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

```plaintext
function START-P-PLANNING-AGENT(percept) returns action
    static: KB, a knowledge base (includes action descriptions)
    p, a plan, initially NoPlan
    t, a counter, initially 0, indicating time
    local variables: G, a goal
    current, a current state description
    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    current ← STATE-DESCRIPTION(KB, t)
    if p = NoPlan then
        G ← ASK(KB, MAKE-GOAL-QUERY(t))
        p ← IDEAL-PLANNER(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
```

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:

\[
\neg \text{At}(\text{Home}, 1) \land \neg \text{Have(\text{milk}, 1)} \land \neg \text{Have(\text{banana}, 1)} \land \neg \text{Have(\text{drill}, 1)}
\]

Operators (successor-state axioms):

\[
\forall a, s. \text{Have}(\text{milk}, a, 1) \iff (a = \text{buy(\text{milk})} \land \neg \text{Pass(\text{buy(\text{milk}), 1})} \lor \neg \text{Have(\text{milk}, 1)} \land a \neq \text{drop(\text{milk})})
\]

Goal conditions (query):

\[
\exists s. \text{At}(\text{home}, 1) \land \text{Have(\text{milk}, 1)} \land \text{Have(\text{banana}, 1)} \land \text{Have(\text{drill}, 1)}
\]

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:

\[
\text{do(\text{go(\text{home})}), do(\text{buy(\text{drill})}), do(\text{go(\text{hardware store})}), do(\text{buy(\text{banana})}), do(\text{buy(\text{milk})}), do(\text{go(\text{supermarket})}, 10)\ldots)}
\]

I.e. the plan (term) would be

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

However, the following plan is also correct:

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

In general, planning by theorem proving is very inefficient

Specialized planning algorithm 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.

STRAIPS 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!).

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

- An **action** is an operator, where all variables have been **instantiated**:
  
- \( \text{Op}(\text{Action: Go(SuperMarket)}, \text{Precond: } At(Home) \land Fath(Home, SuperMarket), \text{Effect: } At(Supermarket) \land \neg At(Home)) \)

- 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:

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

\[ \forall S_j \forall \chi \in \text{Precond}(S_j): \exists S_i \text{ with } S_i \prec S_j \text{ and } \chi \in \text{Effects}(S_i) \text{ and } \forall S_k \text{ with } S_k \prec S_i \forall \tau \not\in \text{Effect}(S_k). \]

Consistent Plan:
if \( S_i \prec S_j \), then \( S_i \neq S_j \), and
if \( \chi = \alpha \), then \( \chi \neq \alpha \) for distinct \( \alpha \) and \( \beta \) for a variable \( \chi \) (unique name assumption = UNA)

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

Example

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

\[ \text{Have(Drill)} \quad \text{Have(Milk)} \quad \text{Have(Banana)} \quad \text{At(Home)} \]

Actions:
\[ \text{Op(\text{Action}: \text{Go}(\text{here})), Precond(\text{At}(\text{here}) \land \text{Path}(\text{here}, \text{there})), Effect(\text{At}(\text{there}) \land \neg \text{At}(\text{here})))} \]
\[ \text{Op(\text{Action}: \text{Buy}(\chi), Precond(\text{At}(\text{store}) \land \text{Sells}(\text{store}, \chi), Effect(\text{Have}(\chi)))} \]

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 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 $\alpha \cup \theta$ in $\phi_{SM}$, choose $\alpha = \text{HWS}$ (with causal relationship)
- **Note:** This threatens the purchase of the drill $\rightarrow$ promotion of $\phi_{SM}$.

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
    insert new operator in the front of operators
    choose-OPERATOR(plan, operators, $\alpha_{new}$)
    REFINING-THREATS(plan)
  end

function SELECT-STEP(plan) returns $\alpha_{new}$
  pick a plan step $\alpha_{new}$ from $\phi_{SM}$(plan)
  with a precondition that has not been achieved
  return $\alpha_{new}$

procedure CHOOSE-OPERATOR(plan, operators, $\alpha_{new}$)
  choose a step $\alpha_{new}$ from operators of $\phi_{SM}$(plan) that has $\alpha_{new}$ as an effect
  if there is no such step then fail
  add the current step $\alpha_{new}$ to $\phi_{SM}$(plan)
  add the pre condition $\alpha_{new}$ to ORDERING plan
  if $\alpha_{new}$ is a newly added step then operators
  else $\alpha_{new}$ to 30(H) plan
  add $\alpha_{new}$ to 30(H) plan

procedure REFINING-THREATS(plan)
  for each $\alpha_{new}$ that threatens a step $\beta_{1}, \ldots \beta_{n}$ to ORDERING plan do
    if $\alpha_{new}$ then fail
    if not CONFLICT?(plan) then fail
```

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).

$\rightarrow$ Instantiation of variables is not addressed.
Variables

If a variable appears in the literal of an effect, for ex: $\neg \alpha(x)$, 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 $\alpha(x,y)$;
- 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 for treating variables](image)

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**: \(\text{move}(x, y, z)\), \(\text{stack}(x, y)\), \(\text{unstack}(x, y)\), ...

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**.