

Foundations of AI

Chapter 9. Planning

Situation Calculus, STRIPS, Partial-Order Planning

Bernhard Nebel

- Domain-Independent Planning
- Planning vs. Problem Solving and Program Synthesis
- Situation Calculus
- STRIPS Formalism
- Partial-Order Planning
- The POP Algorithm
- Variables in the POP Algorithm

Consider

- an **autonomous agent**, e.g., a robot
- taking **actions** in its environment
- trying to **perform some task**

The general problem is, given

- the **current state** of the world
- the **possible actions** to take
- the **goal**
where everything is specified *declaratively*,

try to figure out what to do, i.e.,

Find a sequence/set of actions that achieves the goal

Domain Independent Planning - Why do it ?

Declarative language can specify a large class of different problems

~> Domain independent planning systems are **generic problem solvers**

Useful in evolving systems:

- Everything **hand-coded** ⇒ Must be changed at **implementation level**
- Declarative **specification** ⇒ Changes can be made at **declarative level** (Example: Intelligent lift control, developed at Schindler, Switzerland)

Issues:

- **Efficiency:** Running time should be close to specialized solvers
- **Quality:** Solutions should be (close to) optimal

Planning vs. Problem Solving (and Search)

The main difference is **explicit, logic-based representation**:

- *States* are specified using logical formulae instead of using data structures.
 - This implies that the agent can explicitly reason and communicate about the world state
- The *goal condition* is specified using a logical formula instead of a goal test (black box)
 - The agent is able to reason about its goals
- *Operators*: Axioms or transformation of logical formulae instead of modification of data structures
 - This means the agent knows what the operators do.

Planning as Logical Inference (1)

Planning can be formalized as logical inference using the so-called *Situation Calculus*.

Main ideas

- There exist some special entities called *situations*.
- Logical atoms are true or false in situations (e.g. $At(home, s_0)$).
- Change in the world can be described by using so-called *Successor-State Axioms*, which formalize what changes and what stays unchanged.
- Planning reduces then to proving an existential formula and determining the term that is the witness.

Planning vs. Program Synthesis

- The world description is logic based (while programs manipulated arbitrary structures)
- Plans are (usually) simpler than programs, namely, linear sequences of action (but only with action determinism and full observability)
- Connected with that, we usually do not have iteration or recursion

Planning as Logical Inference (2)

Initial state:

$$At(home, s_0) \wedge \neg Have(milk, s_0) \wedge \neg Have(banana, s_0) \\ \wedge \neg Have(drill, s_0)$$

Operators (successor-state axioms):

$$\forall a, s [Have(milk, do(a, s)) \Leftrightarrow \{(a = buy(milk)) \\ \wedge Poss(buy(milk), s) \\ \vee Have(milk, s) \\ \wedge (a \neq drop(milk))\}]$$

Goal condition (query)

$$\exists s [At(home, s) \wedge Have(milk, s) \wedge Have(banana, s) \\ \wedge Have(drill, s)]$$

If all successor-state axioms and all preconditions (*Poss*) are spelled out, a proof of the existential formula leads to a term (for the variable *s*) that is the desired plan.

Planning as Logical Inference (3)

The variable binding for *s* could for instance be

$$do(go(home), \\ do(buy(drill), \\ do(go(hardware_store), \\ do(buy(banana), \\ do(buy(milk), \\ do(go(super_market), s_0))))))$$

This means the plan would be

$$\langle go(super_market), buy(milk), \dots \rangle$$

However, also this plan would be "correct":

$$\langle go(super_market), buy(milk), drop(milk), buy(milk), \dots \rangle$$

Usually, the search space is much too large for general theorem proving systems.

↪ **specialized inference systems** for limited representation languages

The STRIPS Formalism

STRIPS: STanford Research Institute Problem Solver

System is obsolete, but formalism is still used

State: Set of ground atoms without function symbols except constant symbols. These sets are interpreted under the CWA (closed world assumption).

Goal condition: Set of literals (perhaps with free variables that are interpreted as existentially quantified)

Note: There is no explicit notion of “state” or “situation” in the language.

Sometimes, one also considers *background theories* in STRIPS

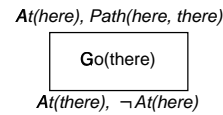
Operators are tuples consisting of

action names: Name with parameters (an in the situation calculus)

preconditions: Conjunction of literals that must be true in order to execute the operator successfully

effects: Conjunction of effect literals. Positive literals are added, negative ones are deleted.

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

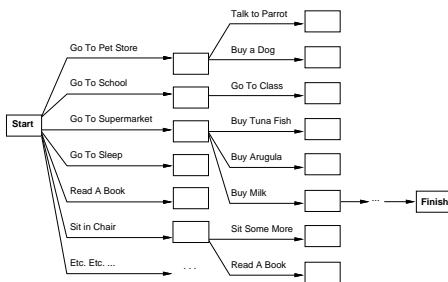


Operators are *schematic actions*. Actions are instantiated operators without any variables.

Search in the State Space

We could now search through the set of states in order to find a path from the initial state to a state satisfying the goal condition – reducing planning to search.

One possibility is to a forward search (progression planning):



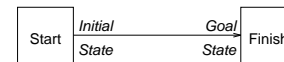
Alternatively, we could start from the goals and search backwards (regression planning).

Possible because the operators contain enough information. Often, regression planning is more efficient because of the smaller branching factor. However, with heuristics we could guide the forward search ...

Plan Space

Instead of searching in the state space, we could also search in the space of possible (partial) plans.

The starting point is then a **partial plan** which contains only a start and finish operator, which code the initial state and the goal condition:



The goal is a **complete plan** which solves the problem



Transformations in plan space:

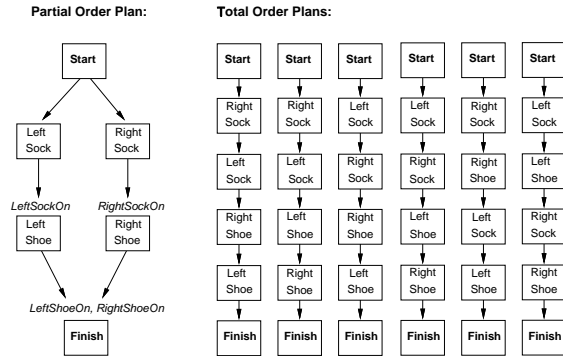
Refinements make the plan more concrete and complete **modifications** which modify the plan

- in the following only refinements

Plan = Sequence of Actions?

Often it is not necessary or reasonable to commit to a particular order of execution:

~> **non-linear** or **partially ordered** plans (*least-commitment planning*)



Foundations of AI/Summer'02

Chapter 9. Planning * Situation Calculus, STRIPS, Partial-Order Planning - 13

Partially Ordered Plans

One plan step = STRIPS operator with a label

A **plan** consists of

- a set of **plan steps** with a partial order (\prec) on them, where $S_i \prec S_j$ means that S_i must be executed before S_j ;
- a set of **variable bindings** $x = t$, where x is a variable and t is a term (constant or variable);
- a set of **causal links**: $S_i \xrightarrow{c} S_j$ means " S_i satisfies the precondition c of S_j " (this implies $S_i \prec S_j$)

Solutions for a planning problem should be **complete** and **consistent**.

Foundations of AI/Summer'02

Chapter 9. Planning * Situation Calculus, STRIPS, Partial-Order Planning - 14

Completeness and Consistency of Plans

Complete plan:

Each precondition of each step is satisfied:

$\forall S_j, c$ with $c \in \text{Precond}(S_j)$ implies
 $\exists S_i$ with $S_i \xrightarrow{c} S_j$ and for all linearizations (\prec is an total order embedding \prec):

$\forall S_k$ with $S_i \prec S_k \prec S_j, -c \notin \text{Effect}(S_k)$.

Consistent plan:

$\forall S_i, S_j$:

if $S_i \prec S_j$, then $S_j \not\prec S_i$ and

if $x = A$, then $x \neq B$ for different A and B

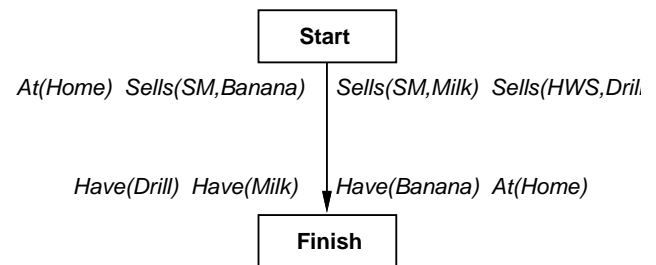
for one var. x (Unique Names Assumption!)

A *complete and consistent plan* solves the planning problem – because every linearization is a solution.

Foundations of AI/Summer'02

Chapter 9. Planning * Situation Calculus, STRIPS, Partial-Order Planning - 15

Example



Operators:

Op (Action: *go(there)*,
 Precond: *At(there)*,
 Effect: $At(there) \wedge \neg At(there)$)

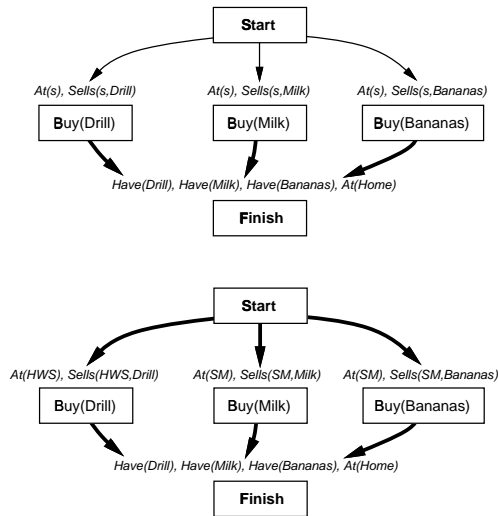
Op (Action: *buy(x)*,
 Precond: $At(store) \wedge Sells(store, x)$,
 Effect: *Have(x)*)

Foundations of AI/Summer'02

Chapter 9. Planning * Situation Calculus, STRIPS, Partial-Order Planning - 16

Refinement (1)

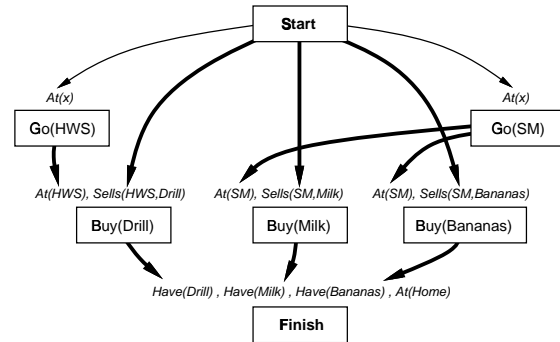
Regression: Try to satisfy the *Have* predicates ... after instantiating the variables:



thin arrows = \prec
fat arrows = causal links (\rightarrow)

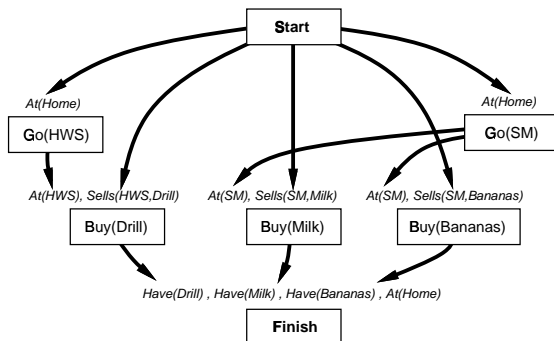
Refinement (2)

Go to the right shops ...



Refinement (3)

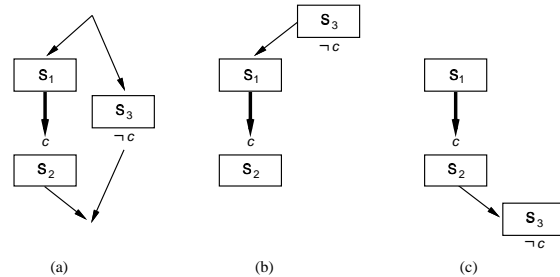
First go to the shops ...



Note: No search yet but only backchaining

Now: *Conflict!* After doing *go(hws)*, we are not any longer *At(home)*. The same holds for *go(sm)*.

Protecting Causal Links



a) Conflict: S_3 threatens the causal link between S_1 and S_2

Conflict resolution:

b) Promotion: the causal link pair is moved to a place after the threat

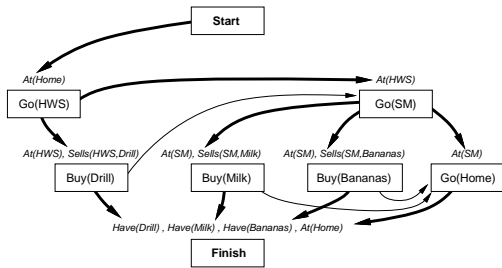
c) Demotion: the causal link pair is moved before the threat

Another Refinement ...

- We cannot resolve the conflict by demotion or promotion

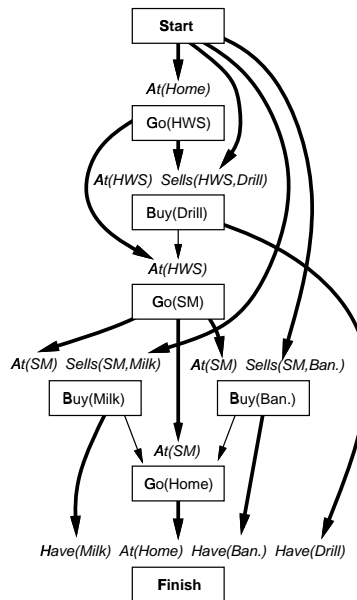
~ The refinement decision was **wrong**.

- **Alternative:** When instantiating $At(x)$ in $go(sm)$ we choose $x = hws$ (with the appropriate causal link)
- **Note:** $go(sm)$ threatens now the link between $go(hws)$ and $buy(drills)$ ~ demotion.



The Complete Solution ...

... with all links



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 Sadd from operators or STEPS(plan) that has c as an effect
  if there is no such step then fail
  add the causal link Sadd →c Sneed to LINKS(plan)
  add the ordering constraint Sadd → Sneed to ORDERINGS(plan)
  if Sadd is a newly added step from operators then
    add Sadd to STEPS(plan)
    add Start → Sadd → Finish to ORDERINGS(plan)

procedure RESOLVE-THREATS(plan)
  for each Sthreat that threatens a link Si →c Sj in LINKS(plan) do
    choose either
      Promotion: Add Sthreat → Si to ORDERINGS(plan)
      Demotion: Add Sj → Sthreat to ORDERINGS(plan)
    if not CONSISTENT(plan) then fail
  end
  
```

choose ... fail: Nondeterminism (implemented by backtracking)
pick: don't care nondeterminism
Solution? checks completeness and consistency

Properties of the POP Algorithm

Soundness: Each return value of the POP algorithm is a complete and consistent plan

Completeness: If the search through the nondeterministic choices is implemented in a breadth-first search fashion, the algorithm finds a solution if one exists.

Systematicity: Two different partial plans do not have identical linear refinements, provided the partial plans are not refinements of each other

- Instantiation of variables are not dealt with yet

Variables

If a variable appears in an effect literal, e.g. $At(x)$, then the variable is a potential threat to causal links with the same predicate symbol.

Possible conflict resolution

- by *equality constraint*, e.g. $x = hws$ in order to avoid a threat to $At(sm)$;
- by *inequality-constraints*, e.g. $x \neq sm$ (but is tricky);
- deferring conflict resolution to a point when all variables are instantiated.

We use the latter alternative.

Variables

```

procedure CHOOSE-OPERATOR(plan, operators, Sneed c)
  choose a step Sadd from operators or STEPS(plan) that has cadd as an effect
  such that  $it = UNIFY(c, c_{add}, BINDINGS(plan))$ 
  if there is no such step
  then fail
  add it to BINDINGS(plan)
  add  $S_{add} \xrightarrow{c} S_{need}$  to LINKS(plan)
  add  $S_{add} \prec S_{need}$  to ORDERINGS(plan)
  if Sadd is a newly added step from operators then
  add Sadd to STEPS(plan)
  add  $Start \prec S_{add} \prec Finish$  to ORDERINGS(plan)

procedure RESOLVE-THREATS(plan)
  for each  $S_i \xrightarrow{c} S_j$  in LINKS(plan) do
  for each Sthreat in STEPS(plan) do
  for each c' in EFFECT(Sthreat) do
  if  $SUBST(BINDINGS(plan), c) = SUBST(BINDINGS(plan), -c')$  then
  choose either
    Promotion: Add  $S_{threat} \prec S_j$  to ORDERINGS(plan)
    Demotion: Add  $S_j \prec S_{threat}$  to ORDERINGS(plan)
  if not CONSISTENT(plan)
  then fail
  end
end
end
  
```

Works if all operators use the effect variables in the preconditions.

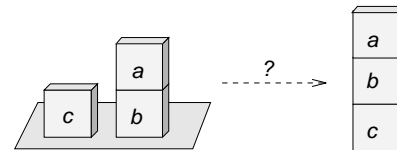
Modeling using STRIPS

The following steps are necessary when one wants to model/formalize a planning problem (similarly to other areas in AI):

- Deciding about the domain of discourse
- Introducing a vocabulary for the relevant objects, predicates, and operators
- Specifying the operators
- Specifying problem instances

After that, a planning system can solve the problem

Example: Blocks World



- We have named blocks and the table;
- we can place as many blocks as we want on the table;
- only one block can be on top of another block;
- only one block can be under another block;
- a block can only be moved if there is no block on top of it;
- it is not allowed to destroy stack etc.

Formalization

- Talk only about blocks, the table is implicit

~ **constants** (denoting blocks): a, b, c

- One represents explicitly whether a block can be moved and whether it sits on the table

~ **predicates:**

- $On(x, y)$: x is on top of y
- $OnTable(x)$: x sits on the table
- $Clear(x)$: there is nothing on top of x

- Operators to move blocks

~ **operators:** $move(x, y, z)$, $stack(x, y)$,
 $unstack(x, y) \dots$

Summary

- Planning is more flexible and declarative than problem-solving/search approaches.
- In principle, we could reduce planning to **logical inference** (= *situation calculus*) but this is very inefficient.
- Instead of searching in the state space, one can also search in the plan space.
- The principle of (*least commitment*) says that one should commit to a choice only when one is forced to do so.
- **Partial order planning** is an example for the application of this principle
- The **POP algorithm** implements such a planning method.