11. Planning

Planning in Situational Calculus, STRIPS Formalism, Non-Linear Planning

Wolfram Burgard & Bernhard Nebel

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.

- Difference between this method and problem solving and search?
- Difference between this method and (automatic) programming?

A Simple Planning Agent

function SIMPLE-PLANNING-AGENT(percept) returns an action
  static: KB, a knowledgebase (includes action descriptions)
  p, a plan, initially NoPlan
  t, a counter, initially 0, indicating time
  local variables: G, a goal

  current ← STATE-DESCRIPTION(KB, t)
  if p = NoPlan then
    G ← ASK(KB, MAKE-GOAL-QUERY())
    p ← INITIAL-PLAN(KB, current, G, KB)
  if p = NoPlan or p is empty then action ← NoOp
  else
    action ← FIRST(p)
    p ← REST(p)
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
Agent Approaches

1. Definition of a goal.
2. Development of a plan to bring the agent from the current state to the goal state.
3. Execution of the plan until the goal state is reached.
4. Repeat from 1).

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 with the world.
- **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.

Difference between this method and programming:
- Logic-based description of the world.
- Plans are usually only linear programs (no control structures).

Planning as Logical Inference (1)

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

**Initial State**:

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

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

\[
\forall a \cdot \text{Have(milk}, a) \Rightarrow \neg \text{Have(milk}, s) \land \text{Pass(buy(milk}, a) \land \text{Have(buy(milk,} s) \lor \text{Have(milk,} s) \land a \neq \neg \text{drop(milk)}}
\]

**Goal conditions** (query):

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

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:

\[
\text{do(gro(home), do(buy(drill)), do(buy(hardware store), do(buy(banana), do(buy(milk),}
\text{do(gro(supermarket), s0))))}
\]

I.e. the plan (term) would be

\[
\langle \text{go(super_market), buy(milk) } \ldots \rangle
\]

However, the following plan is also correct:

\[
\langle \text{go(super_market), buy(milk), drop(milk), buy(milk), } \ldots \rangle
\]

but in general too inefficient; the search space is too large!

→ Specialized inference system for limited representation.
→ Planning algorithm
The STRIPS Formalism

STRIPS: Standard Research Institute Problem Solver (early 70s)

The system is obsolete, but the formalism is still used.

World State (including initial state): Set of ground atoms, no function symbols except for constants, interpreted under CWA (sometimes also standard interpretation, i.e. negative facts must be 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

Actions 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

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

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

Searching in the State Space

We can now search through the state space (the set of all states) — 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, and usually more efficient, since the branching factor is lower.

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:

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

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 \( \chi = i \), where \( \chi \) is a variable and \( i \) 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:

\[
\forall S_i \text{ with } c \in \text{Precond}(S_i) \text{ and } \\
\exists S_j \text{ with } S_i \prec S_j \text{ and } c \in \text{Effects}(S_j) \text{ and } \\
\text{for every linearization of the plan:} \\
\forall S_k \text{ with } S_j \prec S_k \prec S_j \rightarrow c \not\in \text{Effects}(S_k).
\]

Consistent Plan:

- if \( S_j \prec S_k \), then \( S_j \prec S_k \), and
- if \( \chi = \alpha \), then \( \chi \neq \beta \) for distinct \( \alpha \) and \( \beta \) for a variable \( \chi \) (Unique Name Assumption!)

A complete, consistent plan is called a solution to a planning problem.

Example

\[
\text{Start} \rightarrow \text{At(Home)} \rightarrow \text{Sells(SM,Banana)} \rightarrow \text{Sells(SM,Milk)} \rightarrow \text{Sells(HWS,Drill)} \\
\text{Have(Drill)} \rightarrow \text{Have(Milk)} \rightarrow \text{Have(Banana)} \rightarrow \text{At(Home)} \\
\]

Actions:

\[
\text{Op(A): Go(here),} \\
\text{Op(A): Bu(\chi),} \\
\text{Op(A): Prep(At(here) \land \text{Path(here, there)}, \land \text{Prep(At(here) \land \text{Sale(store, \chi)} \land \text{Effect:Have(\chi)})}}} \\
\text{there, here, \chi: store are variables.}
\]
Plan Refinement (1)

Regression Planning: Fulfills 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 backwards 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 \( At(x) \) in \( go(SM) \), choose \( x = HWS \) (with causal relationship)
- **Note:** This threatens the purchase of drill → promotion of \( go(SM) \).

![Diagram of the plan refinement process](image)

---

The Complete Solution

![Diagram of the complete solution process](image)

---

The POP Algorithm

```plaintext
function POP(initial goal, operators) returns plan
  plan = make-MovelessPlan(initial goal)
  keep do
    if solution-plan then return plan
    \( S_{dom} \), \( c \) = select-NLMove(plan)
    choose-ORIGIN-plan(operators, \( S_{dom} \), \( c \))
    REORDER-THINGS(plan)
  end

function SELECT-NLMove(plan) returns \( S_{dom} \), \( c \)
  pick a plan step \( S_{dom} \) from \( STOP(\) plan) with a precondition that has not been achieved
  return \( S_{dom} \), \( c \)

procedure choose-ORIGIN-plan(operators, \( S_{dom} \), \( c \))
  choose a step \( S_{dom} \) from operators of \( STOP(\) plan) that has \( c \) as an effect
  if there is no such step then fail
  add the current \( S_{dom} \) \( \rightarrow \) \( S_{dom} \) to \( ENSU(\) plan)
  add the preceding event \( S_{dom} \) \( \rightarrow \) \( S_{dom} \) to \( ORDERING(\) plan)
  If \( c \) is a newly added step from \( S_{dom} \) from \( STOP(\) plan) then
    add \( S_{dom} \) \( \rightarrow \) \( S_{dom} \) to \( ORDERING(\) plan)

procedure REORDER-THINGS(plan)
  for each \( S_{dom} \) that threatens a \( ENSU(\) \( \rightarrow \) \( S_{dom} \) \) to \( ORDERING(\) plan) do
    choose other
    Promotion: Add \( S_{dom} \) \( \rightarrow \) \( ORDERING(\) plan)
    Deletion: Add \( S_{dom} \) \( \rightarrow \) \( ORDERING(\) 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 or depth-first-search is used, the algorithm finds a solution, given one exists.

**Systematization:** Two distinct partial plans do not have the same total ordered plans as a refinement, is the partial plans are not refinements of one another (and total ordered plans contain causal relationships).

→ Instantiation of variables is not addressed.
Variables

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

- through an equality constraint, e.g. \( x = \text{HWS} \), so as not to threaten \( \mathcal{A}(\text{SM}) \);
- through an inequality constraint (language extension), e.g. \( x \neq \text{SM} \) (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

\begin{verbatim}
procedure ChainOnObject(object o, operator oOp) 
    choose a new oOp from operators on STRIPS plan (that has \( x \) as an effect 
    each but \( x \) = \( \neg \neg \mathcal{A}(\text{op}) \) (\text{STRIPS plan})
    if there is no such way 
        then fail 
        add \( x \) to BEGINING plan
        add \( \neg x \) to ENDING plan
        add \( \mathcal{A}(\text{op}) \) to ORIENTATION plan
        If \( \mathcal{A}(\text{op}) \) is newly added step from operator then 
            add \( \neg \neg x \) to \( \mathcal{A}(\text{op}) \) in BEGINING plan
        \end
procedure REVERSE(TRAN/) 
for each \( \neg \mathcal{A}(\text{op}) \) in \( \mathcal{A}(\text{op}) \) in BEGINING plan do 
    for each \( \neg \neg x \) in \( \mathcal{A}(\text{op}) \) in BEGINING plan do 
        if \( \neg \neg x \) in BEGINING plan 
            then fail 
            choose plan 
            Procedure: Add \( \neg \neg x \) in \( \mathcal{A}(\text{op}) \) in BEGINING plan 
            Generation: Add \( \neg \neg x \) in \( \mathcal{A}(\text{op}) \) in BEGINING plan 
            if \( \neg \neg x \) in \( \mathcal{A}(\text{op}) \) in ENDING plan 
                then \neg \neg \neg \neg 
        end
    end
end
\end{verbatim}

Works if the initial state contains no variables, and every operator uses all its variables in its precondition. Otherwise we must change the 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:** \( \text{move}(x;y;z) \), \( \text{stack}(x;y) \), \( \text{unstack}(x;y) \), ...