SAT Modelling

Idea: Problem Transformation

SAT

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:

\[
p\text{ cnf } 5 \ 3
\]

\[
1 \ -5 \ 4 \ 0 \quad \equiv \quad (v_1 \lor \neg v_5 \lor v_4) \land \\
-1 \ 5 \ 3 \ 4 \ 0 \quad (\neg v_1 \lor v_5 \lor v_3 \lor v_4) \land \\
-3 \ -4 \ 0 \quad (\neg v_3 \lor \neg v_4)
\]

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?

![Graph with nodes and edges]

Variables for choosing the colour of each node

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

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

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

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

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
Computational Complexity

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

Theorem (Bylander’94)
**PlanEx is PSPACE-complete.**

Theorem (Bylander’94)
**PlanEx with bounded plan length \(k\) is PSPACE-complete.**

PSPACE with NP calculus?

Transformation Idea

- Bounded plan length assumes binary encoding of \(k\).
- What if we assume \(k\) in unary encoding?
- PlanEx “becomes” 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

Sequential Classical Planning in SAT
Classical Planning via SAT

[Kautz & Selman ’92]

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!

February 5th, 2020 B. Nebel, R. Mattmüller, G. Behnke – AI Planning 13 / 44

Decision Variables

We only need two types of decision variables!
- \( a^t_i \) – Action \( i \) is executed at time \( t \).
- \( v^t_i \) – State variable \( i \) is true at time \( t \).

February 5th, 2020 B. Nebel, R. Mattmüller, G. Behnke – AI Planning 14 / 44

Overall Formula

Constraints to check:
- Correctly applying actions at each time step (\( \tau(t) \)).
- \( s^t \) and \( g \) must be respected.

\[
\mathcal{F} = \bigwedge_{t=0}^{k-1} \tau(t) \land \bigwedge_{v_i \in s^t} v^0_i \land \bigwedge_{v_i \in g} \neg v^0_i \land \bigwedge_{v_i \in g} v^t_i \quad \text{here: } k = 9
\]

February 5th, 2020 B. Nebel, R. Mattmüller, G. Behnke – AI Planning 15 / 44

Classical Planning via SAT

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.

February 5th, 2020 B. Nebel, R. Mattmüller, G. Behnke – AI Planning 16 / 44
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:
  \[
  F_2 = \left[ \bigwedge_{a \in A} a^{t+1} \rightarrow \bigwedge_{v \in \text{add}(a)} v^{t+1} \right] \land \left[ \bigwedge_{a \in A} a^{t+1} \rightarrow \bigwedge_{v \in \text{del}(a)} \neg v^{t+1} \right]
  \]

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

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:
\[
\bigwedge_{x_1 \in X, x_2 \in X \setminus \{x_1\}} \neg x_1 \lor \neg x_2 \land (x_1 \Rightarrow \neg x_2) \land (x_2 \Rightarrow \neg x_1)
\]

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:
\[
\bigwedge_{i=1}^{|X|-1} \neg x_i \lor f_i \\
\bigwedge_{i=2}^{|X|-1} \neg f_{i-1} \lor f_i \\
\bigwedge_{i=1}^{|X|} \neg x_i \lor \neg f_{i-1} \land (x_i \Rightarrow \neg f_{i-1}) \land (f_{i-1} \Rightarrow \neg x_i)
\]
At-most-one

Maybe this is a bit much ...

\[ n_i - \text{bit } i \ (0\text{-index}) \text{ of a } \lceil \log(|X|) \rceil \text{-digit binary number if one} \]

Binary encoding:

\[-x_i \lor n_j \quad \text{if } \left\lfloor \frac{i}{2^j} \right\rfloor \text{ mod } 2 = 1 \]

\[-x_i \lor \neg n_j \quad \text{if } \left\lfloor \frac{i}{2^j} \right\rfloor \text{ mod } 2 = 0 \]

Different AMO Implementations

<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{1}{2}n )</td>
<td>( \frac{n}{2} )</td>
</tr>
<tr>
<td>product</td>
<td>( 2(n + n^{\frac{m+1}{2}}) )</td>
<td>( 2n^{\frac{1}{2}} )</td>
</tr>
</tbody>
</table>

where \( n \) is the number of atoms, i.e., \(|X|\)

\(^1\)Frisch and Giannaros; SAT Encodings of the At-Most-k Constraint – Some Old, Some New, Some Fast, Some Slow; 2010

Classical Planning via SAT

There are a lot of improvements to this formula.

- Invariants.
- \( \forall \)-step semantics.
- \( \exists \)-step semantics.
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. \( \forall b \in \text{Block} : (\exists b' \in \text{Block} : \text{on}(b', b)) \lor \text{free}(b) \) – \textbf{Yes}.
2. \( \forall b \in \text{Block} : \text{on}(b, \text{table}) \) – \textbf{No}.
3. \( \forall b, b' \in \text{Block} : \neg \text{on}(b', b) \lor \neg \text{on}(b, b') \) – \textbf{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 $eff = add(a) \cup \{v | v \in del(a)\}$ its effects (as a literal set).

$V = \{\neg v \mid v \in V\}$ ($\ell \in V^{-V}$ denotes a literal.)

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

$I = \{\ell \in \text{V} \cup \neg \text{V} \mid I \cup \text{pre} \models \ell\}\setminus \{\neg \ell \mid \ell \in \text{eff}\} \cup \text{eff}$

$F_{\langle \text{pre}, \text{eff} \rangle}(G)$ is a filter for invariants, returning those that hold after the execution of an action $a = \langle \text{pre}, \text{eff} \rangle$

$R_{\langle A, \ldots, A_n \rangle} := F_{\langle a_1 \rangle} \cdots F_{\langle a_n \rangle}$

Call $R_{\langle A, \ldots, A_n \rangle} := F_{\langle a_1 \rangle} \cdots F_{\langle a_n \rangle} (G)$ with initial invariant $I_\text{init} = \{v \lor \ell | v \in s_1, \ell \in \text{V} \lor \neg \text{V}\} \cup \{\neg v \lor \ell | v \notin s_1, \ell \in \text{V} \lor \neg \text{V}\}$ and arbitrary linearization of action set $A, a_1, \ldots, a_n$, until $G$ 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:

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

Let \( A \) be some set of actions. Parallel execution of \( A \) is safe, if all (\( \forall \)) linearisations of \( A \) 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., \( \forall a_1 \neq a_2 \in A : \text{del}(a_1) \cap \text{prec}(a_2) = \emptyset \), as \( a_1 \) can occur before \( a_2 \).

Sufficient conditions: Necessary conditions are already sufficient.

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.)

Encoding \( \forall \)-step

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

- \( a_1' \rightarrow \neg a_2' \) \quad \forall a_1, a_2 \in A \text{ with } \text{del}(a_1) \cap \text{pre}(a_2) \neq \emptyset \quad \rightarrow \text{quadratic effort.} \)
- \( a' \rightarrow \text{del}_v \) \quad \forall a \in A, v \in \text{del}(a) \quad \rightarrow \text{linear effort.} \)
- \( \text{del}_v \rightarrow \neg a' \) \quad \forall a \in A, v \in \text{pre}(a) \quad \rightarrow \text{linear effort.} \)

\( \exists \)-step

Allow parallel execution of actions.
But when?

- Let \( A \) be some set of actions.
- Parallel execution of \( A \) is safe, if all (\( \forall \)) linearisations of \( A \) 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., \( \forall a_1 \neq a_2 \in A : \text{del}(a_1) \cap \text{prec}(a_2) = \emptyset \), as \( a_1 \) can occur before \( a_2 \).

Sufficient conditions: Necessary conditions are already sufficient.
Parallel Plans are (Still) Bad!

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

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

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.

\( \exists \)-step Parallelism

- 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...

Disabling Graph

[Rintanen, Heljanko, Niemelä’06]

- 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 \( \mathcal{A} \cup \text{pre}(a) \cup \text{pre}(a') \) is inconsistent.
**∃-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 this order.

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

**Chains**

- We are given an SCC and an ordering of its vertices.
- We want choose an acyclic subsequence of \( \pi \).
- Approx.: Do not choose both ends of a forward edge.
- Iterate over causes of these edges: \( v \in \text{del}(a) \cap \text{pre}(a_2) \)
  - \( E_v \rightarrow \) subsequence of \( \pi \) with \( v \in \text{del}(a) \) (Erasing)
  - \( R_v \rightarrow \) subsequence of \( \pi \) with \( v \in \text{pre}(a) \) (Requiring)

\[
\bigwedge \{ \pi' \rightarrow f'_i \mid i < j, \pi' \in E_v, \pi' \in R_v, \{a_{i+1}, \ldots, a_{j-1}\} \cap R_v = \emptyset \} \cup \\
\{ f'_j \rightarrow f'_i \mid i < j, \{\pi', \pi''\} \in R_v, \{a_{i+1}, \ldots, a_{j-1}\} \cap R_v = \emptyset \} \cup \\
\{ f'_i \rightarrow \neg \pi' \mid \pi' \in R_v \}
\]

**Further Improvements**

**Improvements for classical planning:**
- Extension to conditional effects [Rintanen, Heljanko, Niemelä’06].
- Relaxed ∃-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 [Goche & 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