SAT Modelling
Idea: Problem Transformation

- Planning Problem
- TransformerPlanner
- SAT problemXYZ ProblemPlan
- XYZSAT Solver

Invariants
\( \forall \)-step
\( \exists \)-step
Definition (SAT)
Given a propositional formula $\mathcal{F}$, decide whether $\mathcal{F}$ has a satisfying valuation.

Definition (CNF-SAT)
Given a propositional formula $\mathcal{F}$ in conjunctive normal form, decide whether $\mathcal{F}$ has a satisfying valuation.

A valuation is an assignment of decision variables to $\{\top, \bot\}$.

CNF:

$$\mathcal{F} = \bigwedge_{C \in \mathcal{C}} \bigvee_{\ell \in C} \ell$$

($\mathcal{C}$ is the set of clauses; $C$ is a clause, a set of literals.)
SAT Solvers

- SAT solvers are programs that determine whether a satisfying valuation exists and if so output it.
- A lot of research in recent years (annual competitions since 2002).
- Usable OSes have minisat in their package manager.
- Standardised input format DIMACS:

```
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
≡
```

CNF with 5 vars and 3 clauses:

\[(\neg v_1 \lor \neg v_5 \lor v_4) \land
(\neg v_1 \lor v_5 \lor v_3 \lor v_4) \land
(\neg v_3 \lor \neg v_4)\]
Colouring

Definition

Given a graph \( G = (V, E) \) and a number \( k \).
Is there an assignment of \( k \) colours to the vertices of \( G \), such that all adjacent vertices have different colours?
Colouring

Variables for choosing the colour of each node

\[ \text{colour}_v^i \text{ where } v \in V \text{ and } i \in \{1, \ldots, k\} \]

If a node has a colour, all adjacent nodes have a different colour

\[ \text{colour}_v^i \rightarrow \neg \text{colour}_w^i \quad \forall (v,w) \in E \]
\[ \neg \text{colour}_v^i \lor \neg \text{colour}_w^i \quad \forall (v,w) \in E \]

Every node has a colour

\[ \bigvee_{i=1}^{k} \text{colour}_v^i \quad \forall v \in V \]

Every node has at most one colour

\[ \bigwedge_{i=1}^{k} \left[ \text{colour}_v^i \rightarrow \bigwedge_{j=1, j \neq i}^{k} \neg \text{colour}_v^j \right] \quad \forall v \in V \]
Theoretical Background
Definition (PLANEx)

Given a planning problem \( \mathcal{P} \).
Is there a solution \( \pi \) of \( \mathcal{P} \).

Theorem (Bylander’94)

\text{PLANEx is PSPACE-complete.}

Theorem (Bylander’94)

\text{PLANEx with bounded plan length } k \text{ is PSPACE-complete.}

\text{PSPACE with NP calculus?}
Bounded plan length assumes binary encoding of $k$.

What if we assume $k$ in unary encoding?

PLANEx “becomes” $\text{NP}$-“complete”.

For full PLANEx: how to choose the plan length?
- Theoretical limit: $2^{|V|}$.
- Practical limit: usually smaller (sometimes polynomially bounded).

Start with a small $k$ and increase until a solution is found.
Bound Iteration

Planning Problem → Transformer $k = 1 \rightarrow k = 2 \rightarrow k = 3 \rightarrow \ldots \rightarrow k = 2 | V |

SAT problem → SAT Solver

Solution

$\emptyset$ Unsolvable
Sequential Classical Planning in SAT
A plan is just a sequence of state transitions.
- “Mechanics” is identical in all timesteps.
- Just model one timestep and copy’n’paste.
- Edge constraints!
We only need two types of decision variables!

1. \(a_i^t\) – Action \(i\) is executed at time \(t\).
2. \(v_i^t\) – State variable \(i\) is true at time \(t\).
Overall Formula

Constraints to check:

- Correctly applying actions at each time step (\(\tau\)).
- \(s_I\) and \(g\) must be respected.

\[
\mathcal{F} = \bigwedge_{t=0}^{k-1} \tau(t) \wedge \bigwedge_{v_i \in s_I} v_i^0 \wedge \bigwedge_{v_i \in V \setminus s_I} \neg v_i^0 \wedge \bigwedge_{v_i \in g} v_i^k
\]

Here: \(k = 9\)
Classical Planning via SAT

$s \xrightarrow{a} s'$

Constraints to check by $\tau(t)$:

$F_1$ Preconditions must hold (in $s$).

$F_2$ Effects must occur (in $s'$).

$F_3$ Unaffected state variables stay unchanged.

$F_4$ At most one action per timestep.

$F_5$ At least one action per timestep. Necessary? No.
Classical Planning via SAT

- Preconditions must hold:

\[ F_1 = \bigwedge_{a \in A} a^{t+1} \rightarrow \bigwedge_{v \in \text{pre}(a)} v^t \]

- Effects must occur:

\[
F_2 = \left[ \bigwedge_{a \in A} a^{t+1} \rightarrow \bigwedge_{v \in \text{add}(a)} v^{t+1} \right] \wedge \left[ \bigwedge_{a \in A} a^{t+1} \rightarrow \bigwedge_{v \in \text{del}(a)} \neg v^{t+1} \right]
\]
Variables not affected by the executed action must stay the same.

→ Frame Problem!

\[
F_3 = \bigwedge_{v \in V} \left[ \left( \neg v^t \land v^{t+1} \right) \rightarrow \bigvee_{a \in A \text{ with } v \in \text{add}(a)} a^{t+1} \right] \land \\
\bigwedge_{v \in V} \left[ \left( v^t \land \neg v^{t+1} \right) \rightarrow \bigvee_{a \in A \text{ with } v \in \text{del}(a)} a^{t+1} \right]
\]

Only one action at a time:

\[
F_4 = \text{at-most-one}(\{a^t \mid a \in A\})
\]
At-most-one

Given a set of decision variables \( X = \{x_1, \ldots, x_{|X|}\} \). Find a set of clauses that, if satisfied, will ensure that at most one \( x \in X \) is true.

Naive encoding:

\[
\bigwedge_{x_1 \in X} \bigwedge_{x_2 \in X \setminus \{x_1\}} \neg x_1 \lor \neg x_2
\]

\[
(x_1 \Rightarrow \neg x_2) \land (x_2 \Rightarrow \neg x_1)
\]
At-most-one

Idea: Introduce new variables!

\( f_i \) – from index \( i \) on all \( x_i \) will be false

i.e. it is forbidden to use any \( x_i \) after \( i \)

Sequential encoding:

\[
\begin{align*}
|X| - 1 & \quad \bigwedge_{i=1}^{\infty} \neg x_i \lor f_i \\
& \quad \text{(} x_i \Rightarrow f_i \text{)} \\
|X| - 1 & \quad \bigwedge_{i=2}^{\infty} \neg f_{i-1} \lor f_i \\
& \quad \text{(} f_{i-1} \Rightarrow f_i \text{)} \\
|X| & \quad \bigwedge_{i=1}^{\infty} \neg x_i \lor \neg f_{i-1} \\
& \quad (x_i \Rightarrow \neg f_{i-1}) \land \land (f_{i-1} \Rightarrow \neg x_i)
\end{align*}
\]
At-most-one

Maybe this is a bit much ...

\( n_i \) – bit \( i \) (0-index) of a \( \lceil \log(|X|) \rceil \)-digit binary number if one

Binary encoding:

\[
\neg x_i \lor n_j \quad \text{if} \quad \frac{i}{2j} \quad \text{mod} \ 2 = 1
\]

\[
\neg x_i \lor \neg n_j \quad \text{if} \quad \frac{i}{2j} \quad \text{mod} \ 2 = 0
\]
Different AMO Implementations\textsuperscript{1}

<table>
<thead>
<tr>
<th>encoding</th>
<th>#clauses</th>
<th>#new variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>binomial</td>
<td>$n^2$</td>
<td>0</td>
</tr>
<tr>
<td>binary</td>
<td>$n \log n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>sequential</td>
<td>$3n$</td>
<td>$n$</td>
</tr>
<tr>
<td>commander</td>
<td>$\frac{7}{2}n$</td>
<td>$\frac{n}{2}$</td>
</tr>
<tr>
<td>product</td>
<td>$2(n + n^{\frac{1}{m+1}})$</td>
<td>$2n^{\frac{1}{2}}$</td>
</tr>
</tbody>
</table>

where $n$ is the number of atoms, i.e., $|X|$}

\textsuperscript{1}Frisch and Giannaros; SAT Encodings of the At-Most-k Constraint – Some Old, Some New, Some Fast, Some Slow; 2010
Bound Iteration

Planning Problem → Transformer → SAT problem → SAT Solver → Solution

Unsolvable
There are a lot of improvements to this formula.

- Invariants.
- \( \forall \)-step semantics.
- \( \exists \)-step semantics.
Invariants
What are Invariants?

Is there anything we know about states in a planning problem?

**Definition (Invariant)**

An invariant $I$ is a formula over the state variables such that for all states $s$ reachable from $s_I$ it holds $s \models I$. 
What are Invariants?

Predicates:

- $\text{on}(x, y)$ – $x$ lies directly on $y$.
- $\text{free}(x)$ – $x$ has no block above it.

Actions:

- $\text{pickup}(x)$ – pick up $x$, if it is free.
- $\text{putdown}(x, y)$ – put $x$ on $y$, if $y$ is free ($table$ is always free).

Are the following formulae invariants?

1. $\forall b \in \text{Block} : (\exists b' \in \text{Block} : \text{on}(b', b)) \lor \text{free}(b)$ — Yes.
2. $\forall b \in \text{Block} : \text{on}(b, table)$ — No.
3. $\forall b, b' \in \text{Block} : \neg \text{on}(b', b) \lor \neg \text{on}(b, b')$ — Yes.
Invariants are Difficult

How hard is verifying an invariant?
As hard as planning.
Also there are too many invariants.

- Compute an approximation of all invariants of a fixed form.
- Restrict to binary-or invariants:

$$\ell_1 \lor \ell_2$$
Computing Invariants [Rintanen’98]

Note: Here we consider some action \( a = (pre, add, del) \) and denote with \( \text{eff} = add(a) \cup \{ \neg v \mid v \in del(a) \} \) its effects (as a literal set).

\[
\neg V = \{ \neg v \mid v \in V \} \quad (\ell \in V^{-V} \text{ denotes a literal.})
\]

\[U_{\langle pre, eff \rangle}(\mathcal{I})\] gives all properties (positive or negative state variables) that hold after the execution of an action \( a = \langle pre, eff \rangle \)

\[
\equiv (\{ \neg v \mid v \in add \} \cup del)
\]

\[U_{\langle pre, eff \rangle}(\mathcal{I}) = (\{ \ell \in V \cup \neg V \mid \mathcal{I} \cup pre \models \ell \} \setminus \{ \neg \ell \mid \ell \in \text{eff} \}) \cup \text{eff}
\]

\[F_{\langle pre, eff \rangle}(\mathcal{I})\] is a filter for invariants, returning those that hold after the execution of an action \( a = \langle pre, eff \rangle \)

\[
F_{\langle pre, eff \rangle}(\mathcal{I}) = \begin{cases} 
\mathcal{I} & \text{if } \mathcal{I} \cup \text{pre} \models \bot \text{ and otherwise:} \\
\{ \ell_1 \lor \ell_2 \in \mathcal{I} \mid (\neg \ell_1 \notin \text{eff} \text{ or } \ell_2 \in U_{\langle pre, eff \rangle}(\mathcal{I})) \text{ and } \neg \ell_2 \notin \text{eff} \text{ or } \ell_1 \in U_{\langle pre, eff \rangle}(\mathcal{I})) \}
\end{cases}
\]
Computing Invariants [Rintanen’98], cont’d

Call $R_A(I) := F_{a_1}(F_{a_2}(\cdots F_{a_n}(I)\cdots))$ with initial invariant $I_{init} = \{v \lor \ell \mid v \in s, \ell \in V \cup \neg V\} \cup \{\neg v \lor \ell \mid v \not\in s, \ell \in V \cup \neg V\}$ and arbitrary linearization of action set $A, a_1, \ldots, a_n$, until $I$ does not change anymore.

$R$ stands for “reduce invariant set”.
How to Use Invariants

What to do with an invariant $\ell_1 \lor \ell_2$?

Add it to every timestep $t$ as $\ell_1^t \lor \ell_2^t$. 
∀-step
Linear Plans are Bad!

Consider the following (single) planning problem:

\[
\text{drive}(A, B), \text{load}(B), \text{drive}(B, C), \text{unload}(C), \text{drive}(F, D), \text{load}(D), \text{drive}(D, E), \text{unload}(E)
\]

\[
\text{drive}(A, B) \quad \text{load}(B) \quad \text{drive}(B, C) \quad \text{unload}(C) \\
\text{drive}(F, D) \quad \text{load}(D) \quad \text{drive}(D, E) \quad \text{unload}(E)
\]
Allow parallel execution of actions.
But when?

- Let $\mathcal{A}$ be some set of actions.
- Parallel execution of $\mathcal{A}$ is safe, if all ($\forall$) linearisations of $\mathcal{A}$ are executable.

**Necessary conditions:**
- All actions are executable in the previous state as all could be the first.
- No action can have a delete-effect that is a precondition of another action, i.e., $\forall a_1 \neq a_2 \in \mathcal{A} : \text{del}(a_1) \cap \text{prec}(a_2) = \emptyset$, as $a_1$ can occur before $a_2$.

**Sufficient conditions:** Necessary conditions are already sufficient.
Encoding ∀-step

Remove the at-most-one constraints. Two options:

\[ a_1^t \rightarrow \neg a_2^t \quad \forall a_1, a_2 \in A \text{ with } \text{del}(a_1) \cap \text{pre}(a_2) \neq \emptyset \]
\[ \rightarrow \text{quadratic effort.} \]

\[ a^t \rightarrow \text{del}_v^t \quad \forall a \in A, v \in \text{del}(a) \]
\[ \text{del}_v^t \rightarrow \neg a^t \quad \forall a \in A, v \in \text{pre}(a) \]
\[ \rightarrow \text{linear effort.} \]

Further implications?

The resulting state must always be the same!
Thus we forbid two actions \( a_1, a_2 \) with \( \text{del}(a_1) \cap \text{add}(a_2) \neq \emptyset \) to be executed in parallel.
(Otherwise the resulting state would not be unique.)
∃-step
Parallel Plans are (Still) Bad!

(Re-)Consider the following (single) planning problem:

\[
\begin{align*}
\text{drive}(A, B) & \quad \text{load}(B) & \quad \text{drive}(B, C) & \quad \text{unload}(C) \\
\text{drive}(F, D) & \quad \text{load}(D) & \quad \text{drive}(D, E) & \quad \text{unload}(E)
\end{align*}
\]

\[
\begin{align*}
\text{drive}(A, B) & \quad \text{unload}(C) \\
\text{drive}(B, C) \\
\text{drive}(F, D) & \quad \text{load}(D) & \quad \text{unload}(E) \\
\text{drive}(D, E)
\end{align*}
\]
What Kind of Parallelism do we Look for?

- Absolutely safe parallelism.
  - All linearisations will always be executable and lead to the same state.
  - \( \forall \)-step.

- (Sometimes) Safe parallelism.
  - At least one linearisation is executable and all executable linearisations lead to the same state.
  - \( \exists \)-step.
Given a set of actions $\mathcal{A}$. We call them $\exists$-step executable if a linearisation exists that is executable and all executable linearisations lead to the same state.

How difficult to determine? First part is $\text{NP}$-complete.

How to encode?

Results in the Kautz&Selman encoding ...
Approximate $\exists$-step semantics.

- Analyse dependency between actions.
- Similar to $\forall$-step:
  - If $\text{del}(a) \cap \text{pre}(a') \neq \emptyset$, execute $a'$ before $a$.
  - Ignore if $I \cup \text{pre}(a) \cup \text{pre}(a')$ is inconsistent.
\(\exists\)-step [Rintanen, Heljanko, Niemelä’06]

- Disabling Graph: \(a \rightarrow b\) iff after executing \(a\) it may not be possible to execute \(b\).
- We can safely execute actions in reverse topological order.
- DG may not be acyclic.
- Guess an order in every SCC and order SCCs in reverse topological order.
- If executed in parallel, we will always execute actions in \textbf{this} order.
What do we have to assert inside the propositional formula?

- Parallel actions must result in a consistent state. ✓
- Parallel actions must be executable.

1. Actions must be applicable in the previous state.
2. Reverse topological order of DG ensures that later actions are still applicable.
3. In SCCs there might be edges opposite to the chosen order.
4. SCC can be treated separately.
5. If $a_2$ is executed, then $a_4$ must not.
6. Enforced via *chaines*.

$a_5, a_2, a_3, a_4, a_1$

$(a_5), (a_2, a_3, a_4), (a_1)$
Chains

We are given an SCC and an ordering of its vertices.

\[ \pi = (a_5, a_4, a_3, a_2, a_1) \]

- We want choose an acyclic subsequence of \( \pi \).
- Approx.: Do not choose both ends of a forward edge.
- Iterate over causes of these edges: \( v \in del(a_1) \cap pre(a_2) \)
  - \( E_v \) – subsequence of \( \pi \) with \( v \in del(a) \) (Erasing)
  - \( R_v \) – subsequence of \( \pi \) with \( v \in pre(a) \) (Requiring)

\[
\bigwedge \{ \pi^i \rightarrow f^j_v \mid i < j, \pi^i \in E_v, \pi^j \in R_v, \{a_{i+1}, \ldots, a_{j-1}\} \cap R_v = \emptyset \} \cup \\
\{ f^i_v \rightarrow f^j_v \mid i < j, \{\pi^i, \pi^j\} \in R_v, \{a_{i+1}, \ldots, a_{j-1}\} \cap R_v = \emptyset \} \cup \\
\{ f^i_v \rightarrow \neg \pi^i \mid \pi^i \in R_v \}
\]
Further Improvements

Improvments for classical planning:

- Extension to conditional effects [Rintanen, Heljanko, Niemelä’06].
- Relaxed $\exists$-step [Wehrle & Rintanen’07].
- Parallel SAT search [Rintanen’04] [Rintanen, Heljanko, Niemelä’06].
- Specialised heuristics for SAT solvers [Rintanen’10a] [Rintanen’10b].
- Improved memory management [Rintanen’12].
- Incremental SAT-solving [Gocht & Balyo’17].

Extensions to non-classical planning:

- LTL [Mattmüller & Rintanen’07] [Behnke & Biundo’18].
- Partial Observability [Pandey & Rintanen’18].
- Temporal Planning [Rintanen’17].
- HTN Planning [Behnke, Höller, Biundo’17’18].

→ https://users.aalto.fi/~rintanj1/satplan.html