Introduction to AI, Spring 2023

Least Commitment Planning

Least Commitment Planning: Things to Watch For

- The goal of these slides:
- To get you thinking about **planning abstractions**
  - That is, how to reduce the size of the search space.
- To communicate some ideas about **partial order planning** and **least commitment planning**.
  - Which are terms we'll define shortly.
- There is some gnarly stuff here, but it's gonna be OK!

Least Commitment Planning: Things to Watch For

- Ruminate on this for now:
- A **partial order planner** is one which produces plans in which *some* steps (i.e., actions) need to come before others, but it doesn't need to be completely specified.
  - The blocks world planner we just saw was *not* this. It was a **totally ordered** planner.
- **Least Commitment** is ultimately going to refer to variable binding.
  - Eventually variables need to be bound to actual constants…
  - But try to keep things as variables for as long as you can for flexibility.

Least Commitment Planning: Motivation

- We’ve said this a couple times now, but accurate heuristics are very important for planning.
- In the 1980s and 90s, those heuristics hadn’t been discovered yet.
- So back then, the research focus was on reducing the branching factor through abstraction.
- Some ideas:
  - Allow one action in the search to represent lots of actions in the plan
  - Remove unnecessary fluents (variables not involved in the plan).
  - Work backwards from the goal (“regression search”).

Cargo Domain

- We are going to introduce yet another domain! The **Cargo Domain**.
- A wonderful world, full of **cargo**, **airports**, and **airplanes**.
- At any given moment, a plane is at an airport. Cargo is either at an airport, OR inside of a plane.
- There are three actions:
  - **Load**: put cargo into a plane (if both cargo and plane are at same airport).
  - **Unload**: Take cargo out a plane (now the cargo is at the airport the plane is)
  - **Fly**: Fly a plane from one airport to another.

Cargo Domain – Typed Constants

- In Blocks World we had a lot of predicates that never changed.
- i.e. block(A), block(B), block(C)

- Instead of adding these to the initial state, we're going to introduce the notion of **types**.
- That is, constants and variables have types, like “Cargo” or “Plane” or “Airport”
- And if an action is expecting input of a certain type, it better match.
- We'll say that we are **binding** constants to variables here.

Cargo Domain – Action Templates

- *Action: load(Cargo c, Plane p, Airport a)*

  Precondition: $at(c,a) \wedge at(p,a)$

  Effect: $in(c,p) \wedge \neg at(c,a)$

- *Action: unload(Cargo c, Plane p, Airport a)*

  Precondition: $in(c,p) \wedge at(p,a)$

  Effect: $at(c,a) \wedge \neg in(c,p)$

- *Action: fly(Plane p, Airport from, Airport to)*

  Precondition: $at(p,from)$

  Effect: $at(p,to) \wedge \neg at(p,from)$

If cargo and plane are at the same airport, put the cargo in the plane, remove it from the airport.

If cargo is the plane, and the plane is at the airport, put the cargo in the airport, take it out of the plane.

If the plane is at airport "from", then move the plane to airport "to", remove it from airport "from".

Cargo Problem

- Let's say this is what we're starting with.
- Constants:
  - Two Planes: P1 and P2
  - Two Cargos: C1 and C2
  - Two Airports: MSY and ATL (that's New Orleans and Atlanta, fyi).
- Initial State:
  - Initial State: $at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$
- Goal: *at(C1, MSY)*

Both Planes and both Cargos begin in Atlanta: Get Cargo1 to New Orleans!

Cargo Problem

- Initial State: $at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$
- Goal: *at(C1, MSY)*

- Ok, well, how many actions are **applicable** (i.e., do we satisfy the preconditions for) given this initial state?

1. $load(C1, P1, ATL)$
2. $load(C2, P1, ATL)$
3. $load(C1, P2, ATL)$
4. $load(C2, P2, ATL)$
5. $fly(P1, ATL, MSY)$
6. $fly(P2, ATL, MSY)$
7. $fly(P1, ATL, ATL)$
8. $fly(P2, ATL, ATL)$

We never said that you had to fly to a *different* airport!

Cargo Problem

- Initial State: $at(P1, ATL) \land at(P2, ATL) \land at(C1, ATL) \land at(C2, ATL)$
- Goal: *at(C1, MSY)*

- 8 Actions! Where to start!?!
- Well, maybe we aren't asking the right question…. Let's come from the other side…
- **How many actions *achieve* our goal: *at(C1, MSY)***

1. $unload(C1, P1, MSY)$
2. $unload(C1, P2, MSY)$

Backwards Search

- Instead of searching forward from the initial state towards the goal…
- We are going to search backward from the goal to the initial state.
- Also known as **regression search**.
  - Like with bidirectional search (where we combined this with forwards search), only works when you know ”how” to go backwards, but thankfully having clearly articulated preconditions and effects gives us all we need to know!

The Null Plan

- Typically, each step in a plan is an action: it has preconditions and effects. But…
- We are going to treat the initial state and the goal of the problem as steps, too!
  - The initial state will be represented as the **start step**.
    - No preconditions.
    - Its effects are the initial state.
  - The goal will be represented as the **end step**:
    - No effects
    - Its preconditions are the goal
- The **null plan** is how we'll start: a plan with these two dummy steps.
  - Then it's just a matter of filling in the middle!

The Null Plan

$(no\ preconditions)$

$start$

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

Start step

End step

$at(C1, MSY)$

end

$(no\ effects)$

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

So, let's do it! Let's try making a plan for this guy!

$at(C1, MSY)$

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

We already discussed how there are two options that achieve our goal:
Unloading C1 from P1 at MSY, or
Unloading C1 from P2 at MSY

2 options

$at(C1, MSY)$

$$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$$

So for now, let's just pick one: let's pick the P1 version.

$$at(P1, MSY) \wedge in(C1, P1)$$

$$unload(C1, P1, MSY)$$

$$at(C1, MSY) \wedge \neg in(C1, P1)$$

$$at(C1, MSY)$$

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

Now want to satisfy all the preconditions of this action.

Technically there's two options that yield at(P1, MSY) again
either we fly to MSY from ATL, or…
we fly to MSY from MSY

2 options

$at(P1, MSY) \wedge in(C1, P1)$

$unload(C1, P1, MSY)$

$at(C1, MSY) \wedge \neg in(C1, P1)$

$at(C1, MSY)$

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

Let's pick the more reasonable one…,
flying from ATL to MSY.

$at(P1, ATL)$

$fly(P1, ATL, MSY)$

$\neg at(P1, ATL) \wedge at(P1, MSY)$

$at(P1, MSY) \wedge in(C1, P1)$

$unload(C1, P1, MSY)$

$at(C1, MSY) \wedge \neg in(C1, P1)$

$at(C1, MSY)$

$at(P1, ATL) \land at(P2, ATL) \land at(C1, ATL) \land at(C2, ATL)$

And again, there's two options for the in(C1, P1) precondition as well: it can be loaded into the plane at ATL, or in MSY.

2 options

$at(P1, ATL)$

$fly(P1, ATL, MSY)$

$\neg at(P1, ATL) \land at(P1, MSY)$

$at(P1, MSY) \land in(C1, P1)$

$unload(C1, P1, MSY)$

$at(C1, MSY) \land \neg in(C1, P1)$

$at(C1, MSY)$

$$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$$

Let's pick it getting loaded in ATL.

And this is (mostly) looking pretty good! All of the preconditions of these steps are in the start step i.e., the initial state!

$$at(P1, ATL) \wedge at(C1, ATL)$$
$$load(C1, P1, ATL)$$
$$\neg at(C1, ATL) \wedge in(C1, P1)$$

$$at(P1, ATL)$$
$$fly(P1, ATL, MSY)$$
$$\neg at(P1, ATL) \wedge at(P1, MSY)$$

$$at(P1, MSY) \wedge in(C1, P1)$$
$$unload(C1, P1, MSY)$$
$$at(C1, MSY) \wedge \neg in(C1, P1)$$

$$at(C1, MSY)$$

Abstraction

- But we’re not quite done yet, for a couple of reasons…
- One is: we had a lot of hand-wavey “let’s just pick this one” moments. Is there a way to formalize the selection process?
- To illustrate: there were two ways to achieve the goal of at(C1, MSY) – unloading from P1, and unloading from P2.
- But we don’t actually care about which plane was chosen! Both are equally good!

Abstraction

- We are going to use this idea to illustrate the notion behind a **Least Commitment Planner**.

- We only **bind** the variables in an action when we know what values they must have. Until then, we leave them unbound to maximize flexibility.

- This is where “least commitment” comes in, we are *committing* to as few bound variables as we can get away with for as long as possible.

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

So, let's start over again.

We know that, though there's two ground actions that satisfy the goal, there's only one action template that works: unload.

$at(C1, MSY)$

$$at(P1, ATL) \land at(P2, ATL) \land at(C1, ATL) \land at(C2, ATL)$$

Note this action isn't ground! It's full of variables!

$$at(p_1, a_1) \land in(c_1, p_1)$$
$$unload(c_1, p_1, a_1)$$
$$at(c_1, a_1) \land \neg in(c_1, p_1)$$

This step represents *any* unload action – any plane, from any airport, with any cargo.

$$at(C1, MSY)$$

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

Here is where the magic comes in. Some of these variables we care about:

*we don't want to care about just *any* cargo, we specifically care about C1
*Similarly, we know the airport we care about: MSY

But other variables (like the plane) we *don't* care about!

$at(p_1, a_1) \wedge in(c_1, p_1)$

$unload(c_1, p_1, a_1)$

$at(c_1, a_1) \wedge \neg in(c_1, p_1)$

This step represents *any* unload action – any plane, from any airport, with any cargo.

$at(C1, MSY)$

$at(P1, ATL) \land at(P2, ATL) \land at(C1, ATL) \land at(C2, ATL)$

But hey, the goal step has this lovely

$$at(C1, MSY)$$

And the action template has this lovely

$$at(c_1, a_1)$$

If only there was a way to convert one into the other…?

$at(p_1, a_1) \land in(c_1, p_1)$

$unload(c_1, p_1, a_1)$

$at(c_1, a_1) \land \neg in(c_1, p_1)$

This step represents *any* unload action – any plane, from any airport, with any cargo.

$at(C1, MSY)$

IT'S UNIFICATION TIME!

$at(P1, ATL) \land at(P2, ATL) \land at(C1, ATL) \land at(C2, ATL)$

Unify these two fluents

$at(p_1, MSY) \land in(C1, p_1)$

$unload(C1, p_1, MSY)$

$at(C1, MSY) \land \neg in(C1, p_1)$

Bound variables:

c1 = C1

a1 = MSY

$at(C1, MSY)$

$at(P1, ATL) \land at(P2, ATL) \land at(C1, ATL) \land at(C2, ATL)$

And then update the preconditions, action, and effects based on that unification!

This step now represents specifically unloading C1 from *any plane* at MSY

$at(p_1, MSY) \land in(C1, p_1)$

$unload(C1, p_1, MSY)$

$at(C1, MSY) \land \neg in(C1, p_1)$

Bound variables:
c1 = C1
a1 = MSY

$at(C1, MSY)$

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

As before, we want to satisfy the preconditions of this step.

And as before, there's only one action template that has at() as an effect: the fly action

$at(p_1, MSY) \wedge in(C1, p_1)$

$unload(C1, p_1, MSY)$

$at(C1, MSY) \wedge \neg in(C1, p_1)$

Bound variables:
c1 = C1
a1 = MSY

$at(C1, MSY)$

$$at(P1, ATL) \land at(P2, ATL) \land at(C1, ATL) \land at(C2, ATL)$$

Again, we've added the template -- it's full of variables!

Also note the variable names have been standardized apart.

$$at(p_2, a_2)$$
$$fly(p_2, a_2, a_3)$$
$$\neg at(p_2, a_2) \land at(p_2, a_3)$$

$$at(p_1, MSY) \land in(C1, p_1)$$
$$unload(C1, p_1, MSY)$$
$$at(C1, MSY) \land \neg in(C1, p_1)$$

Bound variables:
c1 = C1
a1 = MSY

$$at(C1, MSY)$$

$$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$$

We unify these literals!

So we know that we want to substitute a3 for MSY.

We still don't know what p1 or p2 will be, but we know they will be the same eventually!

$$at(p_2, a_2)$$
$$fly(p_2, a_2, a_3)$$
$$\neg at(p_2, a_2) \wedge at(p_2, a_3)$$

$$at(p_1, MSY) \wedge in(C1, p_1)$$
$$unload(C1, p_1, MSY)$$
$$at(C1, MSY) \wedge \neg in(C1, p_1)$$

Bound variables:
c1 = C1
a1 = MSY
a3 = MSY

$$at(C1, MSY)$$

$$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$$

And we update that, and move on…

$$at(p_2, a_2)$$
$$fly(p_2, a_2, MSY)$$
$$\neg at(p_2, a_2) \wedge at(p_2, MSY)$$

$$p_2 = p_1$$

$$at(p_1, MSY) \wedge in(C1, p_1)$$
$$unload(C1, p_1, MSY)$$
$$at(C1, MSY) \wedge \neg in(C1, p_1)$$

Bound variables:
c1 = C1
a1 = MSY
a3 = MSY

$$at(C1, MSY)$$

$at(P1, ATL) \land at(P2, ATL) \land at(C1, ATL) \land at(C2, ATL)$

$at(p_3, a_4) \land at(c_2, a_4)$

$load(c_2, p_3, a_4)$

$\neg at(c_2, a_4) \land in(c_2, p_3)$

Again, the only thing that could possibly give us in(C1, p1) is a load action.

$at(p_2, a_2)$

$fly(p_2, a_2, MSY)$

$\neg at(p_2, a_2) \land at(p_2, MSY)$

$p_2 = p_1$

Once again, this new step is not ground, full of variables, with names standardized apart…

$at(p_1, MSY) \land in(C1, p_1)$

$unload(C1, p_1, MSY)$

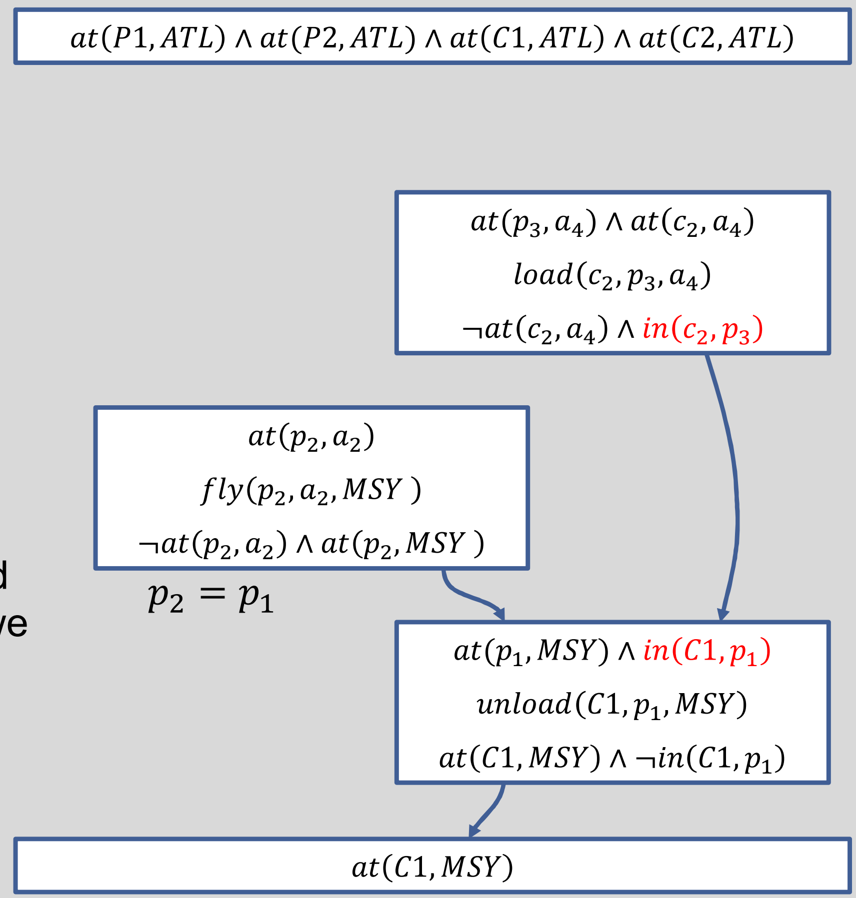
$at(C1, MSY) \land \neg in(C1, p_1)$

Bound variables:
c1 = C1
a1 = MSY
a3 = MSY

$at(C1, MSY)$

We unify again,

Teaching us that C1 will be bound to c2, and that p1 and p3 will be “the same” once we commit to a thing.



Bound variables:
c1 = C1
a1 = MSY
a3 = MSY
c2 = C1

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

$at(p_3, a_4) \wedge at(C1, a_4)$

$load(C1, p_3, a_4)$

$\neg at(C1, a_4) \wedge in(C1, p_3)$

$p_3 = p_1$

So we fill in all the variables we can.

$at(p_2, a_2)$

$fly(p_2, a_2, MSY)$

$\neg at(p_2, a_2) \wedge at(p_2, MSY)$

$p_2 = p_1$

$at(p_1, MSY) \wedge in(C1, p_1)$

$unload(C1, p_1, MSY)$

$at(C1, MSY) \wedge \neg in(C1, p_1)$

$at(C1, MSY)$

Bound variables:

c1 = C1

a1 = MSY

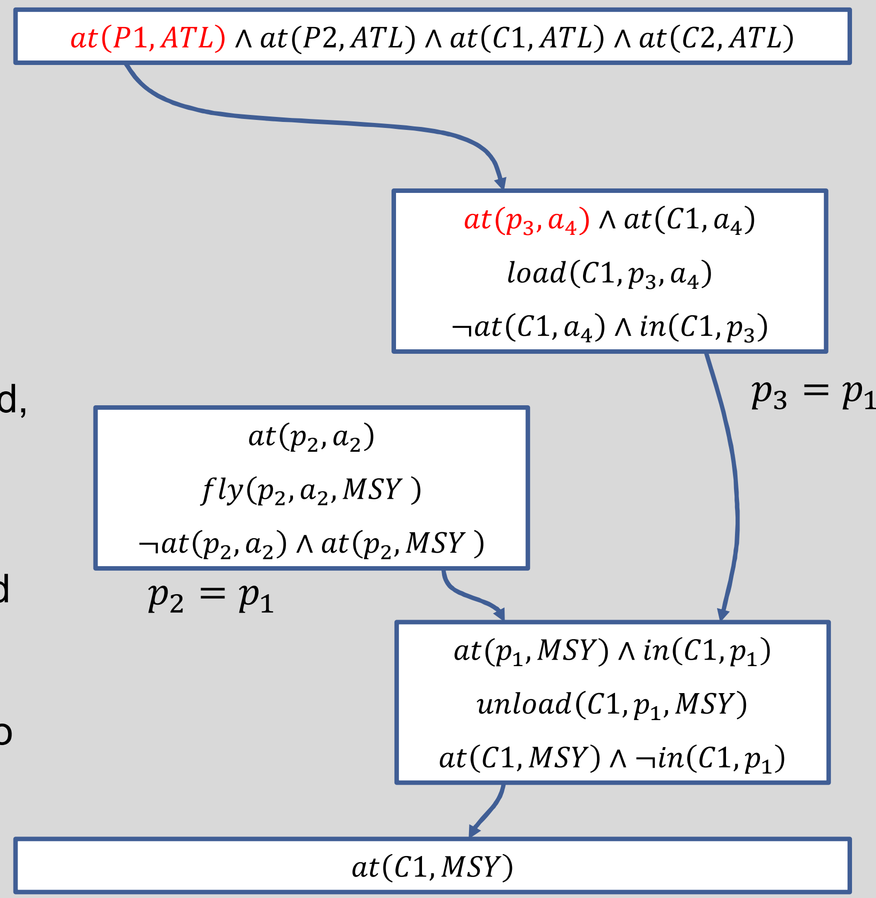
a3 = MSY

c2 = C1

The precondition is already met, so no new action needed, but we need to do another unification with the start step.

Here, ATL is bound to a4, and P1 is bound to p3.

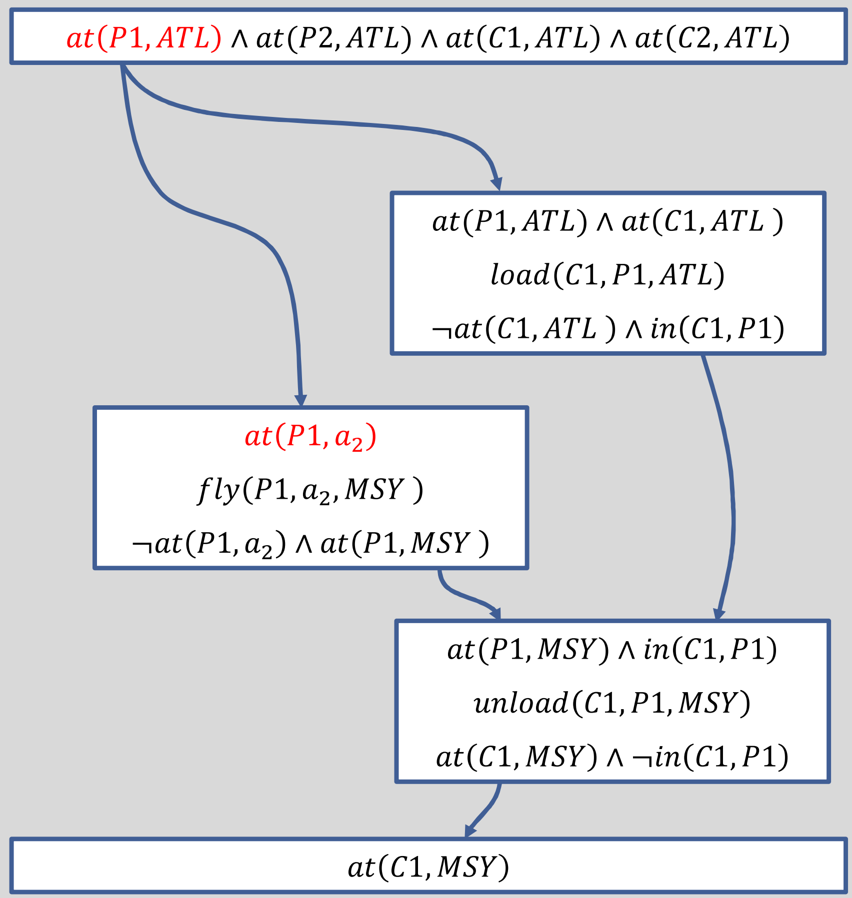
Note p3 = p1, and p2 = p1, so P1 gets filled in everywhere (i.e., replace p1, p2 and p3 with P1).



Bound variables:
c1 = C1
a1 = MSY
a3 = MSY
c2 = C1

And now there is one final unification needed!

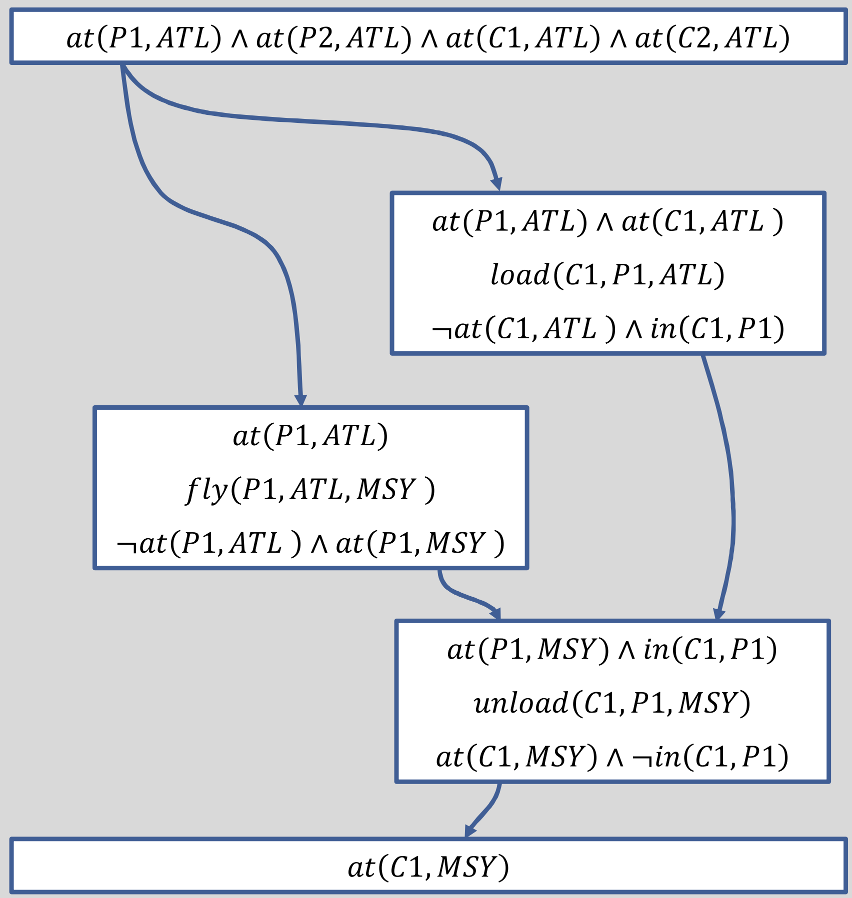
Binding ATL to a2 in the “fly” action template!



Bound variables:
c1 = C1
a1 = MSY
a3 = MSY
c2 = C1
a4 = ATL
p1 = p2 = p3 = P1

And now we are back where we were last time!

But we did it in a “**least commitment**” way!



Bound variables:
c1 = C1
a1 = MSY
a3 = MSY
c2 = C1
a4 = ATL
p1 = p2 = p3 = P1
a2 = ATL

Introduction to Partial-Ordered Plans

- So, we’re actually still *not quite done yet*
- That whole unification work was the least commitment part at play…
- But we’ve also been bandying around the term **partial-ordered**.
- To illustrate what we mean by partial-ordered, let’s visit yet *another* domain: the Shoes and Socks domain!
  - You want both shoes on your feet, but your feet must first be socked up before they can be shoed!

Shoes and Socks Domain

$No\ shoes\ or\ socks\ on$

$Left\ shoe\ on \quad Right\ shoe\ on$

Shoes and Socks Domain

No shoes or socks on

Put on left sock

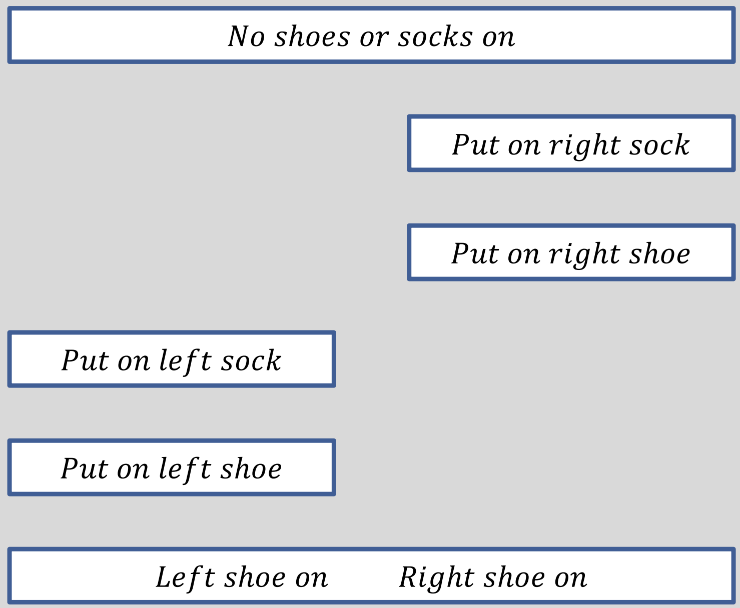
Put on left shoe

Put on right sock

Put on right shoe

Left shoe on *Right shoe on*

Shoes and Socks Domain



Shoes and Socks Domain

No shoes or socks on

Put on left sock

Put on right sock

Put on left shoe

Put on right shoe

Left shoe on *Right shoe on*

Shoes and Socks Domain

No shoes or socks on

Put on right sock

Put on left sock

Put on right shoe

Put on left shoe

Left shoe on *Right shoe on*

Shoes and Socks Domain

No shoes or socks on

Put on right sock

Put on left sock

Put on left shoe

Put on right shoe

Left shoe on *Right shoe on*

Shoes and Socks Domain

No shoes or socks on

Put on left sock

Put on right sock

Put on right shoe

Put on left shoe

Left shoe on *Right shoe on*

Action Ordering

- The takeaway:
- Many possible orders in which the same step can be taken, but the plans are conceptually similar.
- The “conceptually similar part” of shoes and socks domain:
- Put on your left sock before your left shoe, and put on your right sock before your right shoe!

Action Ordering

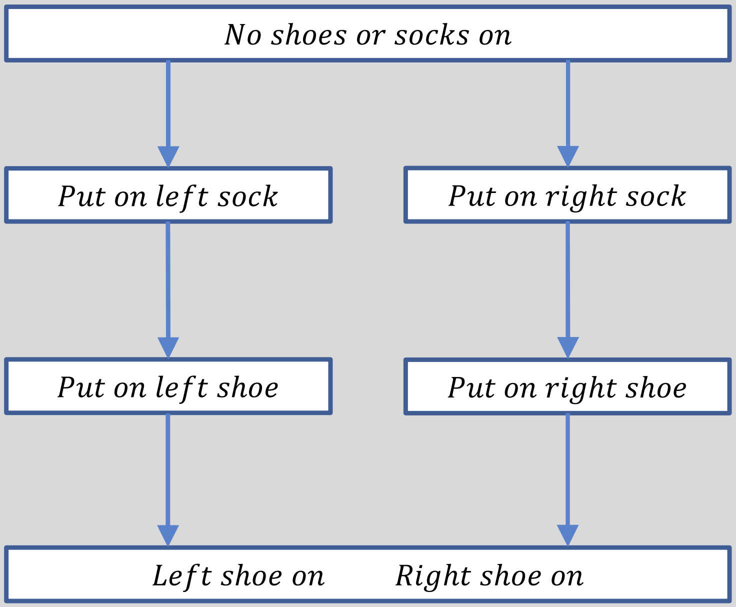
- This is what we mean by **partial ordering** of the steps, instead of total ordering.

- A partial ordering is a set of constraints. The constraints are written like:

$$S1 < S2$$

- S1 and S2 are both steps. The above notation means "Step S1 must happen before S2, but when exactly doesn't matter".

Partial Order Plan



- So here, our partial ordering is:

  Left Sock < Left Shoe

  Right Sock < Right Shoe

(i.e., Left Sock happens before Left Shoe, and Right Sock happens before Right Shoe).

Implied for all plans:

Start Step < All Steps

All Steps < End Step

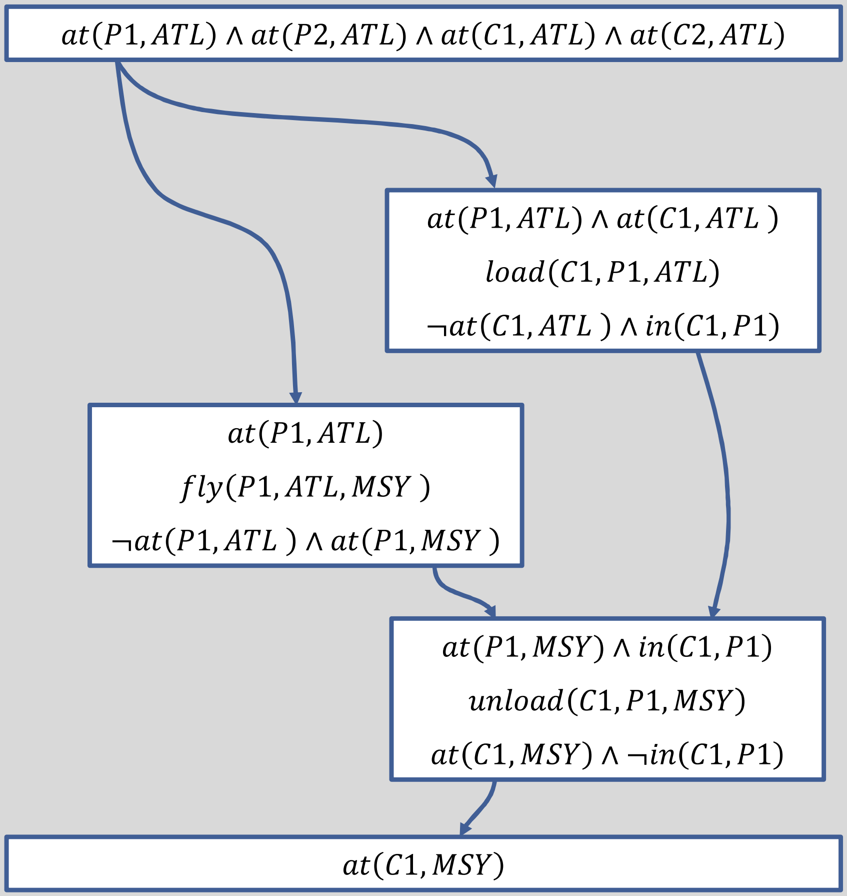
Interleaved Goals

- Partial Ordering allows us to separate one goal from another! Neat!

- *But uh oh…*

- We've already seen how actions needed to achieve one goal might interfere with the actions needed to achieve another goal.

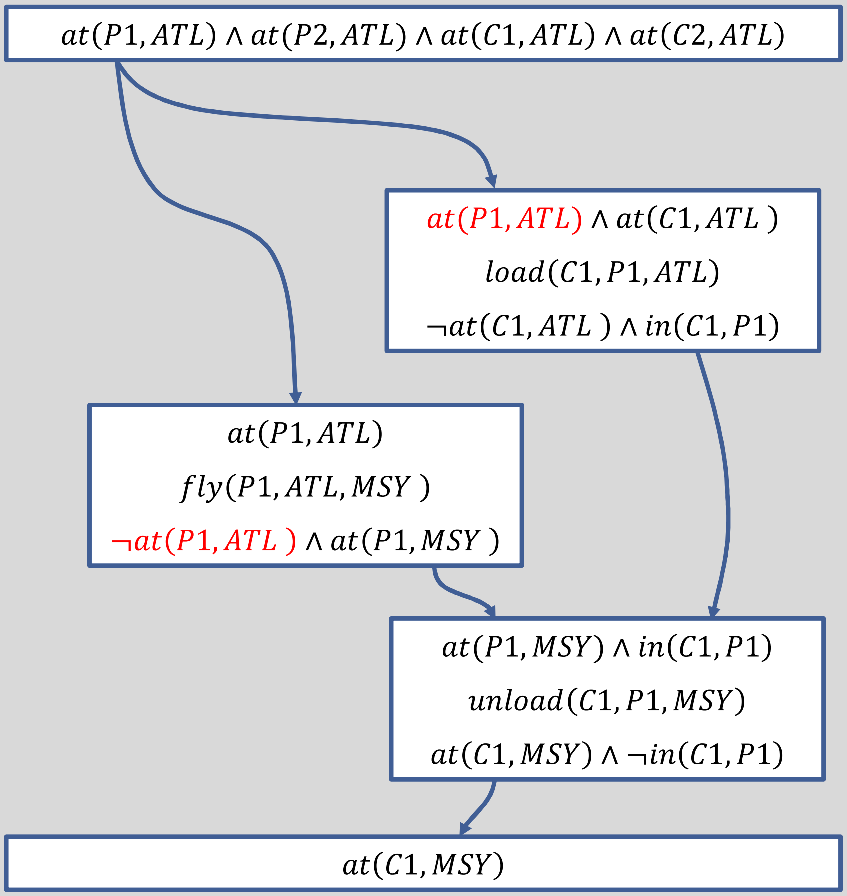
We return to our old friend here…



Yikes! The effect of the fly action takes P1 away from ATL…

But if P1 is away from ATL, it won’t be able to do the load action!

We didn’t mark it before, but it is crucial that load happens prior to fly for this plan to work!



Causal Links

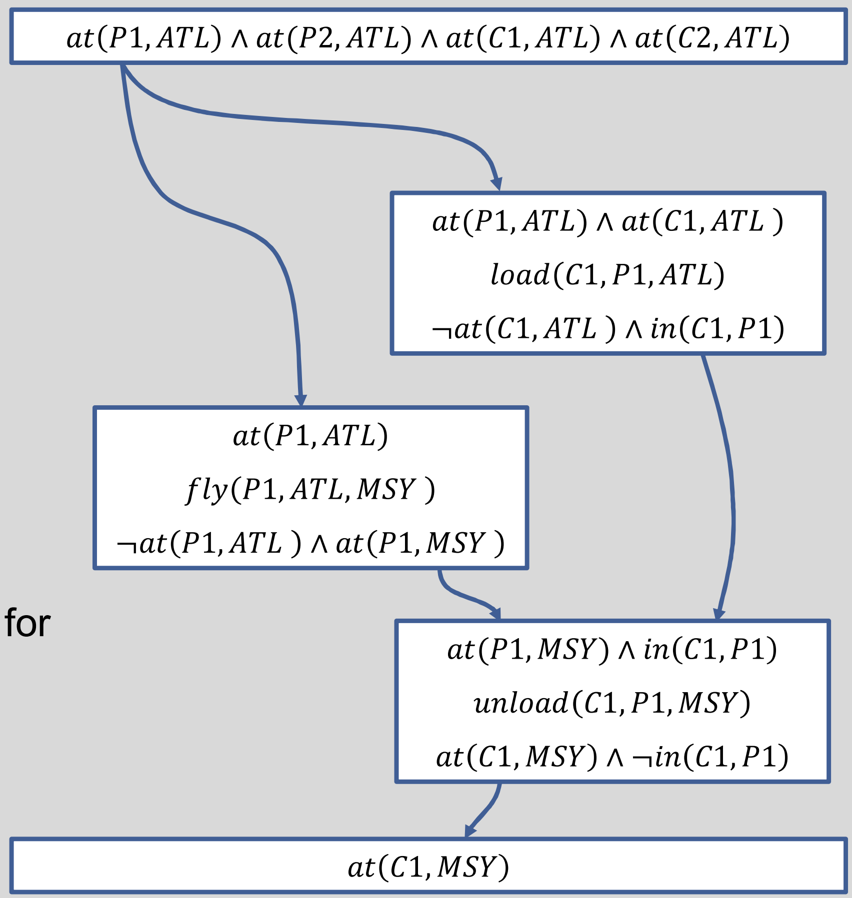
- So, this isn't too big of a surprise: we know that *some* ordering constraints do matter.
- The question is: how do we deal with it?
- The momentary answer**: Causal Links**.
- We write causal links like this: $S1 \xrightarrow{p} S2$

Causal Links

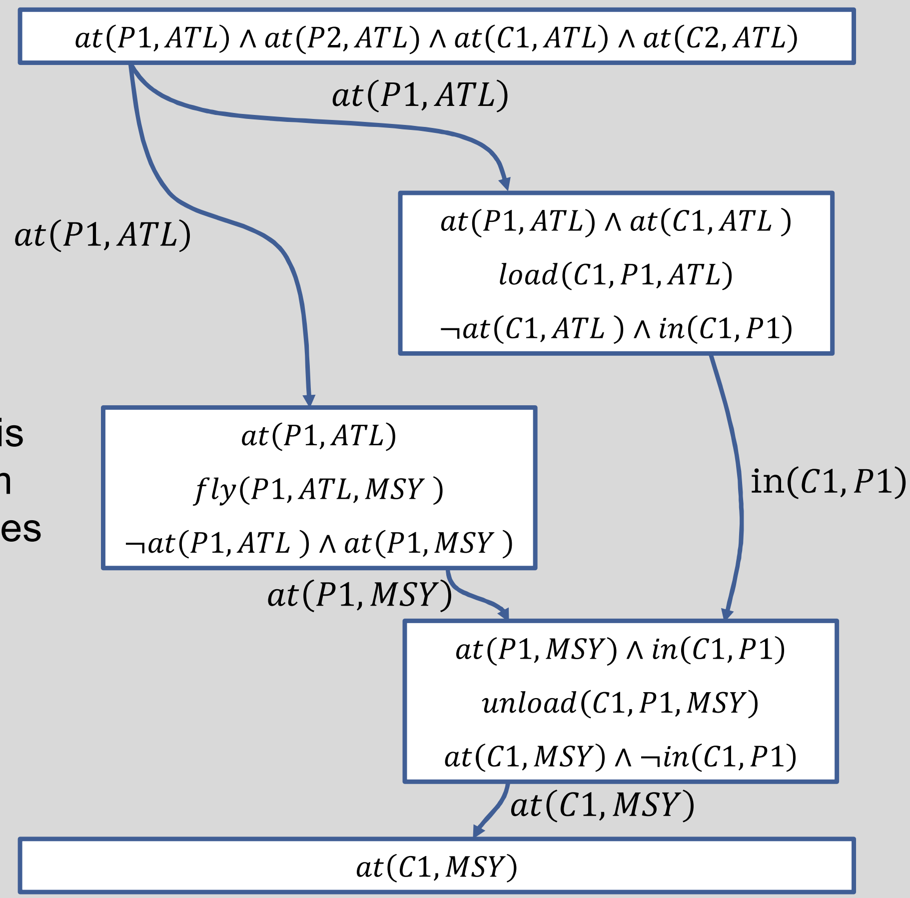
- So, our causal link looks like: $S1 \xrightarrow{p} S2$
- If we consider the plan as a graph, a causal link is a directed edge connecting two steps, S1 and S2, with a label *p, such that:*
  - The **tail** (i.e., the left hand side) is a step with effect *p*
  - The **head** (i.e., the right hand side) is a step with precondition *p*
- Or in other words: When S1 happens, its effect makes p true, and p needs to be true in order for S2 to happen.
- A causal link, therefore, implies the ordering *S1 < S2*

So, returning to our friend here…

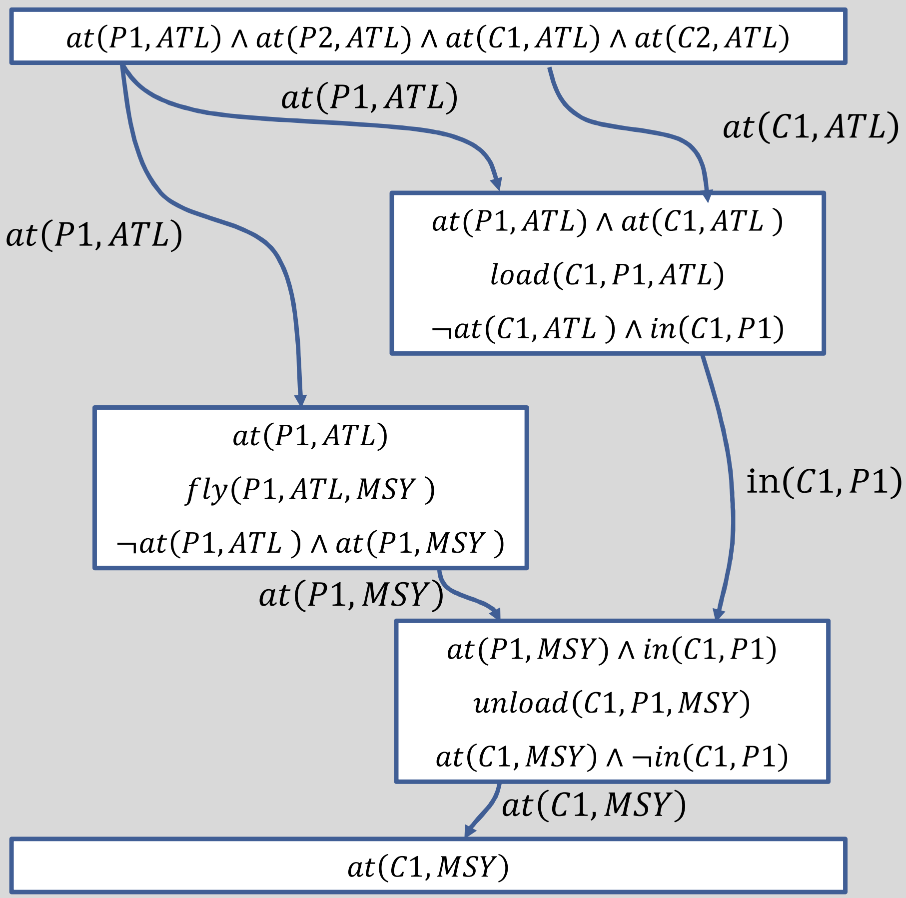
We can add Causal Links to this, by specifying the labels for each edge.



Again, each of these labels is saying "this is the effect from the previous step that satisfies the precondition of the next step"



And let's add in this guy too.



Causal Links

- A Causal Link explains how an earlier step satisfies the preconditions of a later step.
- That's great, but… just specifying the label alone doesn't help us with our problem.
- The problem with interleaving goals is that the commitment a causal link represents can still be undone.

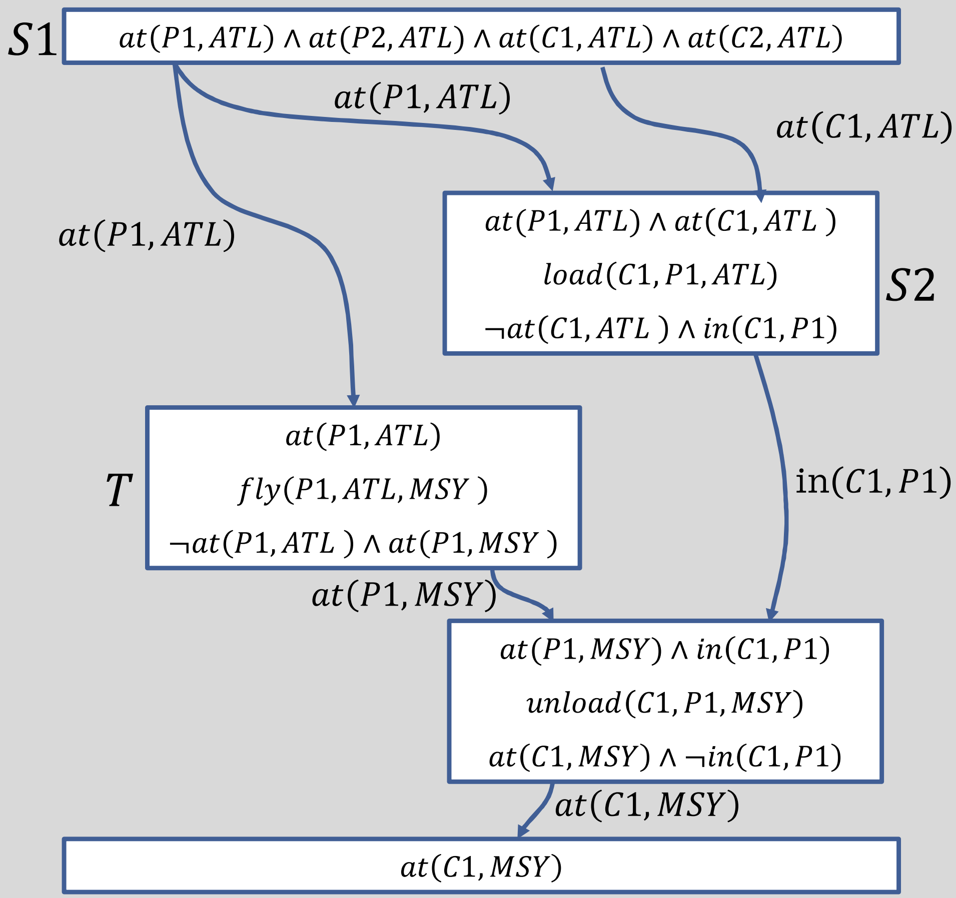
Threatened Causal Links

- We say that a causal link, $S1 \xrightarrow{p} S2$, with Steps S1 and S2 and label p, is **threatened** by a third step, T, if and only if:
  - Step T has the effect $\neg p$
  - The Current Partial Ordering allows for S1 < T1 < S2

- Or in other words: the causal link is threatened if S2 depends on S1's effect to happen, but T can undo S1's effect, AND T can happen in between S1 and S2.

Returning to this guy once again…

Note the convenient labels of S1, S2, and T added to certain steps!



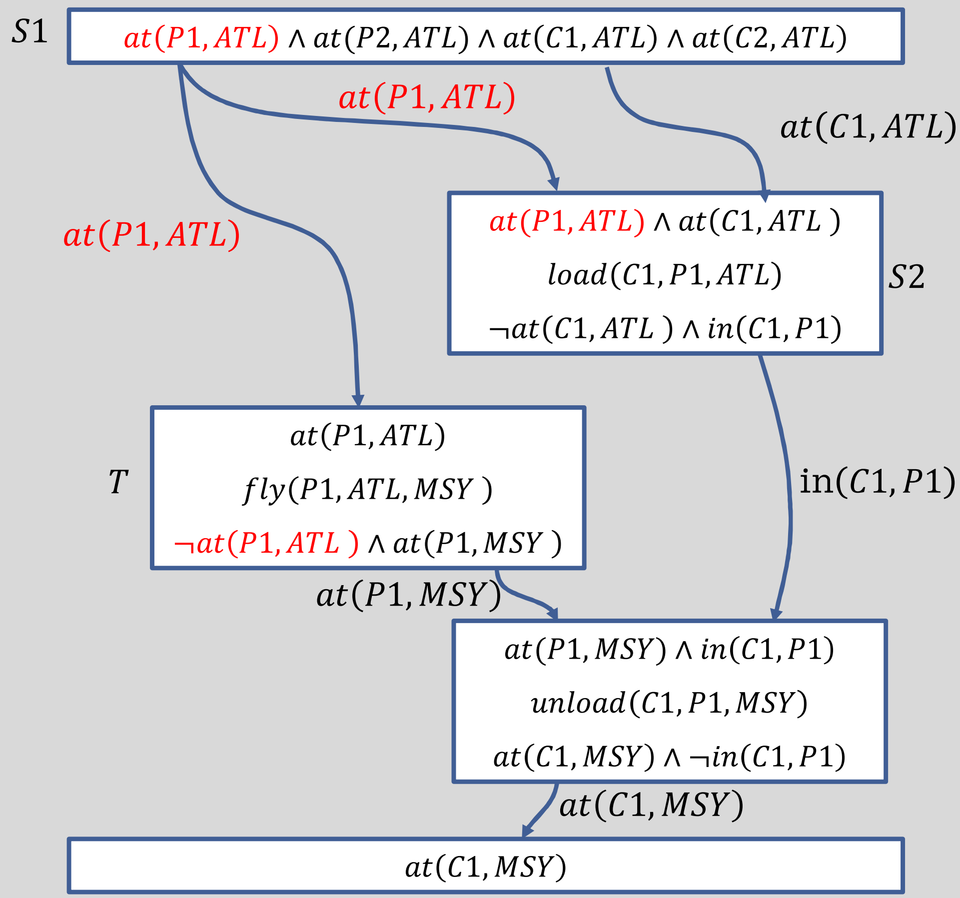
The red text highlights our plight!

S1 needs to happen before S2 because of that at(P1, ATL) predicate, i.e., we have the causal link

$$S1 \xrightarrow{at(P1,ATL)} S2$$

But Step T threatens that causal link, because it has as an effect

$$\neg at(P1, ATL)$$



Fixing Threatened Causal Links

- So, as we’ve seen a couple times now:
  - When a causal link is threatened, it means a step exists that could occur between the tail and the head which undoes the fact established by the link.

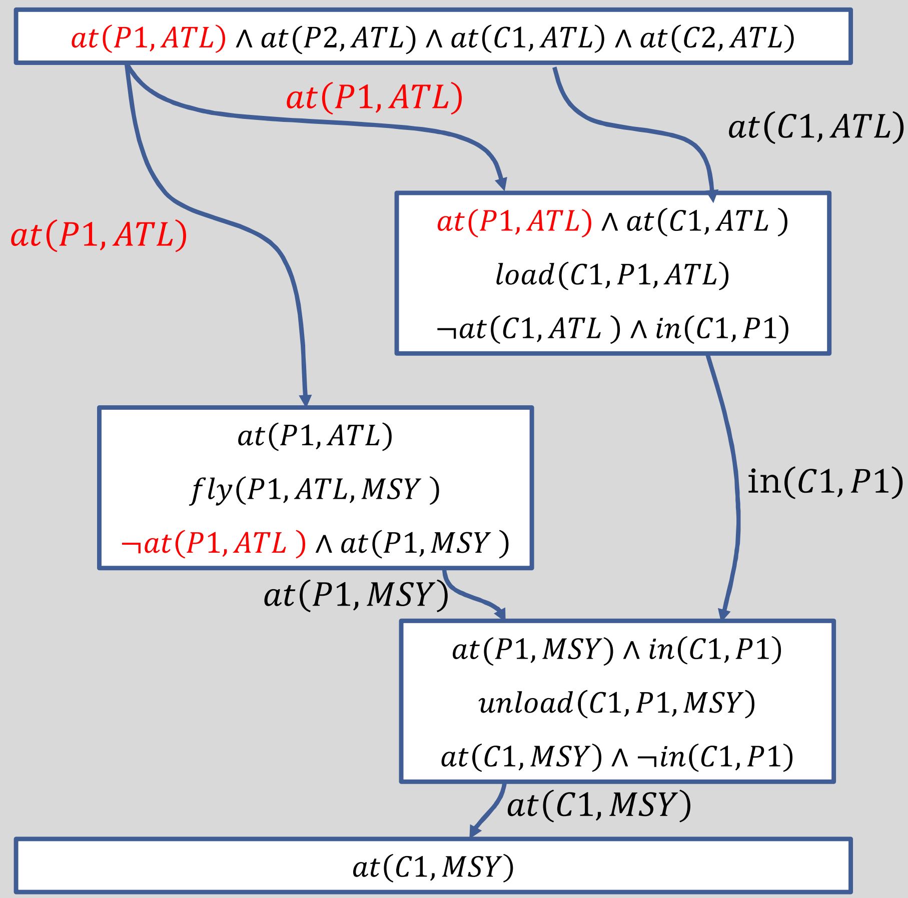
- The solution: we simply fix threatened causal links by adding additional orderings to the plan to ensure that the link is not threatened!

Fixing Threatened Causal Links

- Given a causal link $S1 \xrightarrow{p} S2$ and a step, T, which threatens it, we can remove the threat in one of two ways:
  - **Promotion**: order S2 < T -- have T go AFTER the causal link it threatens
  - **Demotion**: order T < S1 -- have T go BEFORE the causal link it threatens

- Or in other words:
  - S1 and S2 have a good thing going here, and we don't want T to muck it up. So T, you can happen either AFTER S2, once S1 and S2 have done their thing, OR you can happen before S1, and then S1 and S2 happen without worry.
  - Note: you can only do this if it doesn't make the partial ordering impossible (i.e., it doesn't create a cycle in the ordering.)

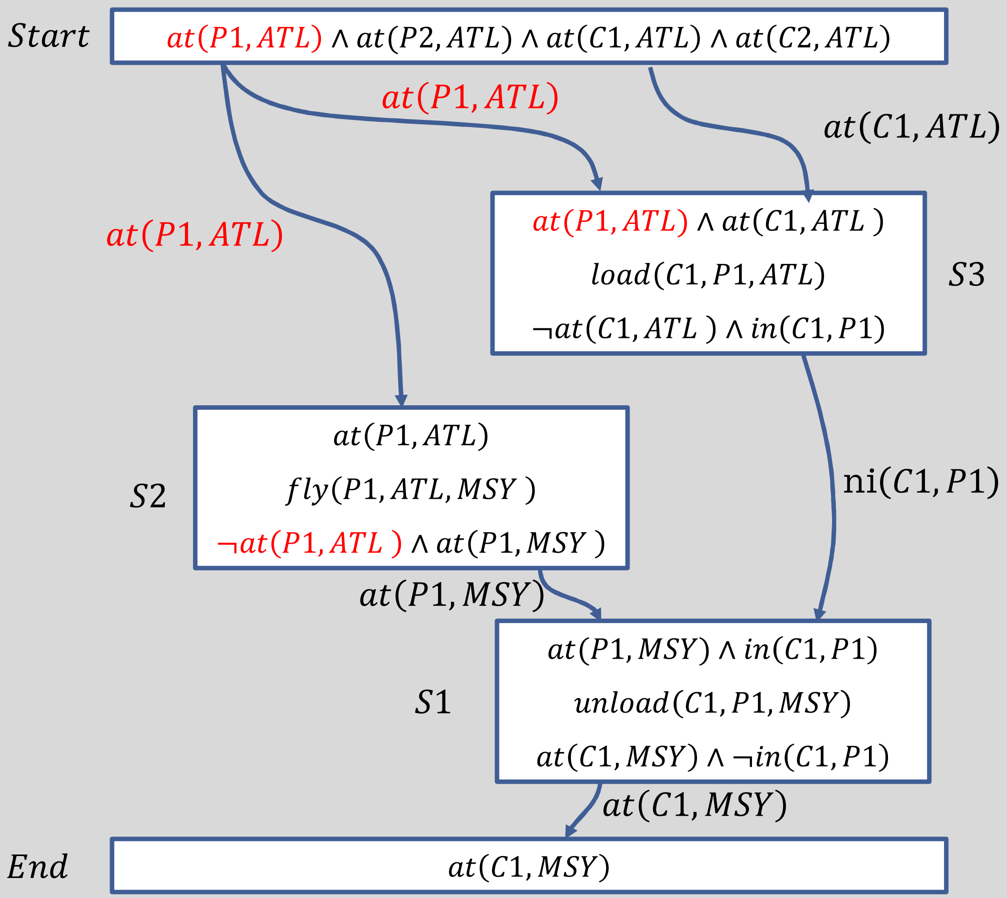
So, with our new promotion and demotion tools in our tool belt…



So, with our new promotion and demotion tools in our tool belt…

Note that we've changed the step names to Start, S1, S2, S3, and End

(S1 is the "bottom" one because we went backwards! It was the first step we added here)!



Here are all of the ordering we have!

It *doesn't yet* address the threat!

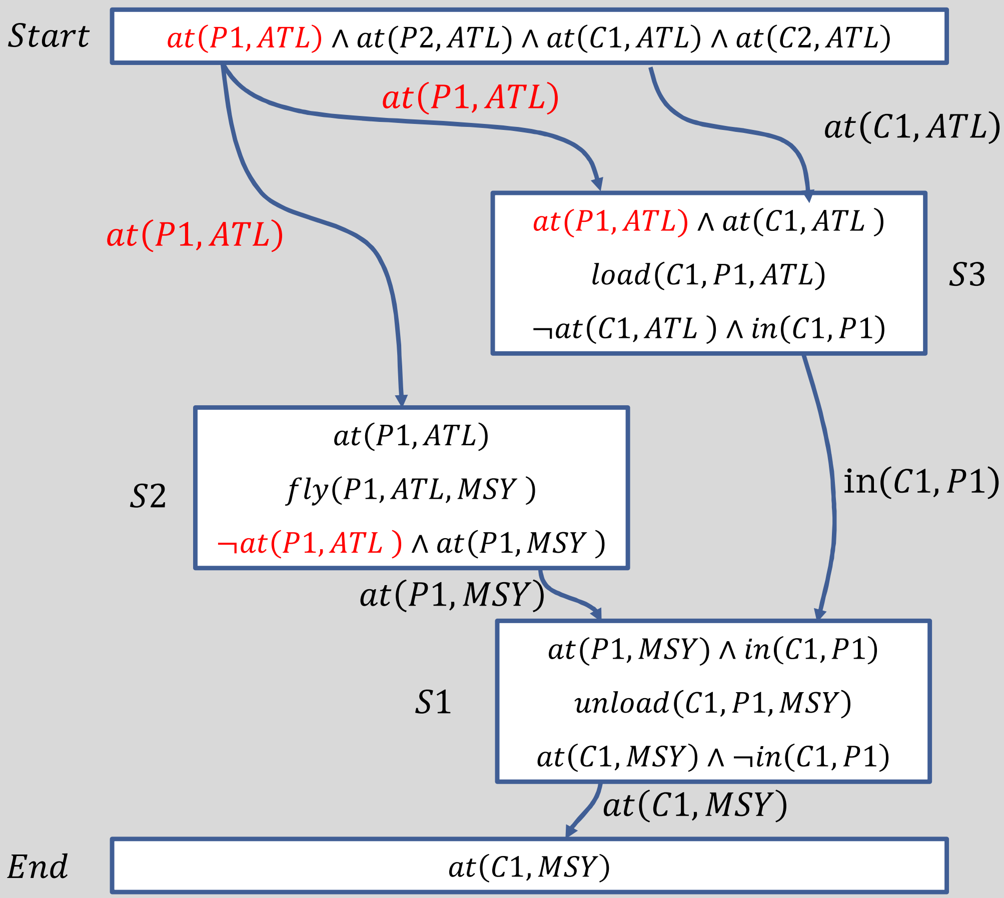
But it does say:

The Start step comes before everything.

The end step comes after everything.

S2 comes before S1

S3 comes before S1



Orderings:

$start < end$

$start < S1$

$S1 < end$

$start < S2$

$S2 < end$

$S2 < S1$

$start < S3$

$S3 < end$

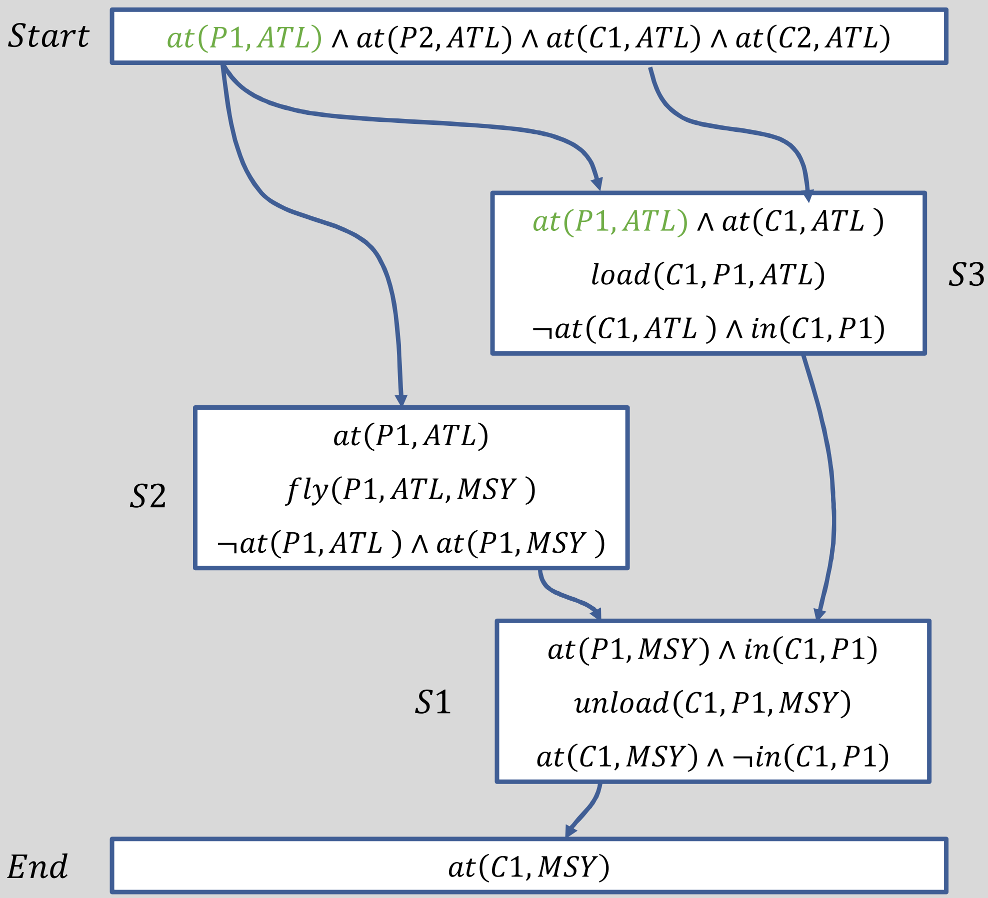
$S3 < S1$

S3 is threatened by S2 here.

So we can “promote” S2, to have it happen after the step it threatens, i.e., after S3

And it seems that this would work!

What would demotion look like?



Orderings:

$start < end$

$start < S1$

$S1 < end$

$start < S2$

$S2 < end$

$S2 < S1$

$start < S3$

$S3 < end$

$S3 < S1$

$S3 < S2$

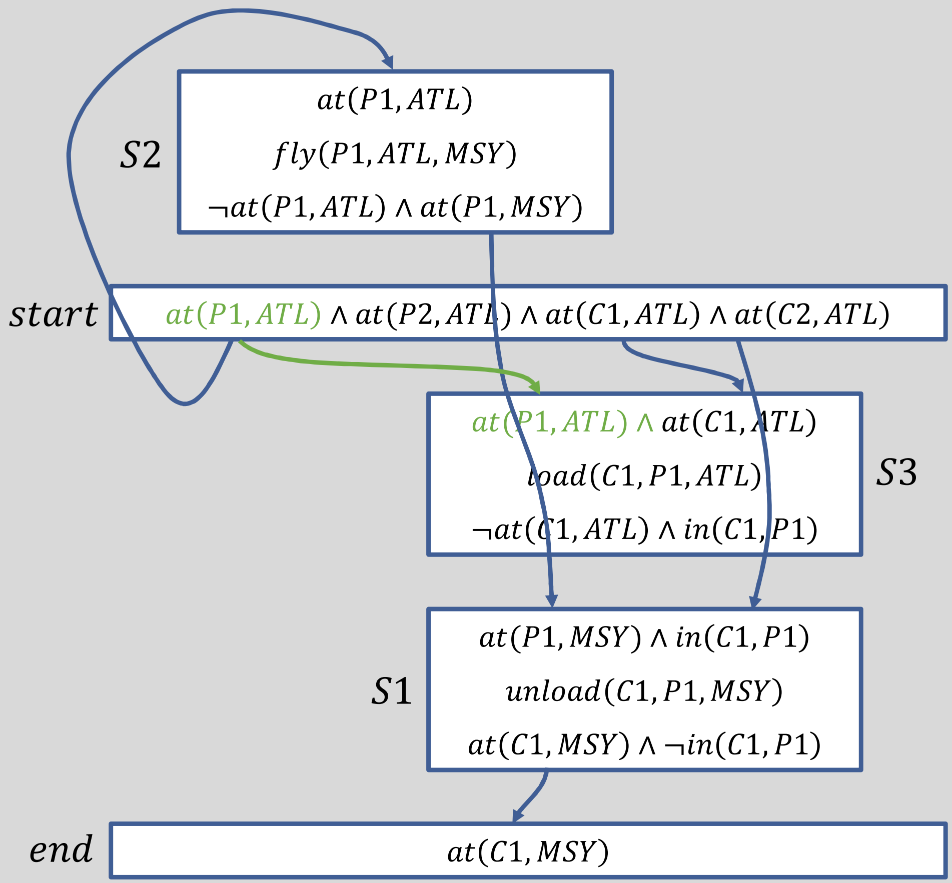
Promote

Demotion is: put the threatening step BEFORE the tail.

The threatening step is S2, and the tail is Start

But this doesn't work!

start is supposed to come before S2, so it violates a previous ordering!



Orderings:

$start < end$

$start < S1$

$S1 < end$

$start < S2$

$S2 < end$

$S2 < S1$

$start < S3$

$S3 < end$

$S3 < S1$

$S2 < start$

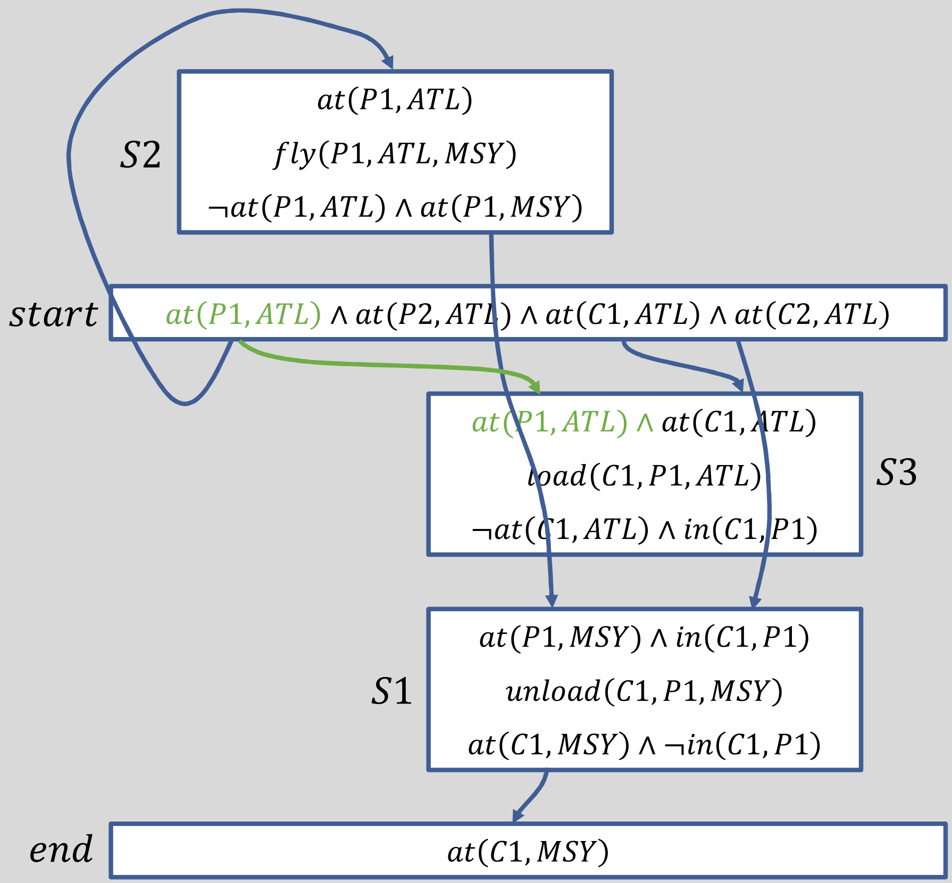
Demote

Demotion is: put the threatening step BEFORE the tail.

The threatening step is S2, and the tail is Start

But this doesn't work!

start is supposed to come before S2, so it violates a previous ordering!



Orderings:

$start < end$

$start < S1$

$S1 < end$

$start < S2$

$S2 < end$

$S2 < S1$

$start < S3$

$S3 < end$

$S3 < S1$

$S2 < start$

Demote Fails

Least Commitment Planning – Brief Recap

- OK. So, we’ve explored three ways to reduce the size of the search space.
  - Backwards search often lowers the branching factor by considering only relevant steps.
  - Leaving some variables unbound in a step allows it to represent many possible steps at once.
  - A partial ordering can represent many possible total orderings (as long as we ensure that no causal links are threatened).

- All of that is wonderful, but, each of these boons has a dark side…

Least Commitment Planning – No Current State!

- The problem with building plans in this way is that the idea of the *current state* no longer exists.
  - When a plan is built backwards, we don’t know what the early steps are until the end.
  - When a step has unbound variables, we don’t exactly know what it’s effects are.
  - We don’t know the exact sequence of steps in a partial order plan, so we don’t know the current state.
- But thinking in terms of state is perhaps the most straight forward way to think about these things!

State Space Search

- The most straight-forward way to view planning as search is state-space search:
  - **State**: A literal or conjunction of literals.
  - **Action**: Take an action whose precondition is met and modify the current state according to its effects.
  - **Goal**: Done when the goal holds in the current state.

- This is called state space search because the nodes in the search space are states. i.e., the literals that are “true” in the current state.

Plan-Space Search

- To perform least-commitment planning, we need to search the *space of plans.*
  - As opposed to a space of states.
- Remember in HW1, how you had to specify all the components of different search problems? For the monkey and the jugs and stuff? Here's the components for a space of plans:
  - **State**: A (possibly incomplete) partial order plan.
  - **Transition Model**: Modifying the plan by adding steps, orderings, or causal links.
  - **Goal Test**: Done when all the goal and preconditions have been satisfied by causal links.
- Because the nodes in the search space are plans, we call this **plan-space search**.

Refinement Search

- **POCL** (partial order causal link) planning is a kind of **refinement search**.

- A plan is a data structure. If that data structure is incomplete, it has a set of **flaws** which describe how it is incomplete.
  - We will talk about what the data structure looks like momentarily!

- Search proceeds by choosing a flaw and fixing it (possibly creating new flaws in the process).

- Search is done when no flaws remain.

POCL Plan Data-Structure

- A plan is composed of four sets:
  - A set of **steps**. Some variables may not be bound.
  - A set of **bindings** which constrain which values the variables can have.
  - A set of **orderings** which define a partial ordering of the steps.
  - A set of **causal links** which keeps track of how goals are achieved.

POCL Plan Flaws

- There are two types of flaws that you might find:

- An **open precondition flaw**: for some step S, with precondition p, indicates that there is no causal link that establishes *p*.
- A **threatened causal link flaw**: indicates that a threatened causal link exists.

- Or in other words: *open precondition flaws* help us create causal links, and *threatened causal link flaws* help us make those causal links play nice with each other!

Partial Order Plan (POP) Algorithm

```
Begin with the null plan and empty set of flaws F.
For each goal conjunct, add an open precondition flaw to F.
To refine a plan with flaws:
    If the plan has no flaws, return it as a solution.
    Choose a flaw X from F to repair.
    If X is an open precondition flaw for literal L of step S:
        Choose some action A which has L as an effect:
            A can be a step already in the plan, or
            A can be a new step (Add open precondition flaws for A’s preconditions to F.)
        Add a causal link from A to S with label L.
        Add any new threatened causal link flaws to F.
    If X is a threatened causal link flaw:
        Promote: Move the threatening step after the head.
        Demote: Move the threatening step before the tail.
    Recursively repair the refined plan.
```

Partial Order Plan (POP) Algorithm

```
Begin with the null plan and empty set of flaws F.
For each goal conjunct, add an open precondition flaw to F.
To refine a plan with flaws:
    If the plan has no flaws, return it as a solution.
    Choose a flaw X from F to repair.
    If X is an open precondition flaw for literal L of step S:
        Choose some action A which has L as an effect:
            A can be a step already in the plan, or
            A can be a new step (Add open precondition flaws for A’s preconditions to F.)
        Add a causal link from A to S with label L.
        Add any new threatened causal link flaws to F.
    If X is a threatened causal link flaw:
        Promote: Move the threatening step after the head.
        Demote: Move the threatening step before the tail.
    Recursively repair the refined plan.
```

The basic premise: We go through each part of the goal, and add it as an “open” Flaw

Partial Order Plan (POP) Algorithm

```
Begin with the null plan and empty set of flaws F.
For each goal conjunct, add an open precondition flaw to F.
To refine a plan with flaws:
    If the plan has no flaws, return it as a solution.
    Choose a flaw X from F to repair.
    If X is an open precondition flaw for literal L of step S:
        Choose some action A which has L as an effect:
            A can be a step already in the plan, or
            A can be a new step (Add open precondition flaws for A’s preconditions to F.)
        Add a causal link from A to S with label L.
        Add any new threatened causal link flaws to F.
    If X is a threatened causal link flaw:
        Promote: Move the threatening step after the head.
        Demote: Move the threatening step before the tail.
    Recursively repair the refined plan.
```

If there’s no flaws, you’re done!

But otherwise, we’ll choose a flaw to repair.

Once you’ve repaired it, you’ll have a “refined plan” (possibly with even more flaws). Recursively repair it!

Partial Order Plan (POP) Algorithm

```
Begin with the null plan and empty set of flaws F.
For each goal conjunct, add an open precondition flaw to F.
To refine a plan with flaws:
    If the plan has no flaws, return it as a solution.
    Choose a flaw X from F to repair.
    If X is an open precondition flaw for literal L of step S:
        Choose some action A which has L as an effect:
            A can be a step already in the plan, or
            A can be a new step (Add open precondition flaws for A’s preconditions to F.)
        Add a causal link from A to S with label L.
        Add any new threatened causal link flaws to F.
    If X is a threatened causal link flaw:
        Promote: Move the threatening step after the head.
        Demote: Move the threatening step before the tail.
    Recursively repair the refined plan.
```

How to repair a flaw? Well, it can be one of two flavors…

If it is the “open” flaw:
1.)choose an action which makes the “missing” thing happen.
2.)And a causal link from that action to where the “open” flaw was.
3.) Did this threaten any other causal links? If so, add new flaws to the list.

Partial Order Plan (POP) Algorithm

How to repair a flaw? Well, it can be one of two flavors…

If it is the “threatened” flaw:

Either promote or demote, whichever works!

```
Begin with the null plan and empty set of flaws F.
For each goal conjunct, add an open precondition flaw to F.
To refine a plan with flaws:
    If the plan has no flaws, return it as a solution.
    Choose a flaw X from F to repair.
    If X is an open precondition flaw for literal L of step S:
        Choose some action A which has L as an effect:
            A can be a step already in the plan, or
            A can be a new step (Add open precondition flaws for A’s preconditions to F.)
        Add a causal link from A to S with label L.
        Add any new threatened causal link flaws to F.
    If X is a threatened causal link flaw:
        Promote: Move the threatening step after the head.
        Demote: Move the threatening step before the tail.
    Recursively repair the refined plan.
```

Partial Order Plan (POP) Algorithm

Let's see it in action!

```
Begin with the null plan and empty set of flaws F.
For each goal conjunct, add an open precondition flaw to F.
To refine a plan with flaws:
    If the plan has no flaws, return it as a solution.
    Choose a flaw X from F to repair.
    If X is an open precondition flaw for literal L of step S:
        Choose some action A which has L as an effect:
            A can be a step already in the plan, or
            A can be a new step (Add open precondition flaws for A's preconditions to F.)
        Add a causal link from A to S with label L.
        Add any new threatened causal link flaws to F.
    If X is a threatened causal link flaw:
        Promote: Move the threatening step after the head.
        Demote: Move the threatening step before the tail.
    Recursively repair the refined plan.
```

$start$

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

Flaw: $at(C1, MSY)$ open

Steps: $start, end$

Bindings:

Orderings: $start < end,$

Causal Links:

We begin by looking at the goals and adding each literal as an “open precondition” flaw.

Note: the red text means “this is a flaw that we haven’t started addressing yet.”

$at(C1, MSY)$

end

$start$

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

Flaw: $at(C1, MSY)$ open

Steps: $start, end, S1$

Bindings:

Orderings: $start < end,$

Causal Links:

We find new action that can potentially satisfy the flaw! This involves a few steps..

$S1$

$at(p_1, a_1) \wedge in(c_1, p_1)$

$unload(c_1, p_1, a_1)$

$at(c_1, a_1) \wedge \neg in(c_1, p_1)$

$at(C1, MSY)$

end

$start$

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

Flaw: $at(C1, MSY)$ open

Steps: $start, end, S1$

Bindings: $c_1 = C1, a_1 = MSY$

Orderings: $start < end,$

Causal Links:

We do the Unification to find bindings to ensure that this action *does* indeed achieve the predicate in our flaw.

$S1$

$at(p_1, MSY) \wedge in(C1, p_1)$

$unload(C1, p_1, MSY)$

$at(C1, MSY) \wedge \neg in(C1, p_1)$

$at(C1, MSY)$

end

start

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

Flaw: $at(C1, MSY)$ open

Steps: $start, end, S1$

Bindings: $c_1 = C1, a_1 = MSY$

Orderings: $start < end,$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$

We add the causal link! Our new step makes at(c1,MSY) happen as an effect, which the end step needs.

$S1$

$at(p_1, MSY) \wedge in(C1, p_1)$

$unload(C1, p_1, MSY)$

$at(C1, MSY) \wedge \neg in(C1, p_1)$

$at(C1, MSY)$

end

start

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

Flaw: $at(C1, MSY)$ open

Steps: $start, end, S1$

Bindings: $c_1 = C1, a_1 = MSY$

Orderings: $start < end, start < S1, S1 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$

And we situate this new step in our orderings. Start happens before S1, and S1 happens before the end step.

Cool! We updated our plan based on this new action – we addressed the flaw *at(C1, MSY).*

$S1$

$at(p_1, MSY) \wedge in(C1, p_1)$

$unload(C1, p_1, MSY)$

$at(C1, MSY) \wedge \neg in(C1, p_1)$

$at(C1, MSY)$

end

$start$

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

Flaw: $at(p_1, MSY)$ open

Steps: $start, end, S1$

Bindings: $c_1 = C1, a_1 = MSY$

Orderings: $start < end, start < S1, S1 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$

But we still have two open precondition flaws at this point.

Let's start tackling $at(p_1, MSY)$

$S1$

$at(p_1, MSY) \wedge in(C1, p_1)$

$unload(C1, p_1, MSY)$

$at(C1, MSY) \wedge \neg in(C1, p_1)$

$at(C1, MSY)$

end

start

$at(P1, ATL) \land at(P2, ATL) \land at(C1, ATL) \land at(C2, ATL)$

Flaw: $at(p_1, MSY)$ open

Steps: $start, end, S1,$ $S2$

Bindings: $c_1 = C1, a_1 = MSY$

Orderings: $start < end, start < S1, S1 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$

We need another new action to satisfy that literal. We add S2 to our set of Steps.

$S2$

$at(p_2, a_2)$

$fly(p_2, a_2, a_3)$

$\neg at(p_2, a_2) \land at(p_2, a_3)$

$S1$

$at(p_1, MSY) \land in(C1, p_1)$

$unload(C1, p_1, MSY)$

$at(C1, MSY) \land \neg in(C1, p_1)$

end

$at(C1, MSY)$

start

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

Flaw: $at(p_1, MSY)$ open

Steps: $start, end, S1,$ $S2$

Bindings: $c_1 = C1, a_1 = MSY,$ $a_3 = MSY, p_1 = p_2, a_1 = a_3$

Orderings: $start < end, start < S1, S1 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end,$

We unify to discover our new bindings.

$S2$

$at(p_2, a_2)$

$fly(p_2, a_2, MSY)$

$\neg at(p_2, a_2) \wedge at(p_2, MSY)$

$S1$

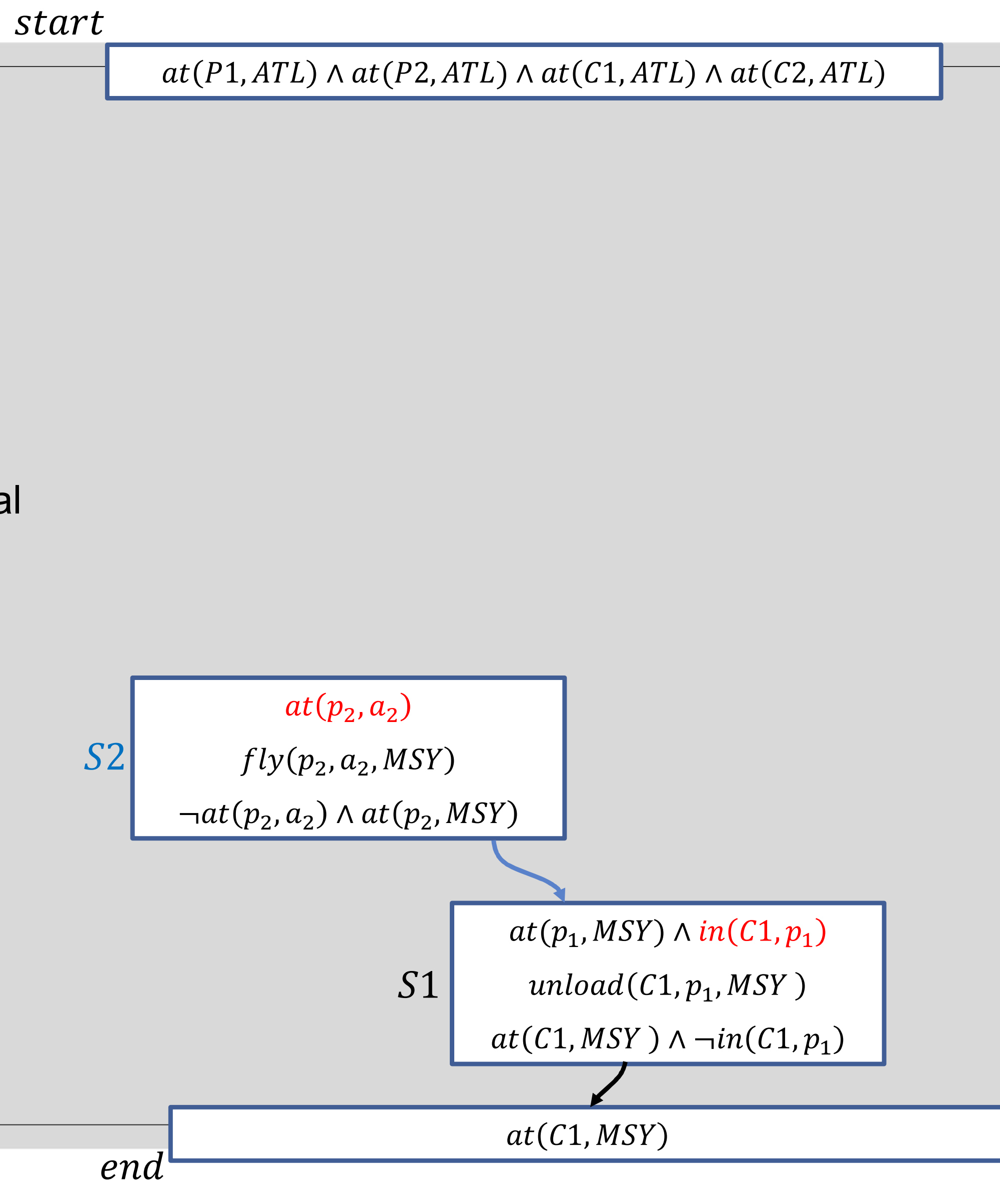
$at(p_1, MSY) \wedge in(C1, p_1)$

$unload(C1, p_1, MSY)$

$at(C1, MSY) \wedge \neg in(C1, p_1)$

$at(C1, MSY)$

end



We add a new causal link

Flaw: $at(p_1, MSY)$ open

Steps: $start, end, S1, S2$

Bindings: $c_1 = C1, a_1 = MSY,$ $a_3 = MSY, p_1 = p_2, a_1 = a_3$

Orderings: $start < end, start < S1, S1 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end,$ $S2 \xrightarrow{at(P1,MSY)} S1$

start

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

Flaw: $at(p_1, MSY)$ open

Steps: $start, end, S1, S2$

Bindings: $c_1 = C1, a_1 = MSY,$ $a_3 = MSY, p_1 = p_2, a_1 = a_3$

Orderings: $start < end, start < S1, S1 < end,$ $start < S2, S2 < S1, S2 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end,$ $S2 \xrightarrow{at(P1,MSY)} S1$

We update our orderings!

Look at us, we're on a roll!
That's two steps in the books!

$S2$

$at(p_2, a_2)$

$fly(p_2, a_2, MSY)$

$\neg at(p_2, a_2) \wedge at(p_2, MSY)$

$S1$

$at(p_1, MSY) \wedge in(C1, p_1)$

$unload(C1, p_1, MSY)$

$at(C1, MSY) \wedge \neg in(C1, p_1)$

$at(C1, MSY)$

end

start

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

We move on to the next open flaw: $at(p_2, a_2)$

Do we need another new action for this guy?

Flaw: $at(p_2, a_2)$ open

Steps: $start, end, S1, S2$

Bindings: $c_1 = C1, a_1 = MSY, a_3 = MSY, p_1 = p_2, a_1 = a_3$

Orderings: $start < end, start < S1, S1 < end, start < S2, S2 < S1, S2 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$

$S2$

$at(p_2, a_2)$

$fly(p_2, a_2, MSY)$

$\neg at(p_2, a_2) \wedge at(p_2, MSY)$

$S1$

$at(p_1, MSY) \wedge in(C1, p_1)$

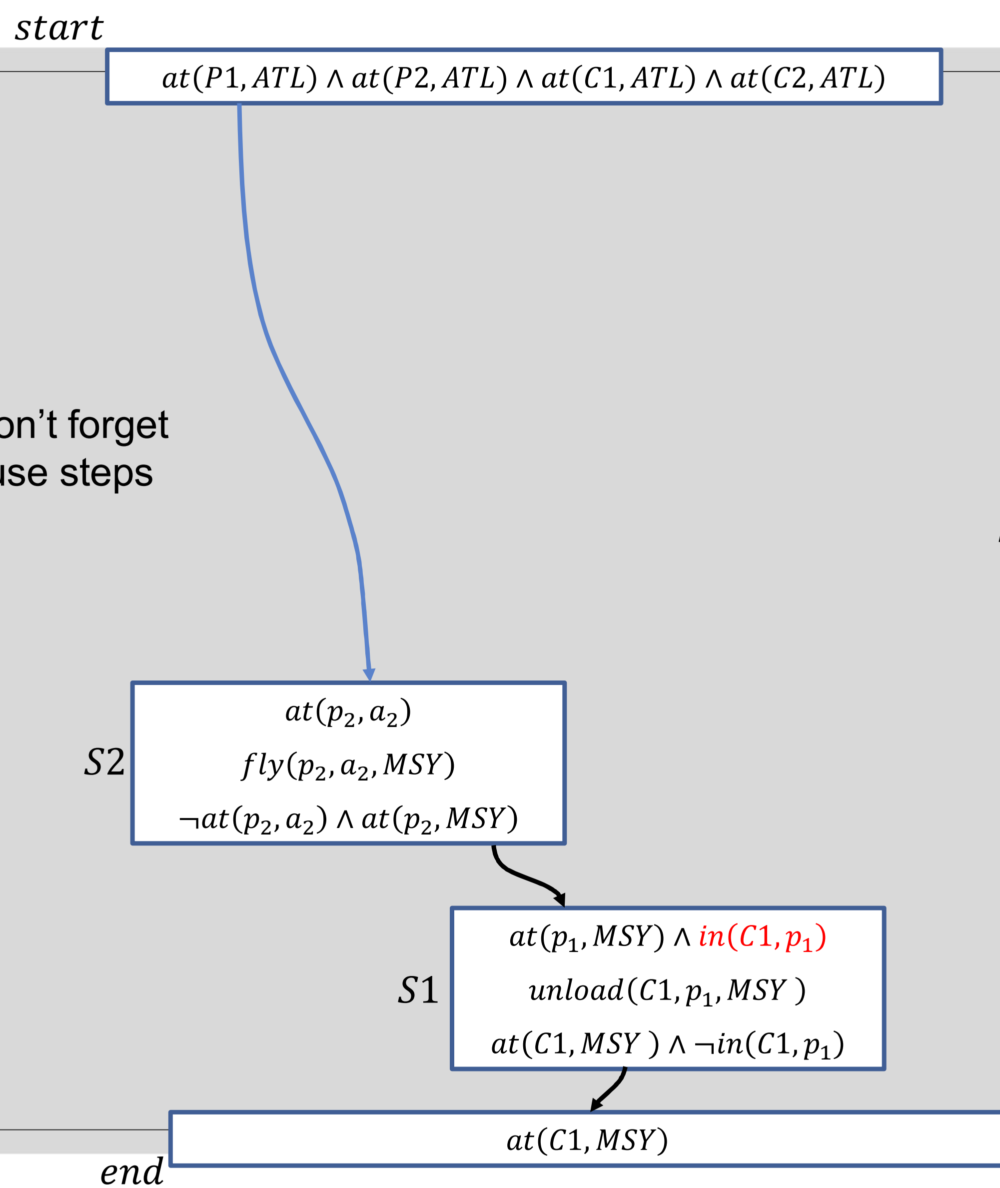
$unload(C1, p_1, MSY)$

$at(C1, MSY) \wedge \neg in(C1, p_1)$

end

$at(C1, MSY)$

Well, no, actually! Don't forget – you're allowed to use steps that already exist!



Flaw: $at(p_2, a_2)$ open

Steps: $start, end, S1, S2$

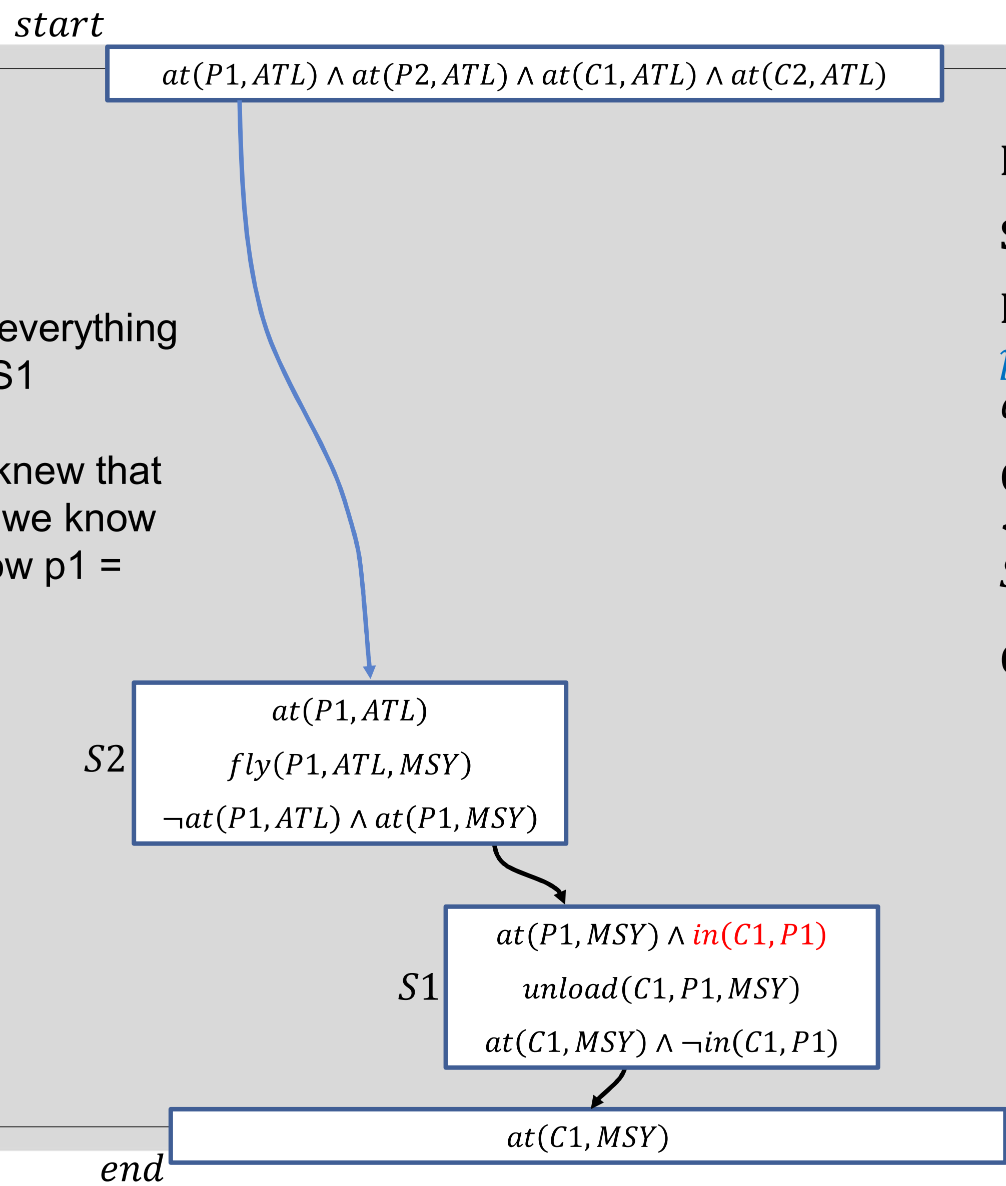
Bindings: $c_1 = C1, a_1 = MSY, a_3 = MSY, p_1 = p_2, a_1 = a_3$

Orderings: $start < end$, $start < S1$, $S1 < end$, $start < S2$, $S2 < S1$, $S2 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$

Unifying here fills in everything from both S2 *and* S1

(before in S1 all we knew that p1=p2, but now that we know that p2 = P1, we know p1 = P1 also!)



Flaw: $at(p_2, a_2)$ open

Steps: $start, end, S1, S2$

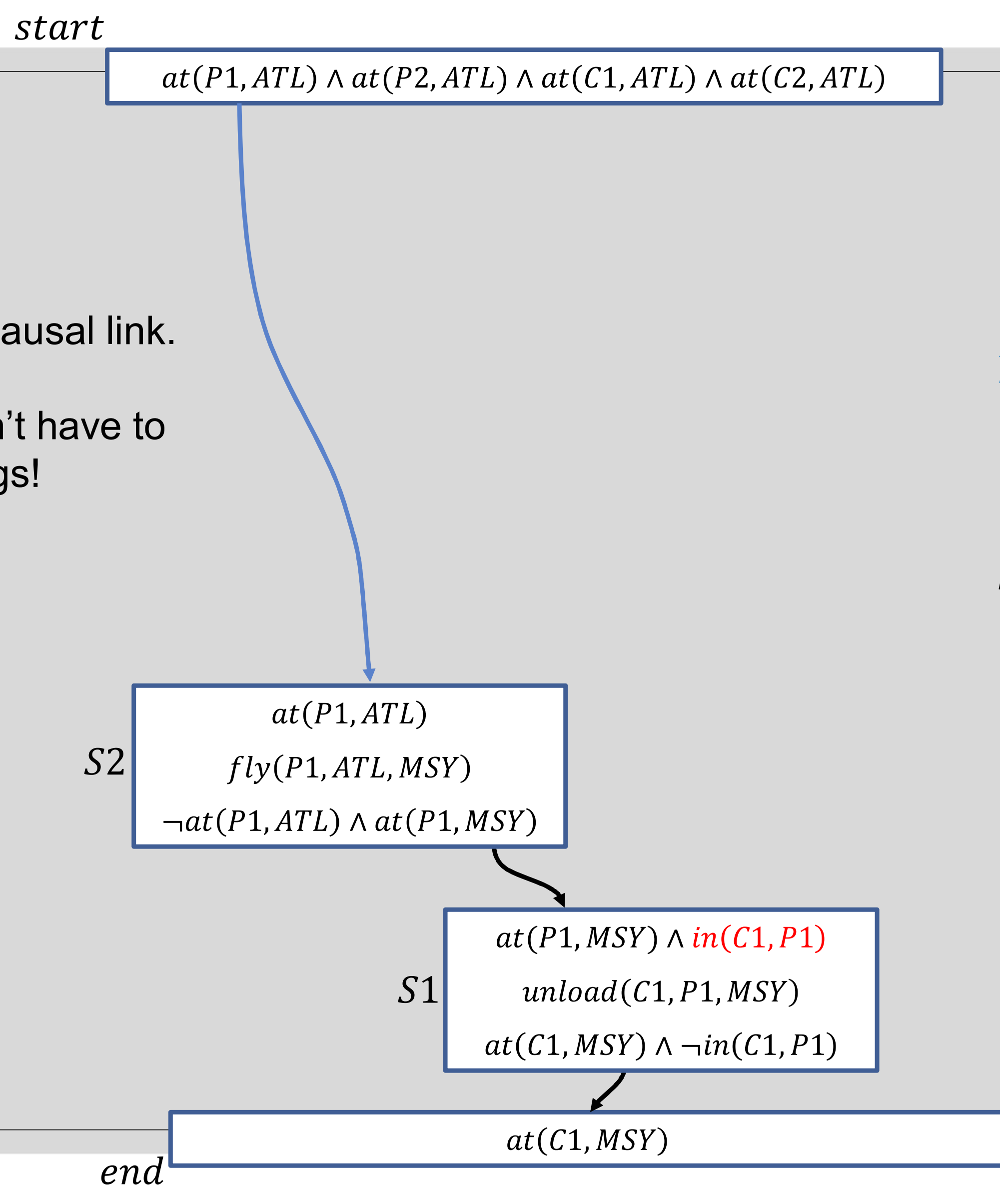
Bindings: $c_1 = C1, a_1 = MSY,$ $p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3$

Orderings: $start < end, start < S1, S1 < end, start < S2, S2 < S1, S2 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end,$ $S2 \xrightarrow{at(P1,MSY)} S1$

We also add in the causal link.

But note that we don't have to futz with the orderings!



Flaw: $at(p_2, a_2)$ open

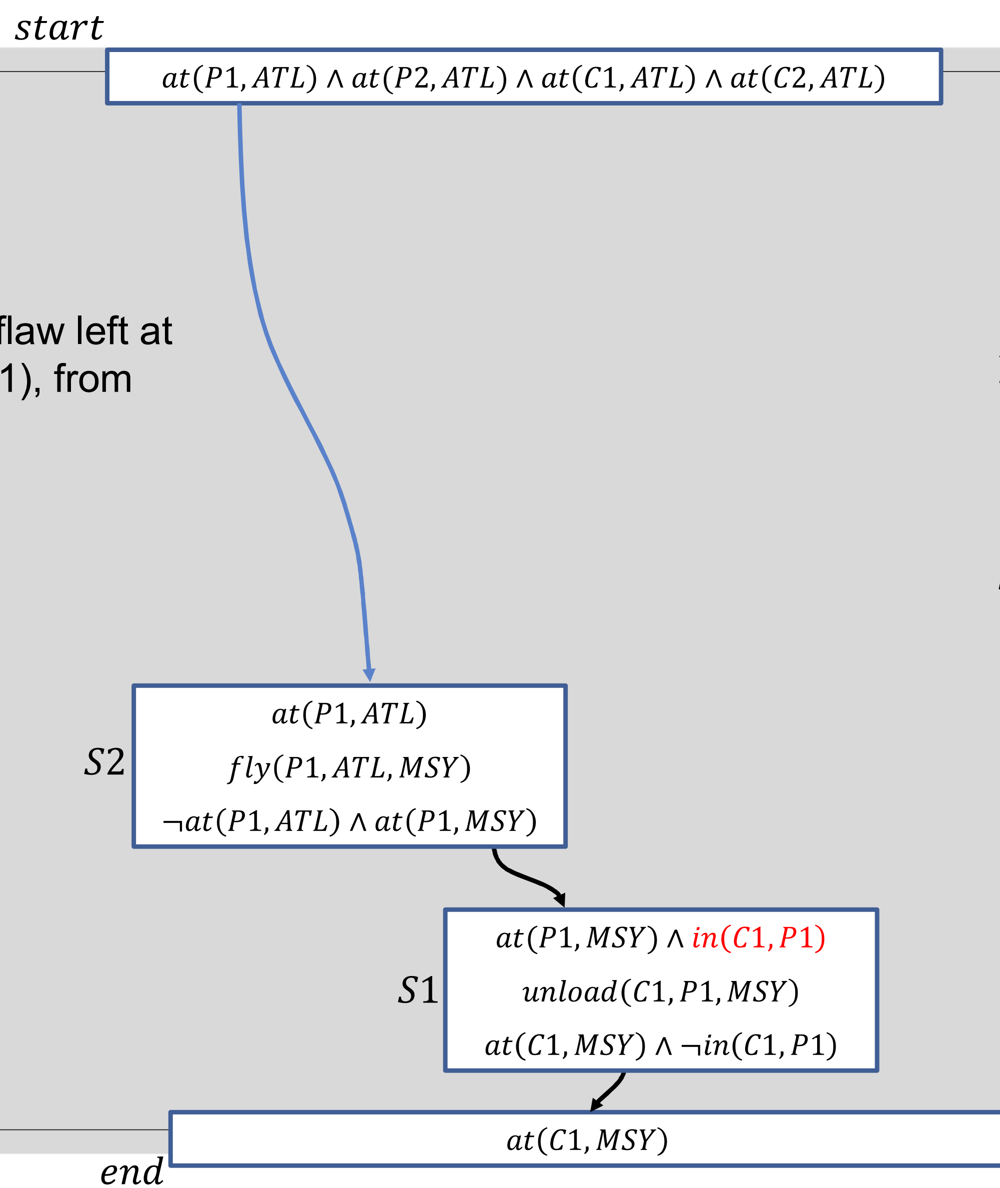
Steps: $start, end, S1, S2$

Bindings: $c_1 = C1, a_1 = MSY,$
$p_1 = P1, p_2 = P1, a_2 = ATL$
$a_3 = MSY, p_1 = p_2, a_1 = a_3$

Orderings: $start < end, start < S1, S1 < end, start < S2, S2 < S1, S2 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end,$
$S2 \xrightarrow{at(P1,MSY)} S1, start \xrightarrow{at(P1,ATL)} S2$

OK, great, only one flaw left at the moment: in(c1,P1), from S1



Flaw: $in(C1, P1)$ open

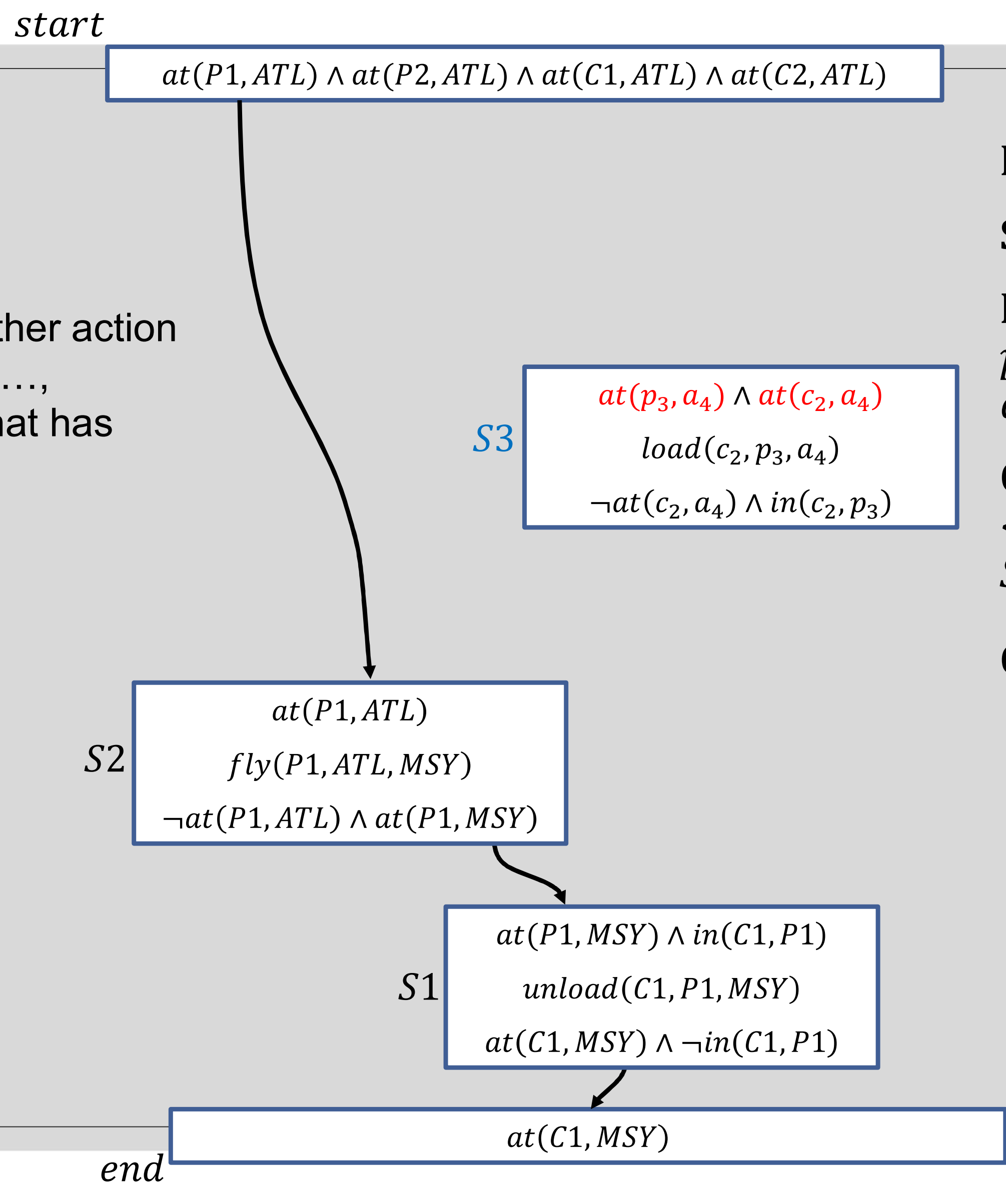
Steps: $start, end, S1, S2$

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$
$a_3 = MSY, p_1 = p_2, a_1 = a_3$

Orderings: $start < end, start < S1, S1 < end, start < S2, S2 < S1, S2 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1, start \xrightarrow{at(P1,ATL)} S2$

Again, we need another action to make this happen…, nothing that exists that has “in” as an effect.



Flaw: $in(C1, P1)$ open

Steps: $start, end$, S1, S2, S3

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3$

Orderings: $start < end, start < S1, S1 < end, start < S2, S2 < S1, S2 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1, start \xrightarrow{at(P1,ATL)} S2$

start

$at(P1, ATL) \wedge at(P2, ATL) \wedge at(C1, ATL) \wedge at(C2, ATL)$

And so we add the bindings
c2 = C1
P3 = P1

And more broadly
c2 = c1 (i.e., the variable that C1 is currently bound to)
p3 = p1 (i.e., the variable that P1 is currently bound to)

$S3$

$at(P1, a_4) \wedge at(C1, a_4)$

$load(C1, P1, a_4)$

$\neg at(C1, a_4) \wedge in(C1, P1)$

$S2$

$at(P1, ATL)$

$fly(P1, ATL, MSY)$

$\neg at(P1, ATL) \wedge at(P1, MSY)$

$S1$

$at(P1, MSY) \wedge in(C1, P1)$

$unload(C1, P1, MSY)$

$at(C1, MSY) \wedge \neg in(C1, P1)$

$at(C1, MSY)$

end

Flaw: $in(C1, P1)$ open

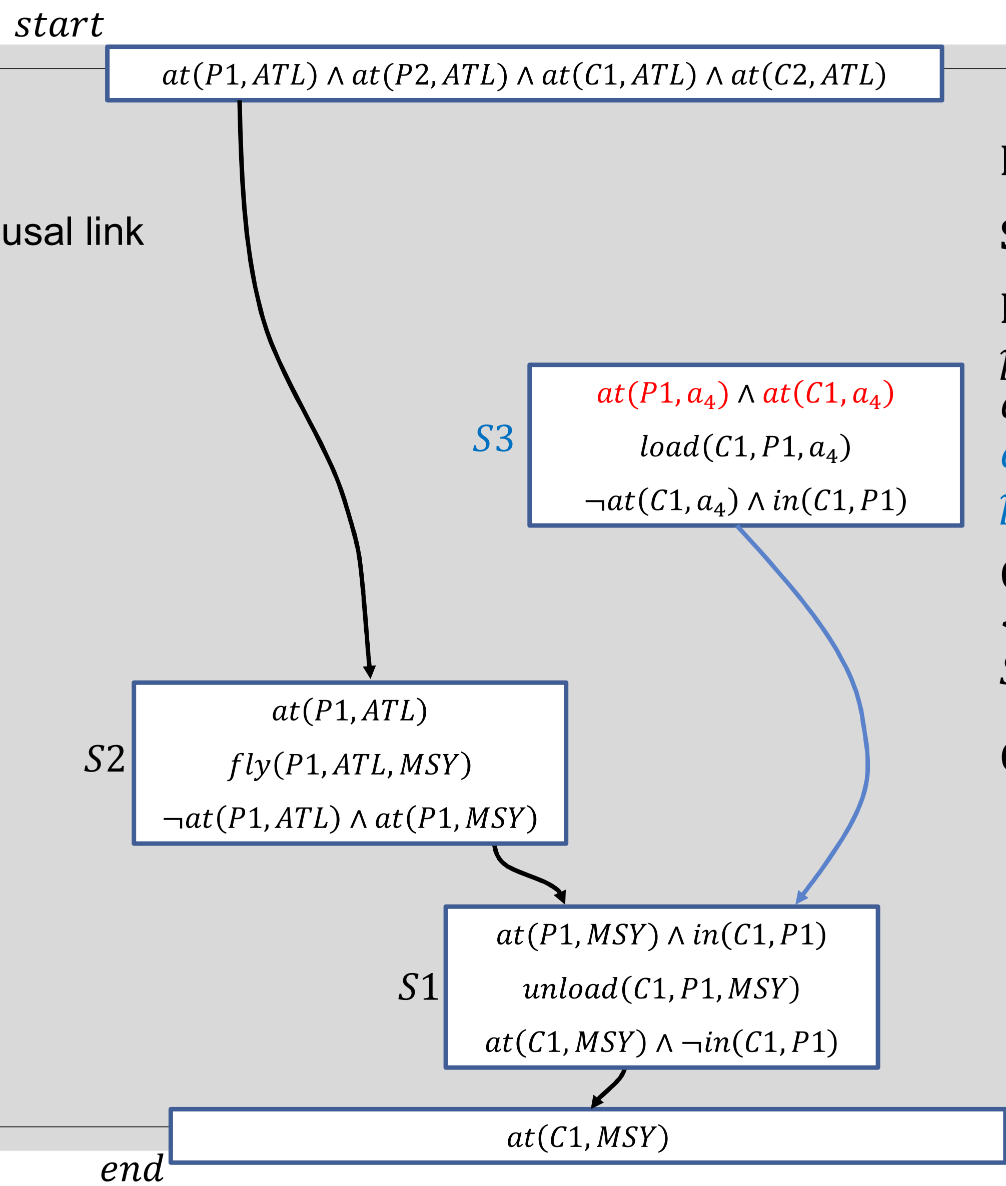
Steps: $start, end, S1, S2, S3$

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$
$a_3 = MSY, p_1 = p_2, a_1 = a_3,$
$c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4$

Orderings: $start < end, start < S1, S1 < end, start < S2, S2 < S1, S2 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$, $start \xrightarrow{at(P1,ATL)} S2$

And we make the causal link official



Flaw: $in(C1, P1)$ open

Steps: $start, end, S1, S2, S3$

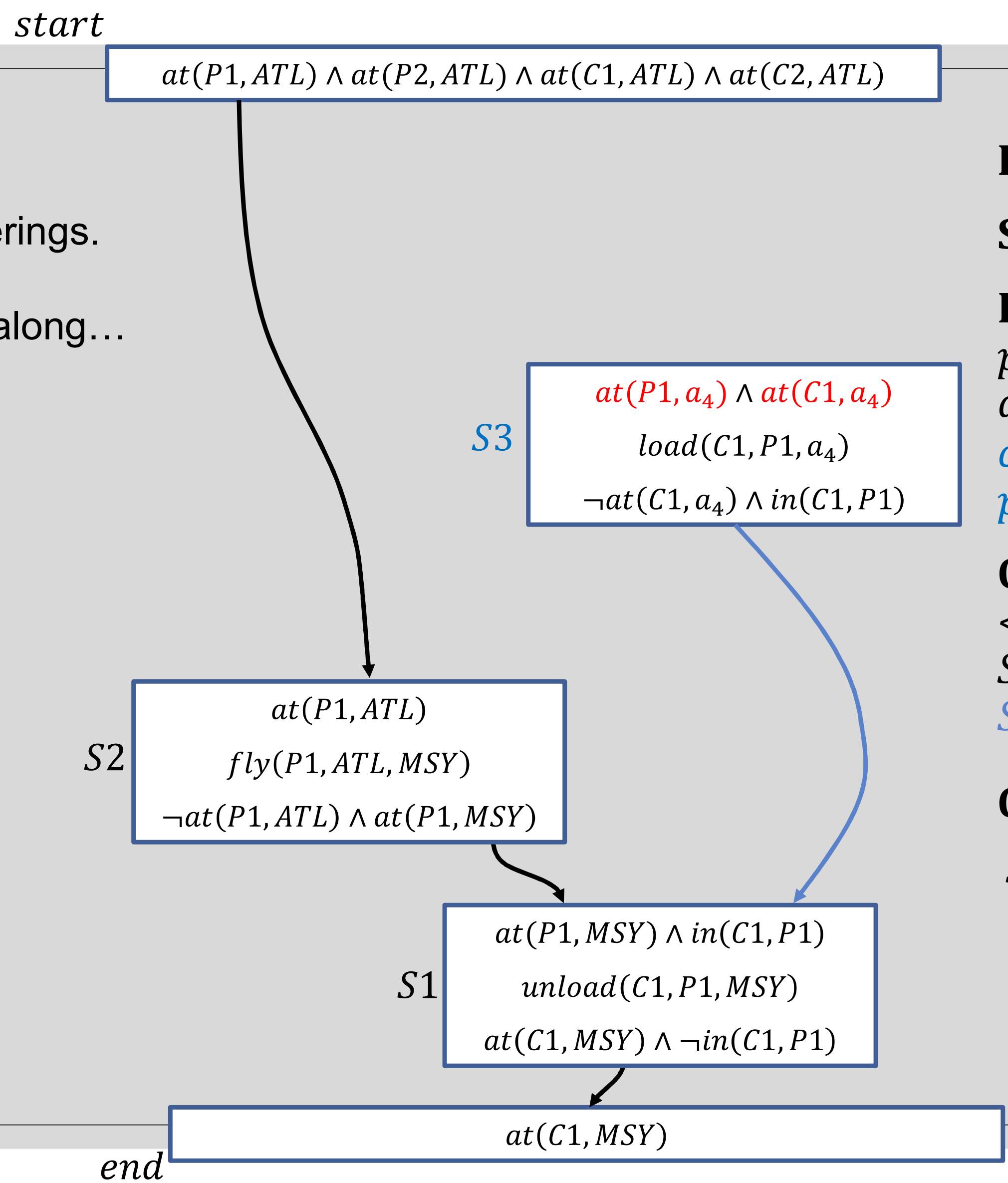
Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3,$ $c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4$

Orderings: $start < end$, $start < S1$, $S1 < end$, $start < S2$, $S2 < S1$, $S2 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$, $start \xrightarrow{at(P1,ATL)} S2$, $S3 \xrightarrow{in(C1,P1)} S1$

And update our orderings.

Great, moving right along…



Flaw: $in(C1, P1)$ open

Steps: $start, end$, S1, S2, S3

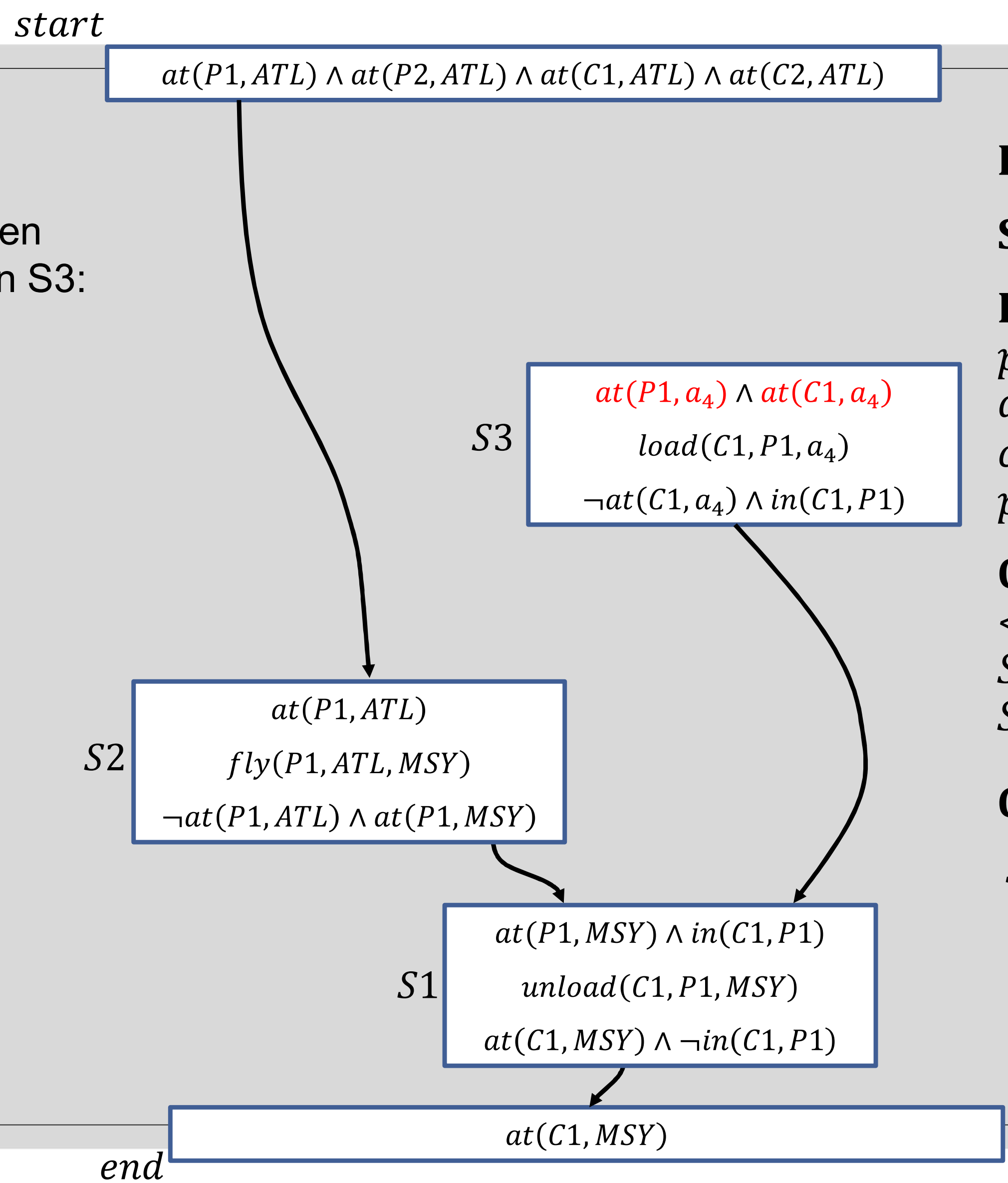
Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3,$ $c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4$

Orderings: $start < end, start < S1, S1 < end, start < S2, S2 < S1, S2 < end, start < S3, S3 < S1, S3 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$, $start \xrightarrow{at(P1,ATL)} S2$, $S3 \xrightarrow{in(C1,P1)} S1$

We have another open condition flaw, now in S3:

$at(P1, a_4)$



Flaw: $at(P1, a_4)$ open

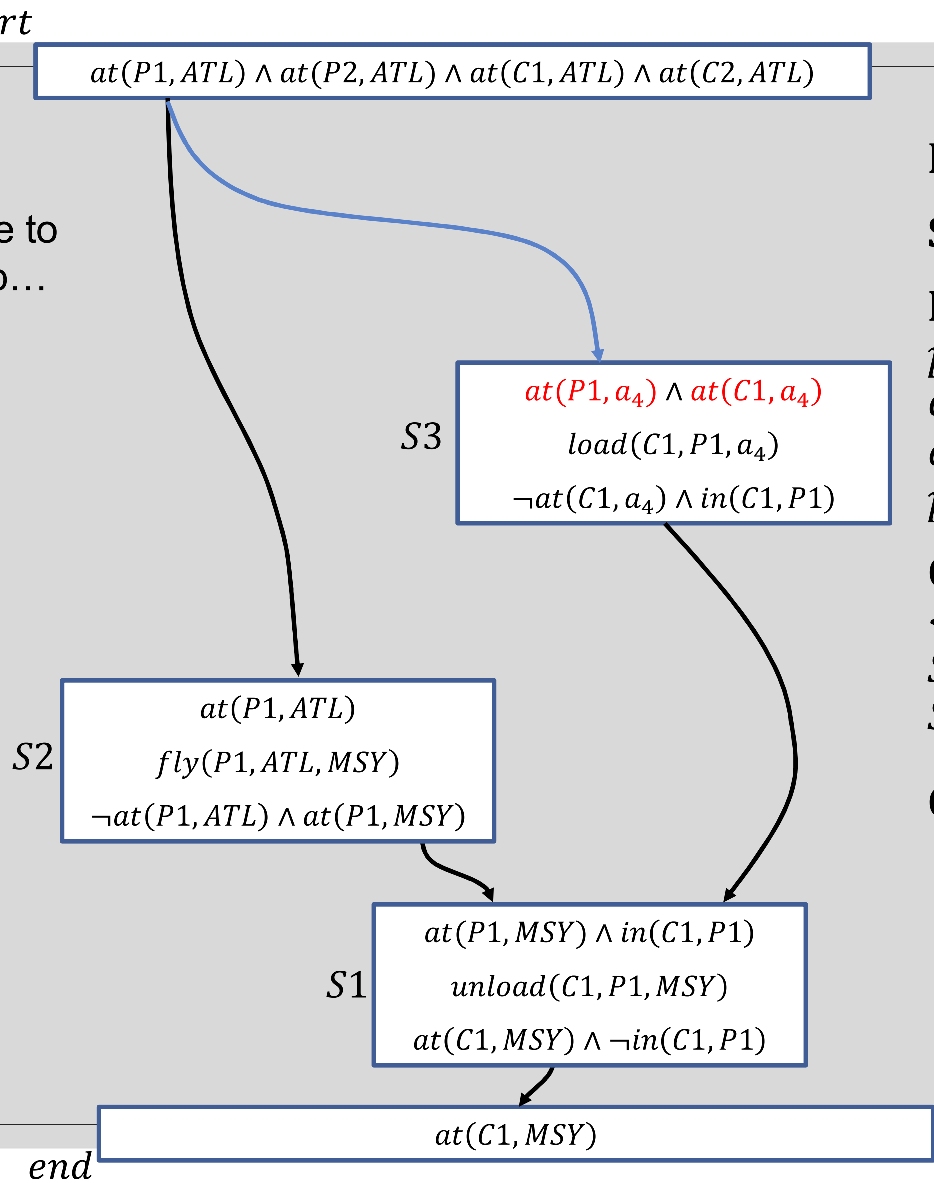
Steps: $start, end, S1, S2, S3$

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3, c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4$

Orderings: $start < end, start < S1, S1 < end, start < S2, S2 < S1, S2 < end, start < S3, S3 < S1, S3 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$, $start \xrightarrow{at(P1,ATL)} S2$, $S3 \xrightarrow{in(C1,P1)} S1$

We see we might be able to leverage an existing step…



Flaw: $at(P1, a_4)$ open

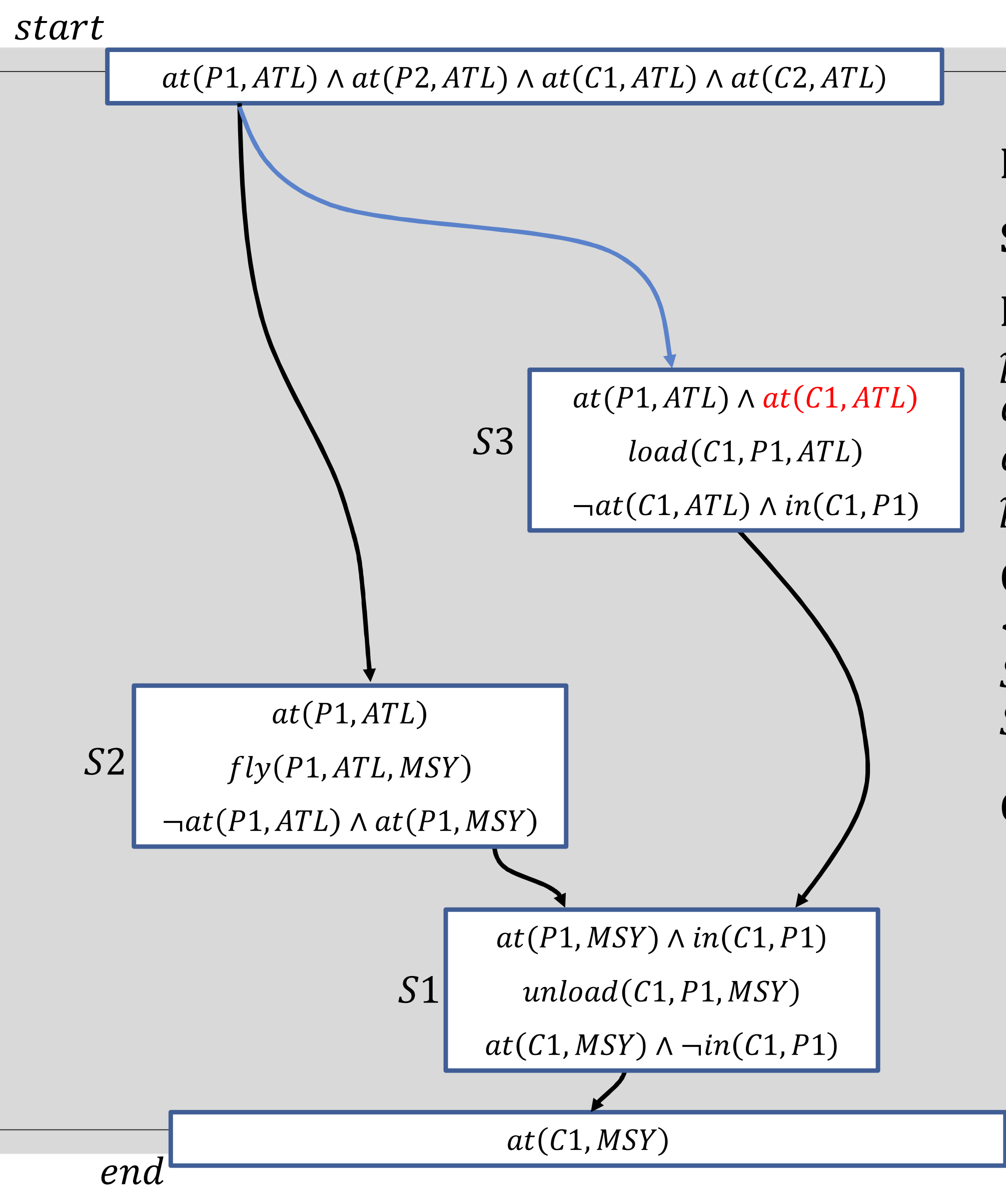
Steps: $start, end, S1, S2, S3$

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3,$ $c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4$

Orderings: $start < end, start < S1, S1 < end, start < S2, S2 < S1, S2 < end, start < S3, S3 < S1, S3 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end,$ $S2 \xrightarrow{at(P1,MSY)} S1, start \xrightarrow{at(P1,ATL)} S2,$ $S3 \xrightarrow{in(C1,P1)} S1$

We unify



Flaw: $at(P1, a_4)$ open

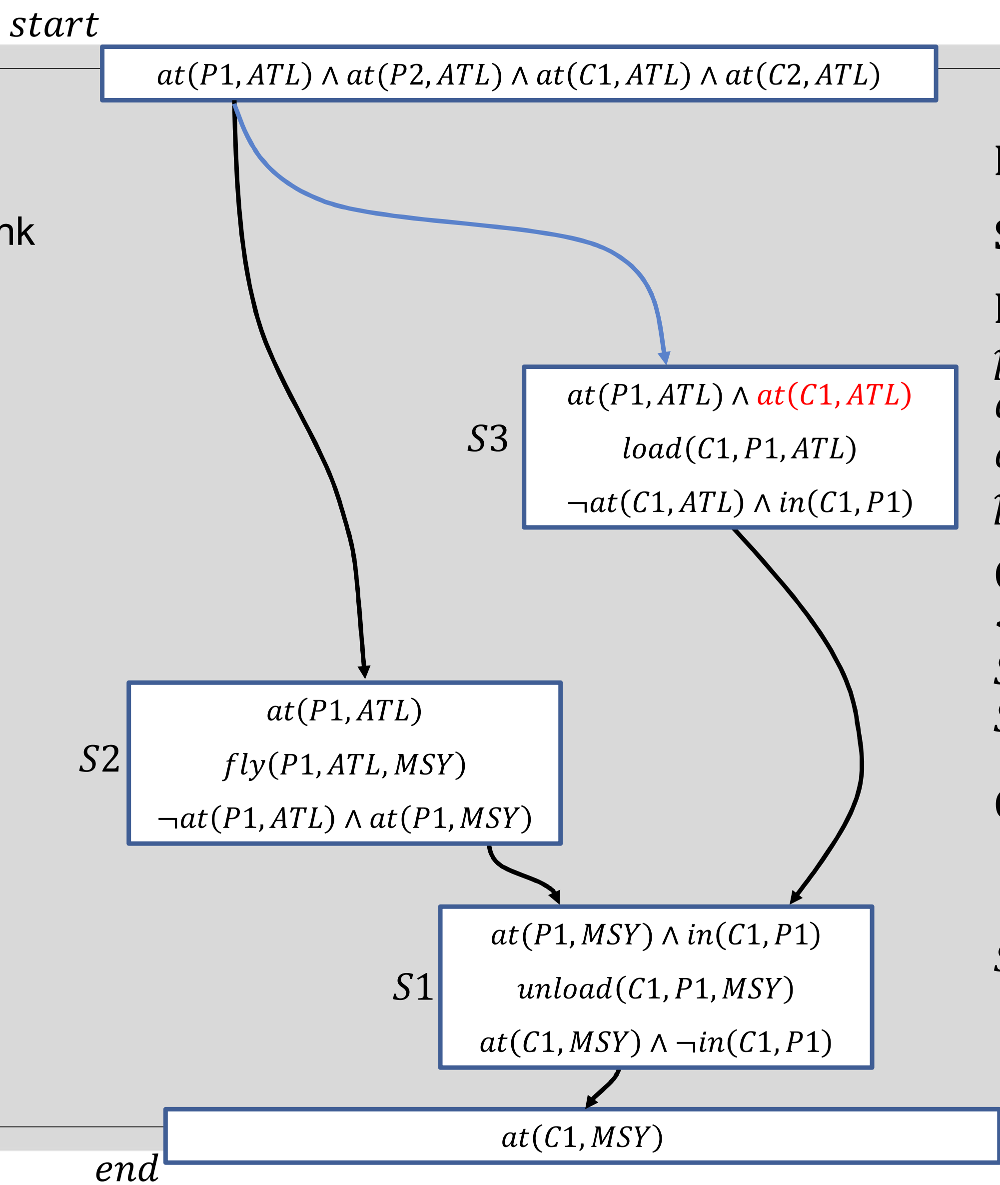
Steps: $start, end, S1, S2, S3$

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3,$ $c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4, a_4 = ATL$

Orderings: $start < end, start < S1, S1 < end, start < S2, S2 < S1, S2 < end, start < S3, S3 < S1, S3 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end,$ $S2 \xrightarrow{at(P1,MSY)} S1, start \xrightarrow{at(P1,ATL)} S2,$ $S3 \xrightarrow{in(C1,P1)} S1$

We add the causal link



Flaw: $at(P1, a_4)$ open

Steps: $start$, end, S1, S2, S3

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3, c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4, a_4 = ATL$

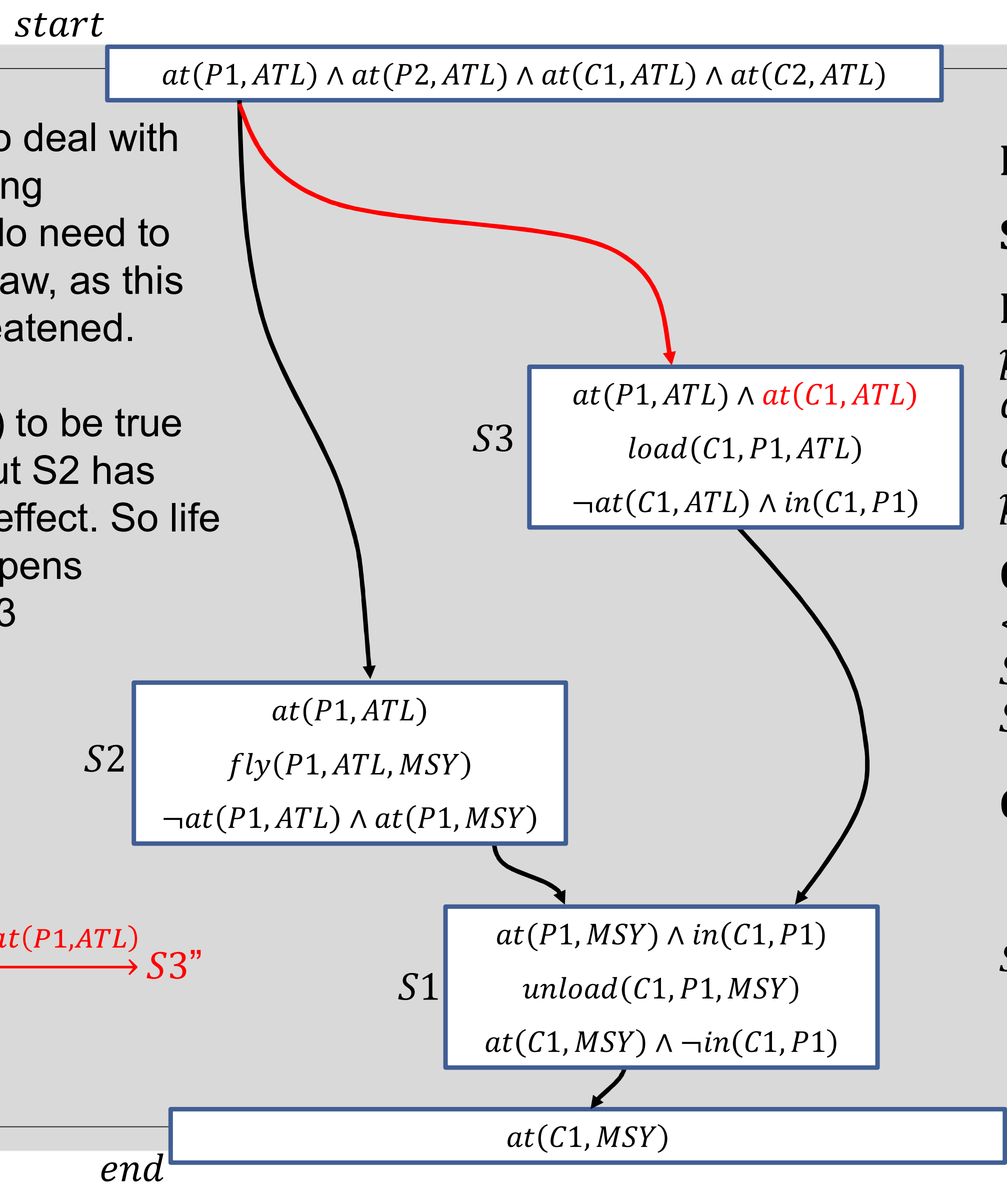
Orderings: $start < end$, $start$ < S1, S1 < end, start < S2, S2 < S1, S2 < end, start < S3, S3 < S1, S3 < end

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$, $start \xrightarrow{at(P1,ATL)} S2$, $S3 \xrightarrow{in(C1,P1)} S1$, $start \xrightarrow{at(P1,ATL)} S3$

And we don't have to deal with any additional ordering constraints, but we do need to add a "threatened" flaw, as this link just made is threatened.

S3 wants at(P1,ATL) to be true as a precondition, but S2 has $\neg at(P1, ATL)$ as an effect. So life will be bad if S2 happens between start and S3

"S2 threatens $start \xrightarrow{at(P1,ATL)} S3$"



Flaw: $at(P1, a_4)$ open

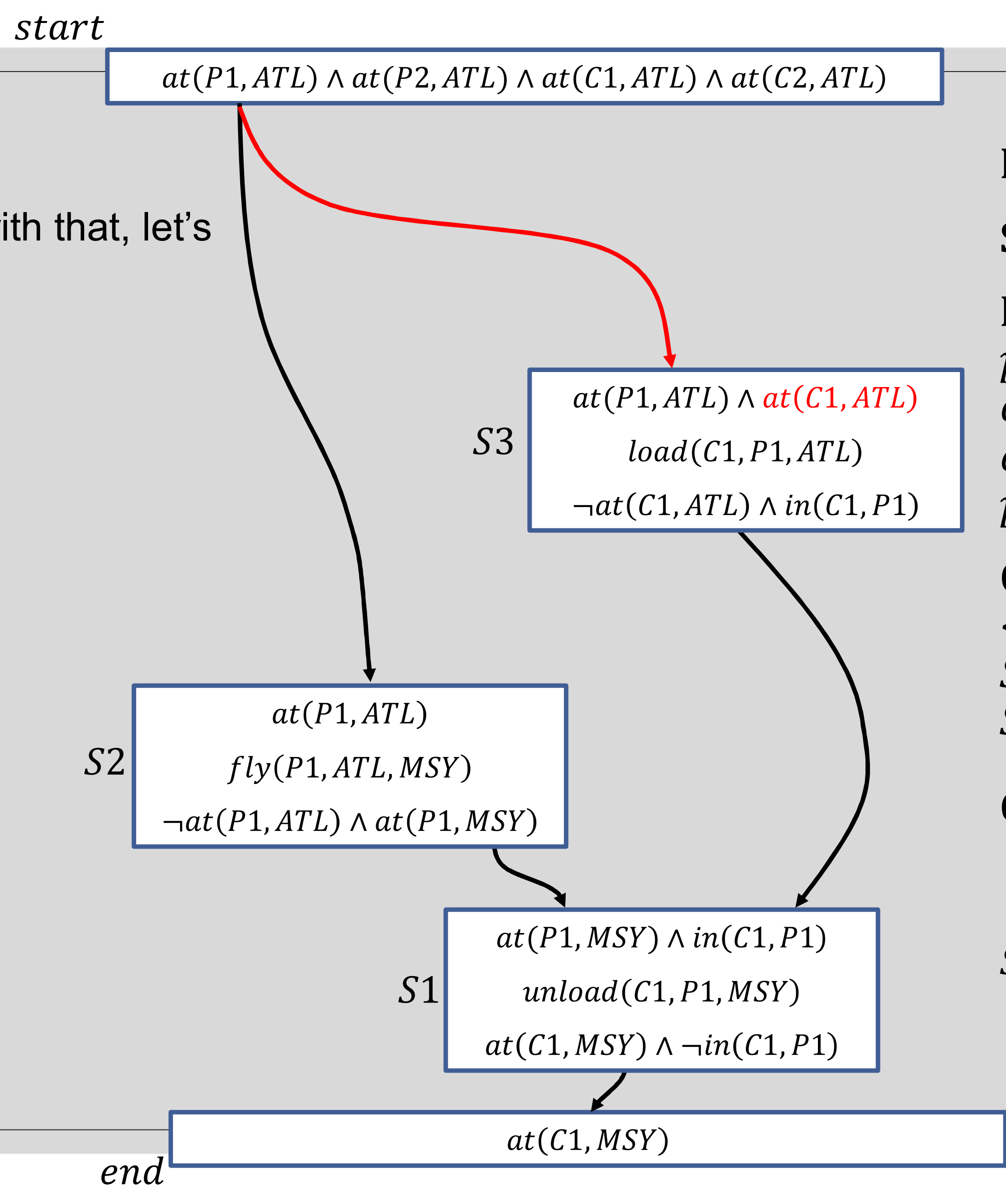
Steps: $start, end, S1, S2, S3$

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3, c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4, a_4 = ATL$

Orderings: *start < end, start < S1, S1 < end, start < S2, S2 < S1, S2 < end, start < S3, S3 < S1, S3 < end*

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$, $start \xrightarrow{at(P1,ATL)} S2$, $S3 \xrightarrow{in(C1,P1)} S1$, $start \xrightarrow{at(P1,ATL)} S3$

But before we deal with that, let's address $at(C1, ATL)$



Flaw: $at(C1, ATL)$open

Steps: $start$, end, S1, S2, S3

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3, c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4, a_4 = ATL$

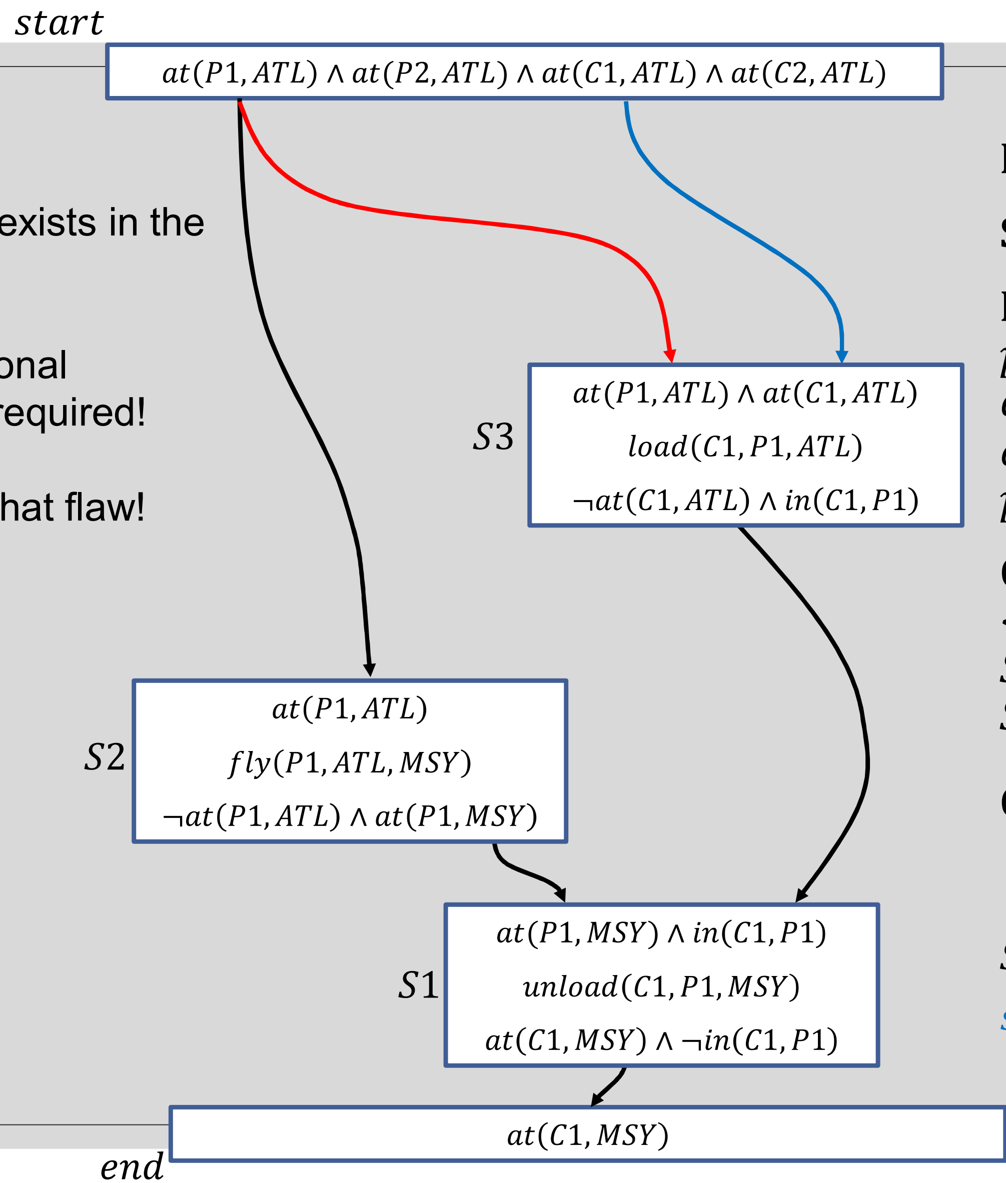
Orderings: $start < end$, $start$ < *S1, S1* < *end, start* < *S2, S2* < *S1, S2* < *end, start* < *S3, S3* < *S1, S3* < *end*

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$, $start \xrightarrow{at(P1,ATL)} S2$, $S3 \xrightarrow{in(C1,P1)} S1$, $start \xrightarrow{at(P1,ATL)} S3$

Happily, this already exists in the start step!

No bindings or additional ordering constraints required!

So, we're done with that flaw!



Flaw: $at(C1, ATL)$open

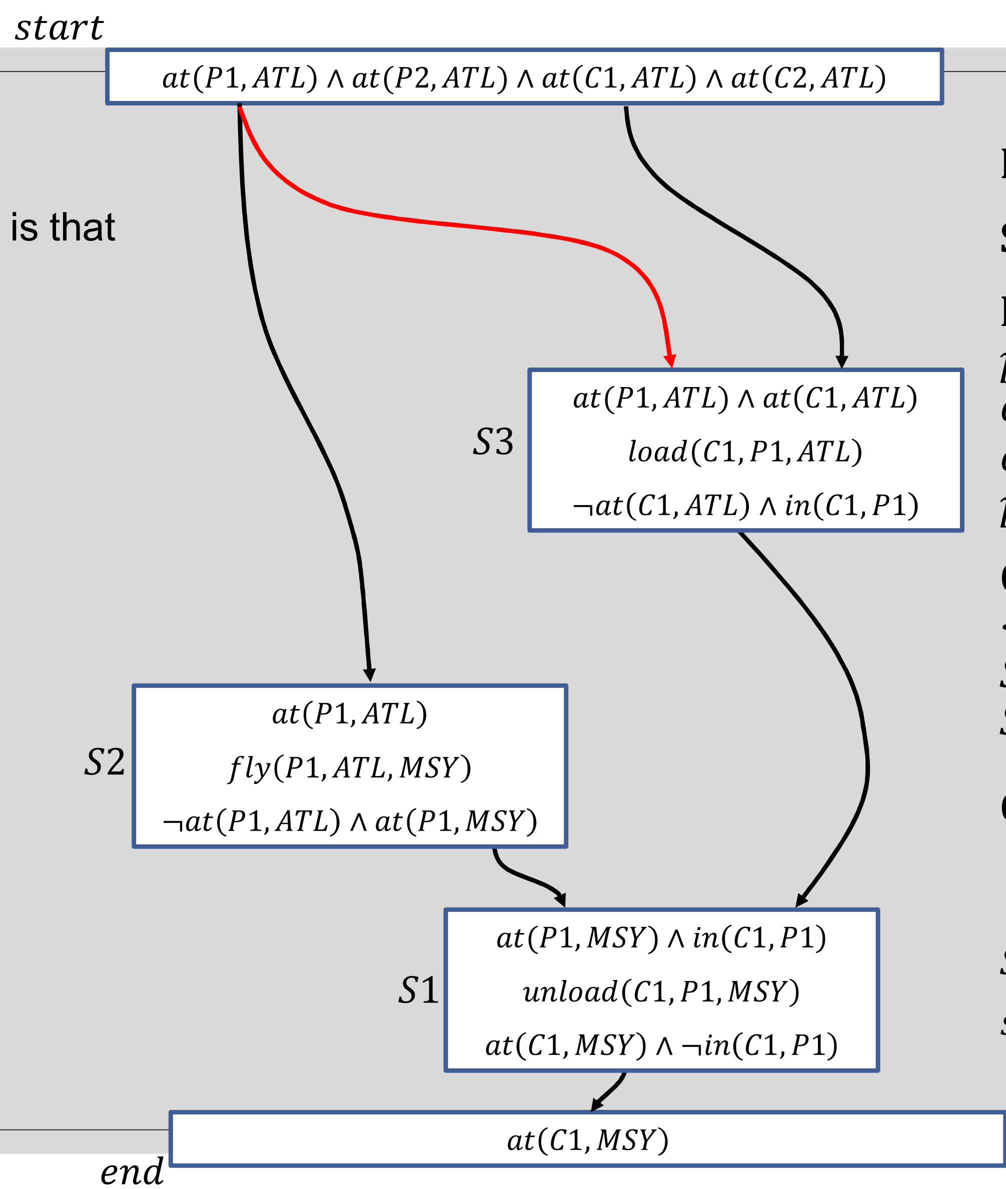
Steps: $start$, end, S1, S2, S3

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3,$ $c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4, a_4 = ATL$

Orderings: $start < end$, $start < S1$, $S1 < end$, $start < S2$, $S2 < S1$, $S2 < end$, $start < S3$, $S3 < S1$, $S3 < end$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$, $start \xrightarrow{at(P1,ATL)} S2$, $S3 \xrightarrow{in(C1,P1)} S1$, $start \xrightarrow{at(P1,ATL)} S3$, $start \xrightarrow{at(C1,ATL)} S3$

Now, all that remains is that threatens flaw!



Flaw: $S2$ threatens $start \rightarrow S3$

Steps: $start$, end, S1, S2, S3

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3, c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4, a_4 = ATL$

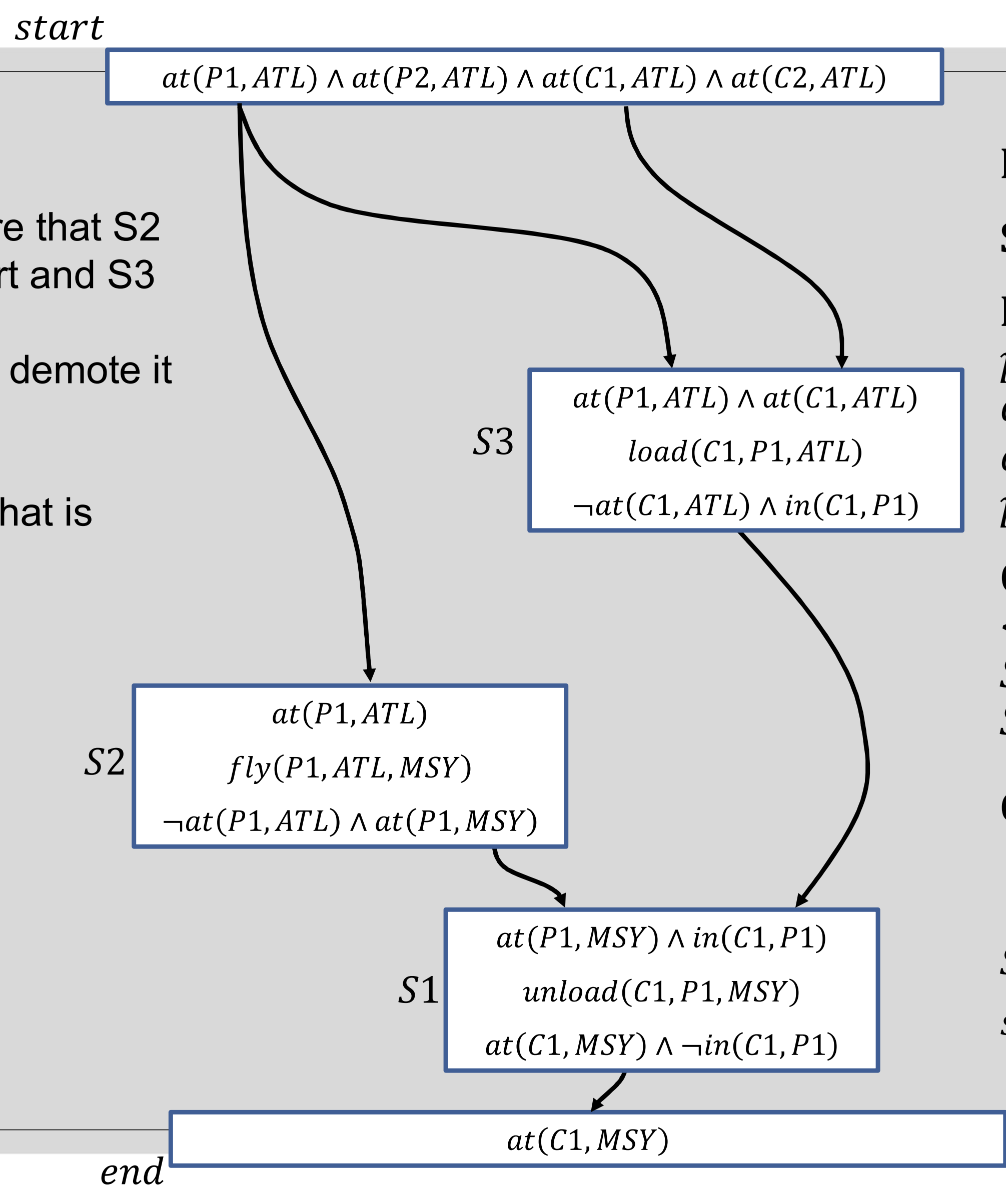
Orderings: $start < end$, *start < S1, S1 < end, start < S2, S2 < S1, S2 < end, start < S3, S3 < S1, S3 < end*

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$, $start \xrightarrow{at(P1,ATL)} S2$, $S3 \xrightarrow{in(C1,P1)} S1$, $start \xrightarrow{at(P1,ATL)} S3$, $start \xrightarrow{at(C1,ATL)} S3$

We want to make sure that S2 isn't "in between" start and S3

We can promote it or demote it to get it there.

Let's promote it!, so that is happens after S3



Flaw: $S2$ threatens $start \rightarrow S3$

Steps: $start, end, S1, S2, S3$

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3, c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4, a_4 = ATL$

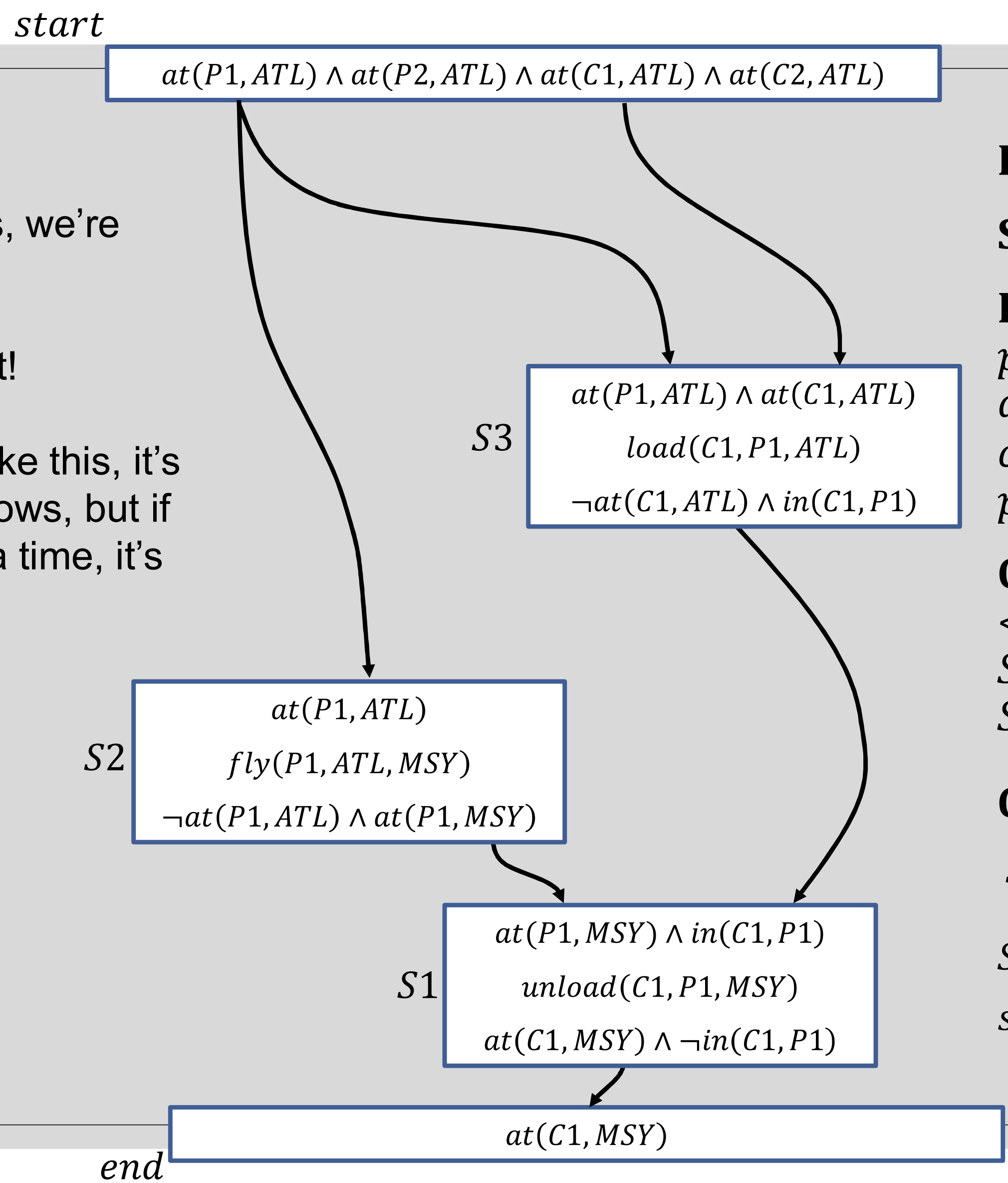
Orderings: $start < end$, $start < S1$, $S1 < end$, $start < S2$, $S2 < S1$, $S2 < end$, $start < S3$, $S3 < S1$, $S3 < end$, $S3 < S2$

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$, $start \xrightarrow{at(P1,ATL)} S2$, $S3 \xrightarrow{in(C1,P1)} S1$, $start \xrightarrow{at(P1,ATL)} S3$, $start \xrightarrow{at(C1,ATL)} S3$

And oh my goodness, we're done!

We have no flaws left!

When you look at it like this, it's a ton of lines and arrows, but if you take it a step at a time, it's not so bad, right?



Flaw: ***None*!**

Steps: $start, end, S1, S2, S3$

Bindings: $c_1 = C1, a_1 = MSY, p_1 = P1, p_2 = P1, a_2 = ATL$ $a_3 = MSY, p_1 = p_2, a_1 = a_3, c_2 = C1, p_3 = P1, c_1 = c_2, p_1 = p_3, a_1 = a_4, a_4 = ATL$

Orderings: $start < end$, *start < S1, S1 < end, start < S2, S2 < S1, S2 < end, start < S3, S3 < S1, S3 < end, S3 < S2*

Causal Links: $S1 \xrightarrow{at(C1,MSY)} end$, $S2 \xrightarrow{at(P1,MSY)} S1$, $start \xrightarrow{at(P1,ATL)} S2$, $S3 \xrightarrow{in(C1,P1)} S1$, $start \xrightarrow{at(P1,ATL)} S3$, $start \xrightarrow{at(C1,ATL)} S3$

Full Disclosure

- In those previous slides, we still “got lucky” with choosing literals and steps to bind to.
- As with other searches that we’ve discussed, it is possible to hit “dead ends” (i.e., we create a partial plan with a flaw that can’t be repaired).
- Just as before, if that happens, we “back up” to an earlier decision point and take a different action.