The Frame Problem in Situation Calculus

Holds(Result(Put_On(Block A, Block B), Situation_0), Black(Block A))?

Intuitively we know that moving an object Doesn’t effect its color, but in FOL we need to explicitly State that many actions do not effect many properties.

“Quick” fix to the frame problem, frame axioms m properties and n actions how many frame axioms?
Introduction to Planning – Differences b/w Uninformed and Informed Search.

- Plan: a **sequence of steps** to achieve a goal.

- Problem solving agent knows: **Actions, states, goals** and **plans**.

- Planning is a **special case** of problem solving: reach a **state** satisfying the **requirements** from the current state using **available actions**.
Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*
Standard search algorithms seem to fail miserably:

After-the-fact heuristic/goal test inadequate
Search vs. planning contd.

Planning systems do the following:
1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

<table>
<thead>
<tr>
<th></th>
<th>Search</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>Java data structures</td>
<td>Logical sentences</td>
</tr>
<tr>
<td>Actions</td>
<td>Java code</td>
<td>Preconditions/outcomes</td>
</tr>
<tr>
<td>Goal</td>
<td>Java code</td>
<td>Logical sentence (conjunction)</td>
</tr>
<tr>
<td>Plan</td>
<td>Sequence from $S_0$</td>
<td>Constraints on actions</td>
</tr>
</tbody>
</table>
Examples of Planning Systems

- Spacecraft assembly, integration and verification
- Job shop scheduling
- Space mission scheduling
- Building construction
- Operations on a flight deck of an aircraft carrier
- For demos: blocks world
Blocks World

- **PickUp(X)**
  - X on table, hand empty, X free

- **PutDown(X)**
  - X in hand

- **Stack(X,Y)**
  - X in hand, y free

- **Unstack(X,Y)**
  - X free, X on Y, hand free
Blocks World (cont)
Assumptions of the "Standard" AI Planning Paradigm

- There is a **single causal agent** and this agent is the planner.

- The planner is given a **well-defined goal** which remains **fixed** over the course of planning.

- The planner is assumed to have **functionally complete** and **accurate knowledge** of the starting situation.

- The planner is assumed to **possess** the **knowledge** required to accurately model the world.

- The planner is assumed to **possess** the **resources** (time and memory) required to use this model to reason about the possible worlds associated with different courses of action that might be pursued.
STRIPS - Linear Planner

- First planner developed by SRI, stands for STanford Research Institute Problem Solver.

- In STRIPS notation, a model of the world is just a list of variables free atomic propositions that hold in the world.

- Operators involving variables are called operator schemas.

- The following expresses an initial state in the block world:
  \(<\text{on(a,t)}, \text{on(b,a)}, \text{clear(b)}, \text{on(c,t)}, \text{clear(c)}>)\>

- Don’t allow negative facts i.e. NOT\text{clear(b)}. Why? What’s this called.
The description of the goal state is again a list of atomic proposition where all variables are interpreted existentially.

The goal state of plan, for example, will be given by such a description (if we want an apple we usually do not refer to a particular apple). An example goal state in the block world is:

<on(X,c), on(c,t)>

This means that some block should be on c, which is itself directly on the table.
STRIPS (cont)

- The main element of the language is the operator description, which has **three parts**:
  1. The **action** name, which may be parametrized
  2. The **precondition**, which is a conjunction of positive literals
  3. The **effect**, which is a conjunction of positive and/or negative literals

- The Preconditions consist of a **conjunctive logical expression** which is intended to describe the conditions that must be true in order to apply the operator.

- The **positive or additions** consist of a set of expressions that must be **added** to a model of the situation if the operator is applied.

- The **negative or deletions** consist of a set of expressions that must be **deleted** from a model of a situation if the operator is applied.
STRIPS (cont)

Figure 10.1. STRIPS representation for (1) opening a door (2) closing a door.

Why do we need deletions and additions?
STRIPS (cont)

Figure 10.1. STRIPS representation for (1) opening a door (2) closing a door.

Why do we need deletions and additions?
STRIPS (cont)

STRIPS representation for (1) opening a door (2) closing a door.

Why do we need deletions and additions?
STRIPS Example

Example of STRIPS Planning (Operator Schema & Initial Model).
STRIPS Example

Example of STRIPS Planning (Operator Schema & Initial Model).
STRIPS Example (cont)

Goal Wf's

\[ \text{G0: (3x) [BOX(x) \land INROOM(x,R1)]} \]
\[ \text{G1: INROOM(BOX,r1) \land INROOM(ROBOT,r1) \land CONNECTS(d,r1,R1)} \]
\[ \text{G2: INROOM(ROBOT,r1) \land CONNECTS(d,r1,R2)} \]

Plan

\[ \text{GOTHRU(D1,R1,R2)} \]
\[ \text{PUSHTHRU(BOX1,D1,R2,R1)} \]

M1: INROOM(ROBOT,R2)
CONNECTS(D1,R1,R2)
CONNECTS(D2,R2,R3)
BOX(Box1)
INROOM(Box1,R2)

\[
(\forall x \forall y \forall z) [\text{CONNECTS}(x,y,z) \Rightarrow \text{CONNECTS}(x,z,y)]
\]

M2: INROOM(ROBOT,R1)
CONNECTS(D1,R1,R2)
CONNECTS(D2,R2,R3)
BOX(Box1)
INROOM(Box1,R1)

\[
(\forall x \forall y \forall z) [\text{CONNECTS}(x,y,z) \Rightarrow \text{CONNECTS}(x,z,y)]
\]
Example STRIPS problem

**Init**($\text{At}(C_1, SFO) \land \text{At}(C_2, JFK) \land \text{At}(P_1, SFO) \land \text{At}(P_2, JFK)$
$\land \text{Cargo}(C_1) \land \text{Cargo}(C_2) \land \text{Plane}(P_1) \land \text{Plane}(P_2)$
$\land \text{Airport}(JFK) \land \text{Airport}(SFO)$)

**Goal**($\text{At}(C_1, JFK) \land \text{At}(C_2, SFO)$)

**Action**(Load($c$, $p$, $a$),

**PRECOND**: $\text{At}(c, a) \land \text{At}(p, a) \land \text{Cargo}(c) \land \text{Plane}(p) \land \text{Airport}(a)$

**EFFECT**: $\neg \text{At}(c, a) \land \text{In}(c, p)$)

**Action**(Unload($c$, $p$, $a$),

**PRECOND**: $\text{In}(c, p) \land \text{At}(p, a) \land \text{Cargo}(c) \land \text{Plane}(p) \land \text{Airport}(a)$

**EFFECT**: $\text{At}(c, a) \land \neg \text{In}(c, p)$)

**Action**(Fly($p$, from, to),

**PRECOND**: $\text{At}(p, \text{from}) \land \text{Plane}(p) \land \text{Airport}(\text{from}) \land \text{Airport}(\text{to})$

**EFFECT**: $\neg \text{At}(p, \text{from}) \land \text{At}(p, \text{to})$)
Expressiveness of STRIPS

STRIPS (STanford Research Institute Problem Solver) provides a “cut-down” first-order logic representation for planning:

- Preconditions and effects must be function-free
- Closed world assumption: Unmentioned literals are false
- Effect $P \land \neg Q$ : add $P$ and delete $Q$
- Only positive literals in states e.g.: $Poor \land Unknown$
- Only ground literals in goals e.g.: $Rich \land Famous$
- Goals and effects are conjunctions e.g.: $Rich \land Famous$
- No support for equality e.g. $x = y$ is not allowed

Note, the closed-world assumption avoids the frame problem

As an example, consider the following air transport problem involving loading and unloading cargo onto and off planes and flying it from place to place:
Planning as state-space search

Forward state-space search: As each action has an effect, planning can be solved using state-space search algorithms

This requires the following components:

An initial state: given in the STRIPS definition
Actions applicable for any given state that specify the successor state:
    given by the STRIPS action effects
A goal test: given in the STRIPS definition
A step cost: each action is given a cost of one

Similarly, as each action has a precondition, we can search backwards from the goal using backward state-space search

In either case we can use heuristics to estimate the cost of reaching the goal and apply algorithms like $A^*$
Partially ordered plans

However, we can also attempt to find a plan by solving several sub-problems *simultaneously* and combining them - this can have the advantage of reducing the size of the search space and providing a more flexible answer:

*Partially ordered* collection of steps with

- *Start* step has the initial state description as its effect
- *Finish* step has the goal description as its precondition
- causal links from outcome of one step to precondition of another
- temporal ordering between pairs of steps

Open condition = precondition of a step not yet causally linked

A plan is complete iff every precondition is achieved

A precondition is achieved iff it is the effect of an earlier step and no possibly intervening step undoes it
Example

Start

At(Home)  Sells(HWS,Drill)  Sells(SM,Milk)  Sells(SM,Ban.)

Have(Milk)  At(Home)  Have(Ban.)  Have(Drill)

Finish
Example

Start

At(Home)  Sells(HWS,Drill)  Sells(SM,Milk)  Sells(SM,Ban.)

At(HWS)  Sells(HWS,Drill)

Buy(Drill)

At(x)

Go(SM)

At(SM)  Sells(SM,Milk)

Buy(Milk)

Have(Milk)  At(Home)  Have(Ban.)  Have(Drill)

Finish
Example
Planning process

Operators on partial plans:
   add a link from an existing action to an open condition
   add a step to fulfill an open condition
   order one step wrt another to remove possible conflicts

Gradually move from incomplete/vague plans to complete, correct plans

Backtrack if an open condition is unachievable or
if a conflict is unresolvable
function POP(initial, goal, operators) returns plan

    plan ← MAKE-MINIMAL-PLAN(initial, goal)
    loop do
        if SOLUTION?(plan) then return plan
        $S_{\text{need}}$, c ← SELECT-SUBGOAL(plan)
        CHOOSE-OPERATOR(plan, operators, $S_{\text{need}}$, c)
        RESOLVE-THREATS(plan)
    end

function SELECT-SUBGOAL(plan) returns $S_{\text{need}}$, c

    pick a plan step $S_{\text{need}}$ from STEPS(plan)
    with a precondition c that has not been achieved
    return $S_{\text{need}}$, c
procedure CHOOSE-OPERATOR(plan, operators, S\textsubscript{need}, c)

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

procedure RESOLVE-THREATS(plan)

for each $S\text{\_threat}$ that threatens a link $S_i \rightarrow c \rightarrow S_j$ in LINKS(plan) do
    choose either
        Demotion: Add $S\text{\_threat} \prec S_i$ to ORDERINGS(plan)
        Promotion: Add $S_j \prec S\text{\_threat}$ to ORDERINGS(plan)
    if not CONSISTENT(plan) then fail
end
Clobbering and promotion/demotion

A clobberer is a potentially intervening step that destroys the condition achieved by a causal link. E.g., Go(Home) clobbers At(Supermarket):

Demotion: put before Go(Supermarket)

Promotion: put after Buy(Milk)
Properties of POP

Nondeterministic algorithm: backtracks at choice points on failure:
  – choice of $S_{add}$ to achieve $S_{need}$
  – choice of demotion or promotion for clobberer
  – selection of $S_{need}$ is irrevocable

POP is sound, complete, and systematic (no repetition)

Extensions for disjunction, universals, negation, conditionals

Can be made efficient with good heuristics derived from problem description

Particularly good for problems with many loosely related subgoals
Example: Blocks world

"Sussman anomaly" problem

Start State

\[ \text{Clear}(x) \text{ On}(x,z) \text{ Clear}(y) \]

\[ \text{PutOn}(x,y) \]

\[ \neg\text{On}(x,z) \neg\text{Clear}(y) \]

\[ \text{Clear}(z) \text{ On}(x,y) \]

Goal State

\[ \text{Clear}(x) \text{ On}(x,z) \]

\[ \text{PutOnTable}(x) \]

\[ \neg\text{On}(x,z) \text{ Clear}(z) \text{ On}(x,\text{Table}) \]
Example contd.

- On(C, A) on(A, Table) Cl(B) on(B, Table) Cl(C)
- On(A, B) on(B, C)
Example contd.

START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

Cl(B) On(B,z) Cl(C)

PutOn(B,C)

On(A,B) On(B,C)

FINISH
Example contd.

START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

Finish

Example contd.

```
START
On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

On(C,z) Cl(C)
PutOnTable(C)

Cl(A) On(A,z) Cl(B)

PutOn(A,B)

Cl(B) On(B,z) Cl(C)

PutOn(B,C)

On(A,B) On(B,C)

FINISH
```

PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

PutOn(B,C) clobbers Cl(C) => order after PutOnTable(C)
Simple STRIPS BlocksWorld

Move(x,y,z) move x which is currently on y onto z

PC:
A:
D:

Start State

Goal State
Simple STRIPS BlocksWorld

Move(x, y, z) move x which is currently on y onto z

PC: \(PC: \text{On}(x, y) \land C(x) \land C(z)\)

A: \(D: \text{On}(x, y), C(z)\)

D: \(A: \text{On}(x, z), C(y), C(T)\)

Start State

\[
\begin{array}{c}
\text{B} \\
\text{C} \\
\text{A}
\end{array}
\]

Goal State

\[
\begin{array}{c}
\text{A} \\
\text{B} \\
\text{C}
\end{array}
\]

\[
\text{On}(C, A) \land \text{On}(A, T) \land \text{On}(B, T) \land C(B) \land C(C) \land C(T)
\]

\[
\text{On}(C, T) \land \text{On}(B, C) \land \text{On}(A, B) \land C(A) \land C(T)
\]
A Simple Plan

- Ontable (A)
- Ontable (B)
- On (D, A)
- On (C, D)
- Clear (C)
- Clear (B)
- Handempty

Init

- Ontable (B)
- Ontable (C)
- On (D, B)
- On (A, D)
- Clear (A)
- Clear (C)
- Handempty

- A
- D
- B
- C
Need Two Additional Rules

1. A block can always be moved onto the table, if it is clear.

\[ \forall x y [(\exists n(x, y) \in S) \land (C(x) \in S) \supset \text{Can}(\text{Move}(x, y, T))] \]

2. If anything, build towers bottom up, since we can only move one block at a time.

\[ \forall b_0, ..., b_n, y [(C(b_0) \land (\exists n(b_0, y) \in S) \land (y \neq b_1) \land (\exists n(b_0, b_1) \in G) \land (\exists n(b_0, b_1) \notin S) \land (\exists n(b_1, b_2) \in G \cap S) \land ... \land (\exists n(b_n, T) \in G \cap S)) \supset \text{Do}(\text{Move}(b_0, y, b_1))] \]