SAT planning

Relations in propositional logic
Actions in propositional logic
Plans in propositional logic
DPLL
Example

Parallel plans
Parallelism
Interference
Parallel actions
Translation
Optimality
Example

Final remarks

Planning in the propositional logic

- Early work on deductive planning viewed plans as proofs that lead to a desired goal (theorem).
- Planning as satisfiability testing was proposed in 1992.
  1. A propositional formula represents all length $n$ action sequences from the initial state to a goal state.
  2. If the formula is satisfiable then a plan of length $n$ exists (and can be extracted from the satisfying valuation).
- Heuristic search and satisfiability planning are currently the best approaches for planning.
  - Satisfiability planning is often more efficient for small, but difficult problems.
  - Heuristic search is often more efficient for big, but easy problems.
- Bounded model-checking in Computer Aided Verification was introduced in 1998 as an extension of satisfiability planning after the success of the latter had been noticed outside the AI community.

1. Represent actions (= binary relations) as propositional formulae.
2. Construct a formula saying “execute one of the actions”.
3. Construct a formula saying “execute a sequence of $n$ actions, starting from the initial state, ending in a goal state”.
4. Test the satisfiability of this formula by a satisfiability algorithm.
5. If the formula is satisfiable, construct a plan from a satisfying valuation.
Satisfiability testing vs. state-space search

- Like our earlier algorithms (progression and regression planning, possibly with heuristics), planning as satisfiability testing can be interpreted as a search algorithm.
- However, unlike these algorithms, satisfiability testing is undirected search:
  - As the first decision, the algorithm may decide to include a certain action as the 7th operator of the plan.
  - As the second decision, it may require a certain state variable to be true after the 5th operator of the plan.
  - ... 

Relations/actions as formulae

Formulae on $A \cup A'$ as binary relations

Let $A = \{a_1, \ldots, a_n\}$ represent state variables in the current state, and $A' = \{a'_1, \ldots, a'_n\}$ state variables in the successor state.

Formulae $\phi$ on $A \cup A'$ represent binary relations on states: a valuation of $A \cup A' \rightarrow \{0, 1\}$ represents a pair of states $s : A \rightarrow \{0, 1\}, s' : A' \rightarrow \{0, 1\}$.

Example

Formula $(a \rightarrow a') \land ((a' \lor b) \rightarrow b')$ on $a, b, a', b'$ represents the binary relation $\{(00, 00), (00, 01), (00, 11), (01, 01), (01, 11), (10, 11), (11, 11)\}$.

Sets (of states) as formulae

Reminder: Formulae on $A$ as sets of states

We view formulae $\phi$ as representing sets of states $s : A \rightarrow \{0, 1\}$.

Example

Formula $a \lor b$ on the state variables $a, b, c$ represents the set $\{010, 011, 100, 101, 110, 111\}$.

Matrices as formulae

Example (Formulae as relations as matrices)

<table>
<thead>
<tr>
<th>$A$</th>
<th>$A'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>00</td>
</tr>
</tbody>
</table>

Representation of big matrices is possible

For $n$ state variables, a formula (over $2n$ variables) represents an adjacency matrix of size $2^n \times 2^n$.

For $n = 20$, matrix size is $2^{20} \times 2^{20} \sim 10^6 \times 10^6$. 

<table>
<thead>
<tr>
<th>$a'$</th>
<th>$b'$</th>
<th>$a' b'$</th>
<th>$a' b'$</th>
<th>$a' b'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### Actions/relations as propositional formulae

**Example**

\[
\phi = (a_1 \leftrightarrow \neg a'_1) \land (a_2 \leftrightarrow \neg a'_2)
\]

as a matrix:

<table>
<thead>
<tr>
<th>(a_1)</th>
<th>(a_2)</th>
<th>(a'_1)</th>
<th>(a'_2)</th>
<th>(\phi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
<td>0 0 1 0</td>
<td>0 0 1 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 1 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

and as a conventional truth table:

<table>
<thead>
<tr>
<th>(a_1)</th>
<th>(a_2)</th>
<th>(a'_1)</th>
<th>(a'_2)</th>
<th>(\phi)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
<td>0 0 1 0</td>
<td>0 0 1 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 1 0</td>
<td>1 0 0 0</td>
<td>1 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 1</td>
<td>1 0 0 1</td>
<td>1 0 0 1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 1 0</td>
<td>1 0 0 0</td>
<td>1 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

This action rotates the value of the state variables \(a_1, a_2, a_3\) one step forward.

### Translating operators into formulae

**Definition (operators in propositional logic)**

Let \(o = (c, e)\) be an operator and \(A\) a set of state variables. Define \(\tau_A(o)\) as the conjunction of

\[
\begin{align*}
\forall a \in A \: (\neg EPC_a(e) \land (a \land \neg EPC_a(e)) \Rightarrow a') \quad (1) \\
\forall a \in A \: (\neg (EPC_a(e) \land EPC_{\neg a}(e))) \quad (2)
\end{align*}
\]

Condition (1) states that the precondition of \(o\) is satisfied. Condition (2) states that the new value of \(a\), represented by \(a'\), is 1 if the old value was 1 and it did not become 0, or if it became 1. Condition (3) states that none of the state variables is assigned both 0 and 1. Together with (1), this encodes applicability of the operator.
Translating operators into formulae

Example

Let the state variables be $A = \{a, b, c\}$.
Consider the operator $\langle a \lor b, (b \triangleright a) \land (c \triangleright \neg a) \land (a \triangleright b) \rangle$.
The corresponding propositional formula is

\[
(a \lor b) \land ((b \lor (a \land \neg c)) \leftrightarrow a')
\land ((a \lor (b \land \bot)) \leftrightarrow b')
\land ((\bot \lor (c \land \bot)) \leftrightarrow c')
\land (\neg (b \land c) \land (a \land \bot) \land (\bot \land \bot))
\equiv (a \lor b) \land ((b \lor (a \land \neg c)) \leftrightarrow a')
\land ((a \lor b) \leftrightarrow b')
\land (c \leftrightarrow c')
\land (\neg (b \land c))
\]

Correctness

Lemma
Let $s$ and $s'$ be states and $o$ an operator. Let $v : A \cup A' \rightarrow \{0, 1\}$ be a valuation such that

1. for all $a \in A$, $v(a) = s(a)$, and
2. for all $a' \in A$, $v(a') = s'(a)$.

Then $v \models \tau_A(o)$ if and only if $s' = \text{app}_o(s)$.

Planning as satisfiability

1. Encode operator sequences of length 0, 1, 2, ... as formulae $\Phi^\text{seq}_0$, $\Phi^\text{seq}_1$, $\Phi^\text{seq}_2$, ... (see next slide).
2. Test satisfiability of $\Phi^\text{seq}_0$, $\Phi^\text{seq}_1$, $\Phi^\text{seq}_2$, ..., .
3. If a satisfying valuation $v$ is found, a plan can be constructed from $v$. 
Planning as satisfiability

Definition (transition relation in propositional logic)
For \( \langle A, I, O, G \rangle \) define \( R_1(A, A') = \bigvee_{o \in O} r_A(o) \).

Definition (bounded-length plans in propositional logic)
Existence of plans of length \( t \) is represented by the following formula over propositions \( A^0 \cup \cdots \cup A^t \), where \( A^i = \{ a^i | a \in A \} \) for all \( i \in \{ 0, \ldots, t \} \):
\[
\Phi^\text{seq}_i = \ell^0 \land R_1(A^0, A^1) \land R_1(A^1, A^2) \land \cdots \land R_1(A^{t-1}, A^t) \land G^t
\]
where \( \ell^0 = \bigwedge_{a \in A, I(a) = 1} a^0 \land \bigwedge_{a \in A, I(a) = 0} \neg a^0 \)
and \( G^t \) is \( G \) with propositions \( a \) replaced by \( a^t \).

Example
Consider
\[
I \models b \land c
\]
\[
G = (b \land \neg c) \lor (\neg b \land c)
\]
\[
o_1 = (\top, c \land \neg c) \land (\neg c \land c)
\]
\[
o_2 = (\top, (b \land \neg b) \land (\neg b \land b))
\]
The formula \( \Phi^\text{seq}_3 \) for plans of length 3 is:
\[
(b^0 \land c^0)
\land (((b^1 \leftrightarrow b^2) \land (c^0 \leftrightarrow \neg c^1)) \lor ((b^0 \leftrightarrow \neg b^1) \land (c^0 \leftrightarrow c^1)))
\land (((b^1 \leftrightarrow b^2) \land (c^1 \leftrightarrow \neg c^2)) \lor ((b^1 \leftrightarrow \neg b^2) \land (c^1 \leftrightarrow c^2)))
\land (((b^2 \leftrightarrow b^3) \land (c^2 \leftrightarrow \neg c^3)) \lor ((b^2 \leftrightarrow \neg b^3) \land (c^2 \leftrightarrow c^3)))
\land ((b^3 \land \neg c^3) \lor (\neg b^3 \land c^3)).
\]

Theorem
Let \( \Phi^\text{seq}_i \) be the formula for \( \langle A, I, O, G \rangle \) and plan length \( t \).
The formula \( \Phi^\text{seq}_i \) is satisfiable if and only if there is a sequence of states \( s_0, \ldots, s_t \) and operators \( o_2, \ldots, o_t \) such that \( s_0 = I, s_i = \text{app}_{o_i}(s_{i-1}) \) for all \( i \in \{ 1, \ldots, t \} \), and \( s_t \models G \).

Consequence
If \( \Phi^\text{seq}_0, \Phi^\text{seq}_1, \ldots, \Phi^\text{seq}_{t-1} \) are unsatisfiable and \( \Phi^\text{seq}_t \) is satisfiable, then the length of shortest plans is \( i \).
Satisfiability planning with \( \Phi^\text{seq}_i \) yields optimal plans, like heuristic search with admissible heuristics and optimal algorithms like \( A^* \) or \( \text{IDA}^* \).

Plan extraction
All satisfiability algorithms give a valuation \( v \) that satisfies \( \Phi^\text{seq}_i \) upon finding out that \( \Phi^\text{seq}_i \) is satisfiable.
This makes it possible to construct a plan.

Constructing a plan from a satisfying valuation
Let \( v \) be a valuation so that \( v \models \Phi^\text{seq}_i \). Then define \( s_i(a) = v(a^i) \) for all \( a \in A \) and \( i \in \{ 0, \ldots, t \} \).
The \( i \)-th operator in the plan is \( o \in O \) if \( \text{app}_o(s_{i-1}) = s_i \). Note: There may be more than one such operator, in which case any of them may be chosen.
Planning as satisfiability

Example, continued

Example
One valuation that satisfies $\Phi_3^{seq}$:

<table>
<thead>
<tr>
<th>time $i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b^i$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$c^i$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Note:
1. There also exists a plan of length 1.
2. No plan of length 2 exists.

Conjunctive normal form

Many satisfiability algorithms require formulas in the conjunctive normal form: transformation by repeated applications of the following equivalences.

- $\neg(\phi \lor \psi) \equiv \neg\phi \land \neg\psi$
- $\neg(\phi \land \psi) \equiv \neg\phi \lor \neg\psi$
- $\neg\neg\phi \equiv \phi$
- $\phi \lor (\psi_1 \land \psi_2) \equiv (\phi \lor \psi_1) \land (\phi \lor \psi_2)$

The formula is a conjunction of clauses (disjunctions of literals).

Example

$(A \lor \neg B \lor C) \land (\neg C \lor \neg B) \land A$

Note: Transformation to conjunctive normal form can increase formula size exponentially. There are also polynomial translations which introduce additional variables.

The unit resolution rule

Unit resolution
From $l_1 \lor l_2 \lor \cdots \lor l_n$ (here $n \geq 1$) and $\overline{l_i}$, infer $l_2 \lor \cdots \lor l_n$.

Example
From $a \lor b \lor c$ and $\neg a$ infer $b \lor c$.

Unit resolution: a special case
From $A$ and $\neg A$ we get the empty clause $\bot$ ("disjunction consisting of zero disjuncts").

Unit subsumption
The clause $l_1 \lor l_2 \lor \cdots \lor l_n$ can be eliminated if we have the unit clause $l_1$.

The Davis-Putnam-Logemann-Loveland procedure

- The first efficient decision procedure for any logic (Davis, Putnam, Logemann & Loveland, 1960/62).
- Based on binary search through the valuations of a formula.
- Unit resolution and unit subsumption help pruning the search tree.
- The currently most efficient satisfiability algorithms are variants of the DPLL procedure.
  (Although there is currently a shift toward viewing these procedures as performing more general reasoning: clause learning.)
Satisfiability test by the DPLL procedure

Davis-Putnam-Logemann-Loveland Procedure

def DPLL(C: clauses):
    while there are clauses \( h \lor \cdots \lor l_n \) \( \in C \) and \( \overline{h} \in C \):
        \( C := (C \setminus \{ h \lor \cdots \lor l_n \}) \cup \{ \overline{l} \lor \cdots \lor \overline{l_n} \} \)
    while there are clauses \( h \lor \cdots \lor l_n \) \( \in C \) \( (n \geq 2) \) and \( l_i \in C \):
        \( C := C \setminus \{ h \lor \cdots \lor l_n \} \)
    if \( \bot \in C \):
        return false
    if \( C \) contains only unit clauses:
        return true
    Pick some variable \( a \) such that \( a \notin C \) and \( \neg a \notin C \).
    return DPLL(\( C \cup \{ a \} \)) or DPLL(\( C \cup \{ \neg a \} \))

Planning as satisfiability

Example: plan search with DPLL

Consider the problem from a previous slide, with two operators each inverting the value of one state variable, for plan length 3.

\[
(b^0 \land c^0) \\
\land (((b^0 \leftrightarrow b^1) \land (c^0 \leftrightarrow \neg c^1)) \lor ((b^0 \leftrightarrow \neg b^1) \land (c^0 \leftrightarrow c^1))) \\
\land (((b^1 \leftrightarrow b^2) \land (c^1 \leftrightarrow \neg c^2)) \lor ((b^1 \leftrightarrow \neg b^2) \land (c^1 \leftrightarrow c^2))) \\
\land (((b^2 \leftrightarrow b^3) \land (c^2 \leftrightarrow \neg c^3)) \lor ((b^2 \leftrightarrow \neg b^3) \land (c^2 \leftrightarrow c^3))) \\
\land ((b^3 \land \neg c^3) \lor (\neg b^3 \land c^3)).
\]

To obtain a short CNF formula, we introduce auxiliary variables \( a_i \) and \( o_j \) for \( i \in \{1, 2, 3\} \) denoting operator applications.

\[
\begin{align*}
  b^0 & \rightarrow ((b^0 \leftrightarrow b^1) \land (c^0 \leftrightarrow \neg c^1)) \\
  c^0 & \rightarrow ((b^0 \leftrightarrow \neg b^1) \land (c^0 \leftrightarrow c^1)) \\
  o_1^1 \lor o_2^1 & \rightarrow ((b^1 \leftrightarrow b^2) \land (c^1 \leftrightarrow \neg c^2)) \\
  o_2^2 \lor o_3^2 & \rightarrow ((b^1 \leftrightarrow \neg b^2) \land (c^1 \leftrightarrow c^2)) \\
  o_3^3 \lor o_2^3 & \rightarrow (b^2 \leftrightarrow b^3) \land (c^2 \leftrightarrow \neg c^3) \\
  (b^3 \land \neg c^3) \lor (\neg b^3 \land c^3) & \rightarrow ((b^2 \leftrightarrow \neg b^3) \land (c^2 \leftrightarrow c^3))
\end{align*}
\]
Planning as satisfiability
Example: plan search with DPLL

Eliminate implications with \((\{l_1 \land l_2\} \rightarrow h_1) \equiv (\neg l_1 \lor l_2 \lor h_1)\).

\[
\begin{array}{c}
\text{Valuation constructed by the DPLL procedure} \\
\hline
b' & 0 & 1 & 2 & 3 \\
\hline
c' & 1 & 2 & 3 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Valuation constructed by the DPLL procedure} \\
\hline
b' & 1 \\
\hline
c' & 1 \\
\end{array}
\]

Planning as satisfiability
Example: plan search with DPLL

Identify unit clauses.

\[
\begin{array}{c}
\begin{aligned}
b^0 &\rightarrow -b^0 \lor b^1 \\
\neg c^0 &\rightarrow -c^0 \lor c^1 \\
b^1 &\rightarrow -b^0 \lor -b^1 \\
c^1 &\rightarrow -c^0 \lor -c^1 \\
b^2 &\rightarrow -b^1 \lor -b^2 \\
\neg c^2 &\rightarrow -c^1 \lor -c^2 \\
b^3 &\rightarrow -b^2 \lor -b^3 \\
\neg c^3 &\rightarrow -c^2 \lor -c^3 \\
\end{aligned}
\end{array}
\]

\[
\begin{array}{c}
\begin{aligned}
b^0 &\rightarrow -b^0 \lor b^1 \\
\neg c^0 &\rightarrow -c^0 \lor c^1 \\
b^1 &\rightarrow -b^0 \lor -b^1 \\
c^1 &\rightarrow -c^0 \lor -c^1 \\
b^2 &\rightarrow -b^1 \lor -b^2 \\
\neg c^2 &\rightarrow -c^1 \lor -c^2 \\
b^3 &\rightarrow -b^2 \lor -b^3 \\
\neg c^3 &\rightarrow -c^2 \lor -c^3 \\
\end{aligned}
\end{array}
\]

Valuation constructed by the DPLL procedure

\[
\begin{array}{c}
\begin{aligned}
b' &\rightarrow 0 \\
c' &\rightarrow 1 \\
\end{aligned}
\end{array}
\]

Planning as satisfiability
Example: plan search with DPLL

Perform unit subsumption with \(b^0\) and \(c^0\).

\[
\begin{array}{c}
\begin{aligned}
b^0 &\rightarrow -b^0 \lor b^1 \\
\neg c^0 &\rightarrow -c^0 \lor c^1 \\
b^1 &\rightarrow -b^0 \lor -b^1 \\
c^1 &\rightarrow -c^0 \lor -c^1 \\
b^2 &\rightarrow -b^1 \lor -b^2 \\
\neg c^2 &\rightarrow -c^1 \lor -c^2 \\
b^3 &\rightarrow -b^2 \lor -b^3 \\
\neg c^3 &\rightarrow -c^2 \lor -c^3 \\
\end{aligned}
\end{array}
\]

Valuation constructed by the DPLL procedure

\[
\begin{array}{c}
\begin{aligned}
b' &\rightarrow 1 \\
c' &\rightarrow 1 \\
\end{aligned}
\end{array}
\]
Planning as satisfiability

Example: plan search with DPLL

No unhandled unit clauses exist. Must branch.

\[-o_1 \lor b_1\]
\[-o_2 \lor \neg b_1 \lor b_2\]
\[-o_3 \lor \neg b_1 \lor b_3\]
\[-o_1 \lor \neg c_1\]
\[-o_2 \lor \neg c_1 \lor c_2\]
\[-o_3 \lor \neg c_1 \lor c_3\]
\[-o_2 \lor \neg b_1\]
\[-o_3 \lor \neg b_1 \lor b_2\]
\[-o_3 \lor \neg b_1 \lor b_3\]
\[-c_1 \lor \neg b_2\]
\[-c_1 \lor \neg c_2 \lor c_3\]
\[-c_1 \lor \neg c_2 \lor c_3\]

Valuation constructed by the DPLL procedure

\[
\begin{array}{c|ccc}
{b'} & 0 & 1 & 2 & 3 \\
\hline
\ o_1 & 1 & 2 & 3 \\
\ o_2 & 1 & 1 & 1 \\
\ c' & 1 & 1 & 1 \\
\end{array}
\]

Perform unit resolution and unit subsumption with \(b_1\).

\[-o_1 \lor b_1\]
\[-o_2 \lor \neg b_1 \lor b_2\]
\[-o_3 \lor \neg b_1 \lor b_3\]
\[-o_1 \lor \neg c_1\]
\[-o_2 \lor \neg c_1 \lor c_2\]
\[-o_3 \lor \neg c_1 \lor c_3\]
\[-o_2 \lor \neg b_1\]
\[-o_3 \lor \neg b_1 \lor b_2\]
\[-o_3 \lor \neg b_1 \lor b_3\]
\[-c_1 \lor \neg b_2\]
\[-c_1 \lor \neg c_2 \lor c_3\]
\[-c_1 \lor \neg c_2 \lor c_3\]

Valuation constructed by the DPLL procedure

\[
\begin{array}{c|ccc}
{b'} & 0 & 1 & 2 & 3 \\
\hline
\ o_1 & 1 & 2 & 3 \\
\ o_2 & 1 & 1 & 1 \\
\ c' & 1 & 1 & 1 \\
\end{array}
\]
Planning as satisfiability
Example: plan search with DPLL

Perform unit resolution and unit subsumption with $o_1$.

$\neg o_1 \lor b^2$
$\neg o_2 \lor c^1$
$\neg o_3 \lor o_4$
$\neg o_5 \lor o_6$
$b^1 \lor c^3$
$c^2 \lor \neg b^3$
$\neg c^3 \lor \neg b^3$

Valuation constructed by the DPLL procedure

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b'$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$c'$</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

M. Helmert, B. Nebel (Universität Freiburg)  
AI Planning  
November 24th, 2006  
38 / 74
Planning as satisfiability

Example: plan search with DPLL

Perform unit resolution and unit subsumption with $c^3$.

\[-o_1^1 \lor b^2 \lor -o_1^1 \lor -b^2 \lor b^3 \lor -o_1^1 \lor -b^2 \lor -b^3 \lor -o_1^1 \lor -c^2 \lor -c^3\]

\[-o_1^2 \lor o_2^1 \lor o_3^1 \lor -b^3 \lor o_4^1 \lor b^2 \lor -b^2 \lor -b^3 \lor -o_1^1 \lor -c^2 \lor -c^3\]

Valuation constructed by the DPLL procedure

\[
\begin{array}{ccc|ccc}
0 & 1 & 2 & 3 & 1 & 2 & 3 \\
\hline
b' & 1 & 1 & 1 & 0 & 1 & 0 \\
c' & 1 & 0 & 1 & 0 & 1 & 0 \\
\end{array}
\]

M. Helmer, B. Nebel (Universität Freiburg)  AI Planning November 24th, 2006 41 / 74

Planning as satisfiability

Example: plan search with DPLL

No unhandled unit clauses exist. Must branch a third time.

\[-o_1^1 \lor b^2 \lor -o_1^1 \lor -b^2 \lor -o_1^1 \lor -c^2 \lor -c^3\]

\[-o_1^2 \lor o_2^1 \lor o_3^1 \lor -b^3 \lor o_4^1 \lor b^2 \lor -b^2 \lor -b^3 \lor -o_1^1 \lor -c^2 \lor -c^3\]

Valuation constructed by the DPLL procedure

\[
\begin{array}{ccc|ccc}
0 & 1 & 2 & 3 & 1 & 2 & 3 \\
\hline
b' & 1 & 1 & 0 & 1 & 0 & 1 \\
c' & 1 & 0 & 1 & 0 & 1 & 0 \\
\end{array}
\]

M. Helmer, B. Nebel (Universität Freiburg)  AI Planning November 24th, 2006 43 / 74
Planning as satisfiability
Example: plan search with DPLL

Perform unit resolution and unit subsumption with $o_2^2$.

\[-o_1^1 \lor b^2 \quad -o_1^1 \lor \neg b^2\]
\[-o_2^2 \lor c^2 \quad -o_2^2 \lor \neg c^2\]
\[o_1^3 \lor o_2^3\]
\[o_1^3 \lor o_2^3\]

Valuation constructed by the DPLL procedure

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b'$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$c'$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

M. Helmert, B. Nebel (Universität Freiburg)  AI Planning November 24th, 2006 46 / 74

Planning as satisfiability
Example: plan search with DPLL

Perform unit resolution and unit subsumption with $\neg b^2$ and $\neg c^2$.

\[-o_1^1 \lor b^2 \quad -o_1^1 \lor \neg b^2\]
\[-o_1^1 \lor c^2 \quad -o_1^1 \lor \neg c^2\]
\[o_1^3 \lor o_2^3\]

Valuation constructed by the DPLL procedure

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b'$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$c'$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

M. Helmert, B. Nebel (Universität Freiburg)  AI Planning November 24th, 2006 46 / 74

Planning as satisfiability
Example: plan search with DPLL

Perform unit resolution and unit subsumption with $o_2^3$.

\[o_1^3\]

Valuation constructed by the DPLL procedure

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b'$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$c'$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

M. Helmert, B. Nebel (Universität Freiburg)  AI Planning November 24th, 2006 48 / 74
Planning as satisfiability

Example: plan search with DPLL

The formula is satisfiable.

Valuation constructed by the DPLL procedure

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b'</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c'</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>α</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>σ</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>σ'</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Efficiency of satisfiability planning is strongly dependent on the plan length because satisfiability algorithms have runtime $O(2^n)$ where $n$ is the formula size, and formula sizes are linearly proportional to plan length.

Parallel plans Parallelism

Example

Parallel actions Non-interleavable actions

According to the definition attempt, the operators $\langle a, \neg b \rangle$ and $\langle b, \neg a \rangle$ may be executed simultaneously in state $\{ a \mapsto 1, b \mapsto 1 \}$, resulting in the state $\{ a \mapsto 0, b \mapsto 0 \}$.

But this state is not reachable by the two operators sequentially, because executing any one operator makes the precondition of the other false.

Definition attempt

Parallel operator application

Definition attempt

Similar to relaxed planning graphs, we consider the possibility of executing several operators simultaneously.

Definition (?)

Let $\sigma$ be a set of operators (a plan step) and $s$ a state.

Define $app_{\sigma}(s)$ as the state that is obtained from $s$ by making the literals in $\bigcup_{(c,e) \in \sigma} [e]_{s}$ true.

For $app_{\sigma}(s)$ to be defined, we require that $s \models c$ for all $o = (c,e) \in \sigma$ and $\bigcup_{(c,e) \in \sigma} [e]_{s}$ is consistent.

Unfortunately, the definition is flawed. Why?
Parallel actions

Comparison to relaxed planning tasks

▶ When discussing relaxed planning tasks, we gave a conservative definition of parallel operator application:
  ▶ It is not guaranteed that each serialization of a plan step \( \sigma \) (or even one of them) leads to the state \( \text{app}_\pi(s) \).
  ▶ However, the resulting state of the serialized plan is guaranteed to be at least as good as \( \text{app}_\pi(s) \).
▶ Our general definition attempt was not conservative — not even if we require positive normal form (as the example shows).
▶ A conservative definition extending the earlier one for relaxed planning tasks is possible, but complicated.
▶ Instead, we use a semantic definition based on serializations.

Parallel plans

Definition (parallel plan)

A parallel plan for a general planning task \( \langle A, I, O, G \rangle \) is a sequence of plan steps \( \sigma_1, \ldots, \sigma_n \) of operators in \( O \) with:
  ▶ \( s_0 := I \)
  ▶ For \( i = 1, \ldots, n \), step \( \sigma_i \) is applicable in \( s_{i-1} \)
    and \( s_i := \text{app}_\pi(s_{i-1}) \).
  ▶ \( s_n \models G \)

Remark: By ordering the operators within each single step arbitrarily, we obtain a (regular, non-parallel) plan.

Parallel actions

Serializations and semantics

Definition (serialization)

A serialization of plan step \( \sigma = \{ o_1, \ldots, o_n \} \) is a sequence \( o_{\pi(1)}, \ldots, o_{\pi(n)} \) where \( \pi \) is a permutation of \( \{1, \ldots, n\} \).

Definition (semantics of plan steps)

A plan step \( \sigma = \{ o_1, \ldots, o_n \} \) is applicable in a state \( s \) iff each serialization of \( \sigma \) is applicable in \( s \) and results in the same state \( s' \).

The result of applying \( \sigma \) in \( s \) is then defined as \( \text{app}_\pi(s) = s' \).

Note: This definition does not extend the earlier definition for relaxed planning tasks.
Interference

Example

Actions do not interfere

![Diagram showing actions that do not interfere]

Actions can be taken simultaneously.

Actions interfere

![Diagram showing actions that interfere]

If A is moved first, B will not be clear and cannot be moved.

M. Helmert, B. Nebel (Universität Freiburg) AI Planning November 24th, 2006 57 / 74

Interference

Auxiliary definition: affects

Definition (affect)

Let $A$ be a set of state variables and $o = (c, e)$ and $o' = (c', e')$ operators over $A$. Then $o$ affects $o'$ if there is $a \in A$ such that

1. $a$ is an atomic effect in $e$ and $a$ occurs in a formula in $e'$ or it occurs negatively in $c'$, or
2. $\neg a$ is an atomic effect in $e$ and $a$ occurs in a formula in $e'$ or it occurs positively in $c'$.

Example

$\langle c, d \rangle$ affects $\langle \neg d, e \rangle$ and $\langle e, d \triangleright f \rangle$.

$\langle c, d \rangle$ does not affect $\langle d, e \rangle$ nor $\langle e, \neg c \rangle$.

M. Helmert, B. Nebel (Universität Freiburg) AI Planning November 24th, 2006 58 / 74

Interference

Sufficient condition for applying a plan step

Lemma

Let $s$ be a state and $\sigma$ a set of operators so that each operator in $\sigma$ is applicable in $s$, no two operators in $\sigma$ interfere, and $\bigcup_{(c, e) \in \sigma} [e]_s$ is consistent.

Then $\sigma$ is applicable in $s$ and results in the state that is obtained from $s$ by making the literals in $\bigcup_{(c, e) \in \sigma} [e]_s$ true.

M. Helmert, B. Nebel (Universität Freiburg) AI Planning November 24th, 2006 60 / 74
Parallel plans
Parallel operators

Parallel operator application

We cannot simply use our current definition of $\tau_A(o)$ within a satisfiability encoding for parallel planning:

- The formula $\tau_A(o)$ completely defines the relationship between current state and successor state when $o$ is applied.
- It leaves no room for applying another operator in sequence.

Basic idea for parallel plan encodings:

- Decouple the parts of the formula that describe what changes from parts that describe what does not change.

The explanatory frame axioms

The formula states that the only explanation for $a$ changing its value is the application of one operator:

\[
\bigwedge_{a \in A} ((a \land \neg a') \rightarrow EPC_{\neg a}(e))
\]

\[
\bigwedge_{a \in A} ((\neg a \land a') \rightarrow EPC_{a}(e))
\]

When several operators could be applied in parallel, we have to consider all operators as possible explanations:

\[
\bigwedge_{a \in A} ((a \land \neg a') \rightarrow \bigvee_{i=1}^{n} (o_i \land EPC_{\neg a}(e_i)))
\]

\[
\bigwedge_{a \in A} ((\neg a \land a') \rightarrow \bigvee_{i=1}^{n} (o_i \land EPC_{a}(e_i)))
\]

where $\sigma = \{o_1, \ldots, o_n\}$ and $e_1, \ldots, e_n$ are the respective effects.

Parallel plans
Parallel actions

Parallel actions

Definition (plan step application in propositional logic)

Let $\sigma$ be a plan step. Let $\tau_A(\sigma)$ denote the conjunction of formulae:

\[
(o \rightarrow c)
\]

\[
\bigwedge_{a \in A} (o \land EPC_{a}(e) \rightarrow a')
\]

\[
\bigwedge_{a \in A} (o \land EPC_{\neg a}(e) \rightarrow \neg a')
\]

for all $o = (c, e) \in \sigma$ and

\[
\bigwedge_{a \in A} ((a \land \neg a') \rightarrow \bigvee_{i=1}^{n} (o_i \land EPC_{\neg a}(e_i)))
\]

\[
\bigwedge_{a \in A} ((\neg a \land a') \rightarrow \bigvee_{i=1}^{n} (o_i \land EPC_{a}(e_i)))
\]

where $\sigma = \{o_1, \ldots, o_n\}$ and $e_1, \ldots, e_n$ are the respective effects.
Correctness

The formula $\tau_A(\sigma)$ exactly matches the definition of $\text{app}_\sigma(s)$ provided that no actions in $\sigma$ interfere.

Lemma

Let $s$ and $s'$ be states and $\sigma$ a set of operators. Let $\nu : A \cup A' \cup \sigma \rightarrow \{0, 1\}$ be a valuation such that

1. for all $o \in \sigma$, $\nu(o) = 1$,
2. for all $a \in A$, $\nu(a) = s(a)$, and
3. for all $a \in A$, $\nu(a') = s'(a)$.

If $\sigma$ is applicable in $s$, then:

$\nu \models \tau_A(\sigma)$ if and only if $s' = \text{app}_\sigma(s)$.

Translation of parallel plans into propositional logic

Definition

Define $R_2(A, A', O)$ as the conjunction of $\tau_A(O)$ and $\neg(o \land o')$ for all $o \in O$ and $o' \in O$ such that $o$ and $o'$ interfere and $o \neq o'$.

Planning as satisfiability

Existence of plans

Definition (bounded step number plans in propositional logic)

Existence of parallel plans of length $t$ is represented by the following formula over propositions $A^0 \cup \cdots \cup A^t \cup O^1 \cup \cdots \cup O^t$ where $A^i = \{ a^i | a \in A \}$ for all $i \in \{0, \ldots, t\}$ and $O^i = \{ o^i | o \in O \}$ for all $i \in \{1, \ldots, t\}$:

$\Phi^\text{par}_t = l^0 \land R_2(A^0, A^1, O^1) \land \cdots \land R_2(A^{t-1}, A^t, O^t) \land G^t$

where $l^0 = \bigwedge_{a \in A, I(a)=1} a^0 \land \bigwedge_{a \in A, I(a)=0} \neg a^0$ and $G^t$ is $G$ with propositions $a$ replaced by $a^t$.

Theorem

Let $\Phi^\text{par}_t$ be the formula for $\langle A, I, O, G \rangle$ and plan length $t$. The formula $\Phi^\text{par}_t$ is satisfiable if and only if there is a sequence of states $s_0, \ldots, s_t$ and plan steps $\sigma_1, \ldots, \sigma_t$, each consisting of non-interfering operators, such that $s_0 = I$, $s_i = \text{app}_\sigma(s_{i-1})$ for all $i \in \{1, \ldots, t\}$, and $s_t \models G$. 
Why is optimality lost?

Minimal step count does not imply minimal length
That a plan has the smallest number of steps does not guarantee that it has the smallest number of actions.

- Satisfiability algorithms return any satisfying valuation of $\Phi^\text{par}$, and this does not have to be the one with the smallest number of operators.
- There could be better solutions with more time points.
- Moreover, even optimality in the number of time steps is not guaranteed because the non-interference requirement is only sufficient, but not necessary, for parallel applicability.

![Diagram of initial and goal state]

The DPLL procedure solves the problem quickly:
- Formulae for lengths 0 to 4 shown unsatisfiable without any search.
- Formula for plan length 5 is satisfiable: 3 nodes in the search tree.
- Plans have 5 to 7 operators, optimal plan has 5.

Planning as satisfiability

Example

\begin{itemize}
  \item Length 0: \text{branch on } -\text{clear}(b)[1] \text{ depth 0}
  \item Length 1: \text{branch on clear}(a)[3] \text{ depth 1}
  \item Length 5: \text{found a plan}
    \begin{itemize}
      \item 0 \text{ totalable}(e,d)
      \item 1 \text{ totalable}(c,b) \text{ fromtable}(d,e)
      \item 2 \text{ totalable}(b,a) \text{ fromtable}(c,d)
      \item 3 \text{ fromtable}(b,c)
      \item 4 \text{ fromtable}(a,b)
    \end{itemize}
  \end{itemize}

Branches 2 last 2 failed 0; time 0.0
Parallel plans Example

<table>
<thead>
<tr>
<th>clear(a)</th>
<th>clear(b)</th>
<th>clear(c)</th>
<th>clear(d)</th>
<th>clear(e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 0</td>
<td>0 0 0 1 1</td>
<td>0 1 0 0 0</td>
<td>1 1 0 0 0</td>
<td>1 1 0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>on(a,b)</th>
<th>on(a,c)</th>
<th>on(a,d)</th>
<th>on(a,e)</th>
<th>on(b,a)</th>
<th>on(b,c)</th>
<th>on(b,d)</th>
<th>on(b,e)</th>
<th>on(c,a)</th>
<th>on(c,b)</th>
<th>on(c,d)</th>
<th>on(c,e)</th>
<th>on(d,a)</th>
<th>on(d,b)</th>
<th>on(d,c)</th>
<th>on(d,e)</th>
<th>on(e,a)</th>
<th>on(e,b)</th>
<th>on(e,c)</th>
<th>on(e,d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 1 0 0 0</td>
<td>0 0 0 1 1</td>
<td>0 0 0 1 1</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ont(x)</th>
<th>ont(y)</th>
<th>ont(z)</th>
<th>ont(w)</th>
<th>ont(u)</th>
<th>ont(v)</th>
<th>ont(w)</th>
<th>ont(x)</th>
<th>ont(y)</th>
<th>ont(z)</th>
<th>ont(w)</th>
<th>ont(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

1. Infer state variable values from initial values and goals.
2. Branch: $\neg$ clear(b)[1].
3. Branch: clear(a)[3].
4. Plan found:

fromtable(a,b) . . . 1
tfromtable(b,c) . . 1
fromtable(c,d) . 1 .
tfromtable(d,e) . 1 .
totable(b,a) . . 1 .
totable(c,b) . 1 . .
totable(e,d) 1 . . .

definitions

Final remarks

- All successful satisfiability-based planners use some kind of parallel encoding.
- Sequential encodings are not regarded as competitive with (admissible) heuristic search planners.
- In practice, the presented encoding is further refined to be able to rule out bad variable assignments early in the SAT solving procedure.
- The state-of-the-art SATPLAN06 (formerly SATPLAN04, formerly Blackbox) planner supports a number of different encodings.
- The ones that typically perform best are based on (non-relaxed) planning graphs.