Asymptotic Notation and Growth

$f(x) = O(g(x))$ iff $f(x) \leq cg(x)$ for a constant c and large x

Asymptotic Notation and Growth

$f(x) = O(g(x))$ iff $f(x) \leq cg(x)$ for a constant c and large x

- Different models of computation

Asymptotic Notation and Growth

$f(x) = O(g(x))$ iff $f(x) \leq cg(x)$ for a constant c and large x

- Different models of computation
- $O(n^3)$ vs. $O(n^2)$: n^3 will be eventually bigger than $100n^2$

Asymptotic Notation and Growth

$f(x) = O(g(x))$ iff $f(x) \leq cg(x)$ for a constant c and large x

- Different models of computation
- $O(n^3)$ vs. $O(n^2)$: n^3 will be eventually bigger than $100n^2$

A polynomial function $f(x) = O(n^k)$ for some constant k.

- $O(n^k)$ vs. $O(2^n)$: 2^n will be eventually bigger

Asymptotic Notation and Growth

$f(x) = O(g(x))$ iff $f(x) \leq cg(x)$ for a constant c and large x

- Different models of computation
- $O(n^3)$ vs. $O(n^2)$: n^3 will be eventually bigger than $100n^2$

A polynomial function $f(x) = O(n^k)$ for some constant k.

- $O(n^k)$ vs. $O(2^n)$: 2^n will be eventually bigger

If an algorithm $\mathcal{A}$ has a running time $O(2^n)$...

- $n = 100 \Rightarrow 2^{100} \geq 10^{30}$ is already too large

Asymptotic Notation and Growth

$f(x) = O(g(x))$ iff $f(x) \leq cg(x)$ for a constant c and large x

- Different models of computation
- $O(n^3)$ vs. $O(n^2)$: n^3 will be eventually bigger than $100n^2$

A polynomial function $f(x) = O(n^k)$ for some constant k.

- $O(n^k)$ vs. $O(2^n)$: 2^n will be eventually bigger

If an algorithm $\mathcal{A}$ has a running time $O(2^n)$...

- $n = 100 \Rightarrow 2^{100} \geq 10^{30}$ is already too large

An algorithm $\mathcal{A}$ is *efficient* if it runs in polynomial time.

Asymptotic Notation and Growth

$f(x) = O(g(x))$ iff $f(x) \le cg(x)$ for a constant c and large x

- Different models of computation
- $O(n^3)$ vs. $O(n^2)$: n^3 will be eventually bigger than $100n^2$

A polynomial function $f(x) = O(n^k)$ for some constant k.

- $O(n^k)$ vs. $O(2^n)$: 2^n will be eventually bigger

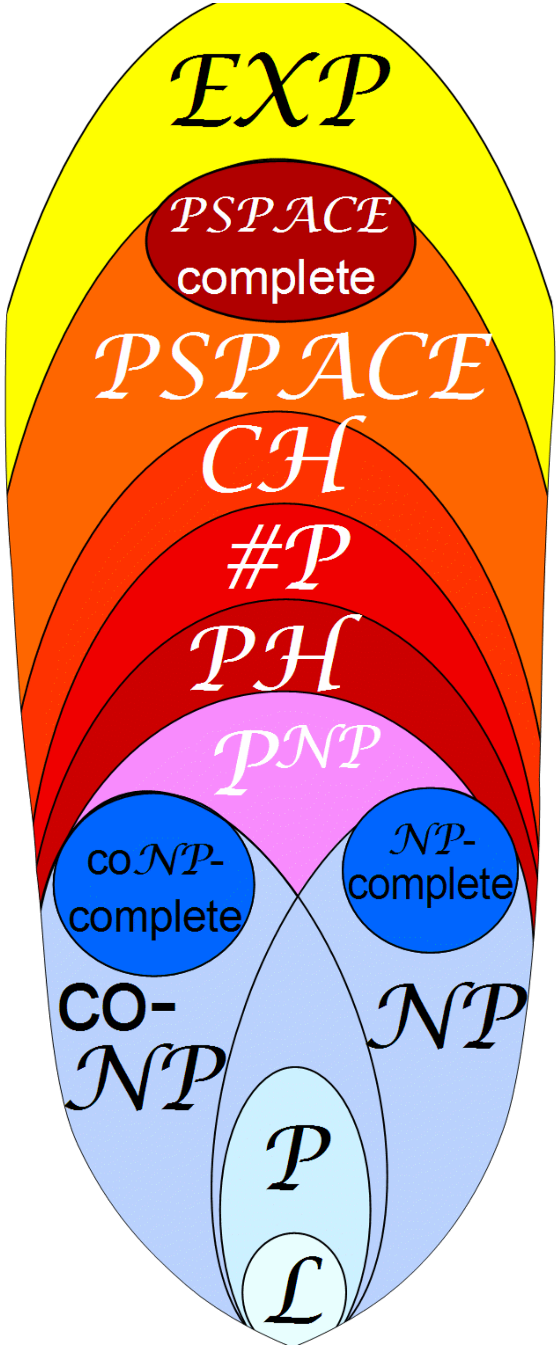
If an algorithm $\mathcal{A}$ has a running time $O(2^n)$...

- $n = 100 \Rightarrow 2^{100} \ge 10^{30}$ is already too large

An algorithm $\mathcal{A}$ is *efficient* if it runs in polynomial time.

- $100n^{100}$ vs. $2^{0.01n}$?

Computability and Complexity

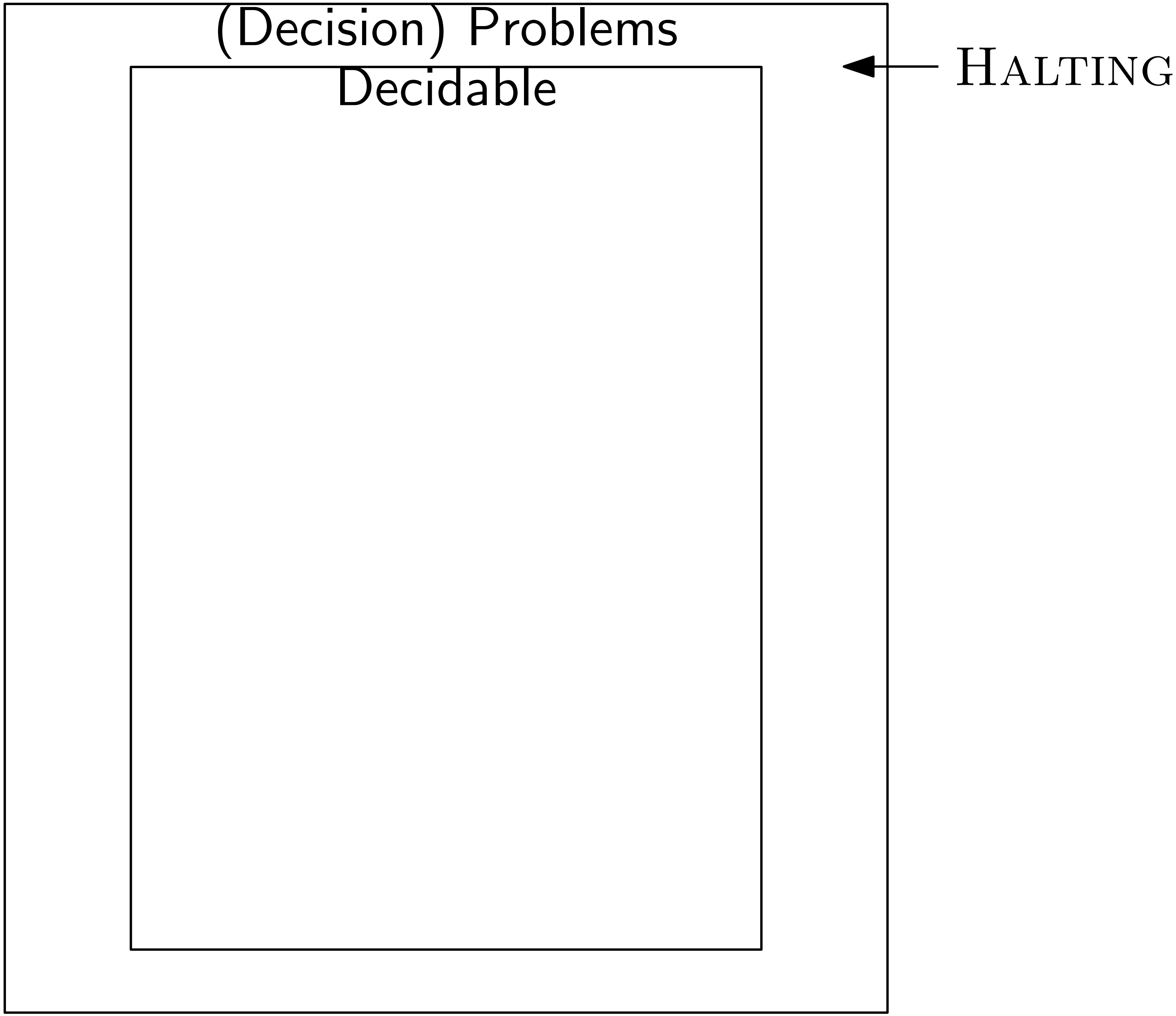


Computability and Complexity

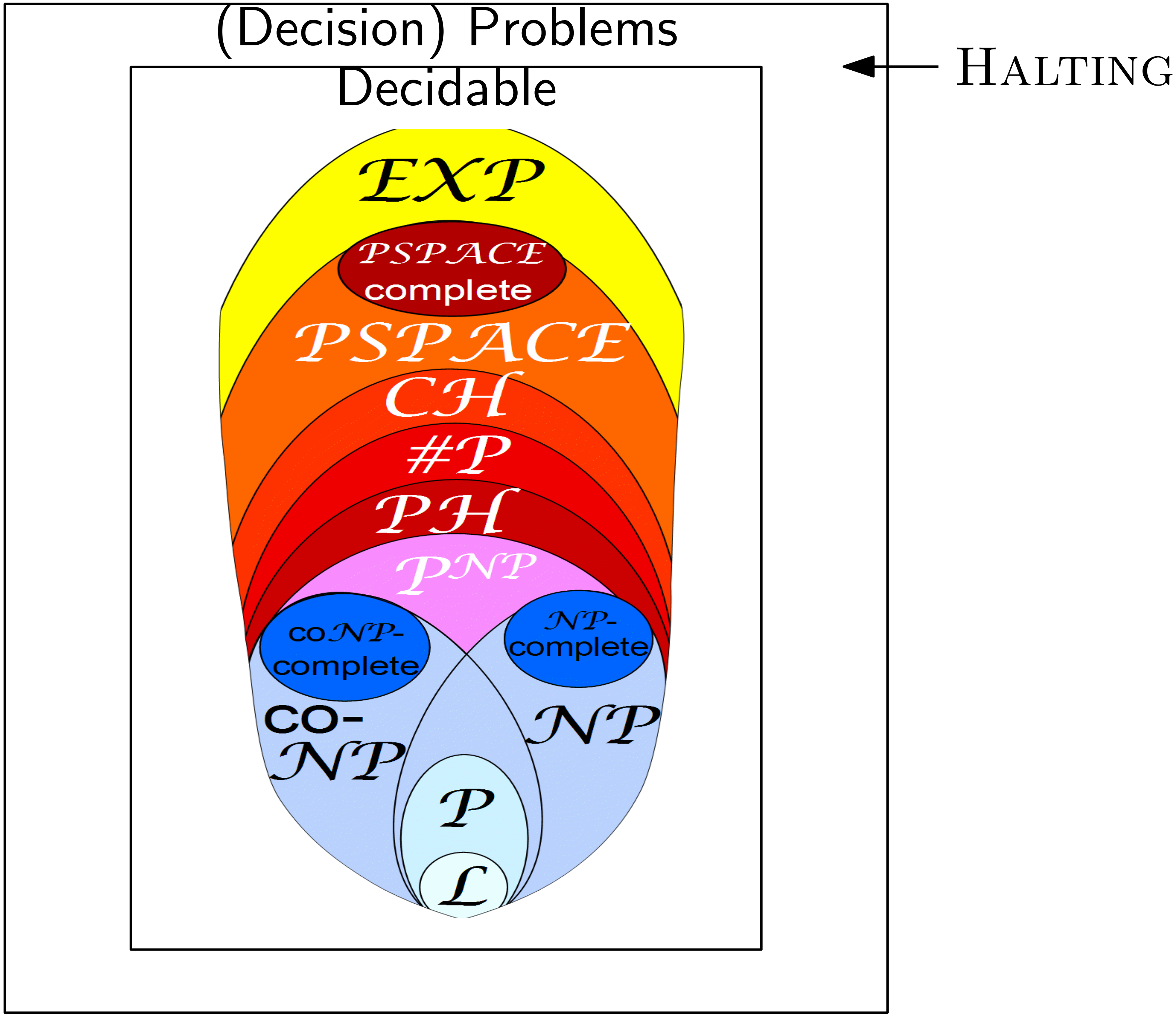
(Decision) Problems

Decidable

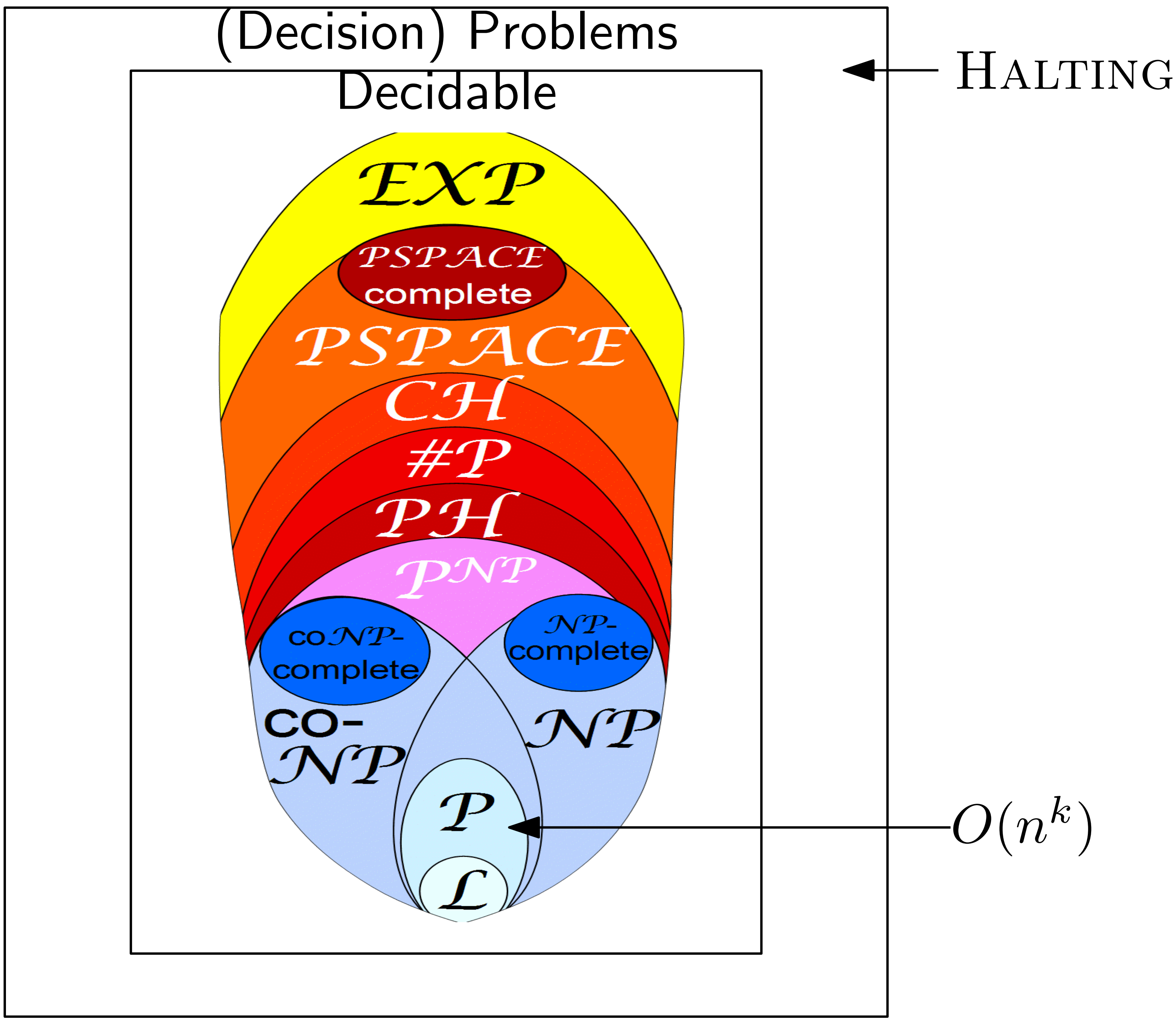
Computability and Complexity



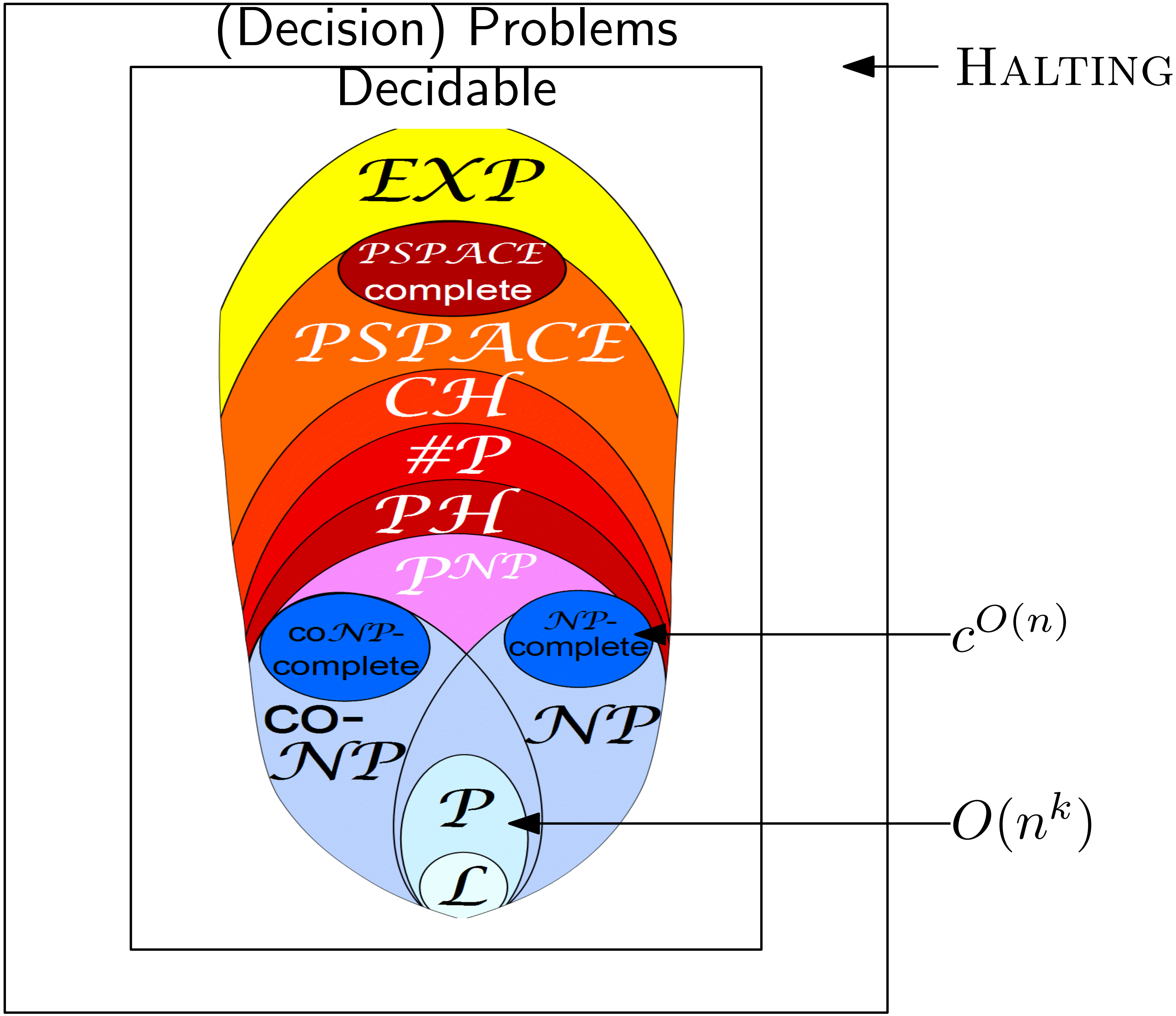
Computability and Complexity



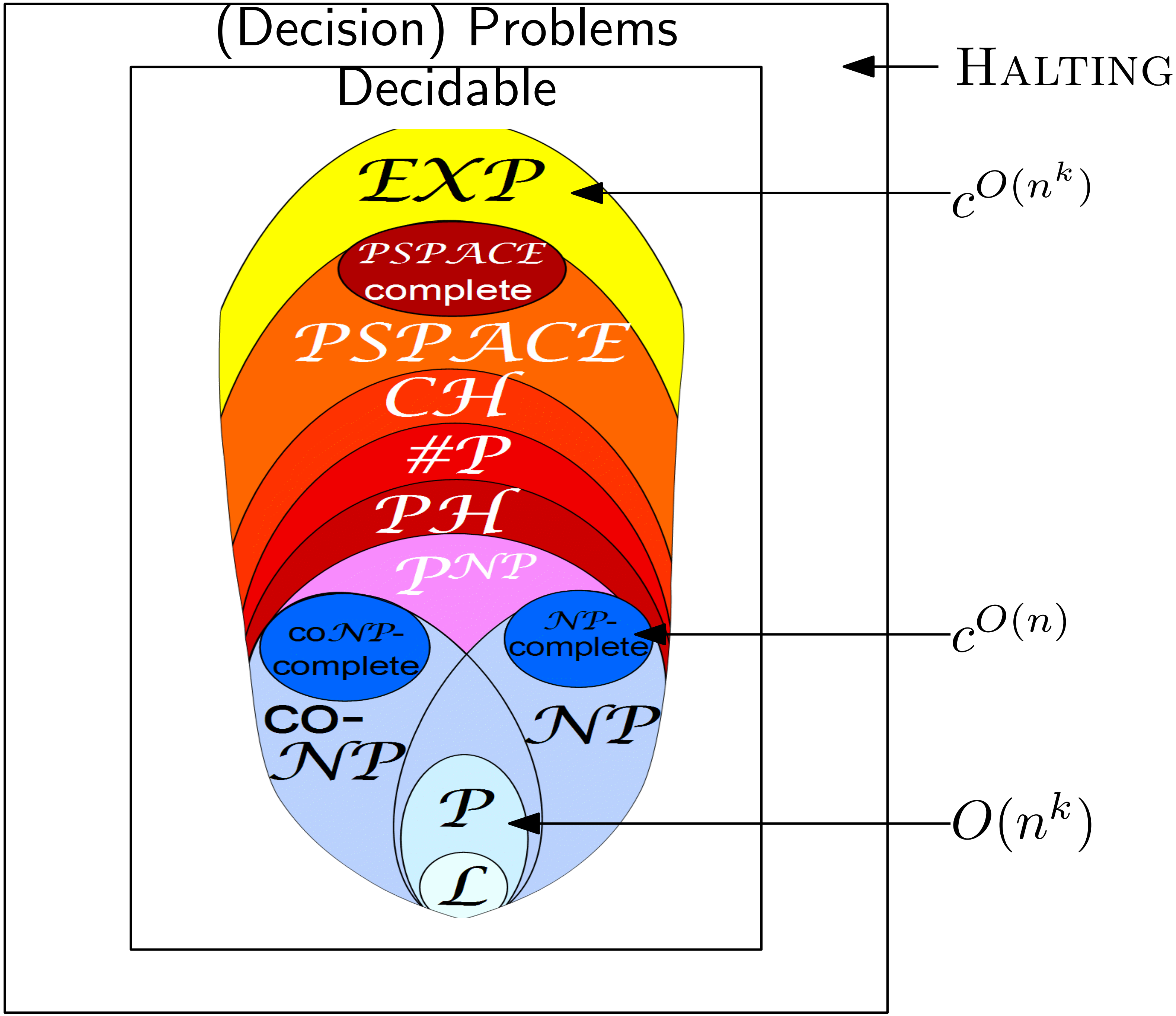
Computability and Complexity



Computability and Complexity



Computability and Complexity



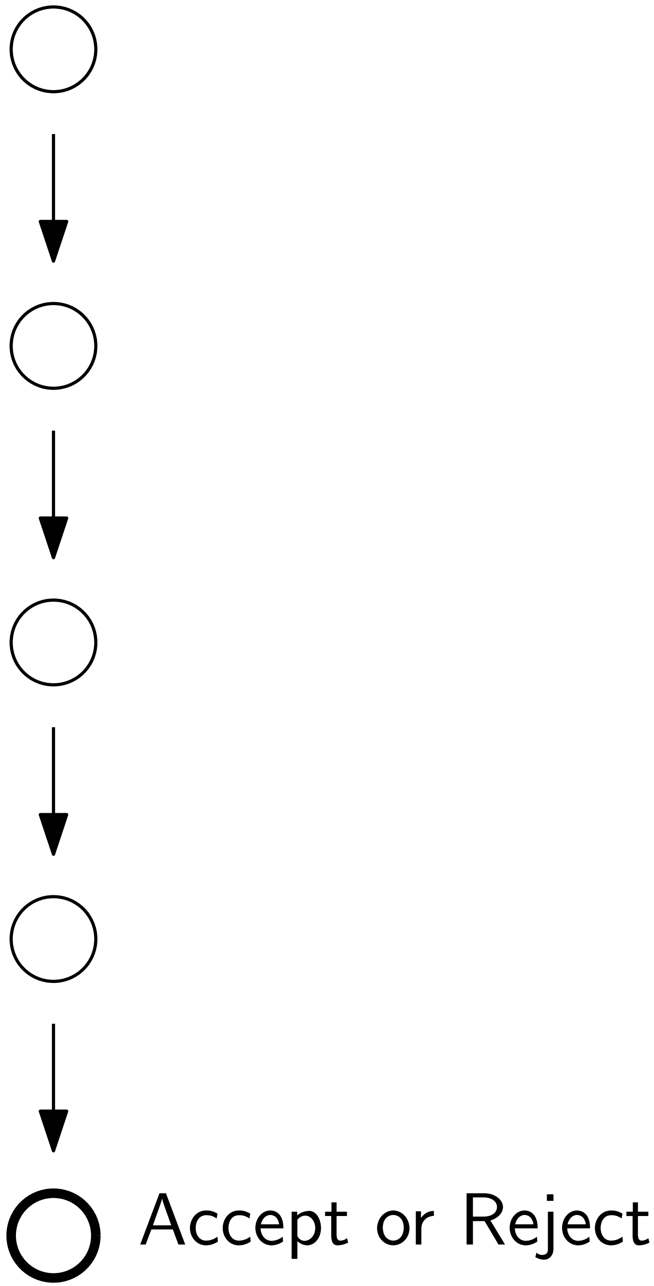
Class P and NP

- P is the class of decision problems that can be solved in *Polynomial time*.

Class P and NP

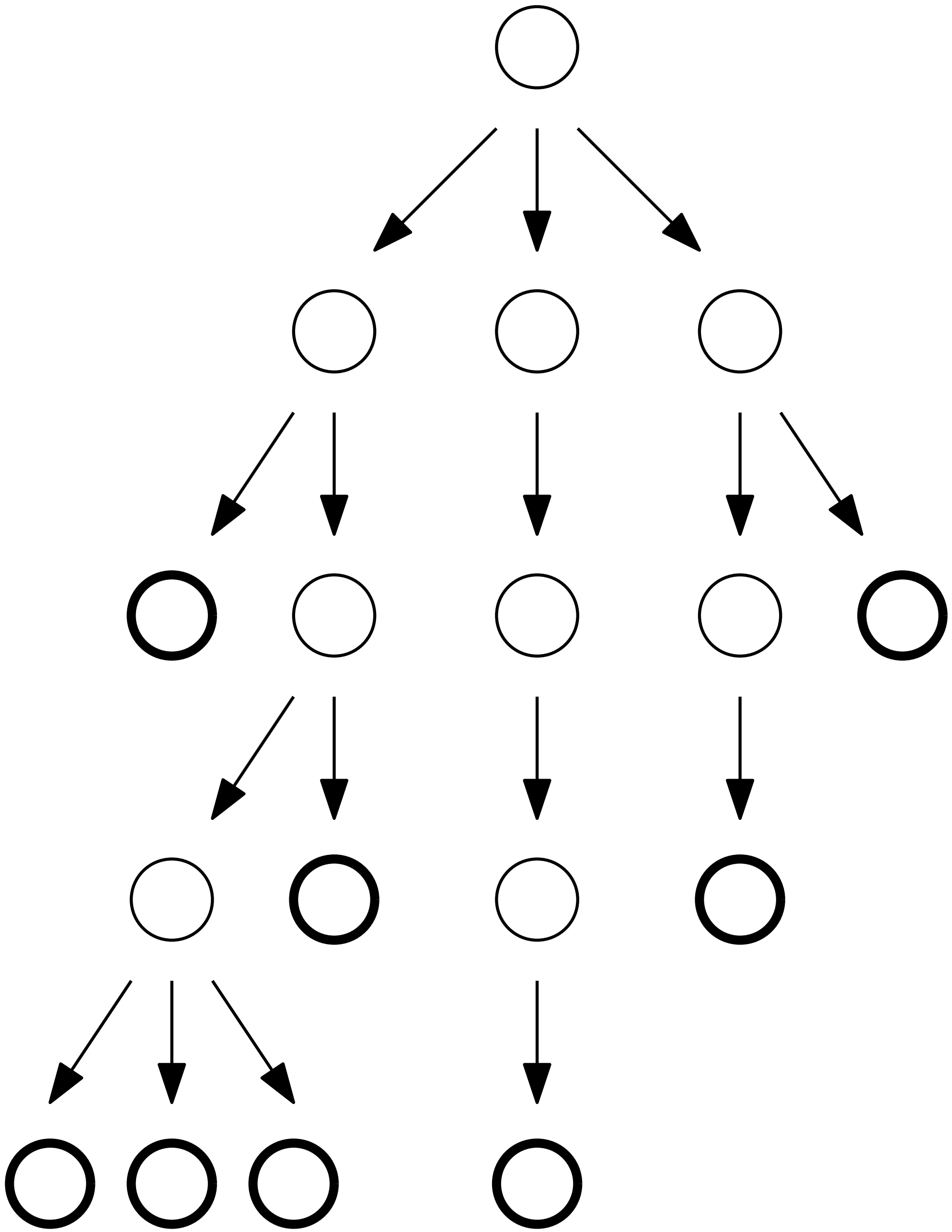
- P is the class of decision problems that can be solved in *Polynomial time*.
- NP is the class of decision problems that can be solved in *Nondeterministic Polynomial time*.

Nondeterminism



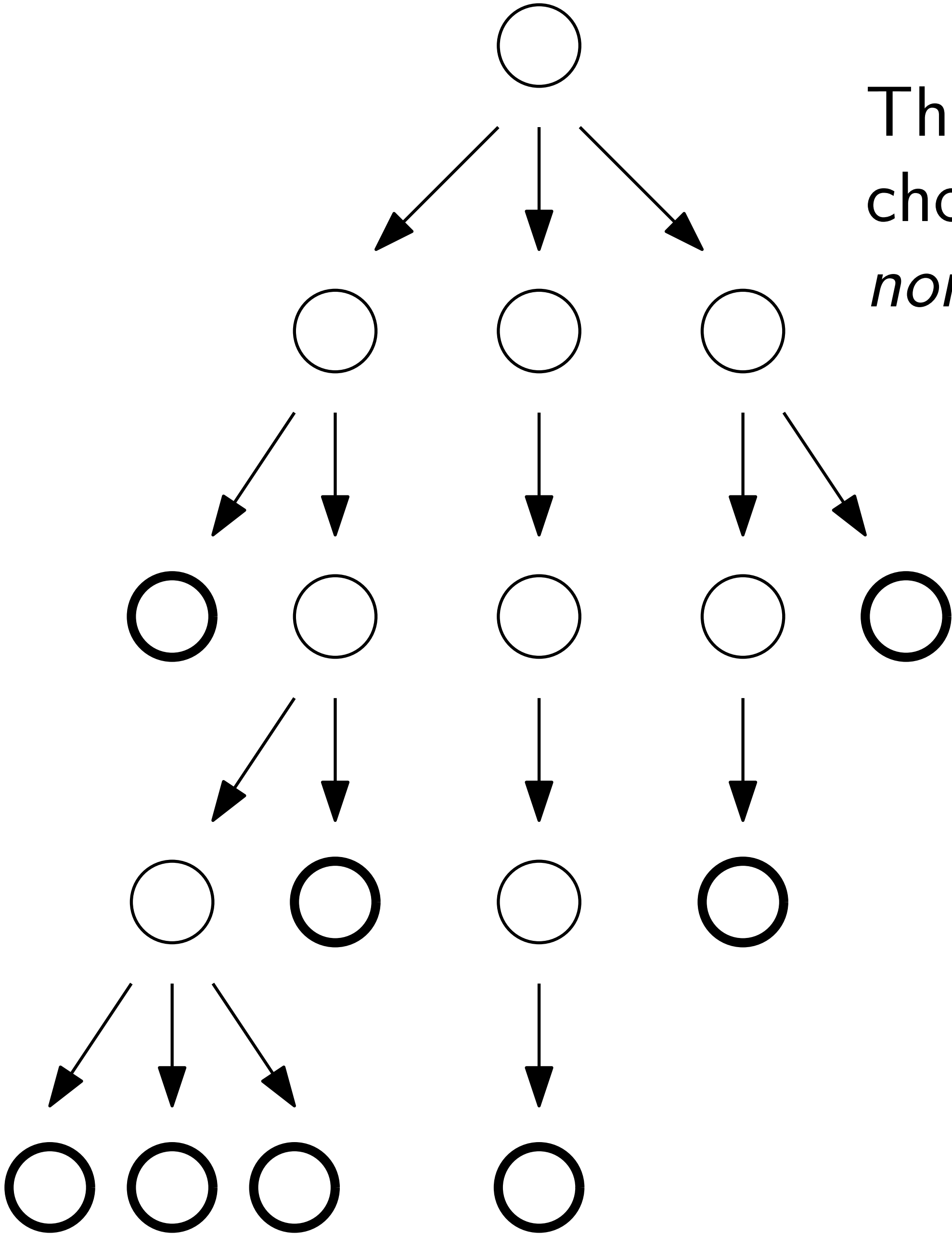
Nondeterminism

Now we consider a program with multiple execution paths.



Nondeterminism

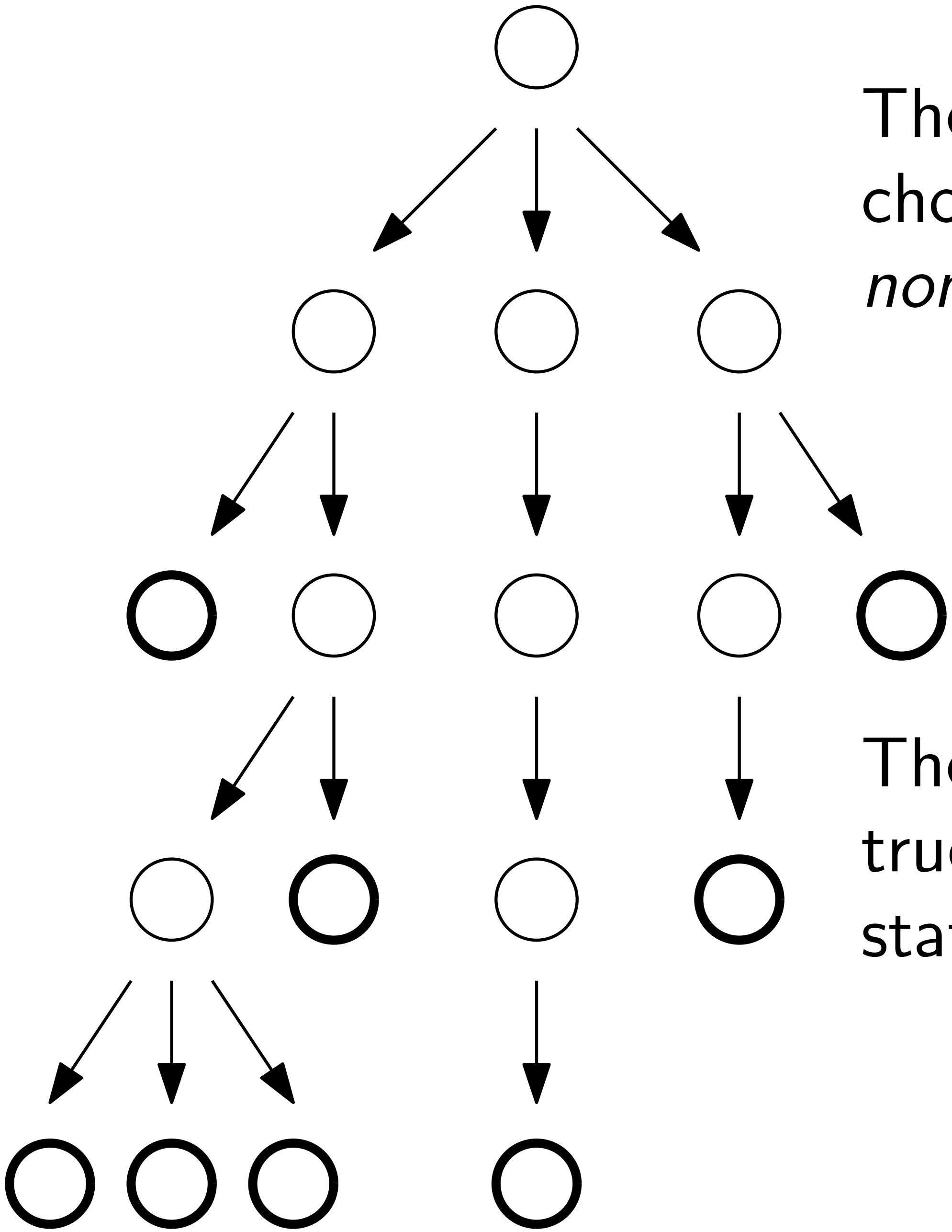
Now we consider a program with multiple execution paths.



The program makes a choice for each branch *nondeterministically*.

Nondeterminism

Now we consider a program with multiple execution paths.



The program makes a choice for each branch *nondeterministically*.

The program returns true iff one of final states accepts!

Examples

COMPOSITE: given an integer n, decide if n is a composite number.

Examples

COMPOSITE: given an integer n, decide if n is a composite number.

1. Choose an integer $1 < i < n$ *nondeterministically*.
2. Return true if i divides n.

Examples

$\textsc{Composite}$: given an integer n, decide if n is a composite number.

1. Choose an integer $1 < i < n$ *nondeterministically*.
2. Return true if i divides n.

The program returns true iff there is a divisor!
$\Rightarrow \textsc{Composite} \in$ NP

Examples

$\textsc{Composite}$: given an integer n, decide if n is a composite number.

1. Choose an integer $1 < i < n$ *nondeterministically*.
2. Return true if i divides n.

The program returns true iff there is a divisor!
$\Rightarrow \textsc{Composite} \in$ NP

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

$\textsc{VertexCover}$: given a graph G and integer k, decide if G has a vertex cover of size k.

Examples

$\textsc{Composite}$: given an integer n, decide if n is a composite number.

1. Choose an integer $1 < i < n$ *nondeterministically*.
2. Return true if i divides n.

The program returns true iff there is a divisor!
$\Rightarrow \textsc{Composite} \in$ NP

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

$\textsc{VertexCover}$: given a graph G and integer k, decide if G has a vertex cover of size k.

1. Choose k vertices *nondeterministically*.
2. Return true if the chosen vertex set covers all edges.

Nondeterminism and Certifier

- NP is the class of decision problems that can be solved in *Nondeterministic Polynomial time*.

If a nondeterministic algorithm returns true for an instance $\mathcal{I}$, there exists an *execution path* that accepts $\mathcal{I}$!

Nondeterminism and Certifier

- NP is the class of decision problems that can be solved in *Nondeterministic Polynomial time*.

If a nondeterministic algorithm returns true for an instance $\mathcal{I}$, there exists an *execution path* that accepts $\mathcal{I}$!

The algorithm can be viewed as a *certifier* $\mathcal{A}(s, t)$ that returns true for s given a proper *certificate* t.

- NP is the class of decision problems that have *efficient* certifiers.

Nondeterminism and Certifier

- NP is the class of decision problems that can be solved in *Nondeterministic Polynomial time*.

If a nondeterministic algorithm returns true for an instance $\mathcal{I}$, there exists an *execution path* that accepts $\mathcal{I}$!

The algorithm can be viewed as a *certifier* $\mathcal{A}(s,t)$ that returns true for s given a proper *certificate* t.

- NP is the class of decision problems that have *efficient* certifiers.

  $\mathcal{A}(s,t)$ is an efficient certifier for a problem if $\mathcal{A} \in$ P and there is a polynomial p s.t. s is a YES-instance iff there is a certificate t satisfying followings:

  1. $|t| \leq p(|s|)$
  2. $\mathcal{A}(s,t) =$ true

Examples Revisited

$\textsc{Composite}$: given an integer n, decide if n is a composite number.

1. Choose an integer $1 < i < n$ *nondeterministically*.
2. Return true if i divides n.

The program returns true iff there is a divisor!
$\Rightarrow \textsc{Composite} \in$ NP

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

$\textsc{VertexCover}$: given a graph G and integer k, decide if G has a vertex cover of size k.

1. Choose k vertices *nondeterministically*.
2. Return true if chosen vertex set covers all edges.

Examples Revisited

$\textsc{Composite}$: given an integer n, decide if n is a composite number.

The certificate is a divisor of n.

1. Choose an integer $1 < i < n$ *nondeterministically*.
2. Return true if i divides n.

The certifier verifies if i divides n.

The program returns true iff there is a divisor!
$\Rightarrow \textsc{Composite} \in$ NP

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

$\textsc{VertexCover}$: given a graph G and integer k, decide if G has a vertex cover of size k.

1. Choose k vertices *nondeterministically*.
2. Return true if chosen vertex set covers all edges.

Examples Revisited

COMPOSITE: given an integer n, decide if n is a composite number.

The certificate is a divisor of n.

1. Choose an integer $1 < i < n$ *nondeterministically*.
2. Return true if i divides n.

The certifier verifies if i divides n.

The program returns true iff there is a divisor!
$\Rightarrow$ COMPOSITE $\in$ NP

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

VERTEXCOVER: given a graph G and integer k, decide if G has a vertex cover of size k.

1. Choose k vertices *nondeterministically*.
2. Return true if chosen vertex set covers all edges.

The certificate is a subset of $V(G)$ of size k;
the certifier verifies it is a vertex cover.

P vs. NP

- P is the class of decision problems that can be solved in *Polynomial time*.
- NP is the class of decision problems that can be solved in *Nondeterministic Polynomial time*.

$\Rightarrow$ P $\subseteq$ NP.

P vs. NP

- P is the class of decision problems that can be solved in *Polynomial time*.
- NP is the class of decision problems that can be solved in *Nondeterministic Polynomial time*.

$\Rightarrow$ P $\subseteq$ NP.

- Every efficient algorithm ($\in$ P) is in NP.

P vs. NP

- P is the class of decision problems that can be solved in *Polynomial time*.
- NP is the class of decision problems that can be solved in *Nondeterministic Polynomial time*.

$\Rightarrow$ P $\subseteq$ NP.

- Every efficient algorithm ($\in$ P) is in NP.
- $\textsc{Composite} \in$ P.
- $\textsc{VertexCover} \notin$ P if P $\neq$ NP
  $\Rightarrow$ $\textsc{VertexCover}$ is really *hard*.

P vs. NP

- P is the class of decision problems that can be solved in *Polynomial time*.
- NP is the class of decision problems that can be solved in *Nondeterministic Polynomial time*.

$\Rightarrow$ P $\subseteq$ NP.

- Every efficient algorithm ($\in$ P) is in NP.
- $\textsc{Composite} \in$ P.
- $\textsc{VertexCover} \notin$ P if P $\neq$ NP
  $\Rightarrow$ $\textsc{VertexCover}$ is really *hard*.
- P $\neq$ NP ?

Reduction

Suppose $A \leq B$.

1. A is at least as easy as B.

Reduction

Suppose $A \leq B$.

1. A is at least as easy as B.
2. B is at least as hard as A.

Reduction

- Cook reduction: A is Turing reducible to B if A can be *efficiently* solvable given an oracle for B.

Reduction

- Cook reduction: A is Turing reducible to B if A can be *efficiently* solvable given an oracle for B.

  $\textsc{RealSorting} \leq_T \textsc{Halting}$: Real numbers can be efficiently sorted given an oracle for $\textsc{Halting}$.

Reduction

- Cook reduction: A is Turing reducible to B if A can be *efficiently* solvable given an oracle for B.

  $\textsc{RealSorting} \leq_T \textsc{Halting}$: Real numbers can be efficiently sorted given an oracle for $\textsc{Halting}$.

  $\textsc{Selection} \leq_T \textsc{Sorting}$: m-th largest number can be efficiently selected given an oracle for Sorting.

Reduction

- Cook reduction: A is Turing reducible to B if A can be *efficiently* solvable given an oracle for B.

  $\textsc{RealSorting} \leq_T \textsc{Halting}$: Real numbers can be efficiently sorted given an oracle for $\textsc{Halting}$.

  $\textsc{Selection} \leq_T \textsc{Sorting}$: m-th largest number can be efficiently selected given an oracle for Sorting.

- Karp reduction: A is many-one reducible to B if an instance of A can be *efficiently* converted into an instance of B.

Reduction

- Cook reduction: A is Turing reducible to B if A can be *efficiently* solvable given an oracle for B.

  $\textsc{RealSorting} \leq_T \textsc{Halting}$: Real numbers can be efficiently sorted given an oracle for $\textsc{Halting}$.

  $\textsc{Selection} \leq_T \textsc{Sorting}$: m-th largest number can be efficiently selected given an oracle for Sorting.

- Karp reduction: A is many-one reducible to B if an instance of A can be *efficiently* converted into an instance of B.

  $\textsc{VertexCover} \leq_m \textsc{IndependentSet}$: If we know $\textsc{VertexCover}$ is hard, $\textsc{IndependentSet}$ is hard, too!

$\textsc{VertexCover} \leq \textsc{IndependentSet}$

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

$\textsc{VertexCover}$: given a graph G and integer k, decide if G has a vertex cover of size k.

VertexCover $\leq$ IndependentSet

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

VertexCover: given a graph G and integer k, decide if G has a vertex cover of size k.

$S \subseteq V(G)$ is an *independent set* for a graph G if there are no adjacent vertices in S.

IndependentSet: given a graph G and integer k, decide if G has an independent set of size k.

$\textsc{VertexCover} \le \textsc{IndependentSet}$

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

$\textsc{VertexCover}$: given a graph G and integer k, decide if G has a vertex cover of size k.

$S \subseteq V(G)$ is an *independent set* for a graph G if there are no adjacent vertices in S.

$\textsc{IndependentSet}$: given a graph G and integer k, decide if G has an independent set of size k.

Observation: S is a vertex cover iff $V(G) \setminus S$ is an independent set.

VertexCover $\leq$ IndependentSet

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

VertexCover: given a graph G and integer k, decide if G has a vertex cover of size k.

$S \subseteq V(G)$ is an *independent set* for a graph G if there are no adjacent vertices in S.

IndependentSet: given a graph G and integer k, decide if G has an independent set of size k.

Observation: S is a vertex cover iff $V(G) \setminus S$ is an independent set.

An instance (G, k) of VertexCover can be converted into an instance $(G, |V(G)| - k)$ of IndependentSet.

NP-hardness and NP-completeness

- A problem A is *NP-hard* if A is at least as hard as any NP problem.

  That is, for *any* NP problem B, $B \leq A$.

NP-hardness and NP-completeness

- A problem A is *NP-hard* if A is at least as hard as any NP problem.

  That is, for *any* NP problem B, $B \leq A$.

- A problem A is *NP-complete* if

  1. A is NP-hard.
  2. $A \in$ NP.

NP-hardness and NP-completeness

- A problem A is *NP-hard* if A is at least as hard as any NP problem.

  That is, for *any* NP problem B, $B \leq A$.

- A problem A is *NP-complete* if

  1. A is NP-hard.
  2. $A \in$ NP.

- CSAT is NP-complete.

NP-hardness and NP-completeness

- A problem A is *NP-hard* if A is at least as hard as any NP problem.

  That is, for *any* NP problem B, $B \leq A$.

- A problem A is *NP-complete* if

  1. A is NP-hard.
  2. $A \in$ NP.

- CSAT is NP-complete.
- $\mathrm{CSAT} \leq \mathrm{3SAT} \leq \textsc{VertexCover} \leq \textsc{IndependentSet}$

NP-hardness and NP-completeness

- A problem A is *NP-hard* if A is at least as hard as any NP problem.

  That is, for *any* NP problem B, $B \leq A$.

- A problem A is *NP-complete* if

  1. A is NP-hard.
  2. $A \in$ NP.

- CSAT is NP-complete.
- $\mathrm{CSAT} \leq \mathrm{3SAT} \leq \textsc{VertexCover} \leq \textsc{IndependentSet}$
- There are plenty of *known* NP-complete problems!

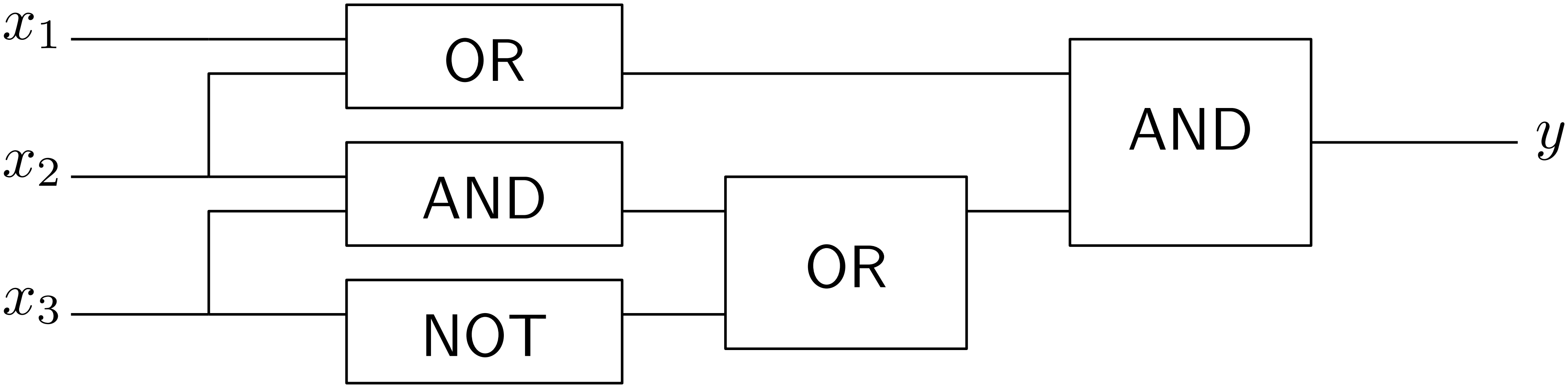
CSAT is NP-complete

A *circuit* consists of input (binary) variables, AND, OR, and NOT gates, and an output (binary) value.

CSAT is NP-complete

A *circuit* consists of input (binary) variables, AND, OR, and NOT gates, and an output (binary) value.

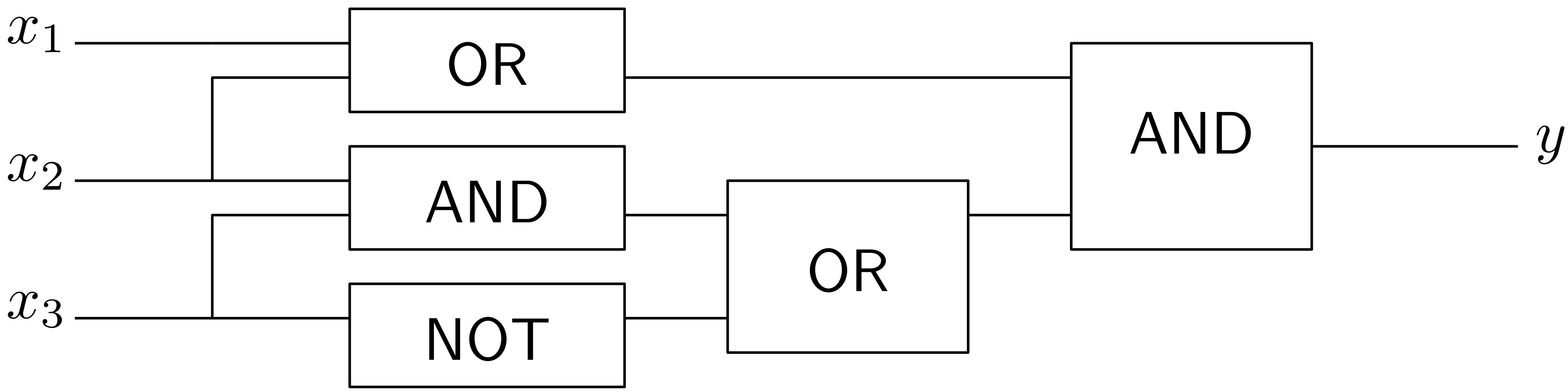
$$y = (x_1 \vee x_2) \wedge (\neg x_3 \vee (x_2 \wedge x_3))$$



CSAT is NP-complete

A *circuit* consists of input (binary) variables, AND, OR, and NOT gates, and an output (binary) value.

$$y = (x_1 \vee x_2) \wedge (\neg x_3 \vee (x_2 \wedge x_3))$$



CSAT: given a circuit C with n input variables, is there an assignment for variables that returns true?

CSAT is NP-complete

CSAT: given a circuit C with n input variables, is there an assignment for variables that returns true?

Theorem. CSAT is NP-complete.

CSAT is NP-complete

CSAT: given a circuit C with n input variables, is there an assignment for variables that returns true?

Theorem. CSAT is NP-complete.

1. $\mathrm{CSAT} \in$ NP.
2. $X \leq \mathrm{CSAT}$ for every NP problem X.

CSAT is NP-complete

CSAT: given a circuit C with n input variables, is there an assignment for variables that returns true?

Theorem. CSAT is NP-complete.

1. CSAT $\in$ NP.
2. $X \leq$ CSAT for every NP problem X.

   Certificate: an assignment for input variables

   Certifier: verify that the output value is true

CSAT is NP-complete

CSAT: given a circuit C with n input variables, is there an assignment for variables that returns true?

Theorem. CSAT is NP-complete.

1. $\mathrm{CSAT} \in$ NP.
2. $X \leq \mathrm{CSAT}$ for every NP problem X.

   Suppose $X \in$ NP. Let $\mathcal{A}$ be an efficient certifier for X.

CSAT is NP-complete

CSAT: given a circuit C with n input variables, is there an assignment for variables that returns true?

Theorem. CSAT is NP-complete.

1. $\mathrm{CSAT} \in$ NP.

2. $X \leq \mathrm{CSAT}$ for every NP problem X.

   Suppose $X \in$ NP. Let $\mathcal{A}$ be an efficient certifier for X.

   s is a YES-instance for X iff there is a certificate t of size $p(|s|)$ and $\mathcal{A}(s,t) =$ true.

CSAT is NP-complete

CSAT: given a circuit C with n input variables, is there an assignment for variables that returns true?

Theorem. CSAT is NP-complete.

1. $\mathrm{CSAT} \in$ NP.

2. $X \leq \mathrm{CSAT}$ for every NP problem X.

   Suppose $X \in$ NP. Let $\mathcal{A}$ be an efficient certifier for X.

   s is a YES-instance for X iff there is a certificate t of size $p(|s|)$ and $\mathcal{A}(s,t) =$ true.

   Convert $\mathcal{A}$ into a circuit for CSAT.

CSAT is NP-complete

CSAT: given a circuit C with n input variables, is there an assignment for variables that returns true?

Theorem. CSAT is NP-complete.

1. $\mathrm{CSAT} \in$ NP.
2. $X \leq \mathrm{CSAT}$ for every NP problem X.

   Suppose $X \in$ NP. Let $\mathcal{A}$ be an efficient certifier for X.

   s is a YES-instance for X iff there is a certificate t of size $p(|s|)$ and $\mathcal{A}(s,t) =$ true.

   Convert $\mathcal{A}$ into a circuit for CSAT.

   Idea: Any turing machine can be imitated by a circuit!

CSAT is NP-complete

CSAT: given a circuit C with n input variables, is there an assignment for variables that returns true?

Theorem. CSAT is NP-complete.

1. $\mathrm{CSAT} \in$ NP.
2. $X \leq \mathrm{CSAT}$ for every NP problem X.

   Suppose $X \in$ NP. Let $\mathcal{A}$ be an efficient certifier for X.

   s is a YES-instance for X iff there is a certificate t of size $p(|s|)$ and $\mathcal{A}(s,t) =$ true.

   Convert $\mathcal{A}$ into a circuit for CSAT.

   Fill in input values for s and leave t.

Constructing a Circuit

Lemma. If a turing machine M runs in time $t(n)$, then we can construct a circuit for M of size $O(t^2(n))$.

Constructing a Circuit

Lemma. If a turing machine M runs in time $t(n)$, then we can construct a circuit for M of size $O(t^2(n))$.

- If M terminates in $t(n)$ steps, then it uses at most $t(n)$ space.

Constructing a Circuit

Lemma. If a turing machine M runs in time $t(n)$, then we can construct a circuit for M of size $O(t^2(n))$.

- If M terminates in $t(n)$ steps, then it uses at most $t(n)$ space.

- The contents of a cell in step $i+1$ depends only on $k=3$ cells and the state of M in step i.
  $|Q|=$ number of states, $|\Sigma|=$ number of alphabets
  $\Rightarrow$ simulated by $O(|Q||\Sigma|)^k$ ($= O(1)$ wrt n) gates.

Constructing a Circuit

Lemma. If a turing machine M runs in time $t(n)$, then we can construct a circuit for M of size $O(t^2(n))$.

- If M terminates in $t(n)$ steps, then it uses at most $t(n)$ space.

- The contents of a cell in step $i+1$ depends only on $k=3$ cells and the state of M in step i.
  $|Q| =$ number of states, $|\Sigma| =$ number of alphabets
  $\Rightarrow$ simulated by $O(|Q||\Sigma|)^k$ ($= O(1)$ wrt n) gates.

- There are $t(n) \times t(n)$ cells to compute.
  $\Rightarrow$ a circuit of size $O(t^2(n))$.

Class P and NP

- P is the class of decision problems that can be solved in *Polynomial time*.
- NP is the class of decision problems that can be solved in *Nondeterministic Polynomial time*.
- NP is the class of decision problems that have *efficient* certifiers.

  $\mathcal{A}(s,t)$ is an efficient certifier for a problem if $\mathcal{A} \in$ P and there is a polynomial p s.t. s is a YES-instance iff there is a certificate t satisfying followings:

  1. $|t| \leq p(|s|)$
  2. $\mathcal{A}(s,t) =$ true

NP-hardness and NP-completeness

- A problem A is *NP-hard* if A is at least as hard as any NP problem.

  That is, for *any* NP problem B, $B \leq A$.

- A problem A is *NP-complete* if

  1. A is NP-hard.

  2. $A \in$ NP.

- CSAT is NP-complete.

NP-hardness and NP-completeness

- A problem A is *NP-hard* if A is at least as hard as any NP problem.

  That is, for *any* NP problem B, $B \leq A$.

- A problem A is *NP-complete* if

  1. A is NP-hard.
  2. $A \in$ NP.

- CSAT is NP-complete.
- $\mathrm{CSAT} \leq \mathrm{3SAT} \leq \textsc{VertexCover} \leq \textsc{IndependentSet}$
- There are plenty of *known* NP-complete problems!

$\mathrm{3SAT}$ is NP-complete

$\mathrm{3SAT}$: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

$3\mathrm{SAT}$ is NP-complete

$3\mathrm{SAT}$: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

- x is a binary variable.
- A literal l is either x or $\neg x$.
- A CNF formula $\phi = C_1 \wedge C_2 \wedge \cdots C_k$ where $C_i = l_{i1} \vee l_{i2} \vee l_{i3}$ for literals l_{ij}.

$\mathrm{3SAT}$ is NP-complete

$\mathrm{3SAT}$: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Theorem. $\mathrm{3SAT}$ is NP-complete.

$\mathrm{3SAT}$ is NP-complete

$\mathrm{3SAT}$: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Theorem. $\mathrm{3SAT}$ is NP-complete.

1. $\mathrm{3SAT} \in$ NP.
2. $X \leq \mathrm{3SAT}$ for every NP problem X.

$3\mathrm{SAT}$ is NP-complete

$3\mathrm{SAT}$: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Theorem. $3\mathrm{SAT}$ is NP-complete.

1. $3\mathrm{SAT} \in$ NP.
2. $X \leq 3\mathrm{SAT}$ for every NP problem X.

$3\mathrm{SAT} \leq \mathrm{CSAT}$ (special case)

$\mathrm{3SAT}$ is NP-complete

$\mathrm{3SAT}$: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Theorem. $\mathrm{3SAT}$ is NP-complete.

1. $\mathrm{3SAT} \in$ NP.
2. $X \leq \mathrm{3SAT}$ for every NP problem X.

We know that CSAT is NP-hard.

$\Rightarrow \mathrm{CSAT} \leq \mathrm{3SAT}$ is sufficient.

$\mathrm{3SAT}$ is NP-complete

$\mathrm{3SAT}$: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Theorem. $\mathrm{3SAT}$ is NP-complete.

1. $\mathrm{3SAT} \in$ NP.

2. $X \leq \mathrm{3SAT}$ for every NP problem X.

Let C be an instance of CSAT.

For each gate, make a variable for its output and simulate the gate.

$3\mathrm{SAT}$ is NP-complete

$3\mathrm{SAT}$: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Theorem. $3\mathrm{SAT}$ is NP-complete.

1. $3\mathrm{SAT} \in$ NP.

2. $X \leq 3\mathrm{SAT}$ for every NP problem X.

Let C be an instance of CSAT.

For each gate, make a variable for its output and simulate the gate.

1) NOT gate

Input value: x_i / Output value: $x_j = \neg x_i$

$\Leftrightarrow (x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)$

3SAT is NP-complete

3SAT: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Theorem. 3SAT is NP-complete.

1. 3SAT $\in$ NP.

2. $X \leq$ 3SAT for every NP problem X.

Let C be an instance of CSAT.

For each gate, make a variable for its output and simulate the gate.

2) AND gate

Input value: x_i, x_j / Output value: $x_k = x_i \wedge x_j$

$\Leftrightarrow (\neg x_k \vee x_i) \wedge (\neg x_k \vee x_j) \wedge (x_k \vee \neg x_i \vee \neg x_j)$

3SAT is NP-complete

3SAT: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Theorem. 3SAT is NP-complete.

1. 3SAT $\in$ NP.

2. $X \leq$ 3SAT for every NP problem X.

Let C be an instance of CSAT.

For each gate, make a variable for its output and simulate the gate.

3) OR gate

Input value: x_i, x_j / Output value: $x_k = x_i \vee x_j$

$\Leftrightarrow (x_k \vee \neg x_i) \wedge (x_k \vee \neg x_j) \wedge (\neg x_k \vee x_i \vee x_j)$

3SAT is NP-complete

3SAT: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Theorem. 3SAT is NP-complete.

1. 3SAT $\in$ NP.

2. $X \leq$ 3SAT for every NP problem X.

Let C be an instance of CSAT.

For each gate, make a variable for its output and simulate the gate.

4) To make some input variable x_i true/false, add x_i / $\neg x_i$

3SAT is NP-complete

$\mathrm{3SAT}$: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Theorem. $\mathrm{3SAT}$ is NP-complete.

1. $\mathrm{3SAT} \in$ NP.

2. $X \leq \mathrm{3SAT}$ for every NP problem X.

Let C be an instance of CSAT.

For each gate, make a variable for its output and simulate the gate.

5) Replace clauses with 1 or 2 varibles

$x_i \vee x_j \Leftrightarrow (x_i \vee x_j \vee z) \wedge (x_i \vee x_j \vee \neg z)$

$x_i \Leftrightarrow (x_i \vee z \vee w) \wedge (x_i \vee \neg z \vee w) \wedge (x_i \vee z \vee \neg w) \wedge (x_i \vee \neg z \vee \neg w)$

$\mathrm{3SAT}$ is NP-complete

$\mathrm{3SAT}$: Is a given 3-CNF formula satisfiable?

e.g. $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Theorem. $\mathrm{3SAT}$ is NP-complete.

1. $\mathrm{3SAT} \in$ NP.

2. $X \leq \mathrm{3SAT}$ for every NP problem X.

   Let C be an instance of CSAT.

   For each gate, make a variable for its output and simulate the gate.

   Claim: C is satisfiable iff ϕ_C is satisfiable.

$\textsc{VertexCover}$ is NP-complete

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

$\textsc{VertexCover}$: given a graph G and integer k, decide if G has a vertex cover of size k.

Theorem. $\textsc{VertexCover}$ is NP-complete.

1. $\textsc{VertexCover} \in$ NP.
2. $3\mathrm{SAT} \leq \textsc{VertexCover}$

$\textsc{VertexCover}$ is NP-complete

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

$\textsc{VertexCover}$: given a graph G and integer k, decide if G has a vertex cover of size k.

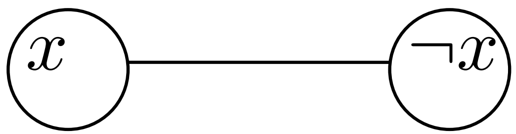
Theorem. $\textsc{VertexCover}$ is NP-complete.

1. $\textsc{VertexCover} \in$ NP.
2. $3\mathrm{SAT} \le \textsc{VertexCover}$

Let ϕ be a 3-CNF with m variables and k clauses.

1) Variable gadget

For each variable x,



VertexCover is NP-complete

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

VertexCover: given a graph G and integer k, decide if G has a vertex cover of size k.

Theorem. VertexCover is NP-complete.

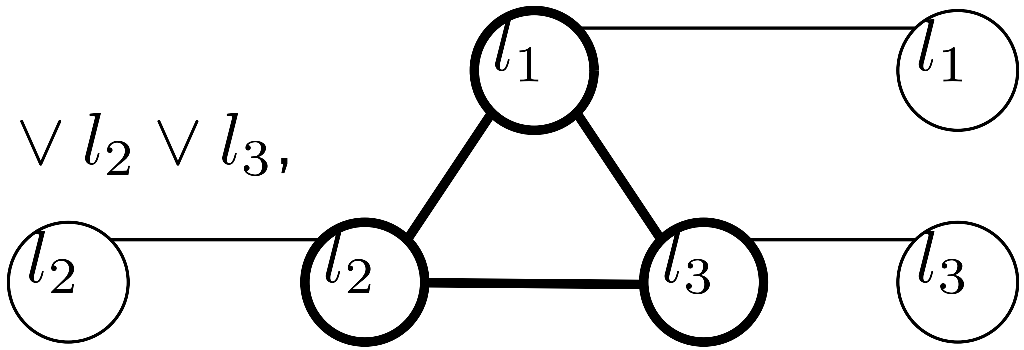
1. VertexCover $\in$ NP.

2. 3SAT $\leq$ VertexCover

Let ϕ be a 3-CNF with m variables and k clauses.

2) Clause gadget

For each clause $C = l_1 \vee l_2 \vee l_3$,



VertexCover is NP-complete

$S \subseteq V(G)$ is a *vertex cover* for a graph G if every edge of G is incident to at least one vertex of S.

VertexCover: given a graph G and integer k, decide if G has a vertex cover of size k.

Theorem. VertexCover is NP-complete.

1. VertexCover $\in$ NP.
2. 3SAT $\leq$ VertexCover

Let ϕ be a 3-CNF with m variables and k clauses.

Claim: ϕ is satisfiable iff G_ϕ has a vertex cover of size $m + 2k$.

SubsetSum is NP-complete

SubsetSum: given a (multi-)set X of integers and an integer s, is there a subset of X whose sum equals to s?

$\textsc{SubsetSum}$ is NP-complete

$\textsc{SubsetSum}$: given a (multi-)set X of integers and an integer s, is there a subset of X whose sum equals to s?

e.g. $X = \{1, 1, 5, 10, 23, 30\}, s = 39$

SubsetSum is NP-complete

SubsetSum: given a (multi-)set X of integers and an integer s, is there a subset of X whose sum equals to s?

e.g. $X = \{1, 1, 5, 10, 23, 30\}, s = 39$ YES! $\{1, 5, 10, 23\}$

$\textsc{SubsetSum}$ is NP-complete

$\textsc{SubsetSum}$: given a (multi-)set X of integers and an integer s, is there a subset of X whose sum equals to s?

Theorem. $\textsc{SubsetSum}$ is NP-complete.

1. $\textsc{SubsetSum} \in \mathsf{NP}$
2. $3\text{SAT} \leq \textsc{SubsetSum}$

SubsetSum is NP-complete

SubsetSum: given a (multi-)set X of integers and an integer s, is there a subset of X whose sum equals to s?

Theorem. SubsetSum is NP-complete.

1. SubsetSum $\in$ NP
2. 3SAT $\leq$ SubsetSum

Certificate: a set of integers Y

Certifer: 1) Y is a subset of X and 2) sum of Y equals to s

SubsetSum is NP-complete

SubsetSum: given a (multi-)set X of integers and an integer s, is there a subset of X whose sum equals to s?

Theorem. SubsetSum is NP-complete.

1. SubsetSum $\in$ NP.
2. $3\mathrm{SAT} \leq$ SubsetSum

Let ϕ be a 3-CNF with m variables and k clauses.

Construct integers t_i, f_i of $m+k$ digits for each variable x_i

Construct integers a_j, b_j of $m+k$ digits for each clause C_j

m digits correspond to T/F assignment for each variable.

t_i, f_i have 1 for i-th digit.

SubsetSum is NP-complete

SubsetSum: given a (multi-)set X of integers and an integer s, is there a subset of X whose sum equals to s?

Theorem. SubsetSum is NP-complete.

1. SubsetSum $\in$ NP.

2. 3SAT $\leq$ SubsetSum

   Let ϕ be a 3-CNF with m variables and k clauses.

   Construct integers t_i, f_i of $m+k$ digits for each variable x_i

   Construct integers a_j, b_j of $m+k$ digits for each clause C_j

   k digits correspond to satisfiability for each clause.

   t_i/f_i has 1 for $(m+j)$-th digit if $x_i/\neg x_i$ appear in C_j.

   $a_j = b_j$ have 1 for $(m+j)$-th digit.

$\textsc{SubsetSum}$ is NP-complete

$\textsc{SubsetSum}$: given a (multi-)set X of integers and an integer s, is there a subset of X whose sum equals to s?

Theorem. $\textsc{SubsetSum}$ is NP-complete.

1. $\textsc{SubsetSum} \in$ NP.
2. $3\mathrm{SAT} \leq \textsc{SubsetSum}$

Let ϕ be a 3-CNF with m variables and k clauses.

Claim: ϕ is satisfiable iff there is a subset of $\{t_1, f_1, \cdots, t_m, f_m, a_1, b_1, \cdots, a_k, b_k\}$ of sum:

$$\underbrace{11\cdots1}_{m}\underbrace{33\cdots3}_{k}$$

DNF-SAT

DNF-SAT: Is a given DNF formula satisfiable?

e.g. $\phi = (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge \neg x_2) \vee \neg x_3$

DNF-SAT

DNF-SAT: Is a given DNF formula satisfiable?

e.g. $\phi = (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge \neg x_2) \vee \neg x_3$

- x is a binary variable.
- A literal l is either x or $\neg x$.
- A DNF formula $\phi = C_1 \vee C_2 \vee \cdots C_k$ where $C_i = l_{i1} \wedge l_{i2} \cdots \wedge l_{ik_i}$ for literals l_{ij}.

DNF-SAT

DNF-SAT: Is a given DNF formula satisfiable?

e.g. $\phi = (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge \neg x_2) \vee \neg x_3$

- DNF-SAT $\in$ P.

DNF-SAT

DNF-SAT: Is a given DNF formula satisfiable?

e.g. $\phi = (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge \neg x_2) \vee \neg x_3$

- DNF-SAT $\in$ P.
- 3SAT $\leq$ DNF-SAT

DNF-SAT

DNF-SAT: Is a given DNF formula satisfiable?

e.g. $\phi = (x_1 \land x_2 \land x_3) \lor (\neg x_1 \land \neg x_2) \lor \neg x_3$

- DNF-SAT $\in$ P.
- 3SAT $\leq$ DNF-SAT

  Let ϕ be a 3-CNF with m variables and k clauses.

  We can convert CNF into DNF as follows:

  $(a \lor b \lor c) \land (d \lor e \lor f)$
  $= (a \land d) \lor (a \land e) \lor (a \land f) \lor (b \land d) \lor (b \land e) \lor (b \land f) \lor$
  $(c \land d) \lor (c \land e) \lor (c \land f)$

DNF-SAT

DNF-SAT: Is a given DNF formula satisfiable?

e.g. $\phi = (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge \neg x_2) \vee \neg x_3$

- DNF-SAT $\in$ P.
- 3SAT $\leq$ DNF-SAT

  Let ϕ be a 3-CNF with m variables and k clauses.

  We can convert CNF into DNF as follows:

  $$(a \vee b \vee c) \wedge (d \vee e \vee f)$$
  $$= (a \wedge d) \vee (a \wedge e) \vee (a \wedge f) \vee (b \wedge d) \vee (b \wedge e) \vee (b \wedge f) \vee$$
  $$(c \wedge d) \vee (c \wedge e) \vee (c \wedge f)$$

NOT a polynomial-time reduction!