vee \neg u_1) \wedge (u_2 \vee \neg u_3 \vee u_4)$

Literals

SAT and *3SAT*

Defn: A CNF formula is a boolean formula in the form of AND of ORs of variables or negation of variables.

Example: $(u_1 \vee \neg u_2 \vee u_3) \wedge (u_3 \vee u_2 \vee \neg u_1) \wedge (u_2 \vee \neg u_3 \vee u_4)$

Literals

SAT and *3SAT*

Defn: A CNF formula is a boolean formula in the form of AND of ORs of variables or negation of variables.

Example: $(u_1 \vee \neg u_2 \vee u_3) \wedge (u_3 \vee u_2 \vee \neg u_1) \wedge (u_2 \vee \neg u_3 \vee u_4)$

Literals

Clause

SAT and *3SAT*

Defn: A CNF formula is a boolean formula in the form of AND of ORs of variables or negation of variables.

Example: $(u_1 \vee \neg u_2 \vee u_3) \wedge (u_3 \vee u_2 \vee \neg u_1) \wedge (u_2 \vee \neg u_3 \vee u_4)$

Literals

Clause

Definition: 1) *SAT* $= \{\phi \mid \phi$ is a satisfiable CNF formula$\}$

SAT and *3SAT*

Defn: A CNF formula is a boolean formula in the form of AND of ORs of variables or negation of variables.

Example: $(u_1 \vee \neg u_2 \vee u_3) \wedge (u_3 \vee u_2 \vee \neg u_1) \wedge (u_2 \vee \neg u_3 \vee u_4)$

Literals

Clause

Definition: 1) *SAT* $= \{\phi \mid \phi$ is a satisfiable CNF formula$\}$

2) *3SAT* $= \{\phi \mid \phi$ is a satisfiable 3CNF formula$\}$

SAT and *3SAT*

Defn: A CNF formula is a boolean formula in the form of AND of ORs of variables or negation of variables.

Example: $(u_1 \vee \neg u_2 \vee u_3) \wedge (u_3 \vee u_2 \vee \neg u_1) \wedge (u_2 \vee \neg u_3 \vee u_4)$

Literals

Clause

Definition: 1) *SAT* $= \{\phi \mid \phi$ is a satisfiable CNF formula$\}$

2) *3SAT* $= \{\phi \mid \phi$ is a satisfiable 3CNF formula$\}$

At most 3 literals per clause