Back to computation as search

\[
\text{search(Node) :- goal(Node).}
\]
\[
\text{search(Node) :- arc(Node,Next), search(Next).}
\]
Back to computation as search

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

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

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

\[\Rightarrow\] non-determinism
Back to 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 \textit{Next} may satisfy \text{arc}(\text{Node}, \text{Next})

\[\Rightarrow\] non-determinism

Choose \textit{Next} closest to goal (heuristic \(h, Q\))

perhaps keeping track of costs (min cost, \(A^*\))
Back to 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 \( h, Q \))
perhaps keeping track of costs (min cost, A*)

Available choices depend on \text{arc}
- actions specified by Turing machine (graph)
Back to computation as search

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

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

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

Choose `Next` closest to goal (heuristic \( h, Q \))

\[ \text{perhaps keeping track of costs (min cost, A*)} \]

Available choices depend on `arc`

- actions specified by Turing machine (graph)

Computation eliminates non-determinism (determinization)
Back to computation as search

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

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

Choose Next closest to goal (heuristic \( h, Q \))
perhaps 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)
Terminating search

Search times out after too many ticks

\[\text{bSearch}(\text{Node}, \_ ) \leftarrow \text{goal}(\text{Node}).\]
\[\text{bSearch}(\text{Node}, \text{Bound}) \leftarrow \text{arc}(\text{Node}, \text{Next}), \]
\[\text{tick}(\text{Bound}, \text{Less}),\]
\[\text{bSearch}(\text{Next}, \text{Less}).\]
Terminating search

Search times out after too many ticks

\[
\text{bSearch(Node,\_)} :- \text{goal(Node)}.
\]
\[
\text{bSearch(Node,Bound)} :- \text{arc(Node,Next)},
\quad \text{tick(Bound,Less)},
\quad \text{bSearch(Next,Less)}.
\]

Design tick to be deterministic

\[y = y' \text{ whenever } \text{tick}(x,y) \text{ and } \text{tick}(x,y')\]

and well-founded: there is no infinite sequence \(x_1, x_2, \ldots\) s.t.

\[\text{tick}(x_i, x_{i+1}) \text{ for every integer } i > 0.\]
Terminating search

Search times out after too many ticks

\[
\text{bSearch}(\text{Node}, _) \leftarrow \text{goal}(\text{Node}).
\]

\[
\text{bSearch}(\text{Node}, \text{Bound}) \leftarrow \text{arc}(\text{Node}, \text{Next}), \\
\quad \text{tick}(\text{Bound}, \text{Less}), \\
\quad \text{bSearch}(\text{Next}, \text{Less}).
\]

Design tick to be deterministic

\[
y = y' \text{ whenever } \text{tick}(x, y) \text{ and } \text{tick}(x, y')
\]

and well-founded: there is no infinite sequence \(x_1, x_2, \ldots\) s.t.

\[
\text{tick}(x_i, x_{i+1}) \text{ for every integer } i > 0.
\]

\[
\text{accept}(\text{String}) \leftarrow \text{node}(\text{String}, \text{Node}), \\
\quad \text{bound}(\text{String}, \text{Bound}), \\
\quad \text{bSearch}(\text{Node}, \text{Bound}).
\]
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.

\[ P = \{ \text{problems a dTm solves in polynomial time} \} \]
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.
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, \( P \subseteq NP \).
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.
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.

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

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

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 $M$. 
A closer look

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

$M$ solves $L$ 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 $L$ 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 $L$ 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 $L$ 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 $x_1, \ldots, x_n$?

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

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 $\overline{x_i}$.

3-SAT is as hard as SAT, 2-SAT is in P.

Horn-SAT: every clause has at most one positive literal — linear.
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 ($P$).

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

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

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

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

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

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 \( \overline{x_i} \)
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 \( P = NP \).

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

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

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
Prolog and SAT

Prolog KB (definite clauses)

\[ x_1 :\, x_2, x_4. \]
\[ x_2 :\, x_3. \]
\[ x_4. \]

\[ \iff \]
\[ [[[x_1, x_2, x_4], [x_2, x_3], [x_4]]] \]

The assignment making all variables TRUE satisfies all CSAT-inputs in which every clause has a positive literal.

(All definite clause KBs are satisfiable.)

From proofs to unsatisfiability:

\[ \text{KB proves } \phi \iff \text{KB}, \phi \text{ is not satisfiable} \]
Prolog and SAT

Prolog KB (definite clauses)

\[
\begin{align*}
x_1 & : \neg x_2, x_4. \\
x_2 & : \neg x_3. \quad \sim \implies \quad [[x_1, x_2, x_4], [x_2, x_3], [x_4]] \\
x_4. 
\end{align*}
\]

CSAT-input

\[
\begin{align*}
x_1 \lor \neg x_2 \lor \neg x_4 \\
x_2 \lor \neg x_3 \quad \sim \implies \quad [[1, -2, -4], [2, -3], [4]] \\
x_4. 
\end{align*}
\]
Prolog and SAT

Prolog KB (definite clauses)

\[
\begin{align*}
x_1 & : - x_2, x_4. \\
x_2 & : - x_3. & \Rightarrow & [[x_1, x_2, x_4], [x_2, x_3], [x_4]] \\
x_4. &
\end{align*}
\]

CSAT-input

\[
\begin{align*}
x_1 \lor \overline{x_2} \lor \overline{x_4} \\
x_2 \lor \overline{x_3} & \Rightarrow [[1, -2, -4], [2, -3], [4]] \\
x_4. &
\end{align*}
\]

The assignment making all variables TRUE satisfies all CSAT-inputs in which every clause has a positive literal. (All definite clause KBs are satisfiable.)
Prolog and SAT

Prolog KB (definite clauses)
\[
\begin{align*}
  x_1 & : - x_2, x_4. \\
  x_2 & : - x_3. \quad \leadsto \quad [[x_1, x_2, x_4], [x_2, x_3], [x_4]] \\
  x_4.
\end{align*}
\]

CSAT-input
\[
\begin{align*}
  x_1 \lor \overline{x_2} \lor \overline{x_4} \\
  x_2 \lor \overline{x_3} \quad \leadsto \quad [[1, -2, -4], [2, -3], [4]] \\
  x_4.
\end{align*}
\]

The assignment making all variables TRUE satisfies all CSAT-inputs in which every clause has a positive literal. (All definite clause KBs are satisfiable.)

From proofs to unsatisfiability:

\[
\begin{align*}
\text{Prolog Horn (linear SAT)} & \quad \iff \quad KB \text{ proves } \varphi \\
& \quad \iff \quad KB, \overline{\varphi} \text{ is not satisfiable}
\end{align*}
\]