Computation as search

\[
\text{search}(\text{Node}) :- \text{goal}(\text{Node}).
\]

\[
\text{search}(\text{Node}) :- \text{arc}(\text{Node}, \text{Next}), \text{search}(\text{Next}).
\]
Computation as search

$$\text{search(Node)} : \text{goal(Node)}.$$

$$\text{search(Node)} : \text{arc(Node,Next)}, \text{search(Next)}.$$

More than one \text{Next} may satisfy \text{arc(Node,Next)}

$\leadsto$ non-determinism
Computation as search

\[ \text{search(Node)} :\!-\! \text{goal(Node)}. \]
\[ \text{search(Node)} :\!-\! \text{arc(Node,Next)}, \text{search(Next)}. \]

More than one Next may satisfy \( \text{arc(Node,Next)} \)
\[ \rightsquigarrow \text{non-determinism} \]

Choose Next closest to goal (heuristic: best-first),
keeping track of costs (min cost, \( \text{A}^* \))
Computation as search

```prolog
search(Node) :- goal(Node).
search(Node) :- arc(Node, Next), search(Next).
```

More than one `Next` may satisfy `arc(Node, Next)`

\[\leadsto\] non-determinism

Choose `Next` closest to goal (heuristic: best-first),
keeping track of costs (min cost, A*)

Available choices depend on `arc`

- actions specified by Turing machine (graph)
Computation as search

\[
\text{search}(\text{Node}) \leftarrow \text{goal}(\text{Node}).
\]

\[
\text{search}(\text{Node}) \leftarrow \text{arc}(\text{Node}, \text{Next}), \text{search}(\text{Next}).
\]

More than one \text{Next} may satisfy \text{arc}(\text{Node}, \text{Next})

\[\leadsto\] non-determinism

Choose \text{Next} closest to goal (heuristic: best-first),
keeping track of costs (min cost, \text{A}^*)

Available choices depend on \text{arc}
- actions specified by Turing machine (graph)

Computation eliminates non-determinism (determinization)
Computation as search

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

More than one `Next` may satisfy `arc(Node,Next)`

\[ \leadsto \text{non-determinism} \]

Choose `Next` closest to goal (heuristic: best-first),
keeping track of costs (min cost, A*)

Available choices depend on `arc`
- actions specified by Turing machine (graph)

Computation eliminates non-determinism (determinization)

Bound number of calls to `arc` (iterations of `search`)
Cobham’s Thesis

A problem is feasibly solvable iff some deterministic Turing machine (dTM) solves it in polynomial time.

\[ P = \{ \text{problems a dTM solves in polynomial time} \} \]
Cobham’s Thesis
A problem is feasibly solvable iff some deterministic Turing machine (dTm) solves it in polynomial time.

\[ P = \{ \text{problems a dTm solves in polynomial time} \} \]

\[ NP = \{ \text{problems a non-deterministic Tm solves in polynomial time} \} \]
Cobham’s Thesis
A problem is feasibly solvable iff some deterministic Turing machine (dTm) solves it in polynomial time.

\[ P = \{ \text{problems a dTm solves in polynomial time} \} \]

\[ NP = \{ \text{problems a non-deterministic Tm solves in polynomial time} \} \]

Clearly, \( P \subseteq NP \).
Cobham’s Thesis

A problem is feasibly solvable iff some deterministic Turing machine (dTm) solves it in polynomial time.

\[ P = \{ \text{problems a dTm solves in polynomial time} \} \]

\[ NP = \{ \text{problems a non-deterministic Tm solves in polynomial time} \} \]

Clearly, \( P \subseteq NP \).

Whether \( P = NP \) is the most celebrated open mathematical problem in computer science.
Cobham’s Thesis

A problem is feasibly solvable iff some deterministic Turing machine (dTm) solves it in polynomial time.

\[P = \{\text{problems a dTm solves in polynomial time}\}\]

\[NP = \{\text{problems a non-deterministic Tm solves in polynomial time}\}\]

Clearly, \(P \subseteq NP\).

Whether \(P = NP\) is the most celebrated open mathematical problem in computer science.

\(P \neq NP\) would mean non-determinism wrecks feasibility.
Cobham’s Thesis
*A problem is feasibly solvable iff some deterministic Turing machine (dTm) solves it in polynomial time.*

\[ P = \{ \text{problems a dTm solves in polynomial time} \} \]

\[ NP = \{ \text{problems a non-deterministic Tm solves in polynomial time} \} \]

Clearly, \( P \subseteq NP \).

Whether \( P = NP \) is the most celebrated open mathematical problem in computer science.

\( P \neq NP \) would mean non-determinism wrecks feasibility.

\( P = NP \) says non-determinism makes no difference to feasibility.
A closer look

Given a set $L$ of strings, and a Tm $M$. 

TIME($n^k$) := \{ $L$ | some $d$Tm solves $L$ in time $n^k$ \}

P := \bigcup_{k \geq 1} \text{TIME}(n^k)

NTIME($n^k$) := \{ $L$ | some $n$Tm solves $L$ in time $n^k$ \}

NP := \bigcup_{k \geq 1} \text{NTIME}(n^k)
A closer look

Given a set $L$ of strings, and a Tm $M$.

$M$ solves in $L$ in time $n^k$ if there is a fixed integer $c > 0$ such that for every string $s$ of size $n$,

$$s \in L \iff M \text{ accepts } s \text{ within } c \cdot n^k \text{ steps.}$$
A closer look

Given a set $L$ of strings, and a Tm $M$.

$M$ solves in $L$ in time $n^k$ if there is a fixed integer $c > 0$ such that for every string $s$ of size $n$,

$$s \in L \iff M \text{ accepts } s \text{ within } c \cdot n^k \text{ steps.}$$

$$\text{TIME}(n^k) := \{ L \mid \text{some dTm solves } L \text{ in time } n^k \}$$

e.g. TIME($n$) includes every regular language
A closer look

Given a set $L$ of strings, and a Tm $M$.

$M$ solves in $L$ in time $n^k$ if there is a fixed integer $c > 0$ such that for every string $s$ of size $n$,

$$s \in L \iff M \text{ accepts } s \text{ within } c \cdot n^k \text{ steps.}$$

$$\text{TIME}(n^k) := \{ L \mid \text{some dTm solves } L \text{ in time } n^k \}$$

e.g. $\text{TIME}(n)$ includes every regular language

$$P := \bigcup_{k \geq 1} \text{TIME}(n^k)$$
A closer look

Given a set $L$ of strings, and a Tm $M$.

$M$ solves in $L$ in time $n^k$ if there is a fixed integer $c > 0$ such that for every string $s$ of size $n$,

$$s \in L \iff M \text{ accepts } s \text{ within } c \cdot n^k \text{ steps.}$$

$$\text{TIME}(n^k) := \{ L \mid \text{some dTm solves } L \text{ in time } n^k \}$$

e.g. TIME($n$) includes every regular language

$$P := \bigcup_{k \geq 1} \text{TIME}(n^k)$$

$$\text{NTIME}(n^k) := \{ L \mid \text{some nTm solves } L \text{ in time } n^k \}$$

$$NP := \bigcup_{k \geq 1} \text{NTIME}(n^k)$$
Boolean satisfiability (SAT)

**SAT.** Given a Boolean expression $\varphi$ with variables $x_1, \ldots, x_n$, can we make $\varphi$ true by assigning true/false to $x_1, \ldots, x_n$?

e.g., $(x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_3)$
SAT. Given a Boolean expression $\varphi$ with variables $x_1, \ldots, x_n$, can we make $\varphi$ true by assigning true/false to $x_1, \ldots, x_n$?

Checking that a particular assignment makes $\varphi$ true is easy ($P$).

e.g., $(x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor \overline{x_3})$
**SAT**. Given a Boolean expression $\varphi$ with variables $x_1, \ldots, x_n$, can we make $\varphi$ true by assigning true/false to $x_1, \ldots, x_n$?

Checking that a particular assignment makes $\varphi$ true is easy ($P$). Non-determinism (guessing the assignment) puts SAT in $NP$.

\[ \text{e.g., } (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor \overline{x_3}) \]
SAT. Given a Boolean expression $\varphi$ with variables $x_1, \ldots, x_n$, can we make $\varphi$ true by assigning true/false to $x_1, \ldots, x_n$?

Checking that a particular assignment makes $\varphi$ true is easy ($P$). Non-determinism (guessing the assignment) puts SAT in $NP$. But is SAT in $P$? There are $2^n$ assignments to try.

e.g., $(x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_3)$
SAT. Given a Boolean expression $\varphi$ with variables $x_1, \ldots, x_n$, can we make $\varphi$ true by assigning true/false to $x_1, \ldots, x_n$?

Checking that a particular assignment makes $\varphi$ true is easy ($P$). Non-determinism (guessing the assignment) puts SAT in $NP$. But is SAT in $P$? There are $2^n$ assignments to try.

**Cook-Levin Theorem.** SAT is in $P$ iff $P = NP$.

e.g., $(x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_3)$
**Boolean satisfiability (SAT)**

**SAT.** Given a Boolean expression $\varphi$ with variables $x_1, \ldots, x_n$, can we make $\varphi$ true by assigning true/false to $x_1, \ldots, x_n$?

Checking that a particular assignment makes $\varphi$ true is easy ($P$). Non-determinism (guessing the assignment) puts SAT in $NP$. But is SAT in $P$? There are $2^n$ assignments to try.

**Cook-Levin Theorem.** SAT is in $P$ iff $P = NP$.

\[ (x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor x_3) \]

**CSAT:** $\varphi$ is a conjunction of clauses, where a *clause* is an OR of literals, and a *literal* is a variable $x_i$ or negated variable $\overline{x}_i$.
**SAT.** Given a Boolean expression $\varphi$ with variables $x_1, \ldots, x_n$, can we make $\varphi$ true by assigning true/false to $x_1, \ldots, x_n$?

Checking that a particular assignment makes $\varphi$ true is easy ($P$). Non-determinism (guessing the assignment) puts SAT in $NP$. But is SAT in $P$? There are $2^n$ assignments to try.

**Cook-Levin Theorem.** $SAT$ is in $P$ iff $P = NP$.

$$e.g., \left(x_1 \lor \overline{x_2} \lor x_3\right) \land \left(\overline{x_1} \lor \overline{x_3}\right)$$

**CSAT:** $\varphi$ is a conjunction of clauses, where a *clause* is an OR of literals, and a *literal* is a variable $x_i$ or negated variable $\overline{x_i}$

$k$-SAT: every clause has exactly $k$ literals

3-SAT is as hard as SAT, 2-SAT is in $P$
SAT. Given a Boolean expression \( \varphi \) with variables \( x_1, \ldots, x_n \), can we make \( \varphi \) true by assigning true/false to \( x_1, \ldots, x_n \)?

Checking that a particular assignment makes \( \varphi \) true is easy (\( P \)). Non-determinism (guessing the assignment) puts SAT in \( NP \). But is SAT in \( P \)? There are \( 2^n \) assignments to try.

Cook-Levin Theorem. SAT is in \( P \) iff \( P = NP \).

\[ \text{e.g., } (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor \overline{x_3}) \]

CSAT: \( \varphi \) is a conjunction of clauses, where a clause is an OR of literals, and a literal is a variable \( x_i \) or negated variable \( \overline{x_i} \)

\( k \)-SAT: every clause has exactly \( k \) literals

3-SAT is as hard as SAT, 2-SAT is in \( P \)

Horn-SAT: every clause has at most one positive literal — linear