Computation as search

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

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

search(Node) :- goal(Node).

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

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

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

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

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

search(Node) :- goal(Node).

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

More than one Next may satisfy arc(Node, Next) \implies\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 as search

search(Node) :- goal(Node).

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

More than one 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, \( A^* \))

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

Computation eliminates non-determinism (determinization)
Computation as search

\begin{verbatim}
search(Node) :- goal(Node).
search(Node) :- arc(Node,Next), search(Next).

More than one Next may satisfy arc(Node,Next) \implies \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)
\end{verbatim}
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, \( P \subseteq NP \).
Feasibility and non-determinism: P vs 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.
Feasibility and non-determinism: P vs 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.

\( P \neq NP \) 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 $M$. 

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

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

$\text{NP} := \bigcup_{k \geq 1} \text{NTIME}(n^k)$
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 $M$ accepts $s$ within $c \cdot n^k$ 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. 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 $n$,

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

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

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

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

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

$NP := \bigcup_{k \geq 1} \text{NTIME}(n^k)$
**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 $NP$.

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.

\[ (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_3) \]
**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 $P = NP$.*

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.

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

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

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

$$e.g., \ (x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor \overline{x}_3)$$

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

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

3-SAT is as hard as SAT, 2-SAT is in $P$
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 \( P = NP \).*

\[
e.g., (x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor \overline{x}_3)
\]

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

\( k \)-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