Planning in the propositional logic

Abstractly

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.

Example

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

Matrices as formulae

Example (Formulae as relations as matrices)

Binary relation \( \{(00, 00), (00, 01), (00, 11), (01, 01), (01, 11), (10, 11), (11, 11)\} \) can be represented as the adjacency matrix:

\[
\begin{array}{ccccccc}
0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

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 \( 2^{20} \times 2^{20} \approx 10^8 \times 10^8 \).

Actions/relations as propositional formulae

Example

\( (a_1 \rightarrow a'_1) \land (a_2 \rightarrow a'_2) \land (a_3 \rightarrow a'_3) \) represents the matrix:

\[
\begin{array}{cccccccccccccccc}
000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
001 & 010 & 011 & 100 & 101 & 110 & 111 & 000 \\
010 & 011 & 100 & 101 & 110 & 111 & 000 & 001 \\
100 & 101 & 110 & 111 & 000 & 001 & 010 & 011 \\
101 & 110 & 111 & 000 & 001 & 010 & 011 & 100 \\
110 & 111 & 000 & 001 & 010 & 011 & 100 & 101 \\
111 & 000 & 001 & 010 & 011 & 100 & 101 & 110 \\
\end{array}
\]

and as a conventional truth-table:

\[
\begin{array}{cccccccccccccccc}
d_1 & d_2 & d_3 & d_4' & d_5' & d_6' & d_7' & d_8' & \\
00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \\
00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \\
00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \\
00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \\
00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \\
00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \\
00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \\
00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & \\
\end{array}
\]

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

Expressiveness of propositional logic

- For every operator there is a corresponding formula (see next slides!)
- Our current definition of operators does not allow expressing nondeterministic actions.
- In the propositional logic they can be expressed.

Example (A nondeterministic action)
The formula $\top$ describes the relation in which any state can be reached from any other state by this action.

A sufficient (but not necessary) condition for determinism

Formulas have the form $(\phi_1 \iff v_1') \land \cdots \land (\phi_n \iff v_n')$ where $A = \{a_1, \ldots, a_n\}$ and $\phi_i$ have no occurrences of propositions in $A'$.

Translating operators into formulae

- Any operator can be translated into a propositional formula.
- Translation takes polynomial time.
- Resulting formula has polynomial size.
- Use in planning algorithms. Two main applications are
  1. Planning as Satisfiability
  2. Progression & regression for state sets as used in symbolic state-space traversal, as typically implemented with the help of binary decision diagrams.

Example

Let the state variables be $A = \{a, b, c\}$.
Consider operator $(a \lor b, b \land a) \land (c \lor \neg a) \land (a \lor b)$).

The corresponding propositional formula is

$$(a \lor b) \land ((b \land (a \land \neg a)) \iff a')$$
$$(a \lor (b \land (a \land \neg a))) \iff b')$$
$$(a \lor (b \land (c \land \neg a))) \iff c')$$
$$\iff (a \lor b) \land ((b \land (a \land \neg a)) \iff a')$$
$$\iff ((a \land b) \land (c \iff c')$$
$$\iff (a \land b) \land (c \iff c')$$

Correctness

Lemma

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

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

Then $\nu = \tau_A(\phi)$ if and only if $s' = \text{app}._s(\phi)$.

Deterministic vs. nondeterministic actions

Example

An action that is applicable if $a$ is false, and that randomly sets values to state variables $b$ and $c$:

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>011</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Corresponding formula: $\neg a \land \neg a'$

Translating operators into formulæ

Definition

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

(1) $\bigwedge_{v \in A} (EPC(v) \lor (a \land \neg EPC(v)))$
(2) $\bigwedge_{v \in A} (EPC(v) \land EPC_{\neg v}(v))$

(2) says that the new value of $a$, represented by $a'$, is 1 if the old value was 1 and it did not become 0, or it became 1.
(3) says that none of the state variables is assigned both 0 and 1. This together with $c$ determine whether the operator is applicable.

Example

Let $A = \{a, b, c, d, e\}$ be the state variables.
Consider operator $(a \land e, (d \lor e))$.
The formula $\tau_A(\phi)$ after simplifications is

\[
(a \land b) \land ((a \iff a') \land (b \iff b') \land (d \iff d') \land ((d \lor e) \iff e'))
\]

Translating operators into formulæ

Example

Let $A = \{a, b, c, d, e\}$ be the state variables.
Consider operator $(a \land e, (d \lor e))$.
The formula $\tau_A(\phi)$ after simplifications is

\[
(a \land b) \land ((a \iff a') \land (b \iff b') \land (d \iff d') \land ((d \lor e) \iff e'))
\]

Correctness

Lemma

Let $\tau_A(\phi)$ be an operator and $\phi$ an operator. Let $\nu : A \cup A' \rightarrow \{0, 1\}$ be a valuation such that

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

Then $\nu \models \tau_A(\phi)$ if and only if $s' = \text{app}._s(\phi)$.

Correctness

Lemma

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

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

Then $\nu \models \tau_A(\phi)$ if and only if $s' = \text{app}._s(\phi)$.
Planning as satisfiability

Definition (Transition relation in CPC)
For \((A, I, O, G)\) define
\[
R_i(A, A') = \bigvee_{o \in O} \tau_i(o).
\]

Definition (Bounded-length plans in CPC)
Existence of plans length 1 is represented by a formula over propositions \(A^0 \cup \cdots \cup A^t\) where \(A^i = \{a[i] \in A\}\) for all \(i \in \{0, \ldots, t\}\) as
\[
\Phi_{eq}^i = \beta \land R_i(A^0, A^i) \land R_i(A^1, A^i) \land \cdots \land R_i(A^{i-1}, A^i) \land G^i
\]
where \(\beta = \bigwedge\{a[i] \in A, I(a) = 1\} \cup \{\neg a[0] | a \in A, I(a) = 0\}\) and \(G^i\) with propositions \(a\) replaced by \(a^i\).

Planning as satisfiability

Theorem
Let \(\Phi_{eq}^i\) be the formula for \((A, I, O, G)\) and plan length \(i\). The formula \(\Phi_{eq}^i\) is satisfiable if and only if there is a sequence of states \(s_0, \ldots, s_t\) and operators \(a_1, \ldots, a_t\) such that \(s_0 = I, s_t = G\) and \(s_i = \text{app}(a_{i-1}, s_{i-1})\) for all \(i \in \{1, \ldots, t\}\).

Consequence
If \(\Phi_{eq}^i, \Phi_{eq}^{i+1}, \ldots, \Phi_{eq}^{k-1}\) are unsatisfiable and \(\Phi_{eq}^k\) is satisfiable, then the length of shortest plans is \(k\).

Satisfiability planning with \(\Phi_{eq}^i\) yields optimal plans, like heuristic search with admissible heuristics and optimal algorithms like A\(^\ast\) or IDA\(^\ast\).

Planning as satisfiability

Example, continued

One valuation that satisfies \(\Phi_{eq}^i\):

\[
\begin{array}{cccc}
\text{time } i & 0 & 1 & 2 & 3 \\
\beta & 1 & 0 & 0 & 0 \\
\epsilon & 1 & 0 & 0 & 1 \\
\end{array}
\]

Notice:
1. Also a plan of length 1 exists.
2. Plans of length 2 do not exist.

Conjunctive normal form

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

\[
\begin{align*}
\neg (\phi \lor \psi) & \equiv \neg \phi \land \neg \psi \\
\neg (\phi \land \psi) & \equiv \neg \phi \lor \neg \psi \\
\neg \neg \phi & \equiv \phi \\
\phi \lor (\psi_1 \land \psi_2) & \equiv (\phi \lor \psi_1) \land (\phi \lor \psi_2)
\end{align*}
\]

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

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

The Davis-Putnam 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 Davis-Putnam procedure (Although there is currently a shift toward viewing these procedures as performing more general resolution: clause-learning.)
**Satisfiability test by the Davis-Putnam procedure**

1. Let $C$ be a set of clauses.
2. For all clauses $I_1 \vee I_2 \vee \cdots \vee I_m \in C$ and $T \in C$, remove $I_1 \vee I_2 \vee \cdots \vee I_m$ from $C$ and add $I_1 \vee I_2 \vee \cdots \vee I_m \vee T$ to $C$.
3. For all clauses $I_1 \vee I_2 \vee \cdots \vee I_m \in C$ and $T \in C$, remove $I_1 \vee I_2 \vee \cdots \vee I_m$ from $C$. (UNIT SUBSUMPON)
4. If $\bot \in C$, return FALSE.
5. If $C$ contains only unit clauses, return TRUE.
6. Pick some $a \in A$ such that $\{a, \neg a\} \cap C = \emptyset$
7. Recursive call: if $C \cup \{a\}$ is satisfiable, return TRUE.
8. Recursive call: if $C \cup \{\neg a\}$ is satisfiable, return TRUE.
9. Return FALSE.

**Planning as satisfiability**

Example: plan search with Davis-Putnam

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

\[
\begin{align*}
    a_1^0 &\equiv \left( (b_1^0 \land b_1^0) \land (b_1^0 \land \neg b_1^0) \right) \\
    a_2^0 &\equiv \left( (b_2^0 \land \neg b_2^0) \land (b_2^0 \land b_2^0) \right) \\
    a_1^1 \lor a_2^1 &\equiv \left( (b_1^1 \land b_1^1) \land (b_1^1 \land \neg b_1^1) \right) \\
    a_1^2 \lor a_2^2 &\equiv \left( (b_1^2 \land b_1^2) \land (b_1^2 \land \neg b_1^2) \right) \\
    (b_1^3 \land \neg b_1^3) \lor (\neg b_1^3 \land b_1^3) &\equiv \left( (b_1^3 \land b_1^3) \land (b_1^3 \land \neg b_1^3) \right) \\
\end{align*}
\]

**Parallel plans**

**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.

- **Formula sizes** can be reduced by allowing several operators in parallel.
- On many problems this leads to big speed-ups.
- However there are no guarantees of optimality.

**Planning as satisfiability with parallel plans**

We consider the possibility of executing several operators simultaneously.

**Definition**

Let $T$ be a set of operators and $s$ a state.

Define $\text{app}_{T}(s)$ as the state that is obtained from $s$ by making the literals in $\bigcup_{a \in T} \text{EPC}(a)$ true.

For $\text{app}_{T}(s)$ to be defined, we require that $s \models c$ for all $a = (c, e) \in T$ and $\bigcup_{a \in T} \text{EPC}(a)$ consistent.

**Parallel operator application**

**Formal definition**

We rewrite the formulae for operator application using the equivalence $a \rightarrow (b \lor c) \equiv ((a \land b) \lor (a \land c))$.

\[
\begin{align*}
    b_0^0 &\equiv a_1^0 \land b_1^0 \land b_2^0 \land b_3^0 \\
    c_0 &\equiv a_1^1 \lor a_2^1 \\
    c_1 &\equiv a_1^2 \lor a_2^2 \\
    d_0 &\equiv (b_1^3 \land \neg b_1^3) \lor (\neg b_1^3 \land b_1^3) \\
\end{align*}
\]

**The explanatory frame axioms**

The formulae say that the only explanation for $a$ changing its value is the application of one operator.

\[
\begin{align*}
    \wedge_{a \in A}((a \land \neg a') \rightarrow \text{EPC}_{a}(c)) \\
    \wedge_{a \in A}((a \land a') \rightarrow \text{EPC}_{a}(c)) \\
\end{align*}
\]

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

\[
\begin{align*}
    \wedge_{a \in A}((a \land \neg a') \rightarrow \bigcup_{a \in A} \text{EPC}_{a}(c)) \\
    \wedge_{a \in A}((a \land a') \rightarrow \bigcup_{a \in A} \text{EPC}_{a}(c)) \\
\end{align*}
\]

where $T = \{a_1, \ldots, a_n\}$ and $c_1, \ldots, c_r$ are the respective effects.
Parallel actions

Formula in CPC

Definition

Let $T$ be a set of operators. Let $\tau_a(T)$ denote the conjunction of formulae

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

for all $(c, e) \in T$ and

\[
\bigwedge_{o \in A}(x \land \neg \neg a') \rightarrow ((o_1 \land EPC_{o_1}(e_1)) \lor \cdots \lor (o_n \land EPC_{o_n}(e_n))) \\
\bigwedge_{o \in E}(\neg x \land a') \rightarrow ((o_1 \land EPC_{o_1}(e_1)) \lor \cdots \lor (o_n \land EPC_{o_n}(e_n)))
\]

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

Parallel actions

Meaning in terms of interleavings

Example

The operators $(a \rightarrow b)$ and $(b \rightarrow c)$ may be executed simultaneously resulting in a state satisfying $\neg a \land \neg b$. But this state is not reachable by the two operators sequentially, because executing any one operator makes the precondition of the other false.

Step plans

Tractable subclass

- Finding arbitrary step plans is difficult: even testing whether a set $T$ of operators is executable in all orders is co-NP-hard.
- Representing the executability test exactly as a propositional formula seems complicated: doing this test exactly would seem to cancel the benefits of parallel plans.
- Instead, all work on parallel plans so far has used a sufficient but not necessary condition that can be tested in polynomial-time.
- This is a simple syntactic test: is the result of executing $a_1$ and $a_2$ in any state both in order $a_1; a_2$ and in $a_2; a_1$ the same.

Interference

Auxiliary definition: affects

Definition (Affect)

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

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

Example

$(c, d)$ affects $(\neg d, c)$ and $(c, d \lor f)$. $(c, d)$ does not affect $(d, e)$ nor $(e, \neg c)$.

Correctness

The formula $\tau_a(T)$ exactly matches the definition of $app_T(s)$.

Lemma

Let $s$ and $s'$ be states and $T$ a set of operators. Let $v : A \cup A' \cup T \rightarrow \{0, 1\}$ be a valuation such that

1. for all $o \in T$, $v(o) = 1$,
2. for all $a \in A$, $v(a) = s(a)$, and
3. for all $a \in A$, $v(a') = s'(a)$.

Then $v \models \tau_a(T)$ if and only if $s' = app_T(s)$.

Interference

Example

Actions do not interfere

Actions can be taken simultaneously.

Actions interfere

If A is moved first, B won’t be clear and cannot be moved.

Interference

Definition (Interference)

Operators $\alpha$ and $\alpha'$ interfere if $\alpha$ affects $\alpha'$ or $\alpha'$ affects $\alpha$.

Example

$(c, d)$ and $(\neg d, e)$ interfere. $(c, d)$ and $(e, f)$ do not interfere.
Lemma
Let $s$ be a state and $T$ a set of operators so that $\text{app}_{s}(s)$ is defined and no two operators interfere. Then $\text{app}_{s}(s) = \text{app}_{\sigma_{1},\ldots,\sigma_{n}}(s)$ for any total ordering $\sigma_{1},\ldots,\sigma_{n}$ of $T$.

Definition (Bounded-length plans in CPC)
Existence of parallel plans length $t$ is represented by a formula over propositions $A^{0} \cup \cdots \cup A^{t} \cup O^{1} \cup \cdots \cup O^{t}$ where $A^{t} = \{a^{t}|a \in A\}$ for all $i \in \{0,\ldots,t\}$ and $O^{t} = \{o^{t}|o \in O\}$ for all $i \in \{1,\ldots,t\}$ as
\[
\Phi^{\text{per}}_{t} = \bigwedge_{i=0}^{t} \bigwedge_{a \in A} \bigwedge_{o \in O} \bigwedge_{i=1}^{t} \bigwedge_{a \in A} \bigwedge_{o \in O} (\neg \phi^{a}_{i} \land \phi^{o}_{i}) \land G^{t}
\]
where $\phi^{a}_{i} = \bigwedge_{a \in A} \bigwedge_{o \in O} (\neg \phi^{a}_{i} \land \phi^{o}_{i})$ and $G^{t}$ is $G$ with propositions $a$ replaced by $a^{t}$.

Definition
Define $R_{A}(A',O)$ as the conjunction of $\tau_{a}(O)$ and $-(a \land a')$ for all $a \in O$ and $a' \in O$ such that $a$ and $a'$ interfere and $a \neq a'$.

Planning as satisfiability
Existence of plans

Theorem
Let $\phi^{\text{per}}_{n}$ be the formula for $(A, I, O, G)$ and plan length $t$. The formula $\phi^{\text{per}}_{n}$ is satisfiable if and only if there is a sequence of states $s_{0},\ldots,s_{t}$ and sets $O_{1},\ldots,O_{t}$ of non-interfering operators such that $s_{0} = I$ and sets $O_{1},\ldots,O_{t}$ of non-interfering operators such that $s_{0} = I$, $s_{i} \models G$ and $s_{i} = \text{app}_{s_{i-1}}(s_{i-1})$ for all $i \in \{1,\ldots,t\}$.

Example
Let $t$ be a state such that $s \models \neg c \land \neg d \land \neg c \land \neg f$.
Let $G = c \land d \land e$.

Example
The Davis-Putnam procedure solves the problem quickly:
- Formulate for lengths 1 to 4 shown unsatisfiable without any search.
- Formulae for plan lengths 1 and 2 shown unsatisfiable without any search.
- Plans 5 to 7 operators, optimal plan has 5.
Planning as satisfiability
Example: valuations after unit propagation, after branching

ON
CLEAR
TABLE

01234

fromtable(a,b) ....T
fromtable(b,c) ...T.
fromtable(c,d) ...T...
totable(b,a) ..T...
totable(c,b) .T...
totable(e,d) T....