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" (changeable) predicates; similar to Davidsonian event arguments, but more like "snapshots" of current world state, than events.

  e.g.,
  
  \[
  \begin{array}{c}
  \text{A} \\
  \text{B} \\
  \text{D} \\
  \text{C}
  \end{array}
  \Rightarrow
  \begin{array}{c}
  \text{On}(A,B,S_0), \text{Clear}(A,S_0), \\
  \text{On}(B,\text{TABLE},S_0),
  \end{array}
  \]

- 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 \(do(\cdot)\) (or \(result\)) function

  \[
  \text{do}(m(A,B,C), S_0) = S_1, \quad \text{new situation (state)}
  \]

  Big advantage (as we'll see): plans are terms \(do(\ldots,do(\ldots))\)

  It can be deduced by C. Green's "answer extraction".

- 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)

  \[
  \text{On}(x,y,z) \land \text{Clear}(x,z, s) \land \text{Clear}(z, s) \land x \neq z \\
  \land s' = \text{do}(m(x,y,z), s) \Rightarrow \text{On}(x',z', s') \land \text{Clear}(y, s')
  \]

  For instance, let \(s = \text{do}(m(A,B,C), S_0)\)

  Then we easily derive

  \[
  \text{On}(A,C,S_1) \land \text{Clear}(B,S_1) \land \neg \text{Clear}(C,S_1)
  \]

  But what about \(\text{On}(B,\text{TABLE},S_1)\), \(\text{On}(C,B,S_1)\), \(\text{Clear}(A,S_1)\), \(\text{On}(D,\text{TABLE},S_1)\)? Not to mention \(\text{Blue}(B,S_0) \land \neg \text{Blue}(B,S_1)\), etc.
So we also need axioms about what doesn't change...

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

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

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

More generally could use

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

E.g., Blue... an individual (a color)

Problems: - too many things don't change!

\(O(mn)\) axioms

no of states, 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 (Hass, Pednault, Schubert, Reiter)
  - Histories (Hayes)

- Nonmonotonic solutions
  - McCarthy, Reiter, Lifschitz, Baker,...

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

Simple example use \(st(x, y)\) instead of \(m(x, y, w)\) for simplicity

1. Stacking causes "On"

2. (not needed)

3. Clear \((A, S_0)\)

4. Clear \((B, S_0)\)

7. denial of concl: \(7\) on \((A, B, s)\) \(\lor\) Anns \((s)\)

8. \(7\) Clear \((B, S_0)\) \(\lor\) Anns \((do(st(A, B), S))\)

9. \(7\) Clear \((B, S_0)\) \(\lor\) Anns \((do(st(A, B), S))\)

10. Anns \((do(stl(A, B), S_0))\)
Suppose we also had:

5. On (B, TABLE, S0)
6. Color (B, BLUE, S0)

Prove: On (B, TABLE, do(st(C,B), S0))
      Clear (A, do(st(A,B), S0))
      Color (B, BLUE, do(st(A,B), S0))

Frame axioms:

In a stack action st(x,y), the only On-relation that becomes false is the one for x:
- \( \neg \text{On}(y,v,s) \lor x = u \lor \text{On}(u,v, \text{do}(st(x,y), s)) \)

Similarly the only Clear-relation that becomes false is the one for y:
- \( \neg \text{Clear}(u,s) \lor y = u \lor \text{Clear}(u, \text{do}(st(x,y), s)) \)

Similarly the color relationships from states are preserved:
- \( \neg \text{Color}(u,v,s) \lor \text{Color}(u,v, \text{do}(st(x,y), s)) \)

We need similar axioms for relations that stay false.

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 \( \neg \text{On}(A,B, 9) \) vs. \( \text{st}(x,y) \)-effects \( \Rightarrow A \) must be clear previously
- resolve denial that B is clear vs. \( \text{st}(x,y) \)-effect, where precondition is that x is on some z (which becomes clear). Also Table must be clear, ok
- resolve denial that A is clear vs. frame axiom for preservation of Clear properly during \( \text{st}(x,y) \)
  \( \Rightarrow \) 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

(1) \( O(#\text{actions}) \) axioms/pred;

(2) \( O(1) \) axioms/pred

"Shakey's" high-level planner/reasoner (Fikes & Nilsson, AI 2 (34): 189-208, 1971)

Formalize actions in terms of operators with specified preconditions & effects

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

E.g., Operator for pushing an object:

- **push** \((k,m,n)\)
  - **Preconds:** \(\text{AT}(k,m) \land \text{ATR}(m)\)
  - **Del:** \(\text{AT}(k,m), \text{ATR}(m)\)
  - **Add:** \(\text{AT}(k,n), \text{ATR}(n)\)
**Algorithm** STRIPS \((S_0, G_0)\)

0. Initialize current plan \(P := \text{nil}; S := S_0;\)

In general, this is \((S, (G_1, G_2, \ldots, G_n)_0, \ldots, \text{associated actions})\)

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., \(P := P \cup O_i;\) also,
   - let \(S := O_i(S)\), the result of applying operator instance \(O_i\) to the given state \(S;\)

3. If unsuccessful then
   - select an unresolvable literal \(\text{in effect, a subgoal}^{1}\) from the failed proof attempt;
   - find an operator instance, say \(O_i;\) that has an effect matching the chosen unresolvable literal;
   - let \(G_i\) be the (conjunction of) preconditions of operator instance \(O_i;\) let \(i := i + 1 \& \text{go to 1;}\)

---

**STRIPS solution of**

\[
\begin{align*}
\text{stack}(x, y) & \quad \text{Init.:} \quad \text{Clear}(A), \text{Clear}(B) \\
\text{del:} \quad \text{Clear}(y) & \quad \text{goal:} \quad \text{On}(A, B) \\
\text{add:} \quad \text{On}(x, y) &
\end{align*}
\]

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

1. Proof of On(A,B) in Init fails:
   
   \[
   \text{On}(A,B) \text{ is unresolvable}
   \]

2. On(A,B) matches On(x,y), therefore select

   \[
   \text{stack}(A, B); \quad \text{Goal-list := (Init, (G_1, On(A,B)))}
   \]

   where \(G_1 := \{ \text{Clear}(A), \text{Clear}(B) \} \)

3. Proofs of Clear(A), Clear(B) in Init succeed, therefore Plan := (Stack(A,B)), Goal-list := (S, (On(A,B)))

   where now \(S := \{ \text{Clear}(A), \text{Clear}(B), \text{On}(A,B) \} \)

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(\text{Table}) in Init, so that \(C\) can be "stacked" to the \text{Table}.
STRIPS in Action

(1) "Planning & the frame problem"

3.5. An Example

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

\[
M_0: \\
\text{A}T(\text{BOX1}, x) \quad \text{A}T(\text{BOX2}, x) \\
\text{A}T(\text{BOX3}, x)
\]

The goal we describing this task in

\[
G_0: \text{A}T(\text{BOX1}, x) \quad \text{A}T(\text{BOX2}, x) \\
\text{in the form is}
\]

\[
\sim \text{G}_0 : \sim \text{AT}(\text{BOX1}, x) \vee \sim \text{AT}(\text{BOX2}, x) \\
\sim \text{AT}(\text{BOX3}, x)
\]

(i.e., \(G_0\), the term \(x\) is a universally quantified variable)

We define the following operators:

1. \text{push}(k, m, n): Robot pushes object \(k\) from place \(m\) to place \(n\).
   - Precondition: \(\text{AT}(k, m) \land \text{AT}(m, n)\)
   - Required precondition: \(\sim \text{AT}(k, n)\)
   - Delete list: \(\text{AT}(k, m), \text{AT}(m, n)\)
   - Add list: \(\sim \text{AT}(k, n)\)

2. \text{grasp}(m, n): Robot goes from place \(m\) to place \(n\).
   - Precondition: \(\text{AT}(m, n)\)
   - Required precondition: \(\sim \text{AT}(m, n)\)
   - Delete list: \(\text{AT}(m, n)\)
   - Add list: \(\text{AT}(m, n)\)

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

\[
\sim \text{AT}(\text{BOX1}, x) \vee \sim \text{AT}(\text{BOX2}, x) \vee \sim \text{AT}(\text{BOX3}, x) \\
\sim \text{AT}(\text{BOX1}, x) \vee \sim \text{AT}(\text{BOX2}, x) \\
\sim \text{AT}(\text{BOX2}, x) \vee \sim \text{AT}(\text{BOX3}, x)
\]

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

The only possible operator is \text{push}(k, m, n). Using the add list clause \(\text{AT}(k, m)\), we can continue the incomplete proof in one of several ways depending on the substitutions made for \(k\) and \(m\).

Each of these substitutions produces a relevant instance of \text{push}. One of these is:

\[
\sim \text{G}_1: \sim \text{push}(k, m, n)
\]

Given by the substitutions \text{BOX2} for \(k\) and \(n\) for \(x\), we construct the proof in negative form:

\[
\sim \text{G}_1: \sim \text{AT}(\text{BOX2}, x) \vee \sim \text{AT}(\text{BOX3}, x)
\]

Having found a contradiction, the proof proceeds as follows:

\[
\sim \text{G}_2: \sim \text{push}(k, m, n) \vee \sim \text{AT}(\text{BOX2}, x) \vee \sim \text{AT}(\text{BOX3}, x)
\]

Thus, STRIPS finds the solution:

\[
M_*: \\
\text{A}T(\text{BOX1}, x) \\
\text{A}T(\text{BOX2}, x) \\
\text{A}T(\text{BOX3}, x)
\]
Some problems STRIPS has trouble solving

1. Sussman anomaly

```
    A
    C
    B  
|____|     GOAL:         Pre:  Clear(x), Clear(z),
    |    |                       On(x,y)
    | A |   Del:               On(x,y), Clear(z)
    | B |   Add:               On(x,z), Clear(y)
    |   |                     Clear(Table)

This is troublesome if STRIPS “protects” goals that have already achieved from subse-
quently change: it gets stuck after achieving just one of 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

```
| N1 | N2 | ----- | N2 | N1 | |
|____|____|       |____|____|___
R1   R2   R3
```

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(S1), Black(S1) V White(S1),

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

$$\exists x \forall y (\text{Have-sock}(x) \land (\exists y) \text{Have-sock}(y) \land x \neq y \land$$
$$[\text{Black}(x) \land \text{Black}(y)] \lor [\text{White}(x) \land \text{White}(y)]$$

It seems more natural to use forward reasoning

Some symbolic AI originators

From Push Singhs blog site

John McCarthy Push Ed Marvin Singh Fredkin Minsky

Some symbolic AI originators
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)

Go to Atlanta
  Goal: be in Atlanta
  - Get tickets for flight to Atlanta
  - Book flight
  - Pay-off: ! Search space reduction!
  ! Don’t have to know detailed circumstances in advance

Take cab to ROC
  - Phone cab company
  - Wait till cab arrives
  - Take cab etc.

Take cab from ROC into Atlanta

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

Towards state that includes goal conditions

Mutex actions: conflicting precond, conflicting effects, or effects of one "obloquing" a precond of the other Mutex literals: every pair of actions producing them are mutext
We ask: What constraints must a successful plan 
\((O, O_2, \ldots O_i \ldots O_n)\) satisfy at each stage \(i\) \((1 \leq i \leq n+1)\)?
- 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)**

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

**Boolean variables:**
- \(OC_i, F_i, C\) — the cover is on the flashlight at stage \(i\).
- \(B_i, F_i\) — battery \(i\) is in the flashlight at stage \(i\).
- \(RC_i\) — remove cover of flashlight at stage \(i\).
- \(PC_i\) — place cover on flashlight at stage \(i\).
- \(I_{1i}\) — insert \(B_1\) into \(F\) at stage \(i\).
- \(I_{2i}\) — insert \(B_2\) into \(F\) at stage \(i\).

**E.g., \(n=4\) (5 stages):**

1. Initial cond's: \(OC(F_0)i \wedge \neg I(B_1,F_0) \wedge I(B_2,F_0)\)
2. Goal cond's: \(OC(F_5)i \wedge I(B_1,F_5) \wedge I(B_2,F_5)\)
3. Precond & effects, for \(i = 1, \ldots, 4\): 
   - \(\neg PC_i \wedge (\neg OC(F_i)i \wedge OC(F_{i+1})i)\)
   - \(\neg RC_i \wedge (\neg OC(F_i)i \wedge RC(F_{i+1})i)\)
   - \(\neg I_{1i} \wedge (\neg OC(F_i)i \wedge I_{1i}(B_1,F_i) \wedge I_{2i}(B_1,F_{i+1})\)
   - \(\neg I_{2i} \wedge (\neg OC(F_i)i \wedge I_{2i}(B_2,F_i) \wedge I_{2i}(B_2,F_{i+1})\)
4. Exclusion cond's
   - \(\neg RC_i \vee \neg PC_i\)
   - \(\neg RC_i \vee \neg PC_i\)
   - \(\neg I_{1i} \vee I_{1i}\)
   - \(\neg I_{2i} \vee I_{2i}\)
5. EC conditions, for \(i = 1, 2, \ldots, 4\): 
   - \(OC(F_i)i \wedge OC(F_{i+1})i \wedge PC_i\)
   - \(OC(F_i)i \wedge \neg OC(F_{i+1})i \wedge RC_i\)
   - \(I_{1i} \wedge \neg I_{1i}(B_1,F_{i+1}) \wedge I_{1i}(B_1,F_{i+1})\)
   - \(I_{2i} \wedge \neg I_{2i}(B_2,F_{i+1}) \wedge I_{2i}(B_2,F_{i+1})\)

**Making \(RC_i, I_{1i}, I_{2i}, PC_i\) true yields a satisfying assignment.**

**Explanation:** 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., \(\neg O_0 \vee O_0\).
- Explanation closure (instead of frame axioms): 
  If a condition \(p\) holds at stage \(i\) while \(\neg p\) holds at stage \(i+1\), then \(O_0, O_0, O_0, \ldots, v_0, v_0, v_0, \ldots\) are the operators that have \(\neg p\) as an effect. (Similarly for \(p, \neg p\) interchanged.)

Similarly, if \(p\) held at \(1\), and \(O_i\) doesn't cause \(p\), \(p\) holds at \(2\).