Foundations of AI

Planning

Planning in the Situation Calculus,
STRIPS Formalism,
Non-Linear Planning

Wolfram Burgard and Luc De Raedt and Bernhard Nebel
Contents

- Planning vs. Problem Solving
- Planning in Situation Calculus
- STRIPS Formalism
- Non-Linear Planning
- The POP Algorithm
- Variable Bindings
Planning

- Given an *logical description* of the *initial situation*,
- a *logical description* of the *goal conditions*, and
- a *logical description* of a set of *possible actions*,
→ find a **sequence of actions** (a **plan of action**) that brings us from the initial situation to a situation in which the goal conditions hold.
Agent Approach

1. Definition of a goal.
2. Identifying current state.
3. Development of a plan to bring the agent from the current state to the goal state.
4. Execution of the plan until the goal state is reached (or goal unachievable with current plan)
5. Repeat from 1.).
A Simple Planning Agent

function SIMPLE-PLANNING-AGENT(\textit{percept}) returns an \textit{action}

\textbf{static:} $KB$, a knowledge base (includes action descriptions)
\hspace{1em} $p$, a plan, initially \textit{NoPlan}
\hspace{1em} $t$, a counter, initially 0, indicating time

\textbf{local variables:} $G$, a goal
\hspace{2em} $current$, a current state description

T\textsc{ell}($KB$, \textsc{Make-Percept-Sentence}($\textit{percept}, t$))

c\textsc{urrent} $\leftarrow$ \textsc{State-Description}($KB, t$)

\begin{itemize}
\item \textbf{if} $p =$ \textit{NoPlan} \textbf{then}
\hspace{1em} $G$ $\leftarrow$ \textsc{Ask}($KB, \textsc{Make-Goal-Query}(t)$)
\hspace{1em} $p$ $\leftarrow$ \textsc{Ideal-Planner}($current, G, KB$)
\item \textbf{if} $p =$ \textit{NoPlan} or $p$ is empty \textbf{then} \textit{action} $\leftarrow$ \textit{NoOp}
\item \textbf{else}
\hspace{1em} \textit{action} $\leftarrow$ \textsc{First}($p$)
\hspace{1em} $p$ $\leftarrow$ \textsc{Rest}($p$)
\end{itemize}

T\textsc{ell}($KB, \textsc{Make-Action-Sentence}(\textit{action}, t))$

$t$ $\leftarrow$ $t + 1$

\textbf{return} \textit{action}
Planning vs. Problem-Solving

Basic difference: **Explicit, logic-based representation**

- **States/Situations**: Through descriptions of the world by logical formula vs. data structures
  This way, the agent can explicitly think about and communicate

- **Goal conditions** as logical formulae vs. goal test (black box)
  The agent can also reflect on its goals.

- **Operators**: Axioms or transformation on formulae vs. modification of data structures by programs
  The agent can gain information about the effects of actions by inspecting the operators.
Planning vs. Automatic Programming

Difference between planning and automatic programming:

- Logic-based description of the world.

- Plans are usually only linear programs (no control structures).
Planning vs. MDP Policy Determination

- Simpler model: deterministic vs. probabilistic and goals vs. utilities
- More abstract, implicit state space description: logical description instead of explicit enumeration of states
- Can handle much larger state spaces!
Planning as Logical Inference (1)

Planning can be elegantly formalized with the help of the situation calculus.

**Initial state:**

\[ At(\text{Home},s_0) \land \neg \text{Have(milk, } s_0) \land \neg \text{Have(banana, } s_0) \land \neg \text{Have(drill, } s_0) \]

**Operators** (successor-state axioms):

\[ \forall a,s \text{ Have(milk, do(a,s)) } \iff \]
\[ \{a = \text{buy(milk)} \land \text{Poss(buy(milk), } s) \lor \text{Have(milk, } s) \land a \neq \neg \text{drop(milk)} \} \]

**Goal conditions** (query):

\[ \exists s \text{ At(home, } s) \land \text{Have(milk, } s) \land \text{Have(banana, } s) \land \text{Have(drill, } s) \]

When the initial state, all prerequisites and all successor-state axioms are given, the **constructive** proof of the existential query delivers a plan that does what is desired.
Planning as Logical Inference (2)

The variable bindings for $s$ could be as follows:

$$do(go(home), do(buy(drill), do(go(hardware_store), do(buy(banana), do(buy(milk),
    do(go(supermarket), s0))))))$$

I.e. the plan (term) would be

$$\langle go(super\_market), buy(milk), \ldots \rangle$$

However, the following plan is also correct:

$$\langle go(super\_market), buy(milk), drop(milk), buy(milk), \ldots \rangle$$

In general, planning by theorem proving is very inefficient

Specialized inference system for limited representation.

→ Planning algorithm
The **STRIPS** Formalism

**STRIPS:** STanford Research Institute Problem Solver (early 70s)

The system is obsolete, but the formalism is still used. Usually simplified version is used:

**World state** (including initial state): Set of ground atoms (called *fluents*), no function symbols except for constants, interpreted under closed world assumption (**CWA**). Sometimes also standard interpretation, i.e. negative facts must be explicitly given

**Goal conditions:** Set of ground atoms

**Note:** No explicit state variables as in situation calculus. Only the current world state is accessible.
STRIPS Operators

Operators are triples, consisting of

**Action Description**: Function name with parameters (as in situation calculus)

**Preconditions**: Conjunction of positive literals; must be true before the operator can be applied (after instantiation)

**Effects**: Conjunction of positive and negative literals; positive literals are added (ADD list), negative literals deleted (DEL list) (no **frame** problem!).

\[
\text{Op} \quad \text{Action: } \text{Go}(\text{there}), \\
\text{Precond: } \text{At}(\text{here}) \land \text{Path}(\text{here}, \text{there}), \\
\text{Effect: } \text{At}(\text{there}) \land \neg \text{At}(\text{here})
\]
Actions and Executions

- An **action** is an operator, where all variables have been *instantiated*:

  \[
  \text{Op}(
  \begin{array}{l}
  \text{Action: } \text{Go(SuperMarket)}, \\
  \text{Precond: } \text{At(Home)} \land \text{Path(Home, SuperMarket)}, \\
  \text{Effect: } \text{At(Supermarket)} \land \neg \text{At(Home)})
  \end{array}
  \]

- An action can be **executed** in a state, if its precondition is satisfied. It will then bring about its effects.
Linear Plans

• A sequence of actions is a **plan**
• For a given initial state $I$ and goal conditions $G$, such a plan $P$ can be **successfully executed** in $I$ iff there exists a sequence of states $s_0, s_1, \ldots, s_n$ such that
  – the $i$th action in $P$ can be executed in $s_{i-1}$ and results in $s_i$
  – $s_0 = I$ and $s_n$ satisfies $G$
• $P$ is called a **solution** to the **planning problem** specified by the **operators**, $I$ and $G$
Searching in the State Space

We can now search through the state space (the set of all states formed by truth assignments to fluents) – and in this way reduce planning to searching.

We can search forwards (progression planning):

Or alternatively, we can start at the goal and work backwards (regression planning).

Possible since the operators provide enough information.
Searching in the Plan Space

Instead of searching in the state space, we can search in the space of all plans.

The initial state is a **partial plan** containing only start and goal states:

![Partial Plan Diagram](image)

The goal state is a **complete plan** that solves the given problem:

![Complete Plan Diagram](image)

Operators in the plan space:

**Refinement operators** make the plan more complete (more steps etc.)

**Modification operators** modify the plan (in the following, we use only refinement operators)
Plan = Sequence of Actions?

Often, however, it is neither meaningful nor possible to commit to a specific order early-on (put on socks and shoes).

→ Non-linear or partially-ordered plans (least-commitment planning)
Representation of Non-Linear Plans

A plan step = STRIPS operator

A plan consists of

- A set of **plan steps** with partial ordering (≺), where $S_i \prec S_j$ implies $S_i$ must be executed before $S_j$.

- A set of **variable assignments** $\chi = t$, where $\chi$ is a variable and $t$ is a constant or a variable.

- A set of **causal relationships** $S_i \rightarrow S_j$ means “$S_i$ produces the precondition $c$ for $S_j$” (implies $S_i \prec S_j$).

Solutions to planning problems must be **complete** and **consistent**.
Completeness and Consistency

**Complete Plan:**

Every precondition of a step is fulfilled:

\[ \forall S_j \forall c \in \text{Precond}(S_j) : \]

\[ \exists S_i \text{ with } S_i < S_j \text{ and } c \in \text{Effects}(S_i) \text{ and } \]

for every linearization of the plan:

\[ \forall S_k \text{ with } S_i < S_k < S_j \text{ and } \neg c \notin \text{Effect}(S_k). \]

**Consistent Plan:**

if \( S_i < S_j \), then \( S_j \not< S_i \) and

if \( x = A \), then \( x \neq B \) for distinct \( A \) and \( B \) for a variable \( x \). (*unique name assumption* = UNA)

A **complete, consistent plan** is called a **solution** to a planning problem (all **linearizations** are **executable linear plans**).
Example

At(Home)  Sells(SM,Banana)  Sells(SM,Milk)  Sells(HWS,Drill)

Have(Drill)  Have(Milk)  Have(Banana)  At(Home)

Actions:

Op( Action: Go(there),
    Precond: At(here) ∧ Path(here, there),
    Effect: At(there) ∧ ¬At(here))

Op( Action: Buy(x),
    Precond: At(store) ∧ Sells(store, x),
    Effect: Have(x))

there, here, x, store are variables.
Plan Refinement (1)

Regression Planning: Fulfils the **Have** predicates:

... after instantiation of the variables:

Thin arrow = <, thick arrow = causal relationship + <
Plan Refinement (2)

Shop at the right store…
Plan Refinement (3)

First, you have to go there…

Note: So far no searching, only simple backward chaining.

Now: Conflict! If we have done go(HWS), we are no longer At(home). Likewise for go(SM).
Protection of Causal Links

(a) Conflict: $S_3$ threatens the causal relationship between $S_1$ and $S_2$.

Conflict solutions:

(b) **Demotion**: Place the threatening step before the causal relationship.

(c) **Promotion**: Place the threatening step after the causal relationship.
A Different Plan Refinement...

- We cannot resolve the conflict by “protection”.
  → It was a mistake to choose to refine the plan.
- **Alternative:** When instantiating $\mathcal{A} t(\chi)$ in $\mathcal{g}_o(SM)$, choose $\chi = \text{HWS}$ (with causal relationship)
- **Note:** This threatens the purchase of the drill $\rightarrow$ promotion of $\mathcal{g}_o(SM)$. 
The Complete Solution
The POP Algorithm

function POP(initial, goal, operators) returns plan

    plan ← MAKE-MINIMAL-PLAN(initial, goal)
    loop do
        if SOLUTION?(plan) then return plan
        Sneed, c ← SELECT-SUBGOAL(plan)
        CHOOSE-OPERATOR(plan, operators, Sneed, c)
        RESOLVE-THREATS(plan)
    end

function SELECT-SUBGOAL(plan) returns Sneed, c

    pick a plan step Sneed from STEPS(plan)
    with a precondition c that has not been achieved
    return Sneed, c

procedure CHOOSE-OPERATOR(plan, operators, Sneed, c)

    choose a step S_add from operators or STEPS(plan) that has c as an effect
    if there is no such step then fail
    add the causal link $S_{add} \rightarrow S_{need}$ to LINKS(plan)
    add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(plan)
    if $S_{add}$ is a newly added step from operators then
        add $S_{add}$ to STEPS(plan)
        add Start $\prec S_{add} \prec Finish$ to ORDERINGS(plan)

procedure RESOLVE-THREATS(plan)

    for each $S_{threat}$ that threatens a link $S_i \rightarrow S_j$ in LINKS(plan) do
        choose either
            Promotion: Add $S_{threat} \prec S_i$ to ORDERINGS(plan)
            Demotion: Add $S_j \prec S_{threat}$ to ORDERINGS(plan)
        if not CONSISTENT(plan) then fail
    end
Properties of the POP Algorithm

Correctness: Every result of the POP algorithm is a complete, correct plan.

Completeness: If breadth-first-search is used, the algorithm finds a solution, given one exists.

Systematicity: Two distinct partial plans do not have the same total ordered plans as a refinement provided the partial plans are not refinements of one another (and totally ordered plans contain causal relationships).

→ Instantiation of variables is not addressed.
Variables

If a variable appears in the literal of an effect, for ex: \( \neg \text{At}(\chi) \), this literal constitutes a potential threat to a causal relationship. Conflict resolution:

- through an equality constraint, e.g. \( \chi = \text{HWS} \), so as not to threaten At(SM);
- through an inequality constraint (language extension), e.g. \( \chi \neq \text{SM} \) (but this is tricky);
- by later execution, if the variable is instantiated (makes it harder to determine if the plan is a solution).

We will choose the last option.
Treatment of Variables

**procedure** \textsc{choose-operator}(plan, operators, S\textsubscript{need}, c)

- **choose** a step \( S_{\text{add}} \) from operators or \textsc{steps}(plan) that has \( c_{\text{add}} \) as an effect such that \( u = \text{unify}(c, c_{\text{add}}, \text{bindings}(\text{plan})) \)
  - If there is no such step
    - then fail
  - add \( u \) to \text{bindings}(\text{plan})
  - add \( S_{\text{add}} \leftarrow S_{\text{need}} \) to \text{links}(\text{plan})
  - add \( S_{\text{add}} \leftarrow S_{\text{need}} \) to \text{orderings}(\text{plan})
  - if \( S_{\text{add}} \) is a newly added step from operators then
    - add \( S_{\text{add}} \) to \text{steps}(\text{plan})
    - add \( \text{start} \leftarrow S_{\text{add}} \leftarrow \text{finish} \) to \text{orderings}(\text{plan})

**procedure** \textsc{resolve-threats}(plan)

- for each \( S_i \leftarrow S_j \in \text{links}(\text{plan}) \) do
  - for each \( S_{\text{threat}} \in \text{steps}(\text{plan}) \) do
    - for each \( c' \in \text{effect}(S_{\text{threat}}) \) do
      - if \( \text{substitute}(\text{bindings}(\text{plan}), c) = \text{substitute}(\text{bindings}(\text{plan}), \neg c') \) then
        - **choose** either
          - \textit{Promotion}: Add \( S_{\text{threat}} \leftarrow S_i \) to \text{orderings}(\text{plan})
          - \textit{Demotion}: Add \( S_j \leftarrow S_{\text{threat}} \) to \text{orderings}(\text{plan})
        - if not \text{consistent}(\text{plan}) then fail
  - end
- end

Works if the initial state contains no variables, and every operator uses all its variables in its precondition. Otherwise we must change the \textit{solution?} function.
Modeling in STRIPS

Similar to what we have already seen (problem-solving, PL1), we must perform the following steps when modeling tasks:

• Decide what to talk about
• Decide on a **vocabulary** of conditions, operators and objects
• Encode **operators**
• Encode problem instances

Then the planner can produce solutions.
Example: The Blocks World

- There are named blocks sitting on a table in the world.
- There can be any number of blocks on the table, but only one block can fit directly on top of another.
- A block can only be moved if there is no other block on top of it.
- Knocking blocks over etc. is not allowed.
Modeling the Blocks World

• Only model the blocks; the table is implicit.

→ Objects: \(a, b, c\)

• We explicitly represent whether a block can be moved and whether it lies directly on the table.

→ Predicates:
  - \(\text{On}(x,y)\): \(x\) is on \(y\)
  - \(\text{OnTable}(x)\): \(x\) is on the table
  - \(\text{Clear}(x)\): there is nothing on top of \(x\)

• There are operators to move blocks from blocks to other blocks.

• There are operators to move blocks from the table onto a block, and vice versa.

→ Operators: \(\text{move}(x,y,z)\), \(\text{stack}(x,y)\), \(\text{unstack}(x,y)\), …
Summary

• Planning differs from problem-solving in that the representation is more flexible.
• In principle, we can reduce planning to logical inference (= situation calculus), although this is very inefficient.
• We can search in the plan space instead of the state space.
• The least commitment principle states that while searching, we should only make decisions when it is absolutely necessary.
• Non-linear planning is an instance of this principle.
• The POP algorithm realizes non-linear planning and is complete and correct.