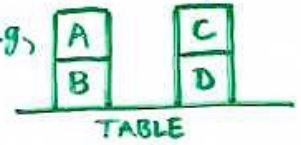
STATES, CHANGE, & THE FRAME PROBLEM

- Most practical domains are not static
- Intelligent agents need to reason about change, & plan actions that cause change

Situation Calculus (McCarthy & Hayes, ca '63, '69)

- Introduce situation arguments (states) into "fluent" (changable) predicates; similar to Davidsonian event arguments, but more like "snapshots" of current world state, than events.

e.g., 

$On(A,B,S_0)$, $Clear(A,S_0)$, $On(B,TABLE,S_0)$, etc.

- View actions as mapping one situation into another; view action types as abstract individuals formed by an action function applied to the individuals involved.

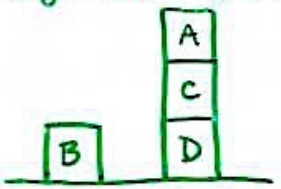
e.g., action type $m(A,B,C)$: move A from B to C

state change: mediated by do (or result) function

$do(m(A,B,C), S_0) = S_1$ — new situation (state)

Big advantage (as we'll see): plans are terms $do(\ldots, do(\ldots))$ & can be deduced by C. Green's "answer extraction".

Things are different in the new situation:



$On(B,TABLE,S_1)$, $Clear(B,S_1)$, $On(A,C,S_1)$, $\neg Clear(C,S_1)$, etc.

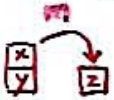
Could also say: $On(B,TABLE,do(m(A,B,C),S_0))$, $Clear(B,do(m(A,B,C),S_0))$, etc.

- Axiomatizing change: effect axioms.

"Given certain preconditions, if we do a certain type of action, then such-and-such conditions hold in the resultant state".

e.g., (taking all free variables to be $\forall$-quantified)

$$On(x,y,s) \wedge Clear(x,s) \wedge Clear(z,s) \wedge x \neq z$$



$$\wedge\, s' = do(m(x,y,z),s) \Rightarrow \underbrace{On(x,z,s') \wedge Clear(y,s')}_{\text{effects}} \wedge \neg Clear(z,s')$$

For instance, Let $S_1 = do(m(A,B,C), S_0)$

Then we easily derive

$$On(A,C,S_1) \wedge Clear(B,S_1) \wedge \neg Clear(C,S_1)$$

But what about $On(B,TABLE,S_1)$, $On(C,D,S_1)$, $Clear(A,S_1)$, $On(D,TABLE,S_1)$? Not to mention $Blue(B,S_0)$ becoming $Blue(B,S_1)$, etc.

- So we also need axioms about what doesn't change...

$Clear(x,s) \wedge s' = do(m(u,v,w),s) \wedge x \neq w \Rightarrow Clear(x,s')$

$On(x,y,s) \wedge s' = do(m(u,v,w),s) \wedge x \neq u \Rightarrow On(x,y,s')$

$Blue(x,s) \wedge s' = do(m(u,v,w),s) \Rightarrow Blue(x,s')$

↳ more generally could use

$Color(x,y,s) \wedge s' = do(m(u,v,w),s) \Rightarrow Color(x,y,s')$

↑ e.g., Blue... an individual (a color)

Problems: - too many things don't change!

$O(mn)$ axioms

no. of fluents, no. of actions

- doesn't allow concurrent actions

Frame Problem: how can we axiomatize (or otherwise express) non-change succinctly?

- Monotonic solutions:
  - quantified state approach (Kowalski)
  - explanation closure (Haas, Pednault, Schubert, Reiter)
  - histories (Hayes)
- Nonmonotonic solutions
  McCarthy, Reiter, Lifschitz, Baker, ... & STRIPS

Before we consider **other** approaches to the frame problem, we look more closely at deductive planning.

Simple example use $st(x,y)$ instead of $m(x,y,z)$ for simplicity

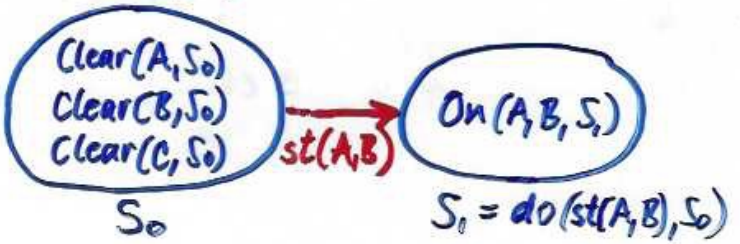
1. Stacking causes "On"

$\neg Clear(x,s) \vee \neg Clear(y,s) \vee On(x,y,do(st(x,y),s))$ — new state

2. (not needed)
3. $Clear(A,S_0)$
4. $Clear(B,S_0)$
7. denial of concl^{n} $\neg On(A,B,s') \vee Ans(s')$

Goal: $\exists s.\ On(A,B,s)$

8. $\neg Clear(A,s) \vee \neg Clear(B,s) \vee Ans(do(st(A,B),s))$
9. $\neg Clear(B,S_0) \vee Ans(do(st(A,B),S_0))$
10. $Ans(do(st(A,B),S_0))$



Suppose we also had:

5. On(B, TABLE, S_0)
6. Color(B, BLUE, S_0)

Prove: On(B, TABLE, do(st(A,B), S_0))
Clear(A, do(st(A,B), S_0))
Color(B, BLUE(do(st(A,B), S_0))

Frame axioms:

In a stack action st(x,y), the only On-relation that becomes false is the one for x:

$\neg On(u,v,s) \vee x=u \vee On(u,v,do(st(x,y),s))$

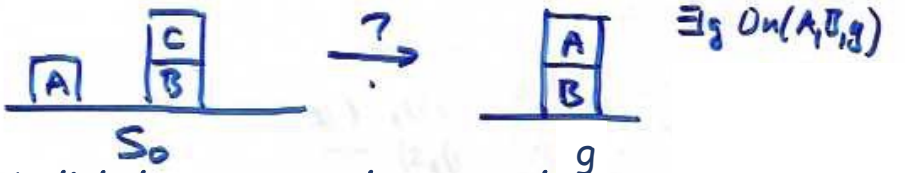
Similarly the only Clear-relation that becomes false is the one for y:

$\neg Clear(u,s) \vee y=u \vee Clear(u,do(st(x,y),s))$

Similarly the color relationships from states are preserved:

$\neg Color(u,v,s)^{color} \vee Color(u,v,do(st(x,y),s))$

We need similar axioms for relations that stay false.



A slightly more complex example

Frame problem arises at intermediate states, in general (preconditions of various moves):

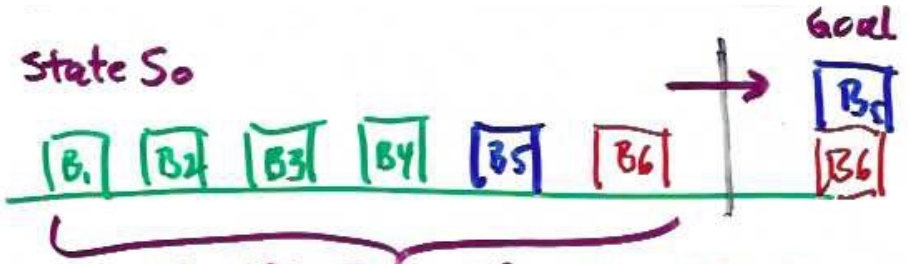
How do we know A is still clear after we've done st(C, Table)?

Outline of proof:

- resolve goal denial $\neg On(A,B,g)$ vs. st(x,y)-effects ⇝ B must be clear previously
- resolve denial that B is clear vs. st(x,y)-effect, where precondition is that x is on some z (which becomes clear) Also Table must be clear, etc
- ≡ resolve denial that A is clear vs. frame axiom for preservation of Clear property during st(x,y) ⇝ new goal that A was clear prior to that action
- resolve against initial state (where A is clear)

P.S.: Early proof algorithms were too inefficient, but later strategies were more practical (SofS, state alignment, unachievability pruning)

Frame Axioms vs. "Explanaton Closure" (EC) in deductive planning



$On(B_i, Table, S_0)$, $Clear(B_i, S_0)$

action function: $do(stack(x,y), s)$

$\forall x, y, s.\ (Clear(x,s) \wedge Clear(y,s)) \Rightarrow On(x, y, do(stack(x,y),s))$

Goal: $\exists s.\ On(B_5, B_6, s)$

To keep blocks $B_1 - B_5$ clear:

(1) $\forall x, y, s, z.\ (Clear(z,s) \wedge z \neq y) \Rightarrow Clear(z, do(stack(x,y),s))$

"Frame axiom"

(2) $\forall x, s, a.\ Clear(x,s) \wedge \neg Clear(x, do(a,s)) \Rightarrow \exists y.\ a = stack(y,x)$

"Explanation Closure"

(1) O(#actions) axioms/pred; (2) O(1) axioms/pred

STRIPS

"Shakey's" high-level planner/reasoner
(Fikes & Nilsson, AI 2 (3/4): 189-208, 1971)

Formalize actions in terms of operators with specified preconditions & effects

delete list, add list

STRIPS ASSUMPTION: All changes are specified by the del. & add lists (& possibly indirectly through state constraints)

We also assume a fully specified, unambiguous initial state (allowing use of CWA).

E.g., Operator for pushing an object:

push (k, m, n)
Precond: AT(k, m) ∧ ATR(m)
Del: AT(k, m), ATR(m)
Add: AT(k, n), ATR(n)

CSC 2V4/VV4

"Algorithm" STRIPS (S_0, G_0)

ignoring failure & backtracking! (i.e. nondeterministic version)

0. Initialize current plan P := nil; S := S_0;
   Initialize current state + goal list to (S_0, (G_0));
   (In general, this is (S, ($G_i, G_{i-1}, \ldots, G_0$))
   O_i, O_{i-1}, ... associated actions
1. Try to prove that goal G_i is already true in state S;
   ← typically, a conjunction of several conditions
2. If successful then
   - remove G_i from goal list
   - if $G_i \neq G_0$, an action instance O_i will be associated with G_i ... append this to P, i.e., P := P ⌢ O_i; also, let S := O_i(S), the result of applying operator instance O_i to the given state S;
   - if the goal list is now (S, ()) (no more goals), return plan P, else [shift G_{i-1} into the role of G_i] i := i-1 & go to 1;
3. If unsuccessful then
   - select an unresolvable literal (in effect, a subgoal) from the failed proof attempt;
   - find an operator instance, say O_{i+1} that has an effect matching the chosen unresolvable literal;
   - Let G_{i+1} be the (conjunction of) preconditions of operator instance O_{i+1}, let i := i+1 & go to 1.

nondeterministic steps

STRIPS solution of

Also try Goal: Above(A,B) where ¬On(x,y) ∨ Above(x,y), ¬Above(x,y) ∨ ¬Above(y,z) ∨ Above(x,z)

On(A,B) false by omission (CWA)

stack (x,y)
pre: Clear(x), Clear(y)
del: Clear(y)
add: On(x,y)

Init: Clear(A), Clear(B), Clear(C)

Goal: On(A,B)

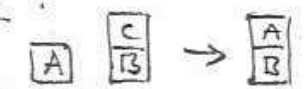
Initially, Plan := ( ), Initially Goal-list := (Init, (On(A,B)))

{Clear(A), Clear(B), Clear(C)}

1. Proof of On(A,B) in Init fails:
   ¬On(A,B) is unresolvable
2. On(A,B) matches On(x,y), therefore select stack(A,B); Goal-list := (Init, (G_1, On(A,B)))
   where G_1 = {Clear(A), Clear(B)} ↕ stack(A,B)
3. Proofs of Clear(A), Clear(B) in Init succeed, therefore Plan := (Stack(A,B)), Goal-list := (S, (On(A,B))), where now S = {Clear(A), Clear(C), On(A,B)}
4. Proof of On(A,B) in S succeeds trivially, so Goal-list := (S, ()), so return Plan = (Stack(A,B))

Try Allen's problem:

Assume Clear(Table) in Init, so that C can be "stacked" to the Table.

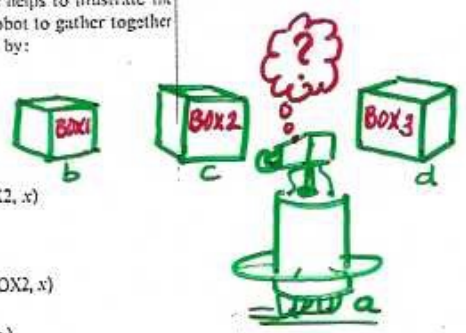


STRIPS in Action (supplementary)

3.5. An Example

Tracing through the main points of a simple example helps to illustrate the various mechanisms in STRIPS. Suppose we want a robot to gather together three objects and that the initial world model is given by:

$$M_0: \begin{Bmatrix} \text{ATR}(a) \\ \text{AT}(\text{BOX1}, b) \\ \text{AT}(\text{BOX2}, c) \\ \text{AT}(\text{BOX3}, d) \end{Bmatrix}.$$



The goal wff describing this task is

$$G_0: (\exists x)\,[\text{AT}(\text{BOX1}, x) \wedge \text{AT}(\text{BOX2}, x) \wedge \text{AT}(\text{BOX3}, x)].$$

Its negated form is

$$\sim G_0: \sim \text{AT}(\text{BOX1}, x) \vee \sim \text{AT}(\text{BOX2}, x) \vee \sim \text{AT}(\text{BOX3}, x).$$

(In $\sim G_0$, the term x is a universally quantified variable.)

We admit the following operators:

(1) *push* (k, m, n): Robot pushes object k from place m to place n.

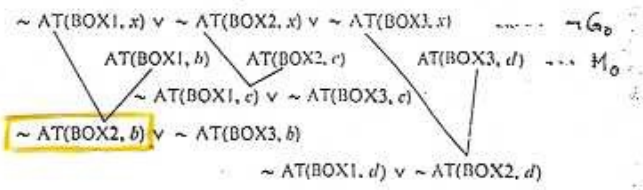
Precondition: $\text{AT}(k, m) \wedge \text{ATR}(m)$
Negated precondition: $\sim \text{AT}(k, m) \vee \sim \text{ATR}(m)$
Delete list: $\text{ATR}(m)$, $\text{AT}(k, m)$
Add list: $\text{AT}(k, n)$, $\text{ATR}(n)$

Note: parameters behave like constants wrt negation (but like variables in unification with ground terms!)

(2) *goto*(m, n): Robot goes from place m to place n.

Precondition: $\text{ATR}(m)$
Negated precondition: $\sim \text{ATR}(m)$
Delete list: $\text{ATR}(m)$
Add list: $\text{ATR}(n)$

Following the flow chart of Fig. 2, STRIPS first creates the initial node $(M_0, (G_0))$ and attempts to find a contradiction to $\{M_0 \cup \sim G_0\}$. This attempt is unsuccessful; suppose the incomplete proof is:



We attach this incomplete proof to the node and then select the node to have a successor computed.

The only candidate operator is push(k, m, n). Using the add list clause $\text{AT}(k, n)$, we can continue the uncompleted proof in one of several ways depending on the substitutions made for k and n. Each of these substitutions produces a relevant instance of push. One of these is:

OP_1: push(BOX2, m, b)

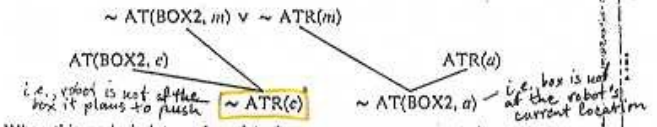
given by the substitutions BOX2 for k and b for n. Its associated precondition (in negated form) is:

$\sim G_1: \sim \text{AT}(\text{BOX2}, m) \vee \sim \text{ATR}(m)$.

(supplementary)

Suppose OP_1 is selected and used to create a successor node. (Later in the search process another successor using one of the other relevant instances of push might be computed if our original selection did not lead to a solution.) Selecting OP_1 leads to the computation of the successor node $(M_0, (G_1, G_0))$.

STRIPS next attempts to find a contradiction for $\{M_0 \cup \sim G_1\}$. The uncompleted proof (difference) attached to the node contains:



When this node is later selected to have a successor computed, one of the candidate operators is goto(m, n). The relevant instance is determined to be

OP_2: goto(m, c)

with (negated) precondition

$\sim G_2$: $\sim$ATR(m).

This relevant operator results in the successor node $(M_0, (G_2, G_1, G_0))$.

Nest STRIPS determines that $(M_0 \cup \sim G_2)$ is contradictory with $m = a$. Thus, STRIPS applies the operator goto(a, c) to M_0 to yield

$$M_1: \begin{Bmatrix} \text{ATR}(c) \\ \text{AT}(\text{BOX1}, b) \\ \text{AT}(\text{BOX2}, c) \\ \text{AT}(\text{BOX3}, d) \end{Bmatrix}.$$

The successor node is $(M_1, (G_1, G_0))$. Immediately, STRIPS determines that $(M_1 \cup \sim G_1)$ is contradictory with $n = c$. Thus, STRIPS applies the operator push(BOX2, c, b) to yield

$$M_2: \begin{Bmatrix} \text{ATR}(b) \\ \text{AT}(\text{BOX1}, b) \\ \text{AT}(\text{BOX2}, b) \\ \text{AT}(\text{BOX3}, d) \end{Bmatrix}.$$

Note: a parameter will match any term denoting an individual (eg, x, $f(a)$) but not a term involving a universally quantified variable

The resulting successor node is $(M_2, (G_0))$, and thus STRIPS reconsiders the original problem but now beginning with world model M_2. The rest of the solution proceeds in similar fashion.

Our implementation of STRIPS easily produces the solution {goto(a, c), push(BOX2, c, b), goto(b, d), push(BOX3, d, b)}. (Incidentally, Green's theorem-proving problem-solver [4] has not been able to obtain a solution to this version of the 3-Boxes problem. It did solve a simpler version of the problem designed to require only two operator applications.)

4. Example Problems Solved by STRIPS

STRIPS has been designed to be a general-purpose problem solver for robot tasks, and thus must be able to work with a variety of operators and with a world model containing a large number of facts and relations. This section

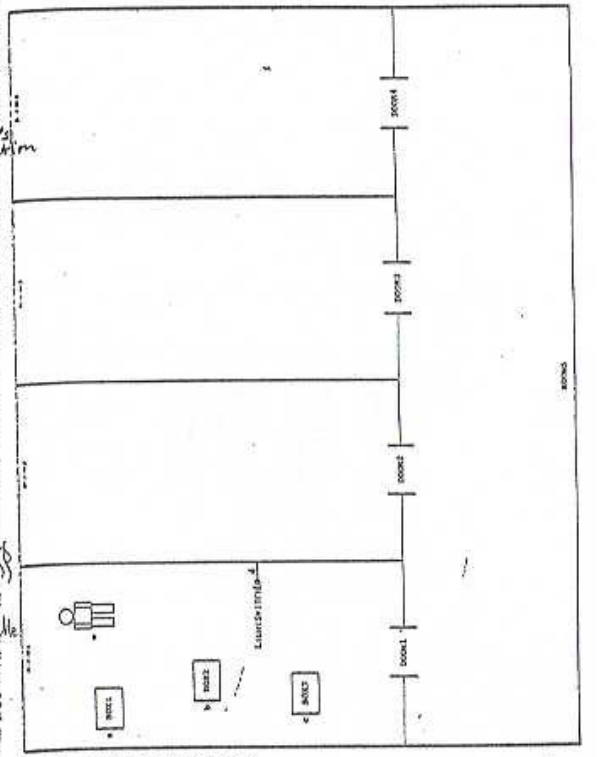
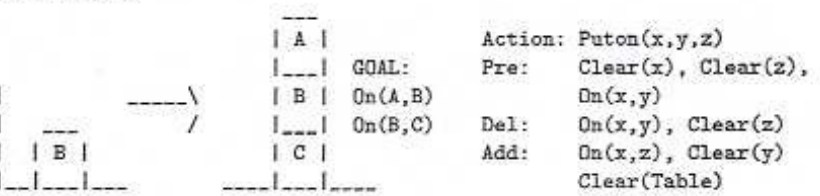


Fig. 3. Room plan for the robot tasks.

Some problems STRIPS has trouble solving

1. Sussman anomaly




```
Action: Puton(x,y,z)
Pre:    Clear(x), Clear(z),
        On(x,y)
Del:    On(x,y), Clear(z)
Add:    On(x,z), Clear(y)
        Clear(Table)
```

This is troublesome if STRIPS "protects" goals that have already achieved from subsequent change: it gets stuck after achieving just one the goals.

If it doesn't use goal protection, then it will undo the goal it achieved first, in the course of achieving the second.

2. Register exchange problem


```
Action: Copy(x,y,z)    ''copy contents x of register y to reg. z"
Pre:    Contains(y,x)
Del:    Contains(z,\$)
Add:    Contains(z,x)

Given:  Contains(R1,N1), Contains(R2,N2)

Goal:   Contains(R1,N2), Contains(R2,N1)
```

3. Matching socks problem

Here it's possible to get a solution, but the method seems awkward...

There is a box with many black and white socks in it. There is one operator, `Take-out-sock`, with no preconditions and add-list containing just

`Have-sock(!S), Black(!S)∨White(!S),`

where `!S` is replaced by a new constant `S1, S2, ...` whenever the operator is instantiated. We also assume unique names, i.e., `unequal(S1,S2,...)`. The goal is to have two socks of the same color; i.e.,

$(\exists x)$Have-sock(x) $\wedge$ $(\exists y)$Have-sock(y) $\wedge$ x $\neq$ y $\wedge$
[Black(x) $\wedge$ Black(y)] $\vee$ [White(x) $\wedge$ White(y)]

It seems more natural to use forward reasoning

From Push Singh's blog site

John McCarthy Push Singh Ed Fredkin Marvin Minsky

Some symbolic AI originators

BEYOND STRIPS

Hierarchical Planning

- Make a plan in terms of high-level (coarse, large-scale) actions & state descriptions
- Refine the steps of the high-level plan into lower-level subplans (when detailed circumstances become known)

Pay-off :
- Search space reduction !
- Don't have to know detailed circumstances in advance

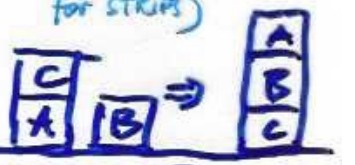
GRAPHPLAN

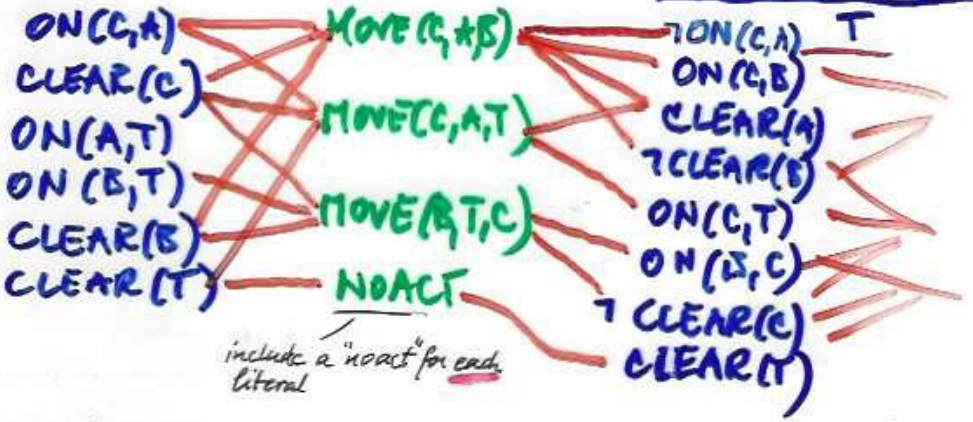
(Blum & Furst '95)

Forward & backward sweeps using

PLANNING GRAPH

("Sussman Anomaly" for STRIPS)





include a "noact" for each literal

- set up graph
- find XOR (exclusive OR) constraints ("mutex")
- find a plan by regression from last state, observing XOR constraints

Towards state that includes goal conditions

Mutex actions: conflicting preconds, conflicting effects, or effects of one "clobbering" a precond of the other
Mutex literals: every pair of actions producing them are mutex.

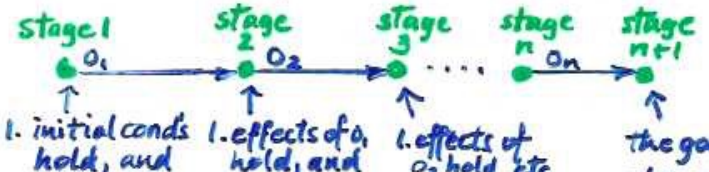
SATPLAN

We ask: What constraints must a successful plan $(O_1, O_2 \cdots O_i \cdots O_n)$ satisfy at each stage i $(1 \le i \le n+1)$?

- express the constraints as a boolean formula
- find a satisfying assignment (e.g., using DPLL)
- extract the plan (the true actions)

How many steps n do we need, for making the goal conditions true at stage $n+1$? Try $n = 1, 2, 3, \ldots$, with the goal asserted to be true at $n+1$, till we succeed.

Constraints implied by a successful plan (STRIPS-like operators)



Similarly, if ~p held at 1, and O_1 doesn't cause p, ~p holds at 2

use frame axiom? No! EC!

Adding state constraints speeds up solutions

Allowing for all plans of a given length

- We can't commit in advance to specific actions (or any actions) but we can say that if O_i occurs, then its precond's were true at stage i and its effects were true at stage $i+1$;
- No two actions occur at the same stage, e.g., $\neg O_1 \vee \neg O_1'$
- Explanation Closure (instead of frame axioms):

  If a condition p holds at stage i while $\neg p$ holds at stage $i+1$, then $O_i \vee O_i' \vee O_i'' \vee \ldots$ occurred at stage i, where $O_i, O_i', O_i'', \ldots$ are the operators that have $\neg p$ as an effect. (Similarly for $p, \neg p$ interchanged.)

Example: Making a flashlight operational by putting in 2 batteries (removing the cover first, and replacing it at the end).

Boolean variables:

$O(C,F,i)$ — the cover is on the flashlight at stage i
$I(B1,F,i)$ — battery 1 is in the flashlight at stage i
$I(B2,F,i)$ — battery 2 ——— " ———
} $1 \le i \le n+1$

RC_i — remove cover of flashlight at stage i
PC_i — place cover on flashlight at stage i
$I1_i$ — insert B1 into F at stage i
$I2_i$ — insert B2 into F at stage i
} $1 \le i \le n$

E.g., $n = 4$ (5 stages)

1. Initial cond's: $O(C,F,1) \wedge \neg I(B1,F,1) \wedge \neg I(B2,F,1)$
2. Goal cond's: $O(C,F,5) \wedge I(B1,F,5) \wedge I(B2,F,5)$
3. Preconds & effects, for $i = 1, \ldots, 4$

$\neg PC_i \vee (\neg O(C,F,i) \wedge O(C,F,i+1))$
$\neg RC_i \vee (O(C,F,i) \wedge \neg O(C,F,i+1))$
$\neg I1_i \vee (\neg O(C,F,i) \wedge \neg I(B1,F,i) \wedge I(B1,F,i+1))$
$\neg I2_i \vee (\neg O(C,F,i) \wedge \neg I(B2,F,i) \wedge I(B2,F,i+1))$

4. Exclusion cond'ns $(1 \le i \le 4)$

$\neg RC_i \vee \neg PC_i$
$\neg PC_i \vee \neg I1_i$
$\neg RC_i \vee \neg I1_i$
$\neg PC_i \vee \neg I2_i$
$\neg RC_i \vee \neg I2_i$
$\neg I1_i \vee \neg I2_i$

5. EC conditions, for $i = 1, \ldots, 4$:

$O(C,F,i) \vee \neg O(C,F,i+1) \vee PC_i$
$\neg O(C,F,i) \vee O(C,F,i+1) \vee RC_i$
$I(B1,F,i) \vee \neg I(B1,F,i+1) \vee I1_i$
$\neg I(B1,F,i) \vee I(B1,F,i+1)$ — 'I' can't become false
$I(B2,F,i) \vee \neg I(B2,F,i+1) \vee I2_i$
$\neg I(B2,F,i) \vee I(B2,F,i+1)$ — 'I' can't become false

A similar technique can be applied to the constraints implied by a planning graph, & this further speeds up solution-finding.

Making $RC_1, I1_2, I2_3, PC_4$ true yields a satisfying assignment