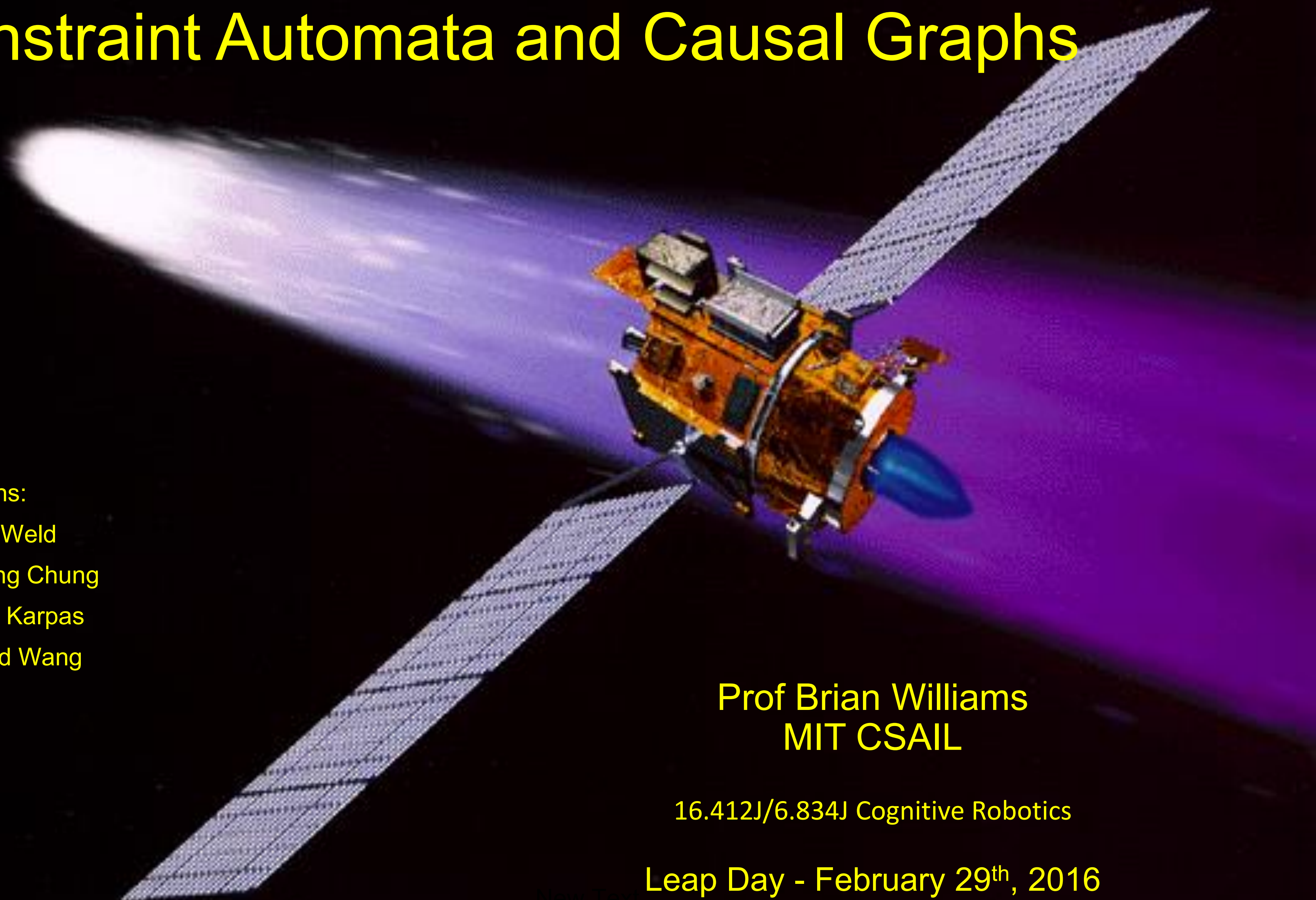
Goal Regression Planning, Constraint Automata and Causal Graphs



Contributions:

- Dan Weld
- Seung Chung
- Erez Karpas
- David Wang

Prof Brian Williams
MIT CSAIL

16.412J/6.834J Cognitive Robotics

Leap Day - February 29th, 2016

courtesy of JPL

This image is in the public domain.

After lecture you will know how to…

- Generate plans by working backwards from goals.
- Generate least commitment, partial order plans.
- Encode actions with indirect effects as concurrent automata with constraints (CCA).
- Analyze action dependence using causal graphs.
- Generate plans with out search,
  for CCA that have tree structured causal graphs.
- Use causal graph planning as a heuristic to HFS.

Assignments

Today:

- D. Weld, "An introduction to least commitment planning," AI Magazine 15(4):27-61, 1994.
- B. Williams and P. Nayak, "A reactive planner for a model-based executive," IJCAI, 1997.

Next:

- D. Wang and B. Williams, "tBurton: A Divide and Conquer Temporal Planner," AAAI, 2015.

Homework:

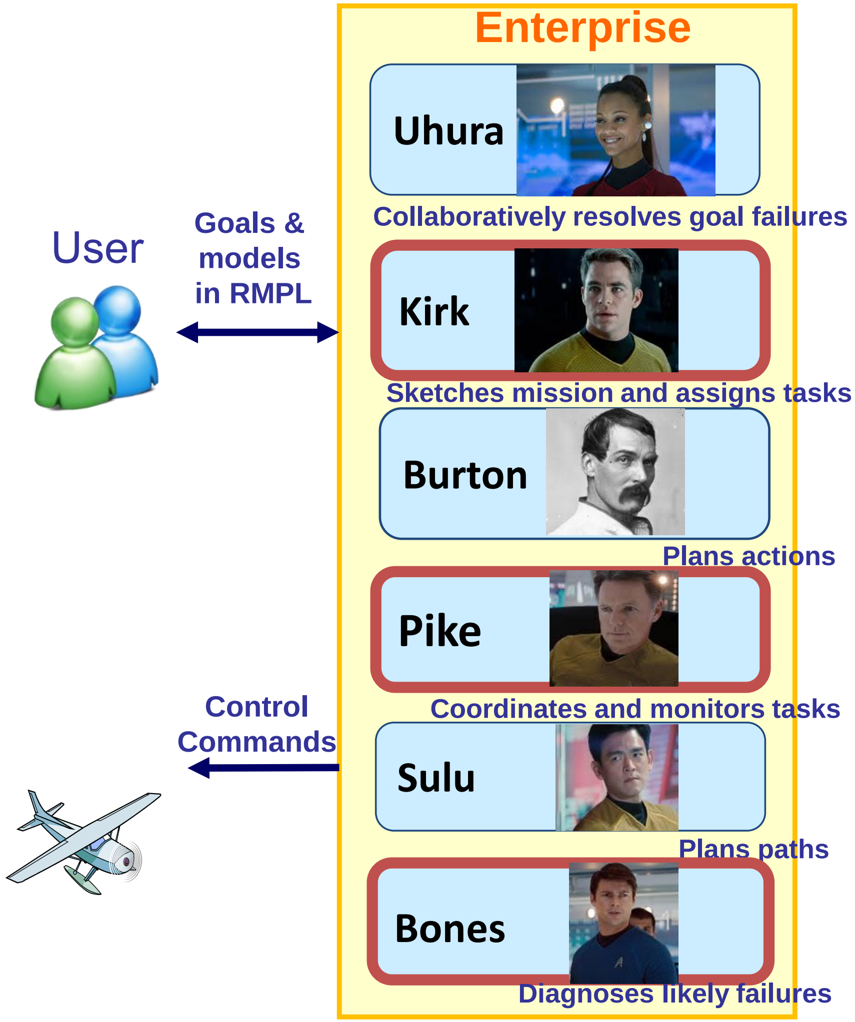
– PSet #3 PDDL Modeling, out today, due Wed, March 9th.

– Advanced Lecture Pset #1, out today, due Fri, March 4th.

Outline

- Review: programs on state
- Planning as goal regression (SNLP)
- Goal regression planning with causal graphs (Burton)
- Appendix: HFS planning with the causal graph heuristic (Fast Downward)

A single "cognitive system" language and executive.

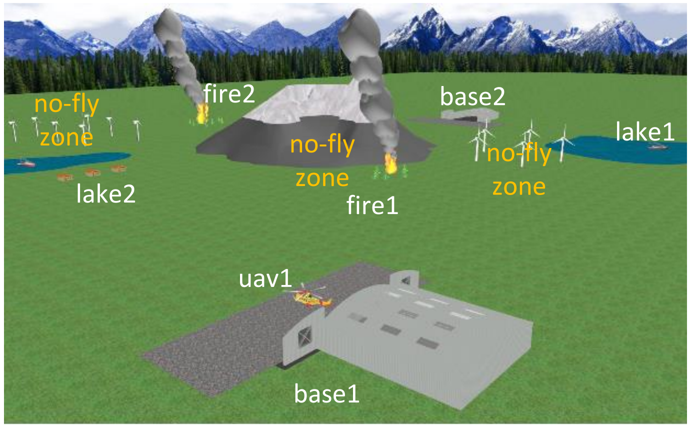


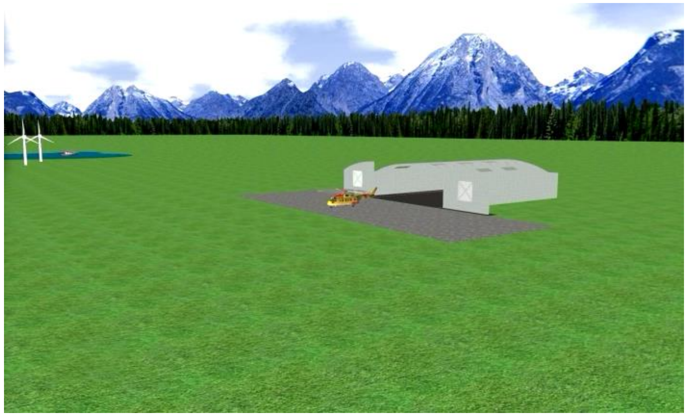
Programs that:

- Make choices
- Adjust timing
- Monitor conditions

© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/.

In a State-Space Program,
Activity Planning Maps Desired States to Actions



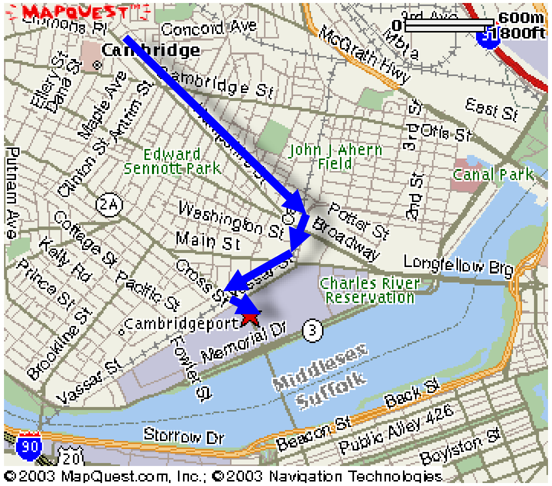


```
class UAV {
  Roadmap location;
  Boolean flying;
  Boolean loaded;

  primitive method takeoff()
    flying == no => flying == yes;

  primitive method land()
    flying == yes => flying == no;

  primitive method load_water(Lake lakespot)
    ((flying == yes) && (loaded == no)
     && (lakespot.location == location)) => loaded == yes;

  primitive method drop_water_high_altiture(Fire firespot)
    ((flying == yes) && (loaded == yes)
     && (firespot.location == location) && (firespot == high))
     => ((loaded == no) && (firespot == medium));

  primitive method drop_water_low_altiture(Fire firespot)
    ((flying == yes) && (loaded == yes)
     && (firespot.location == location) && (firespot == medium))
     => ((loaded == no) && (firespot == out));

  #MOTION_PRIMITIVES(location, fly, flying==yes)
}
```

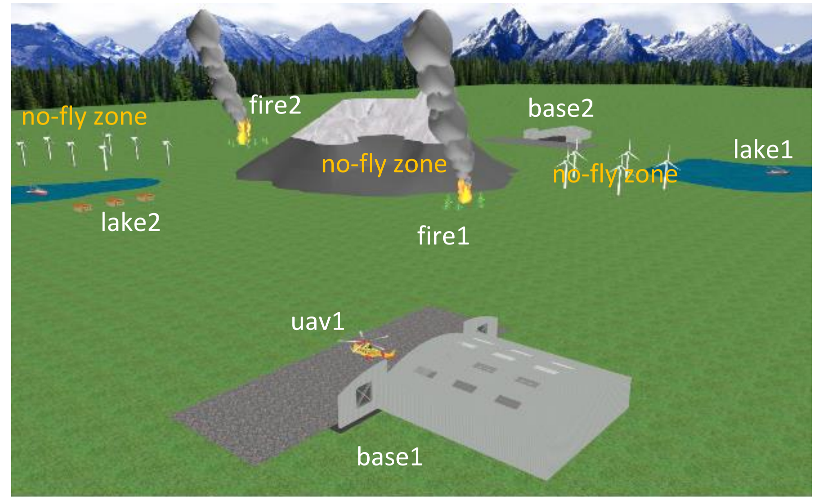
```
class Main{
  UAV uav1;
  Lake lake1;
  Lake lake2;
  Fire fire1;
  Fire fire2;
    ...

 method run() {
   sequence{
     (fire1 == out);
     (fire2 == out);
     (uav1.flying == no &&
      uav1.location == base_1_location);
 }}}
```

```
Main (){
    uav1 = new UAV();
    uav1.location= base_1_location;
    uav1.flying = no;
    uav1.loaded = no;

    lake1 = new Lake();
    lake1.location = lake_1_location;

    lake2 = new Lake();
    lake2.location = lake_2_location;

    fire1 = new Fire();
    fire1.location = fire_1_location;
    fire1 = high;

    fire2 = new Fire();
    fire2.location = fire_2_location;
    fire2 = high;
}
```

Planning maps desired states to actions

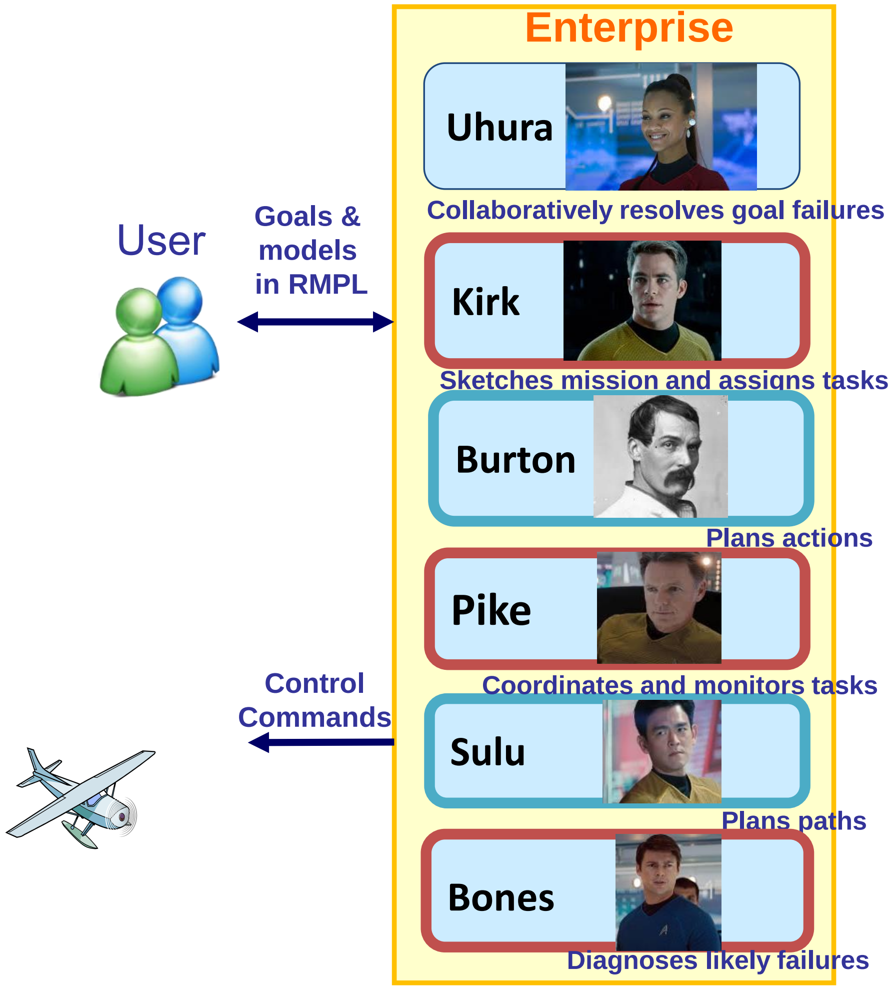


Roadmap Path Planning



“Classical” Action Planning

A single “cognitive system” language and executive.



State Space Programs that:

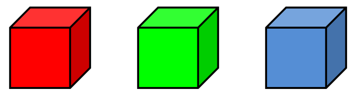
- Plan to achieve discrete states.
- Plan to achieve continuous states.
- Monitor continuous states.

© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/.

Last Week: “Classic” Activity Planning Representation (STRIPS/PDDL)

Action Model:

Objects (things):



Predicates (used to create true or false statements about the world, “facts”):

(On A, B)

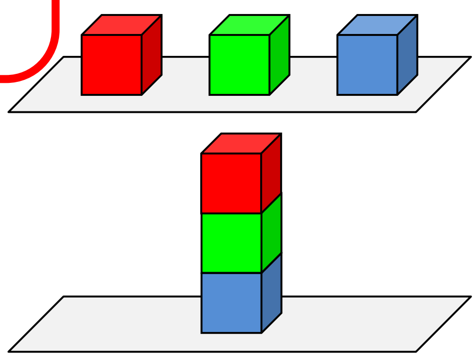
(Clear A)

Actions (used to change truth of predicates):

(Clear A)
(Clear B)
→ **Stack** →
¬(Clear B)
(On A, B)

preconditions

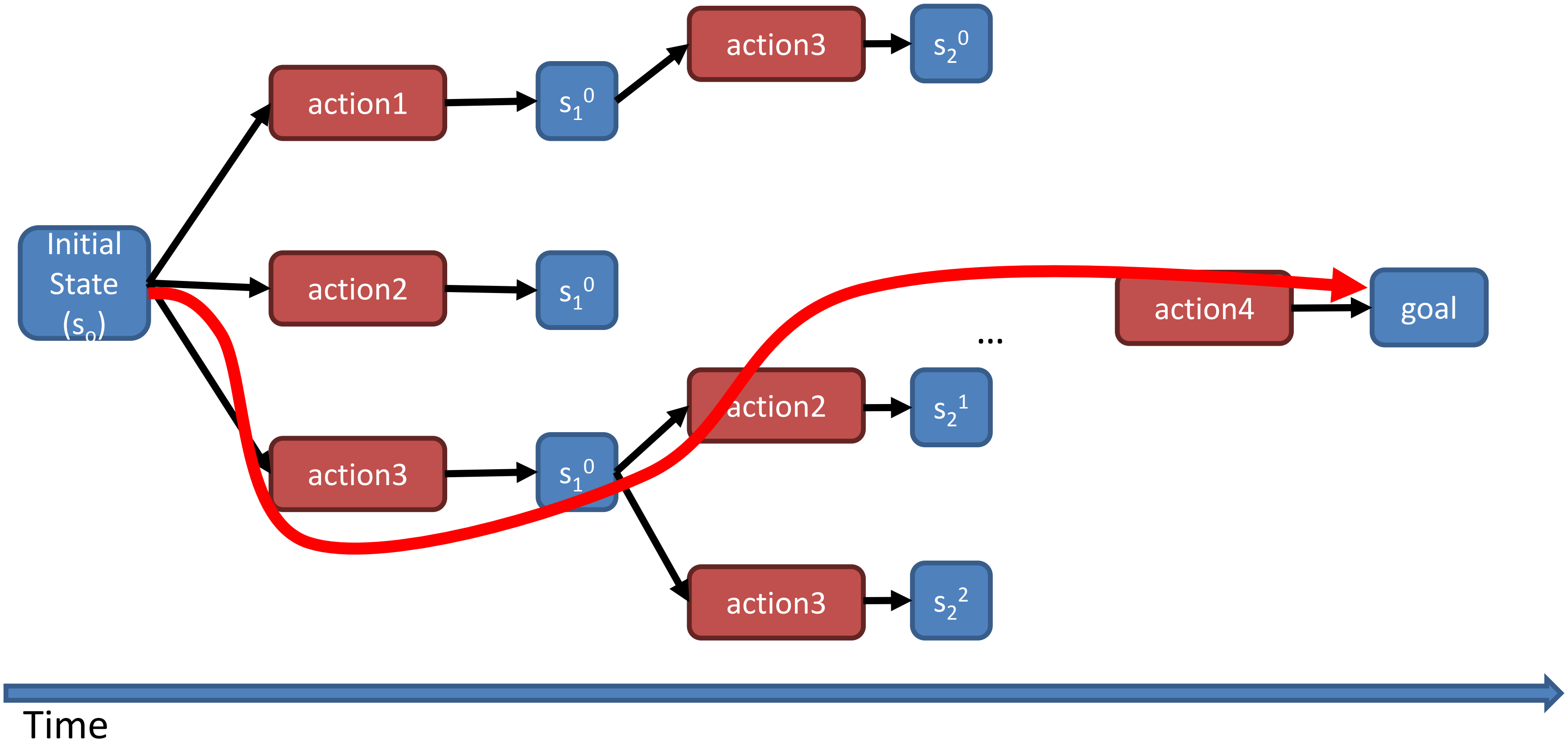
effects



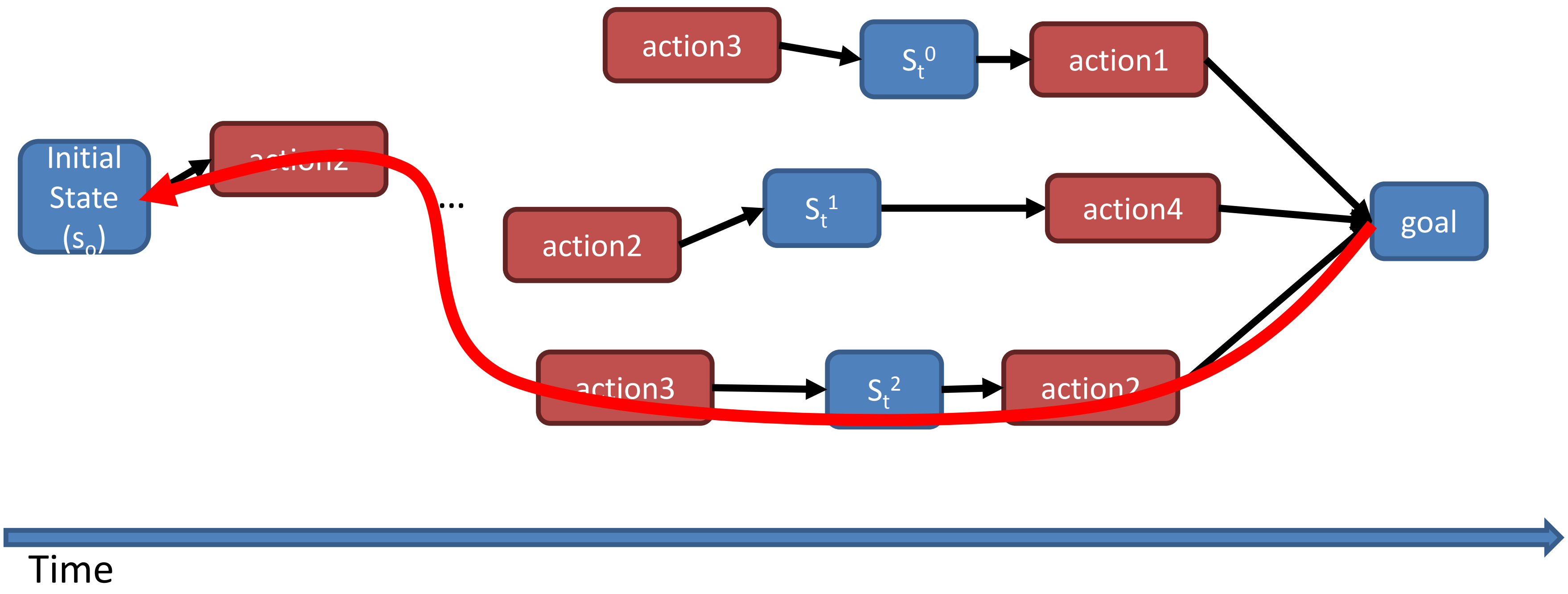
Initial: (Clear) (Clear) (Clear)

Goal: (On ,) (On ,)

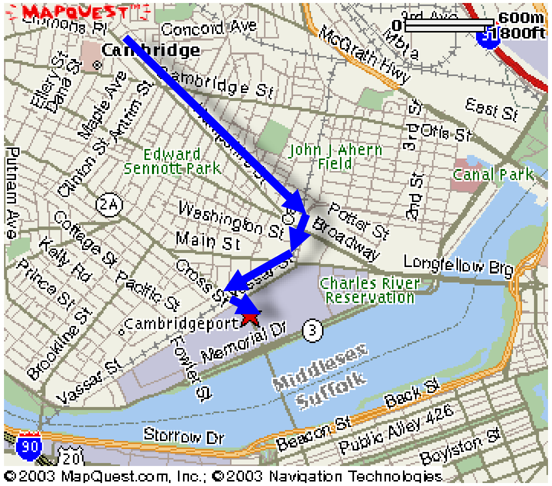
Last Week: Forward Search



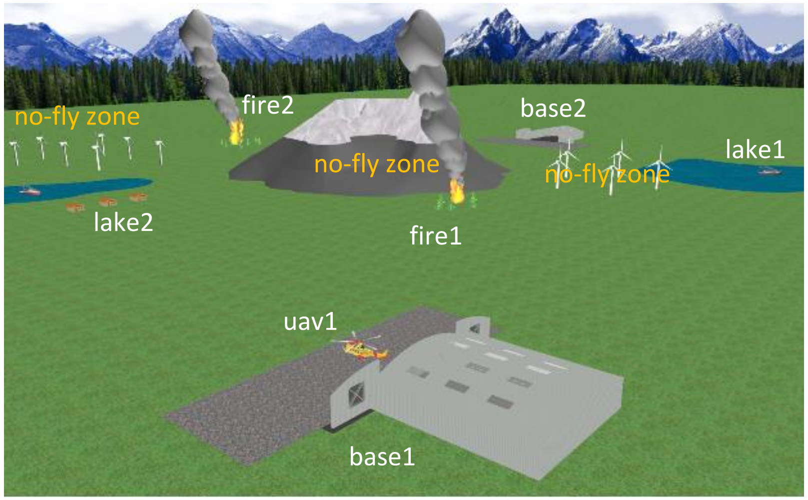
This Week: Goal-Regression Search



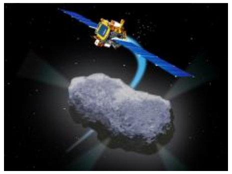
This Week: Planning to Control Complex Devices

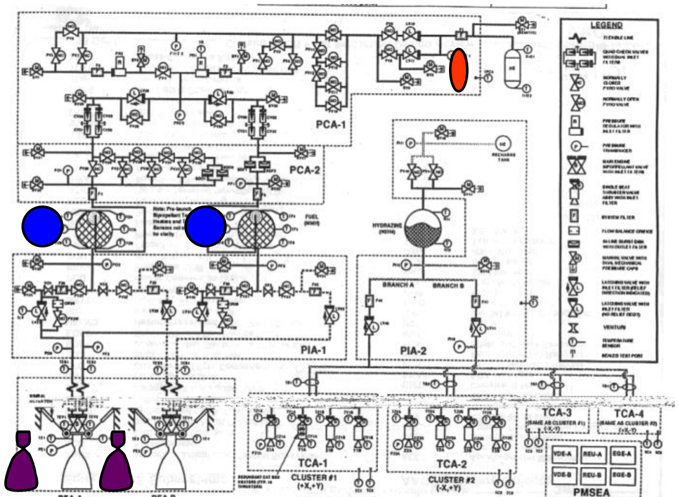


Roadmap Path Planning

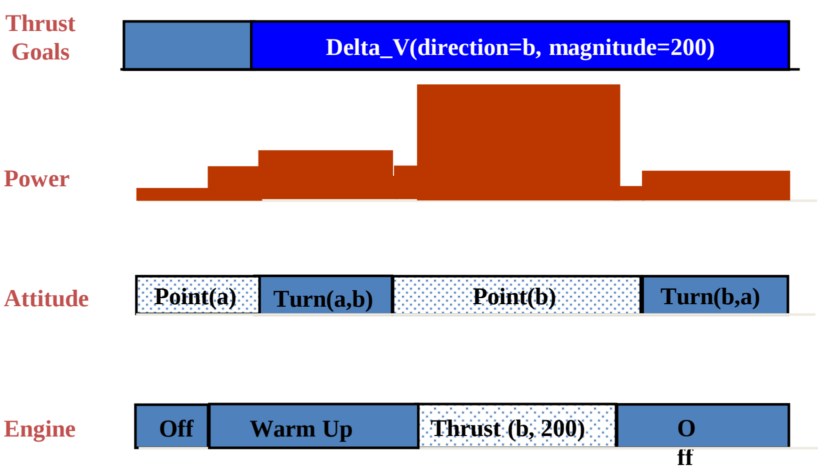


“Classical” Action Planning





Device Reconfiguration & Repair

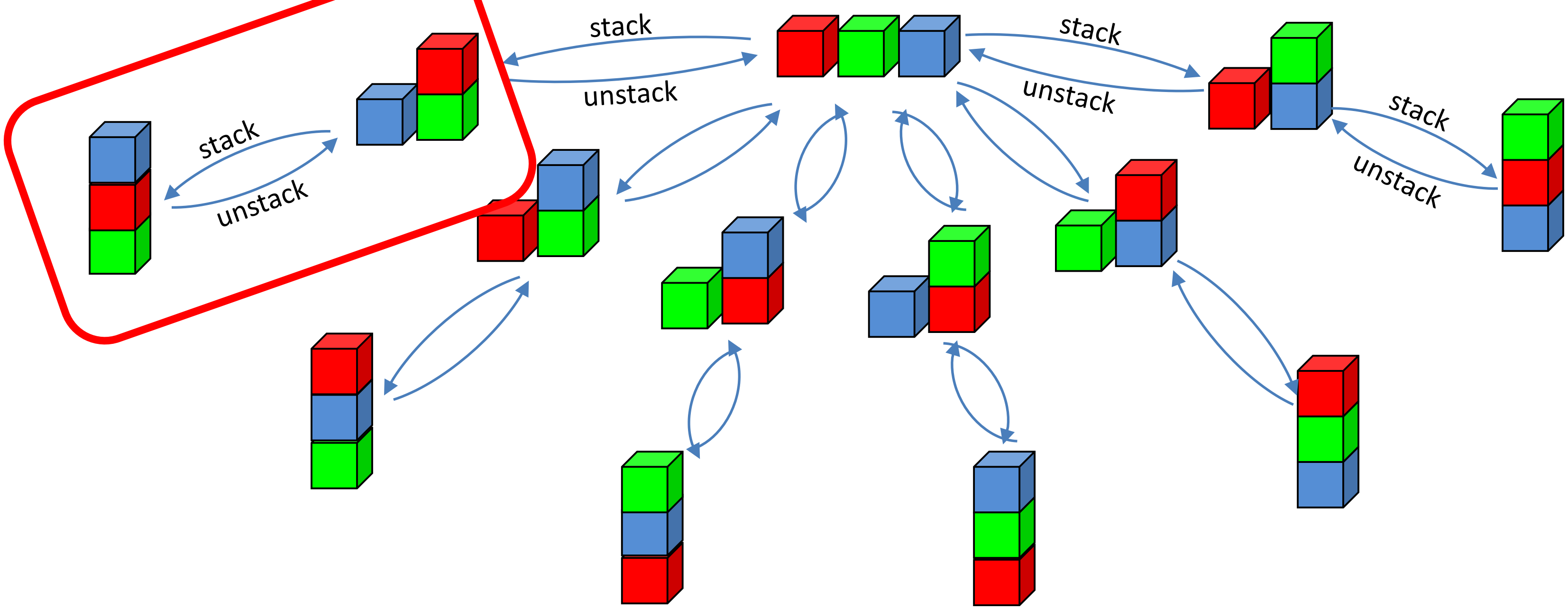


Time-line Planning

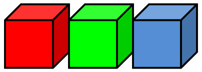
This Week: Automata Representation

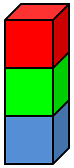
Action Model:

Concurrent Constraint Automata (CCA) (like a state-machine):



Note: This is a very simple example, there are usually many automata, and guards on the transitions.

Initial: 

Goal: 

We will use CCA to support indirect effects, concurrency and metric time .

Algorithms exist to map between the two representations.

Outline

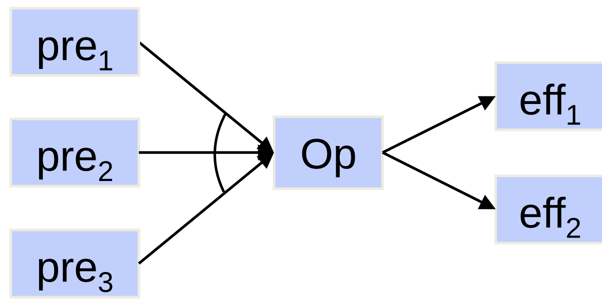
- Review: programs on state
- Planning as goal regression (SNLP/UCPOP)
  – Partial Order Planning Overview
  – Partial Order Planning Problem
  – Partial Order Plan Generation
- Goal regression planning
  with causal graphs (Burton)
- Appendix: HFS planning with the
  causal graph heuristic (Fast Downward)

Classical Planning Problem

Propositions: P_i

Initial Conditions: P_1 P_2 P_3 P_4

Operators:



Goals: $Goal_1$ $Goal_2$ $Goal_3$

Simple Spacecraft Problem

Observation-1
- **target**
- **instruments**

Observation-2

Observation-3

Observation-4

…

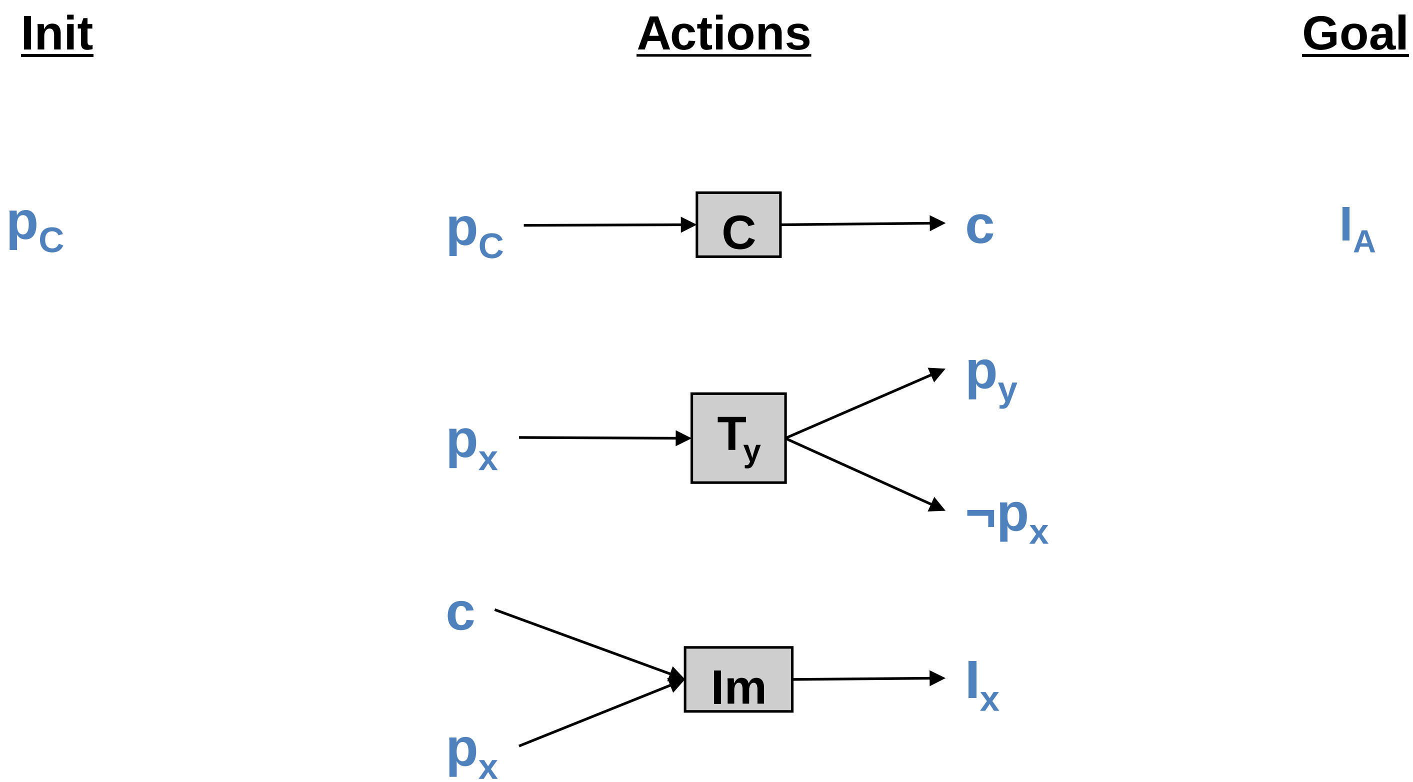
pointing

calibrated

This image is in the public domain.

Propositions: Target Pointed To, Camera Calibrated?, Has Image?
Operators: Calibrate, Turn to Y, and Take Image.

Example



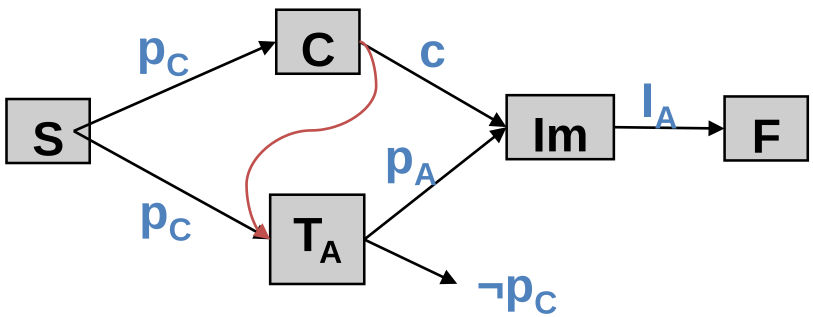
Propositions: Target Pointed To, Camera Calibrated?, Has Image?
Operators: Calibrate, Turn to Y, and Take Image.

Actions in the Planning Domain Description Language (PDDL)

```
(:action TakeImage :parameters (?target, ?instr)
        :precondition (and (Status ?instr Calibrated)
                           (Pointing ?target))
        :effect            (Image ?target))

(:action Calibrate :parameters (?instrument)
        :precondition (and (Status ?instr On)
                           (Calibration-Target ?target),
                           (Pointing ?target)
        :effect            (and (not (Status ?inst On))
                           (Status ?instr Calibrated)))

(:action Turn :parameters (?target)
        :precondition (and (Pointing ?direction)
                           ?direction ≠ ?target
        :effect            (and (not (Pointing ?direction)
                           (Pointing ?target)))
```

By convention, parameters start with "?", as in ?var.

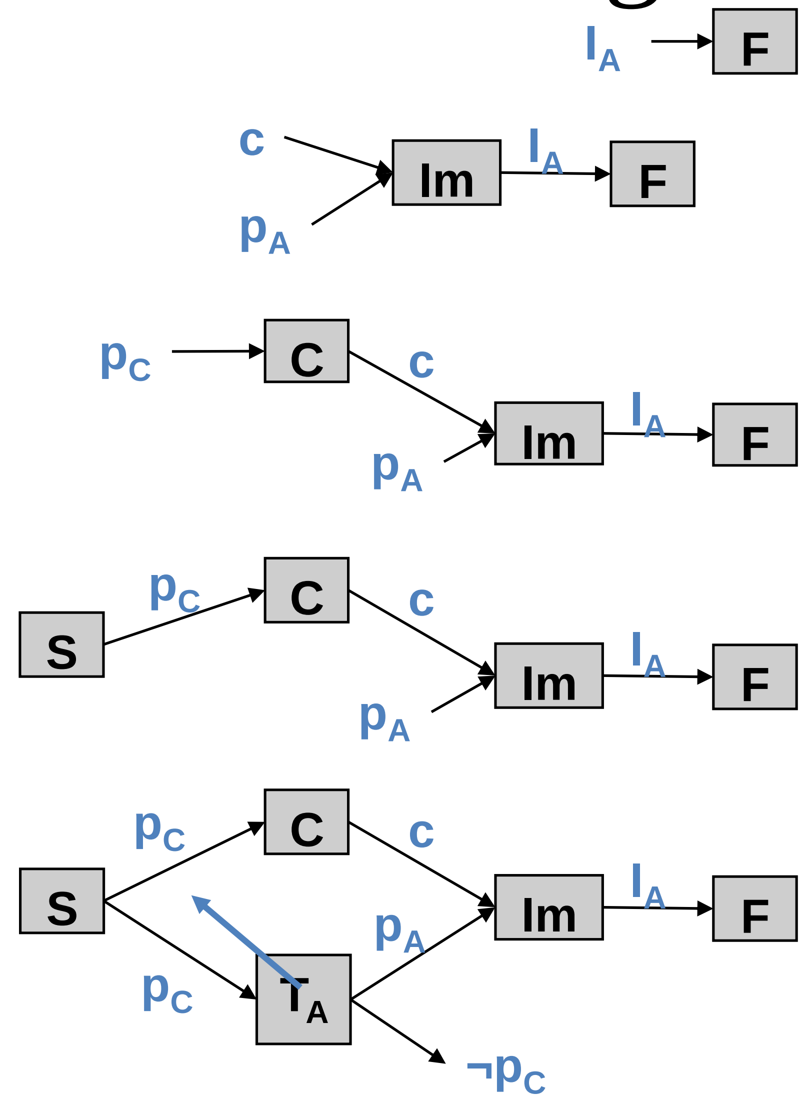
Partial Order Plan

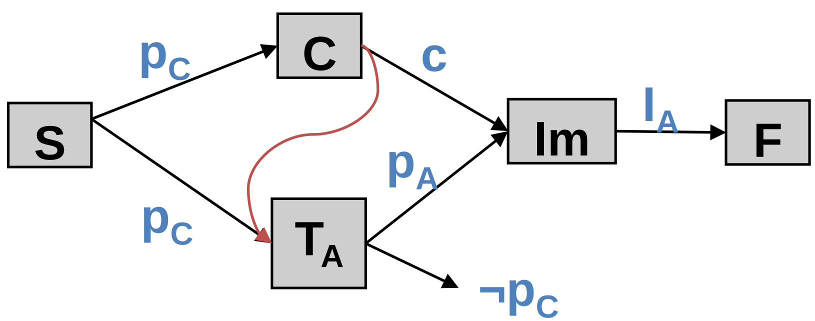


Planning from Goals: Partial Order Causal Link Planning

(SNLP, UCPOP)

1. Select an open condition;
2. Choose an op that can achieve it:
   - Link to an existing instance or
   - Add a new instance;
3. Resolve threats.





Alternatives: Forward Search, CSP

Planning as Goal Regression

– Partial Planning Overview
– Partial Order Planning Problem
  • Problem Encoding
  • Partial Order Plans
  • Plan Correctness
– Partial Order Plan Generation

Example Problem

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

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

Operators:

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

Initial and Goal States Encoded as Operators

Start

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

Why encode as operators?

Don't need to introduce (partial) states as separate objects.

Keeps theory minimal.

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

Finish

Partial Order Plan <Actions,Orderings,Links>

Start

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

At(Home)

Go(HWS)

At(HWS) Sells(HWS,Drill)

Buy(Drill)

At(HWS)

Go(SM)

At(SM), Sells(SM,Milk)

Buy(Milk)

At(SM), Sells(SM,Ban.)

Buy(Ban.)

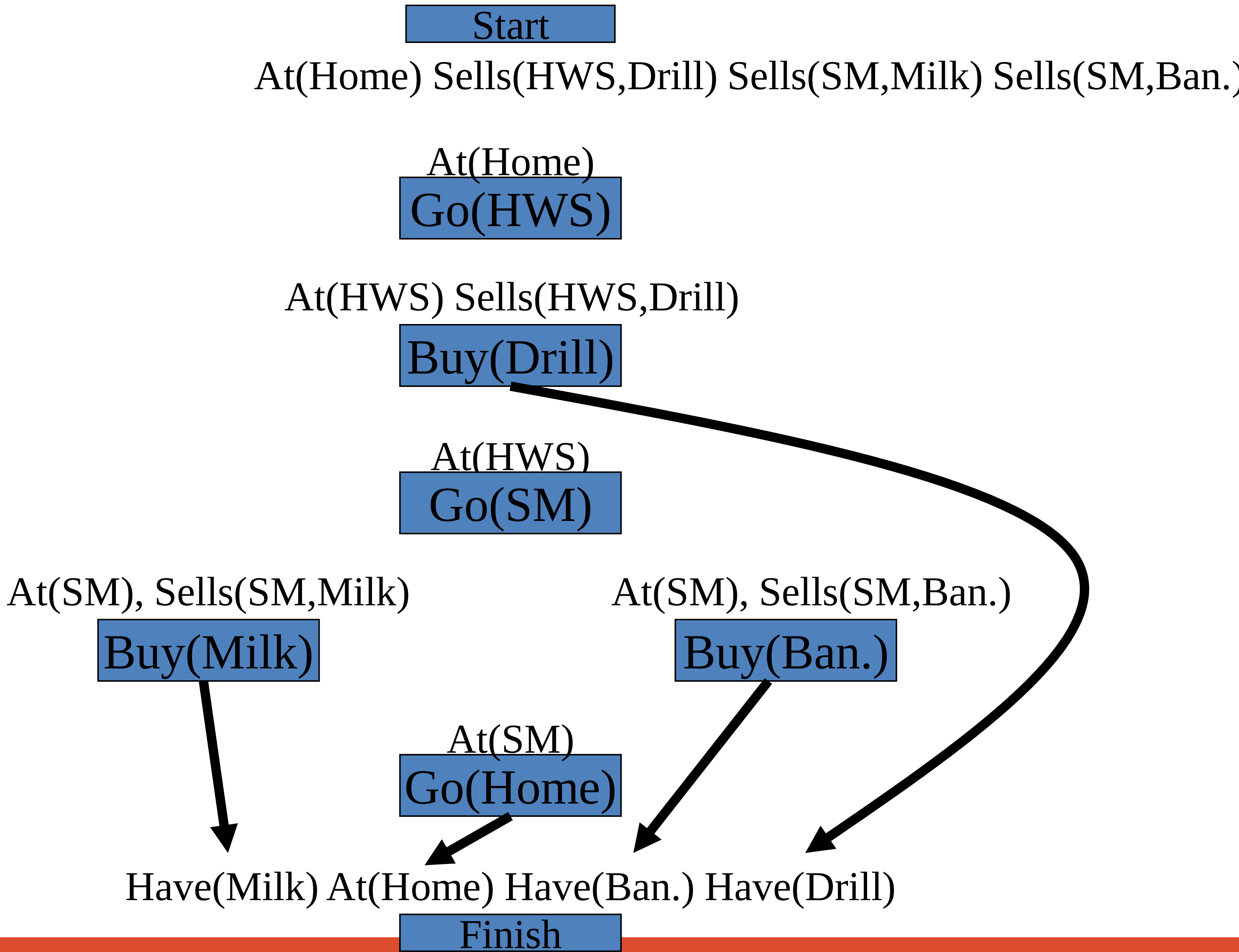
At(SM)

Go(Home)

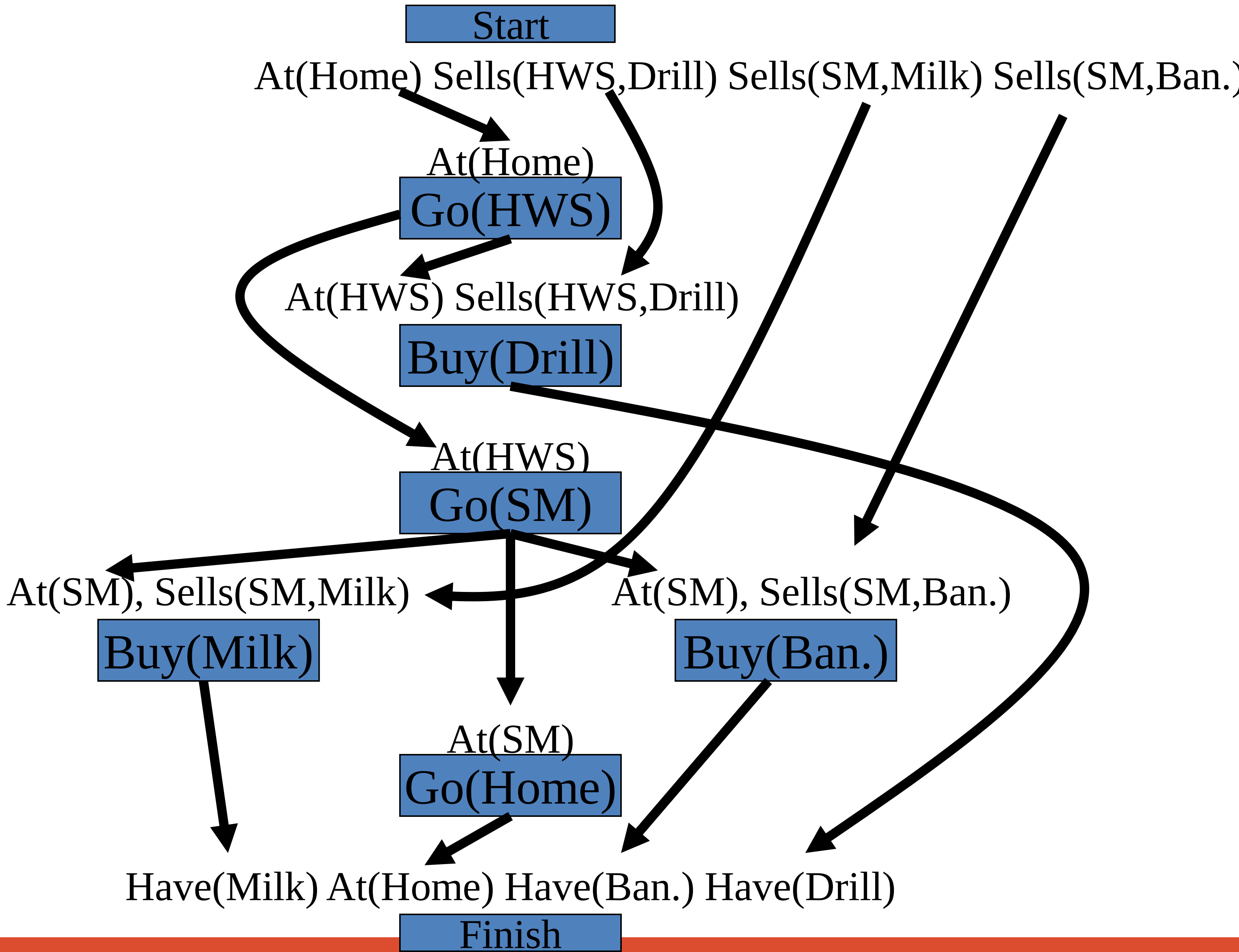
Have(Milk) At(Home) Have(Ban.) Have(Drill)

Finish

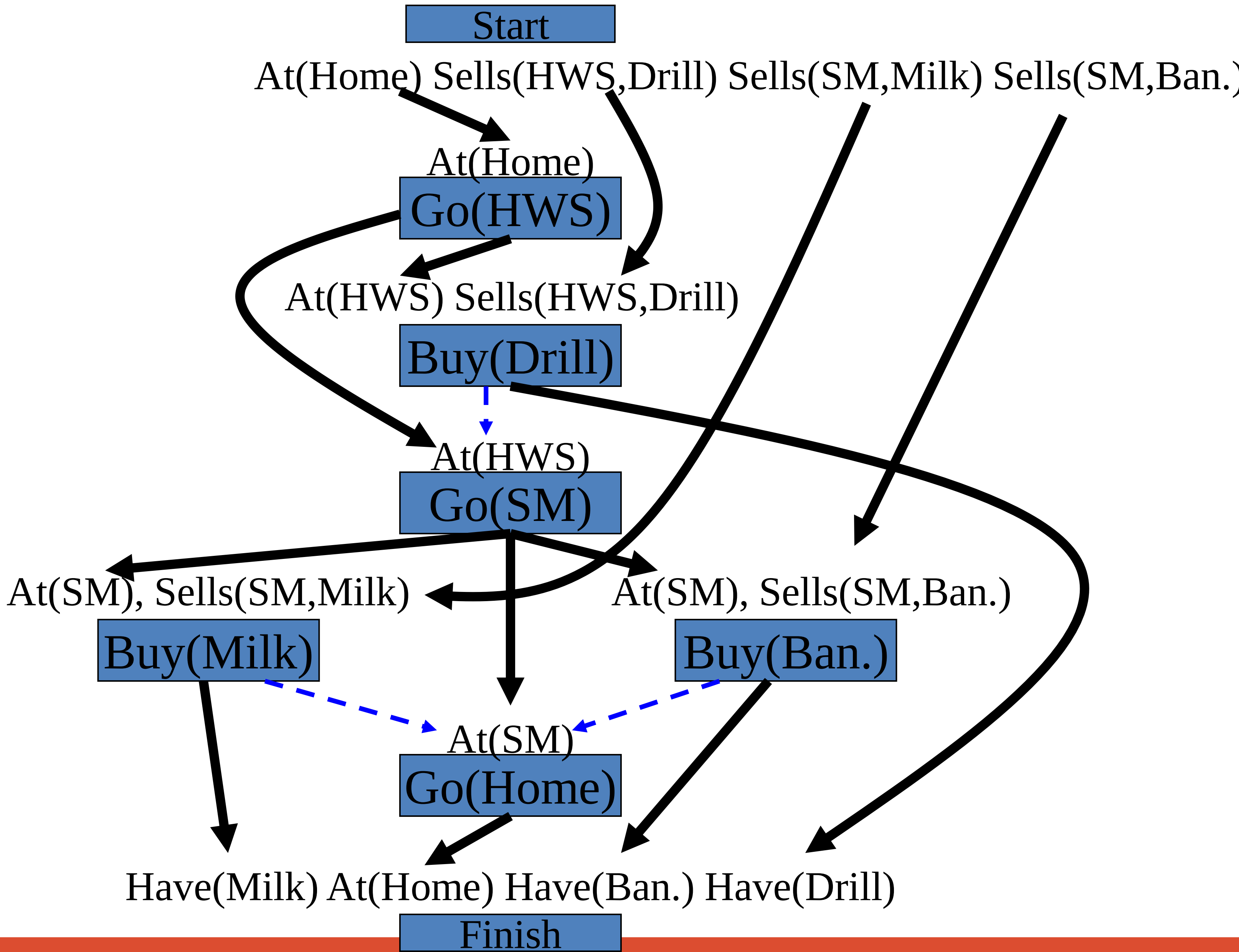
Partial Order Plan <Actions,Orderings,Links>



Partial Order Plan <Actions,Orderings,Links>



Partial Order Plan <Actions,Orderings,Links>

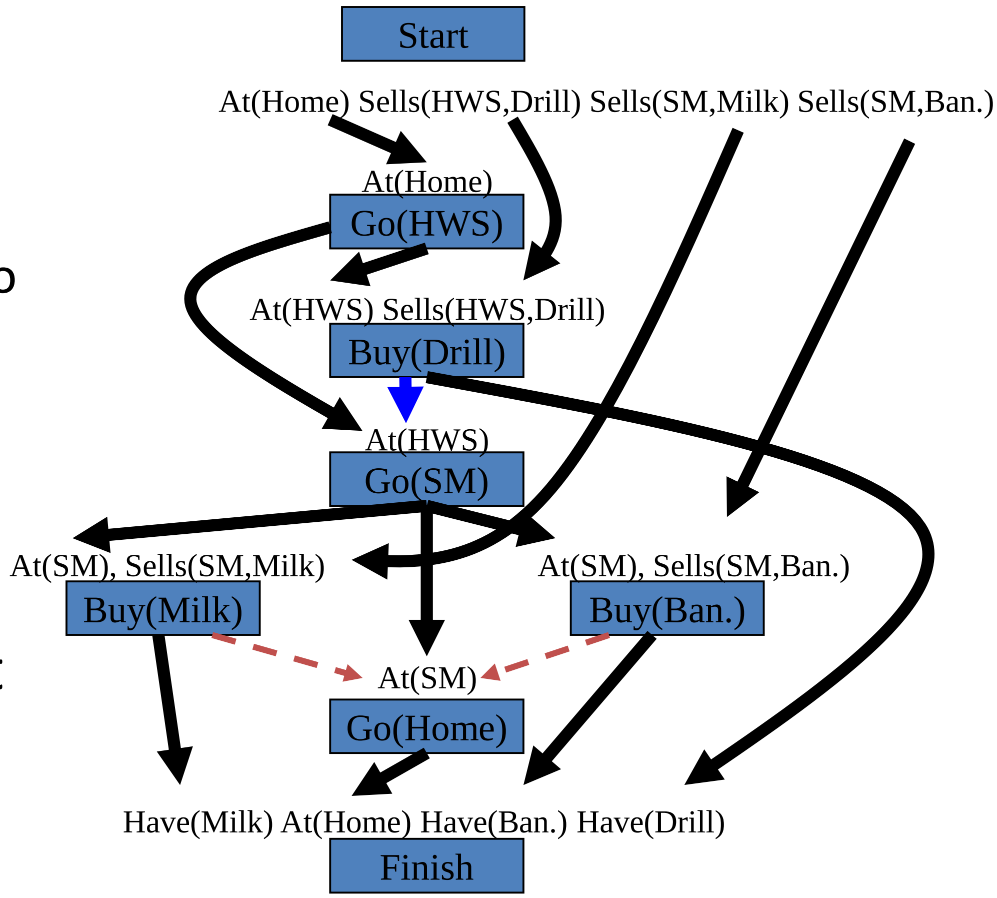


Planning as Goal Regression

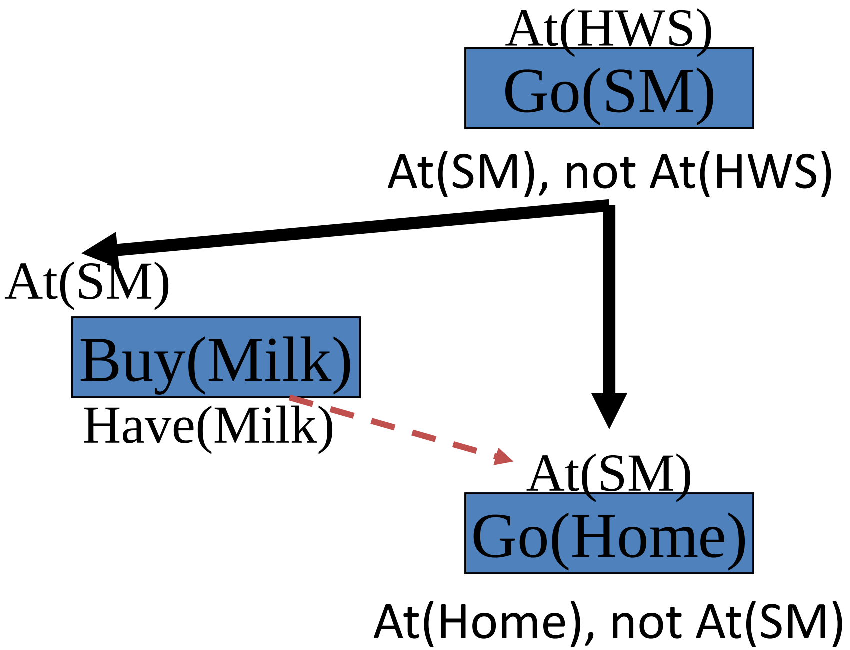
- Partial Planning Overview
- Partial Order Planning Problem
    - Problem Encoding
    - Partial Order Plans
    - Plan Correctness
- Partial Order Plan Generation

What Constitutes a Correct Plan?

- Complete Plan
  - Achieves all Goals
    - Achieves all preconditions …
    - No actions intervene to undo a needed precondition
- Consistent Plan
  - There exists an execution sequence that is consistent with the ordering



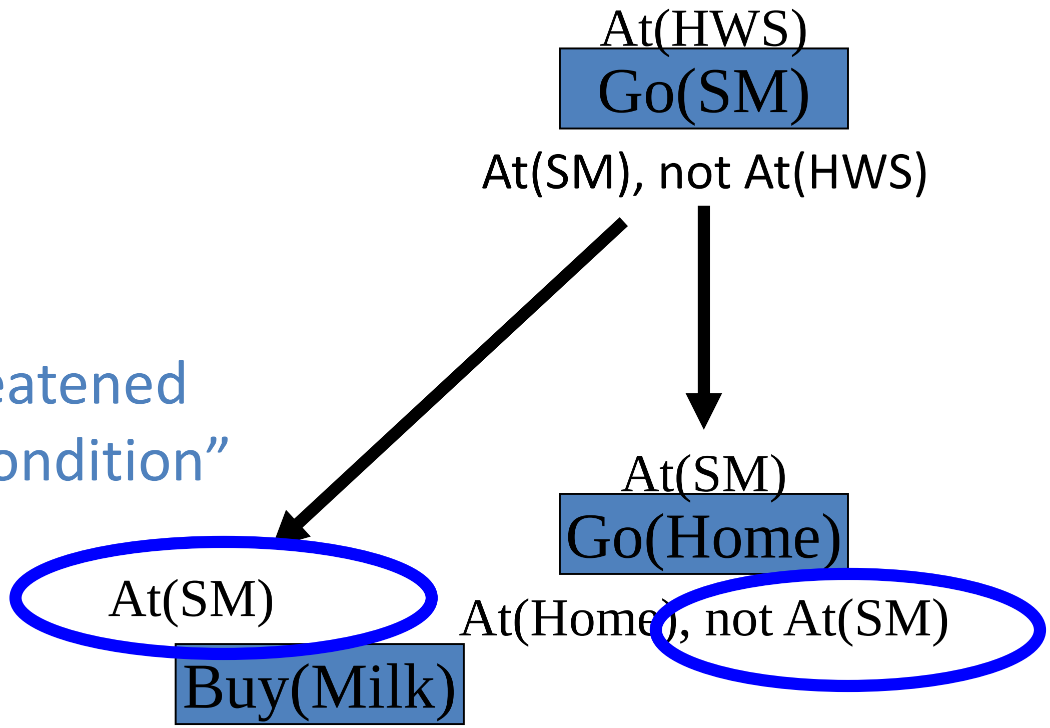
Why is an ordering needed?



Why is an ordering needed?

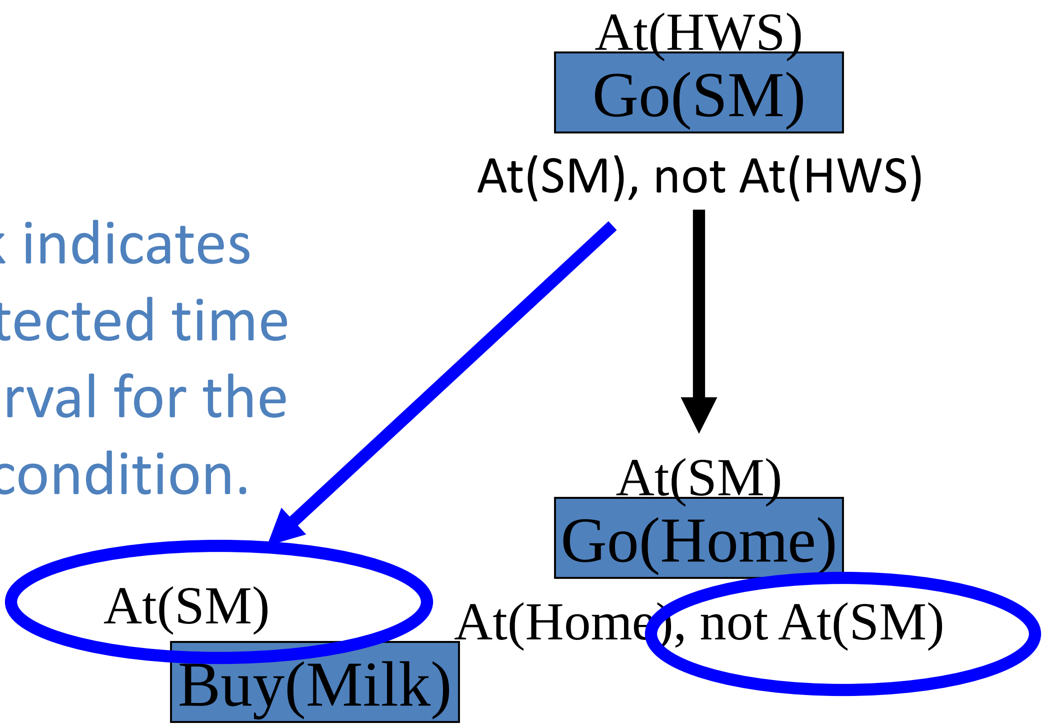
Suppose the other order is allowed, what happens?

“Threatened Precondition”



Why is an ordering needed?

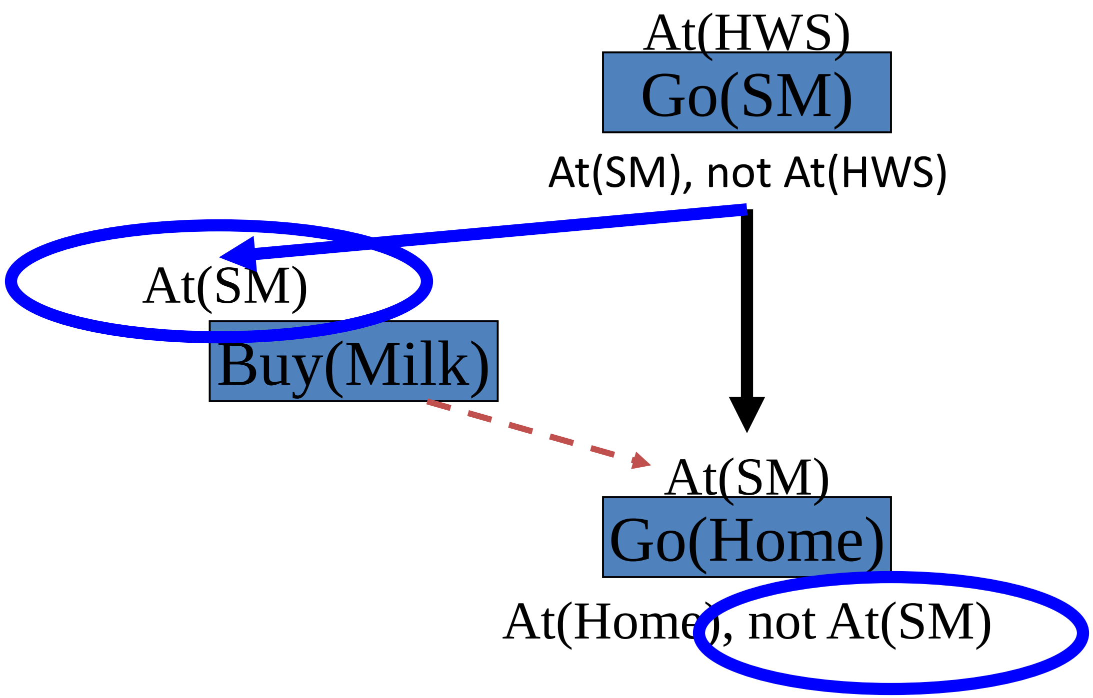
Suppose the other order is allowed, what happens?



Link indicates protected time interval for the precondition.

Why is an ordering needed?

The ordering resolves the threat.

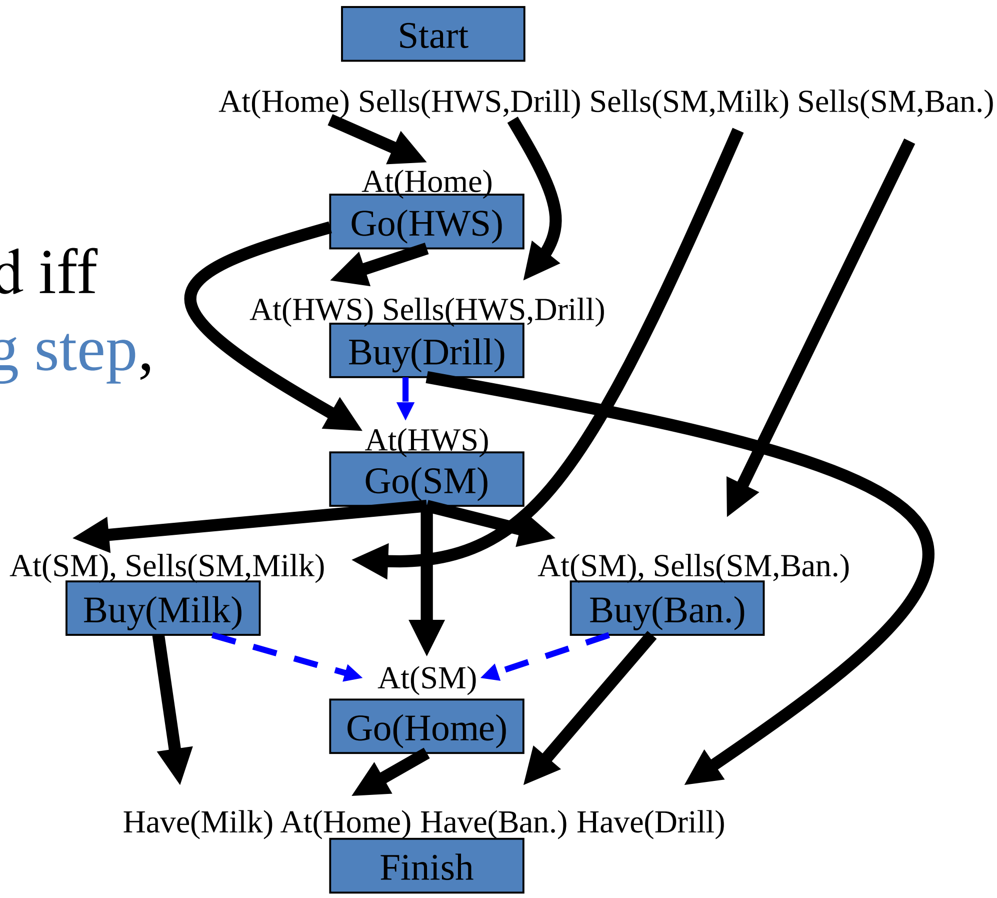


Solution: A Complete and Consistent Plan

- **Complete Plan**

  IFF every precondition of every step is achieved.

  A step's precondition is achieved iff
  - its the effect of some preceding step,
  - no intervening step can undo it.



- **Consistent Plan**

  IFF there is no contradiction in the ordering constraint.
  - i.e., never $s_i < s_j$ and $s_j < s_i$, or
  - the causal links + orderings are loop free.

Planning as Goal Regression

- Partial Planning Overview
- Partial Order Planning Problem
- Partial Order Plan Generation
  - Derivation from Completeness and Consistency
  - Backward Chaining
  - Threat Resolution
  - The POP algorithm

Partial Order Planning Algorithm

The algorithm falls out of Consistency and Completeness

Completeness:

- Must achieve all preconditions

  → Backward chain from goals to initial state,
  by inserting actions and causal links.

- Must avoid intervening actions that threaten

  → After each action is inserted,
  find any action that threatens its effects, and
  impose ordering to resolve.

Consistent:

- Ordering must be consistent

  → After each causal link and ordering is inserted,
  check for loops.

Planning as Goal Regression

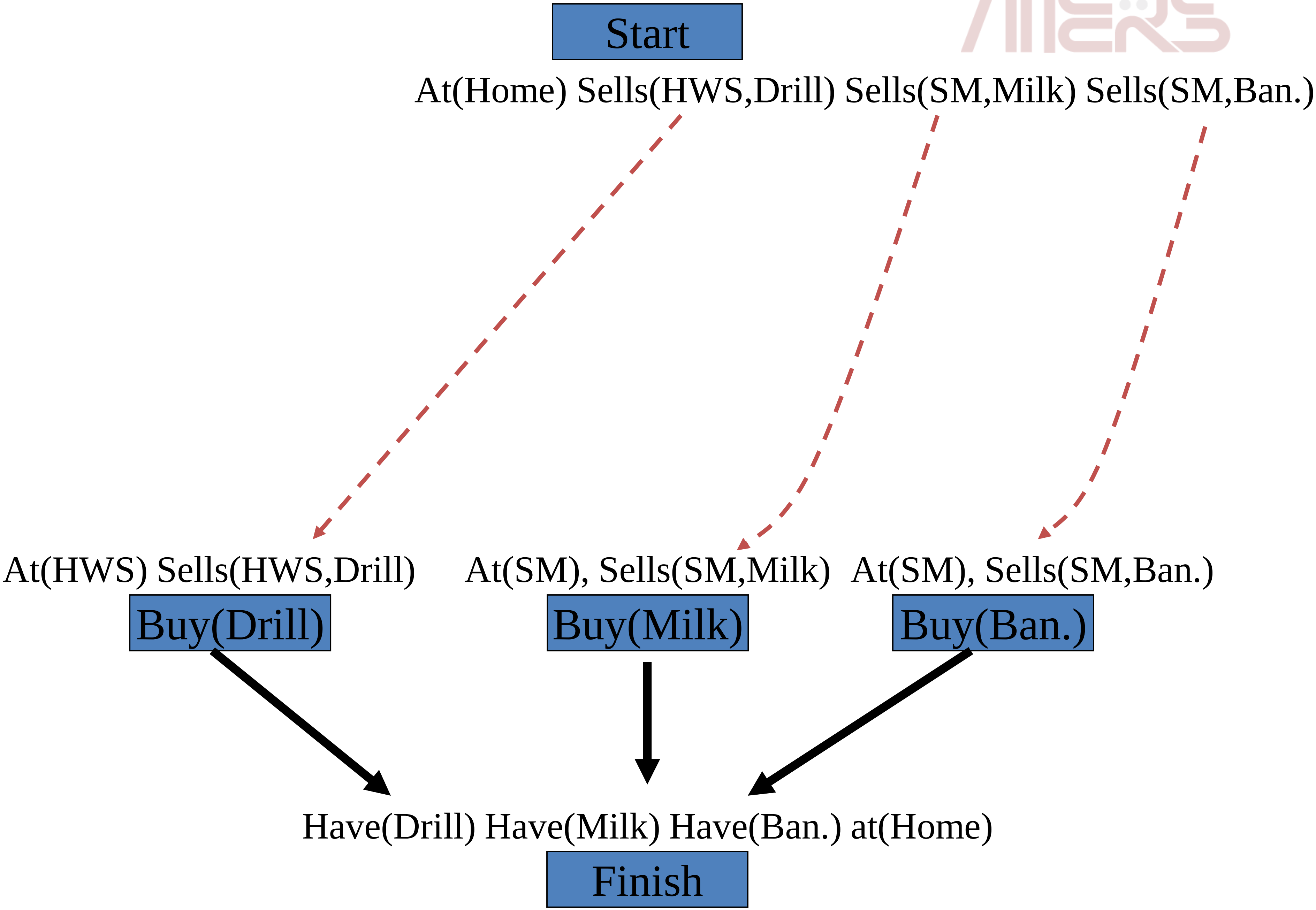
- Partial Planning Overview
- Partial Order Planning Problem
- Partial Order Plan Generation
  - Derivation from Completeness and Consistency
  - Backward Chaining
  - Threat Resolution
  - The POP algorithm

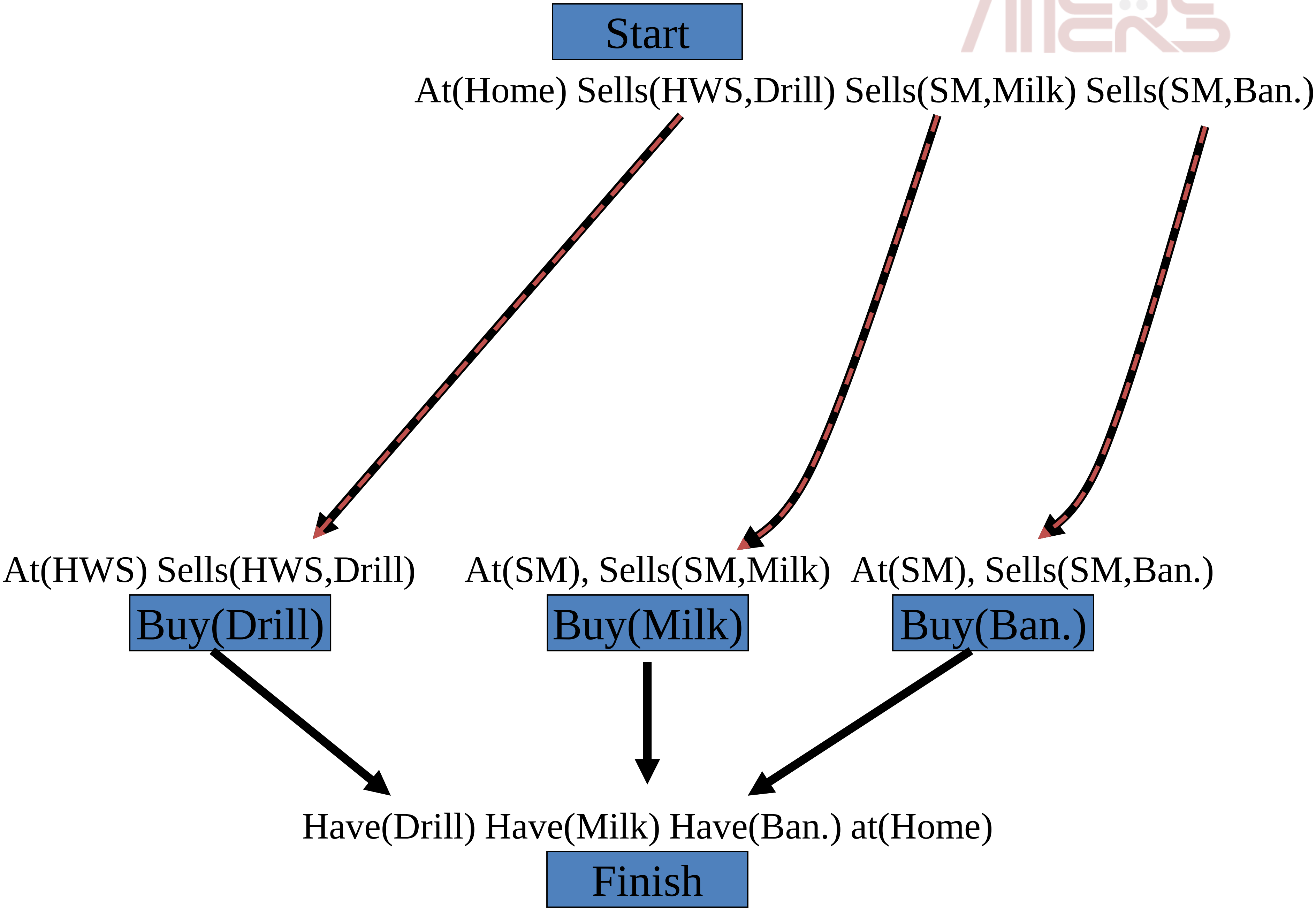
Start

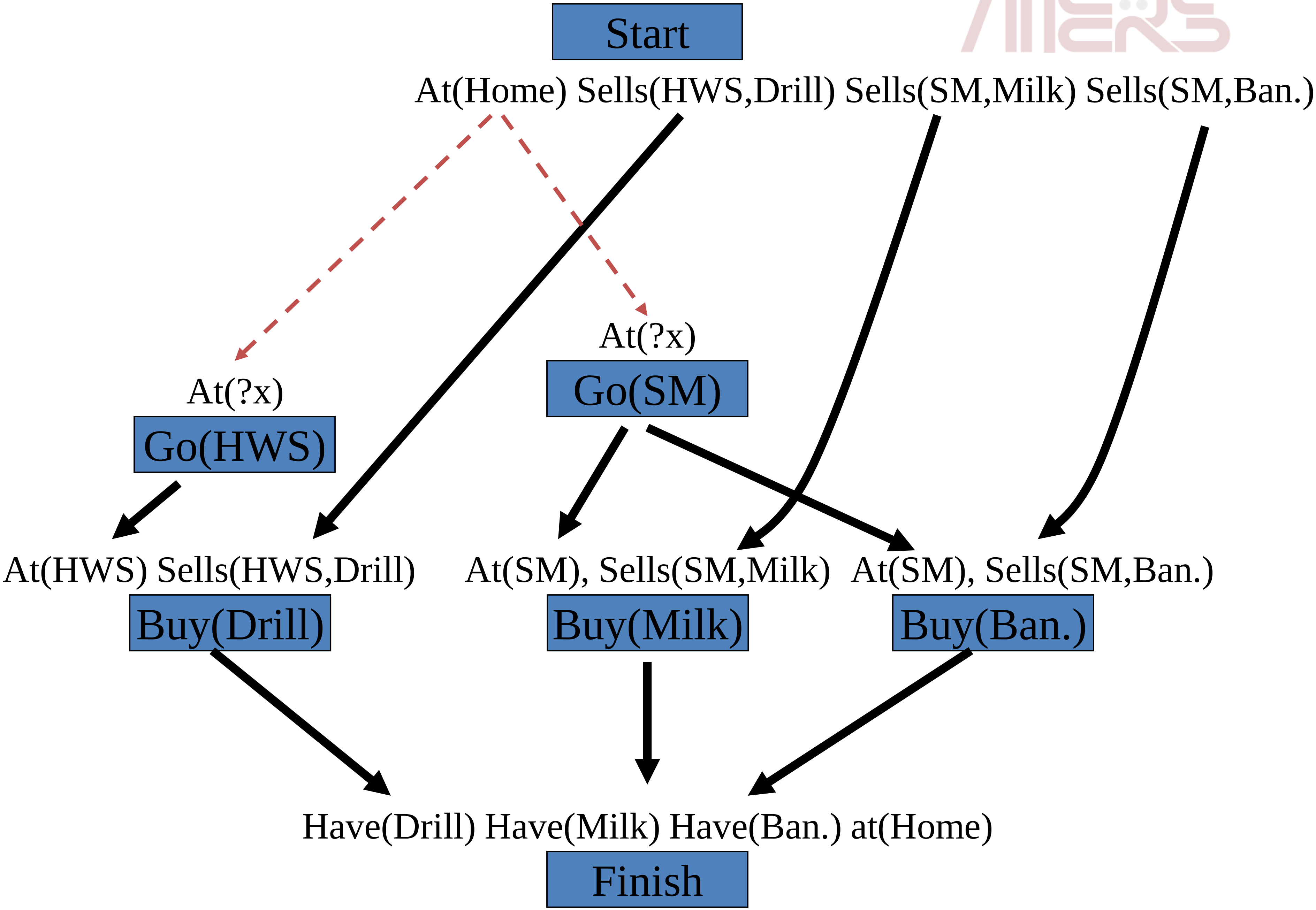
At(Home) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)

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

Finish


Start
At(Home) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)
At(HWS) Sells(HWS,Drill)
Buy(Drill)
At(SM), Sells(SM,Milk)
Buy(Milk)
At(SM), Sells(SM,Ban.)
Buy(Ban.)
Have(Drill) Have(Milk) Have(Ban.) at(Home)
Finish

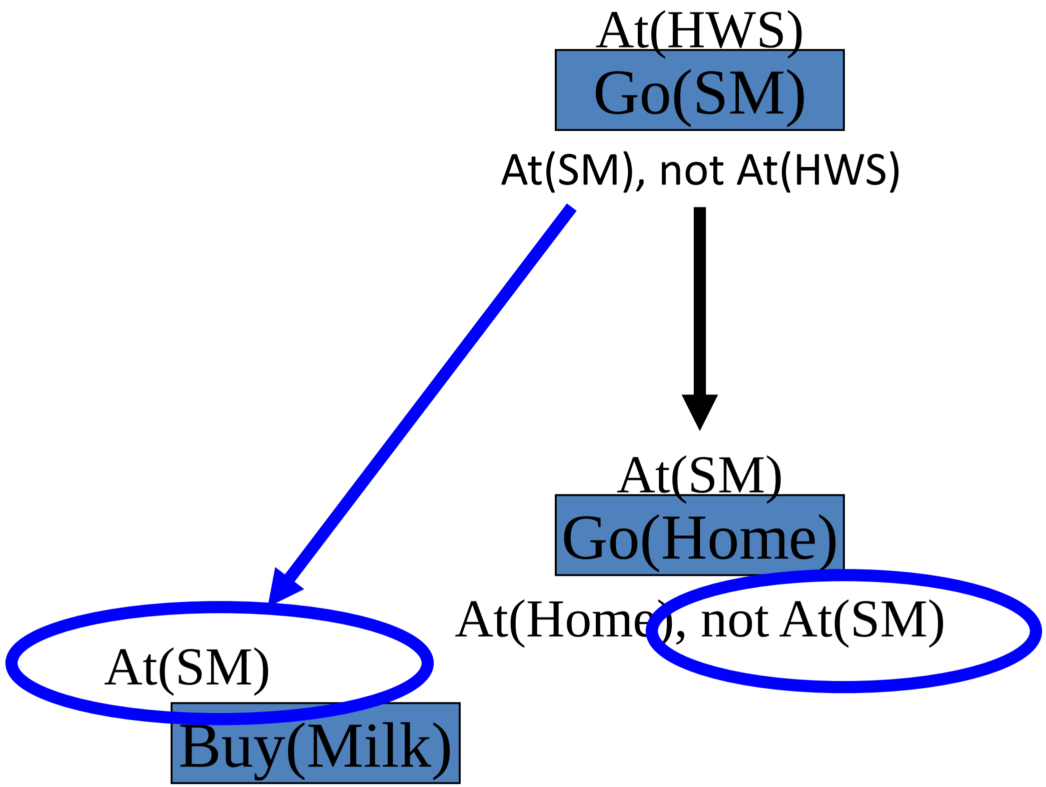

Start
At(Home) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)
At(HWS) Sells(HWS,Drill)
Buy(Drill)
At(SM), Sells(SM,Milk)
Buy(Milk)
At(SM), Sells(SM,Ban.)
Buy(Ban.)
Have(Drill) Have(Milk) Have(Ban.) at(Home)
Finish


Start
At(Home) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)
At(?x)
Go(SM)
At(?x)
Go(HWS)
At(HWS) Sells(HWS,Drill)
Buy(Drill)
At(SM), Sells(SM,Milk)
Buy(Milk)
At(SM), Sells(SM,Ban.)
Buy(Ban.)
Have(Drill) Have(Milk) Have(Ban.) at(Home)
Finish

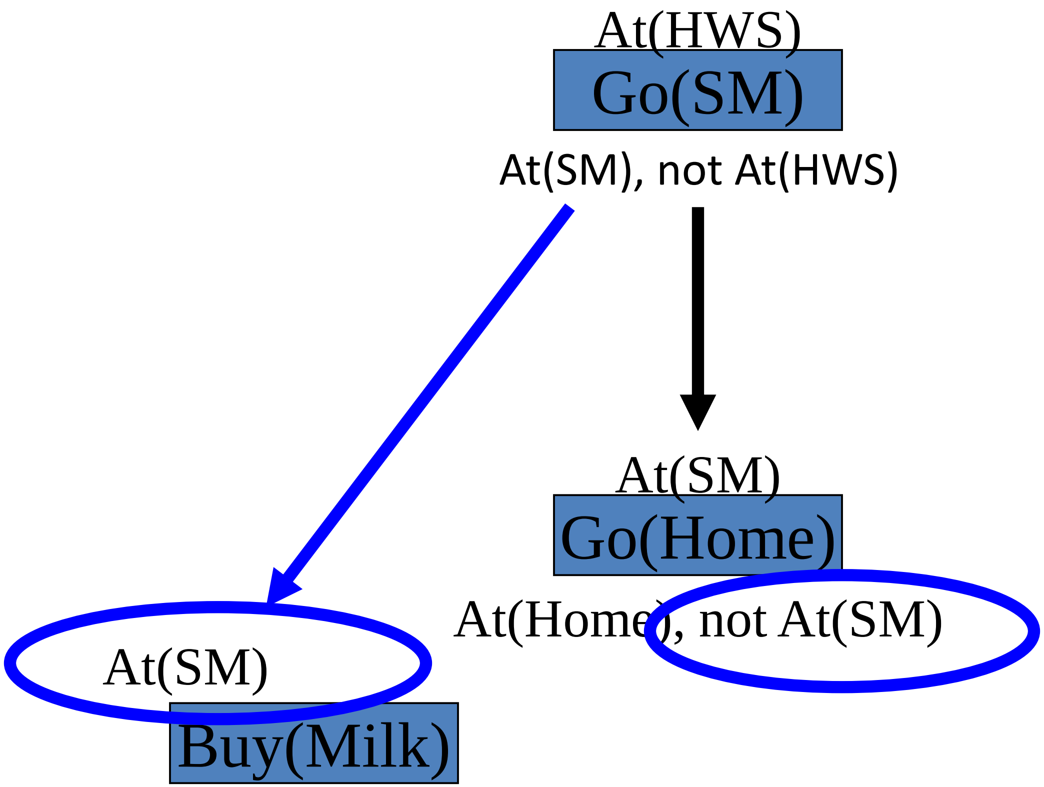
Planning as Goal Regression

- Partial Planning Overview
- Partial Order Planning Problem
- Partial Order Plan Generation
  - Derivation from Completeness and Consistency
  - Backward Chaining
  - Threat Resolution
  - The POP algorithm

After adding a causal link/action, find threat with any existing action/link

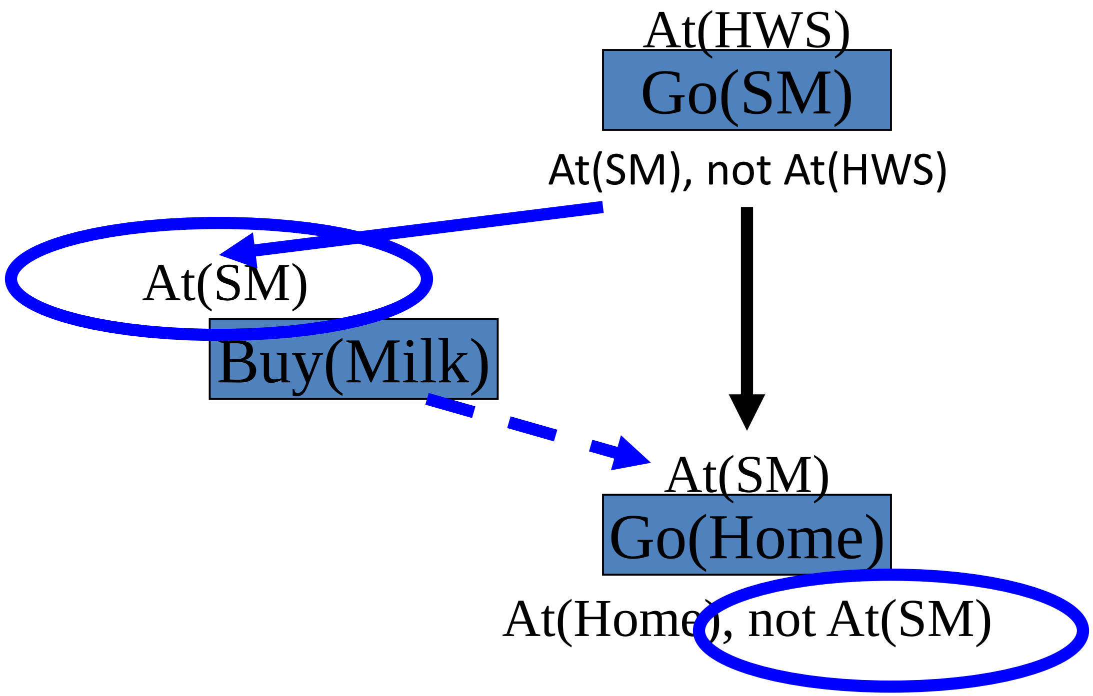


To remove threats…



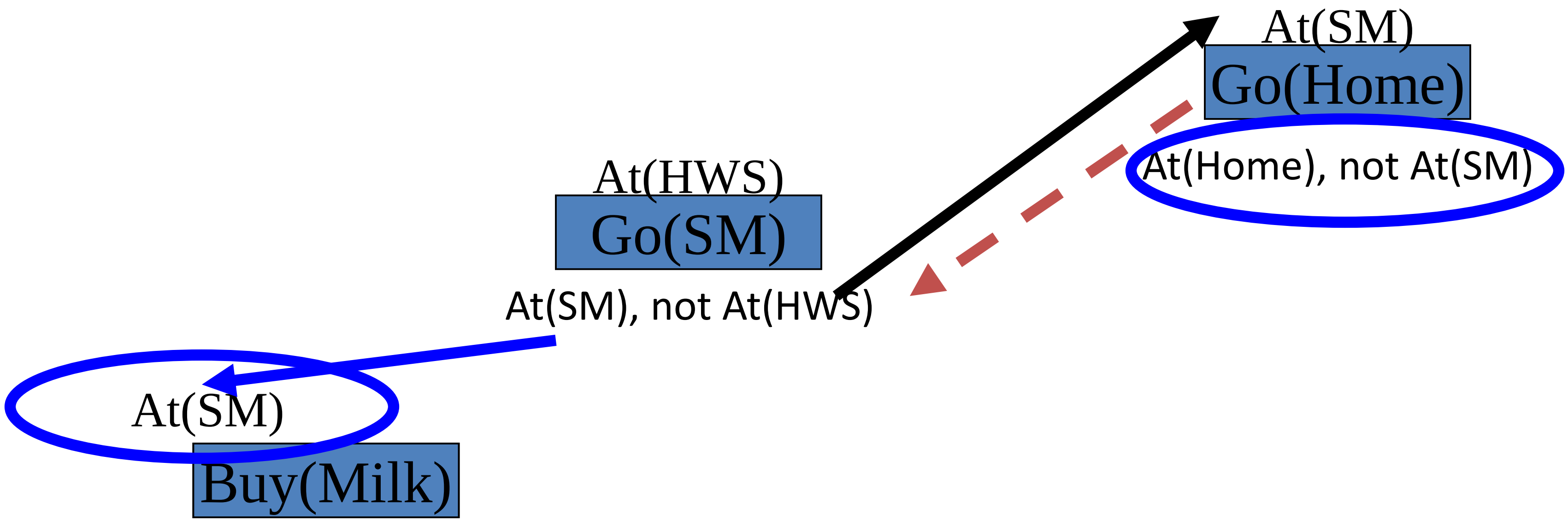
To remove threats…

promote the threat or…



To remove threats…

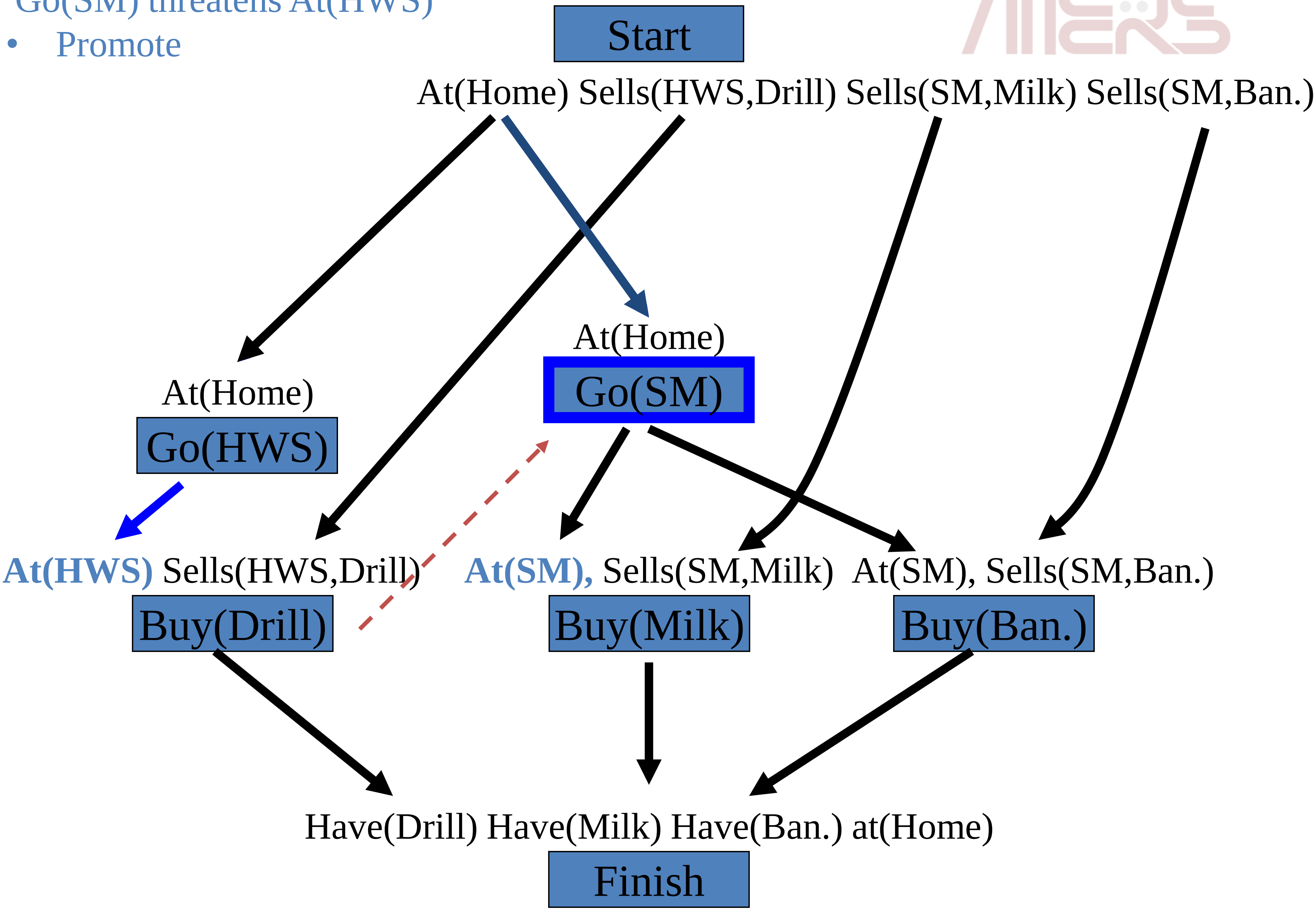
promote the threat or demote the threat



- But only allow demotion/promotion if schedulable
  - consistent = loop free
  - no action precedes initial state

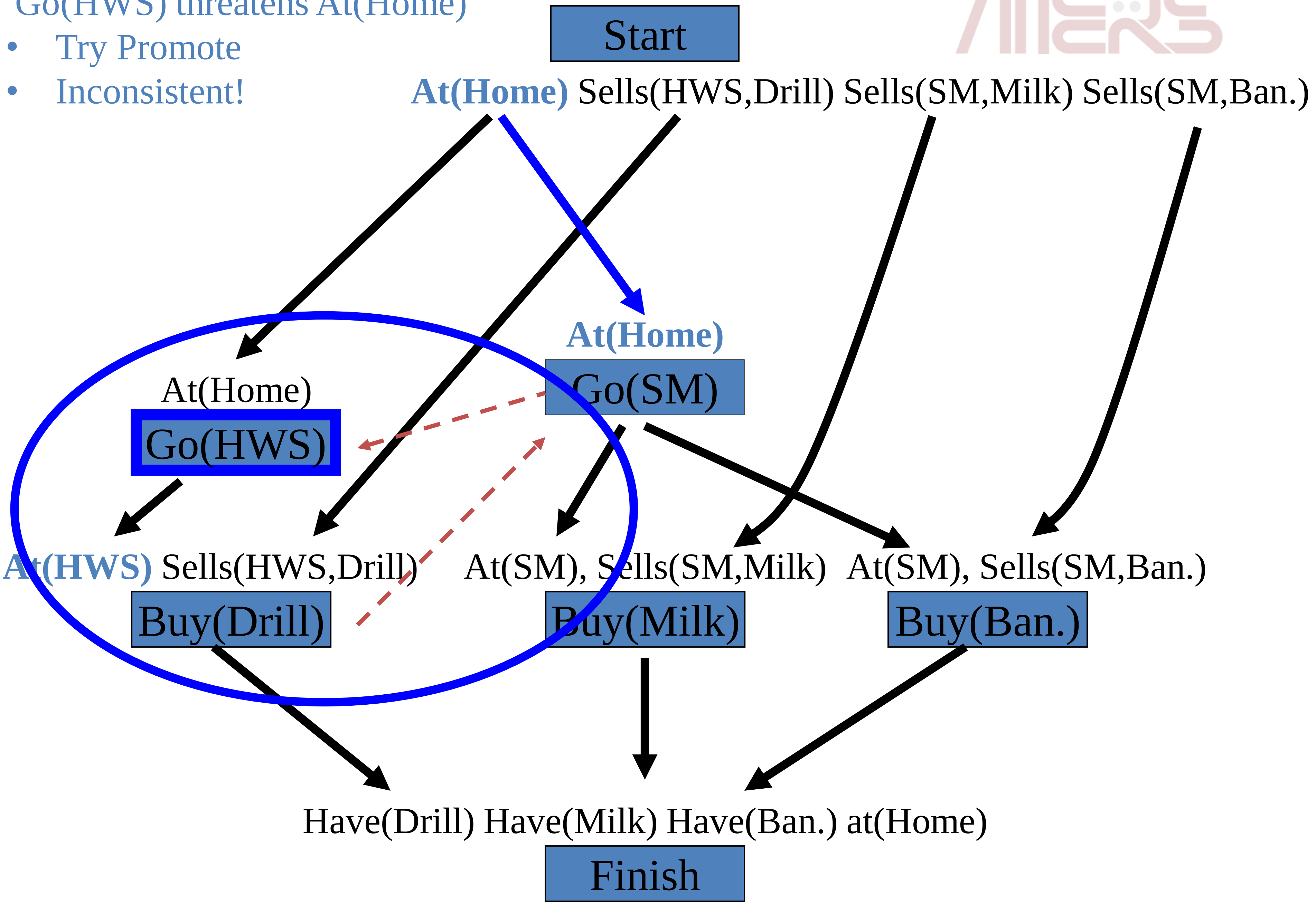
Go(SM) threatens At(HWS)

- Promote



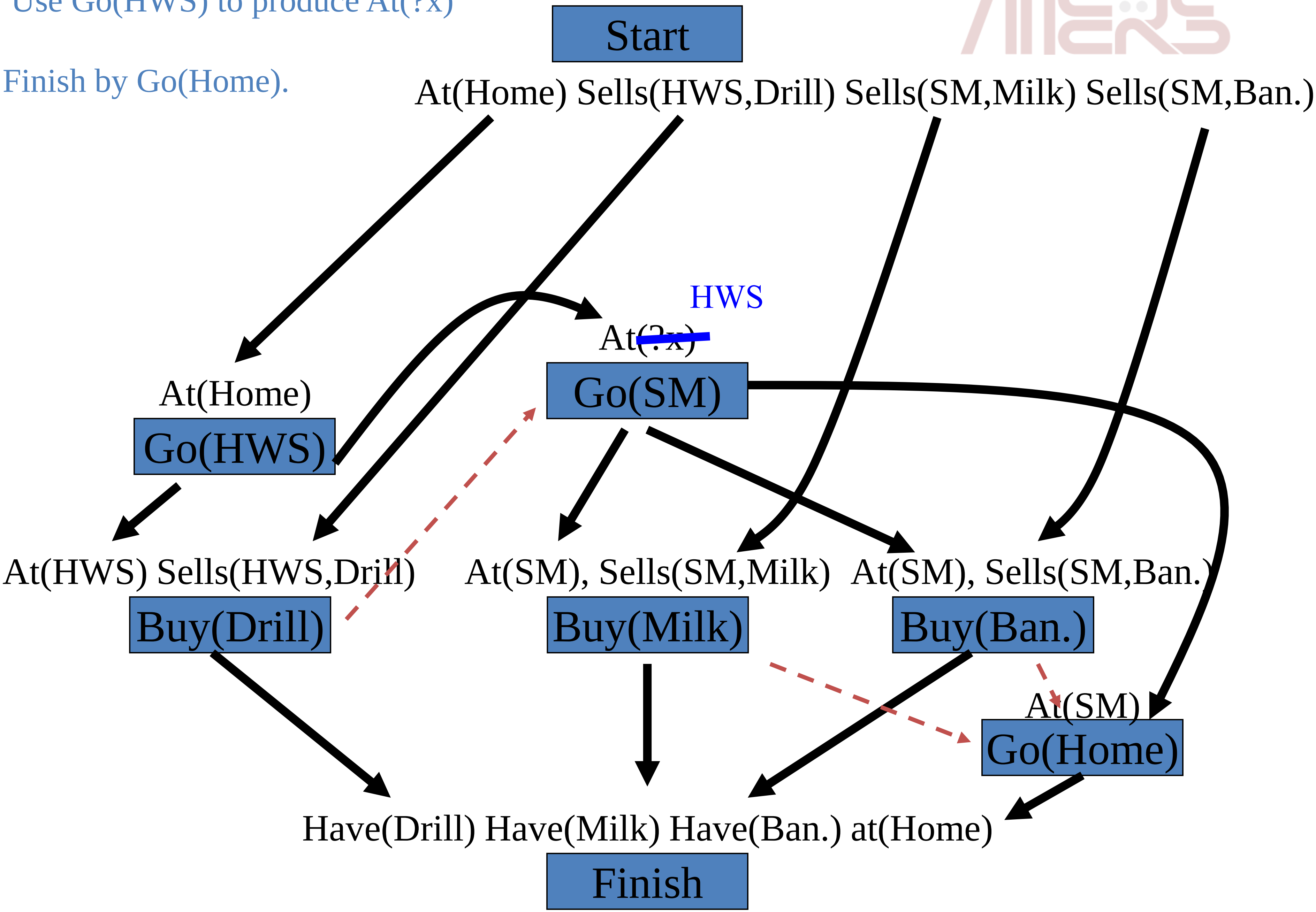
Go(HWS) threatens At(Home)

- Try Promote
- Inconsistent!



Use Go(HWS) to produce At(?x)

Finish by Go(Home).



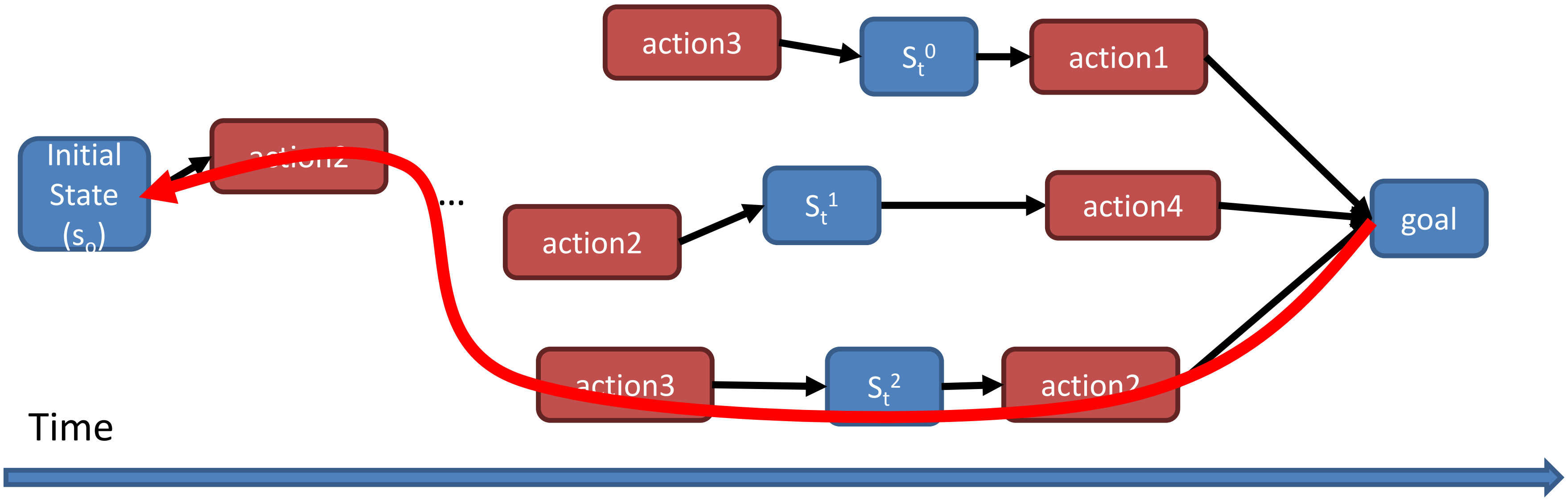
POP(<A,O,L>, agenda, actions)

1. **Termination**: If agenda is empty, return plan <A,O,L>.
2. **Goal Selection**: Select and remove open condition $\langle p, a_{need} \rangle$ from agenda.
3. **Action Selection**: Choose new or existing action a_{add} that can precede a_{need} and whose effects include p. Link and order actions.
4. **Update Agenda**: If a_{add} is new, add its preconditions to agenda.
5. **Threat Detection**: For every action a_{threat} that might threaten some causal link from $a_{produce}$ to $a_{consume}$, choose a consistent ordering:
   a) Demote: Add $a_{threat} < a_{produce}$
   b) Promote: Add $a_{consume} < a_{threat}$
6. **Recurse:** on modified plan and agenda

Choose is nondeterministic

Select is deterministic

Lets Start with Goal-Regression Search



Why can Goal Regression be slow?

- Multiple actions can achieve goals.
- Many possible (sub-)goal orderings.
- Dead-ends can be discovered late.

We try a real-world example next!

What assumptions are implied by the STRIPS representation?

TakeImage (?target, ?instr):
    Pre: Status(?instr, Calibrated),
        Pointing(?target)
    Eff: Image(?target)

Calibrate (?instrument):
    Pre: Status(?instr, On),
        Calibration-Target(?target),
        Pointing(?target)
    Eff: ¬Status(?inst, On),
        Status(?instr, Calibrated)

Turn (?target):
    Pre: Pointing(?direction),
        ?direction ≠ ?target
    Eff: ¬Pointing(?direction),
        Pointing(?target)

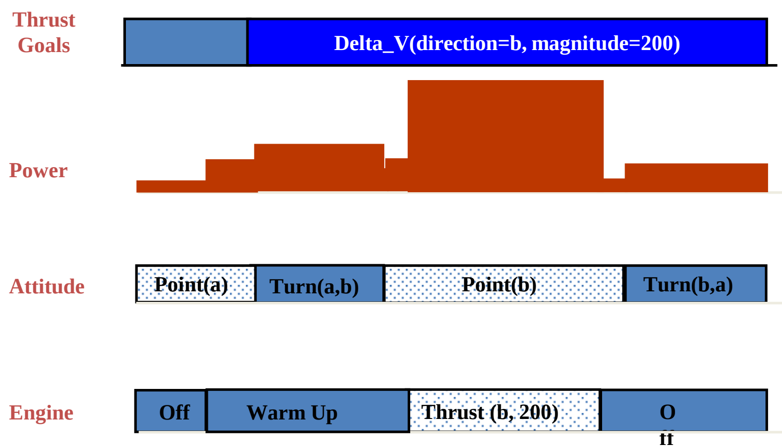
STRIPS Assumptions:

- Atomic time,
- Agent is omniscient (no sensing necessary),
- Agent is sole cause of change,
- Actions have deterministic effects, and
- No indirect effects.
- One action at a time.
- No metric time.
- No goals over time.

Outline

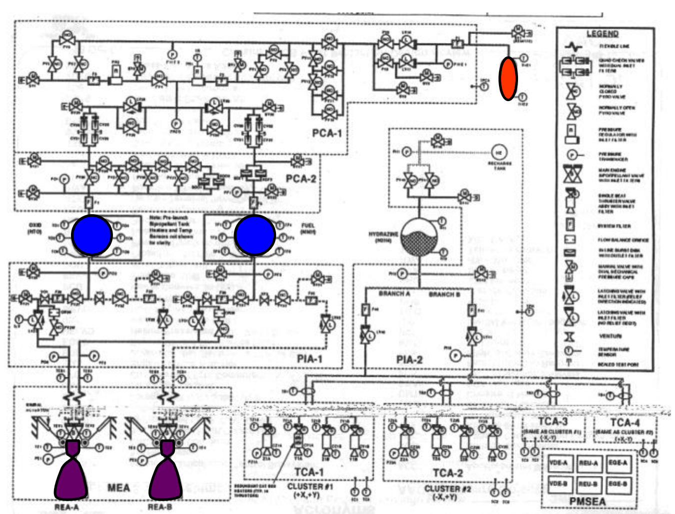
- Review: programs on state
- Planning as goal regression (SNLP/UCPOP)
- Goal regression planning
  with causal graphs (Burton)
- Appendix: HFS planning with the
  causal graph heuristic (Fast Downward)

DS1 Revisited: Planning to Control Interacting Devices over Time



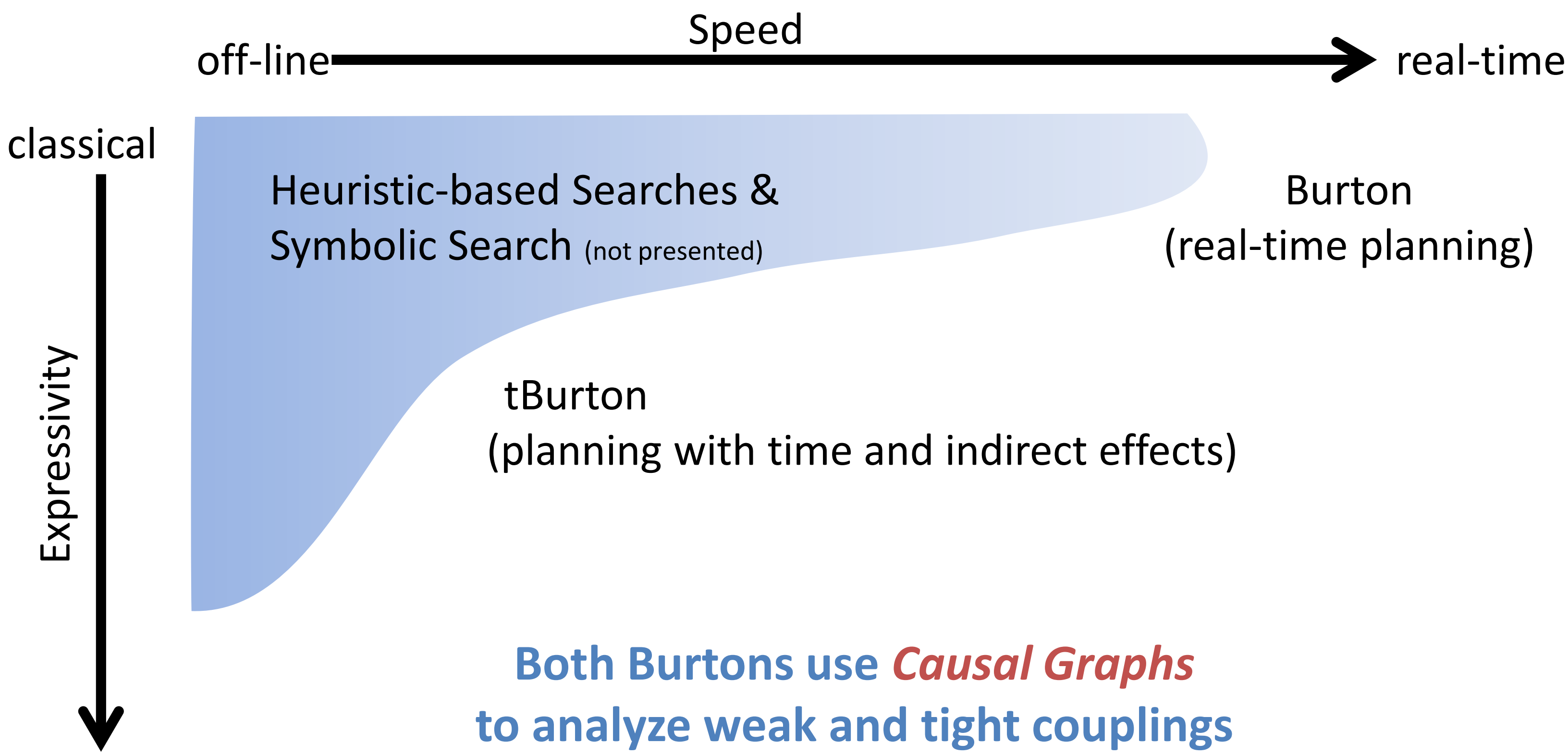
Time-line Planning
(tBurton)

This image is in the public domain.



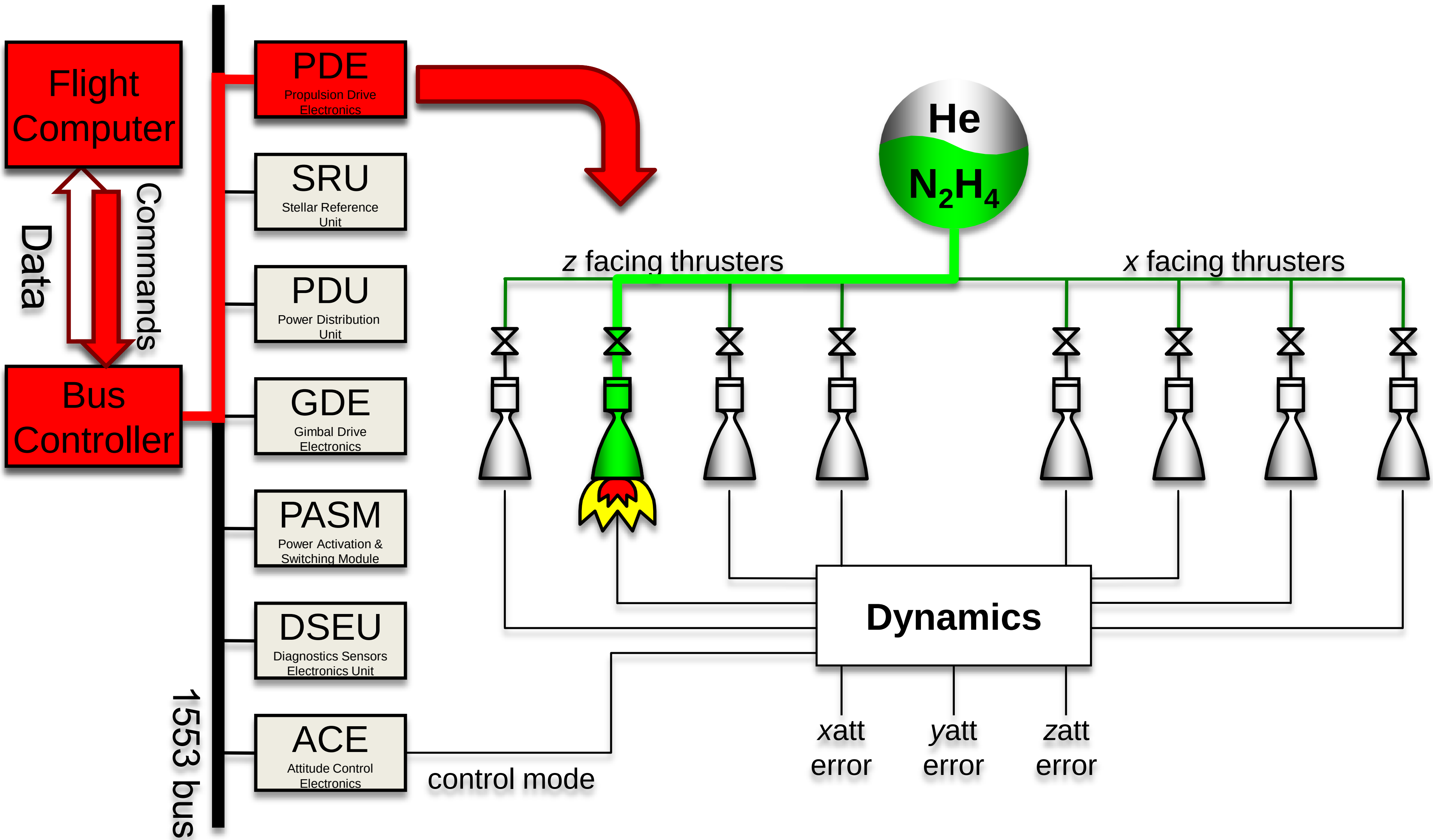
Reactive Device
Reconfiguration and Repair
(Burton)

Domain Independent Planning in the Real-World

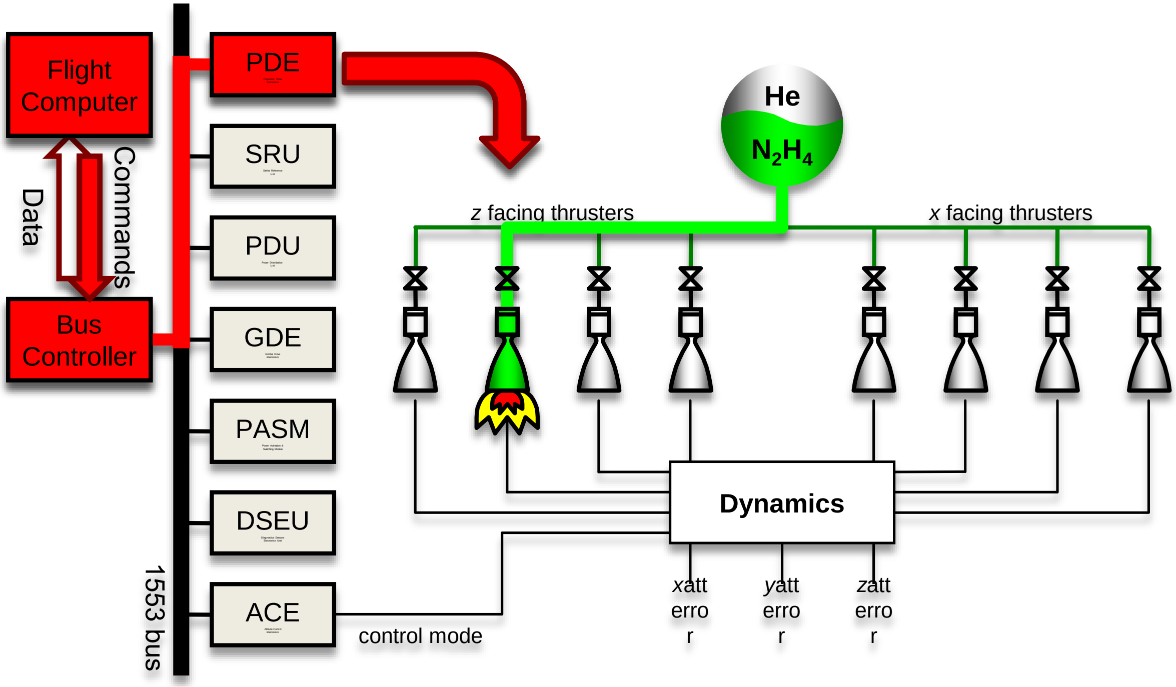


Both Burtons use *Causal Graphs*
to analyze weak and tight couplings
and to Decompose.

Planning with Indirect Effects: DS 1 Attitude Control System Example



Why is Controlling an Engineered Device Easier than a Puzzle?



6	2	8
	3	5
4	7	1

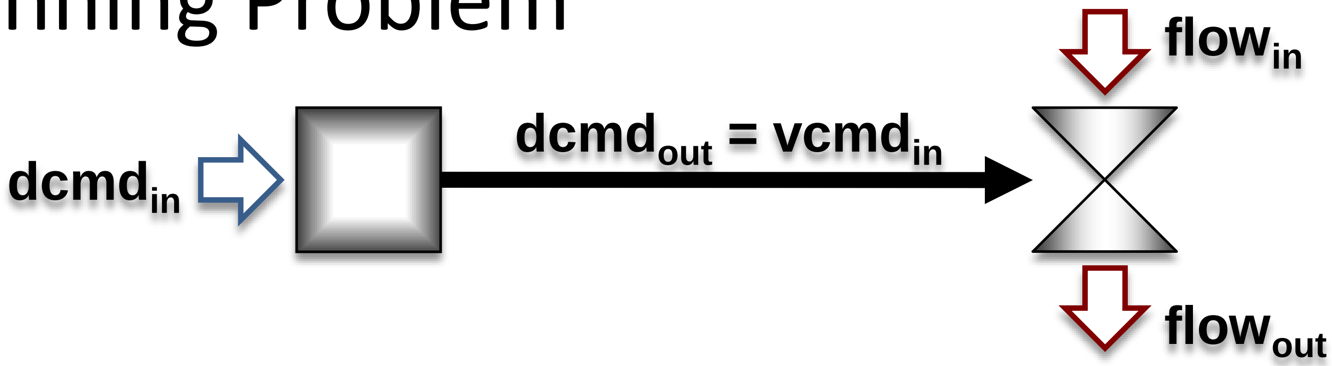
1. Actions are reversible.
2. Devices hold state.
3. Causes and effects form a tree.

1. Actions are reversible.
2. Devices hold state.
3. Causes and effects form tight cycles.

Outline

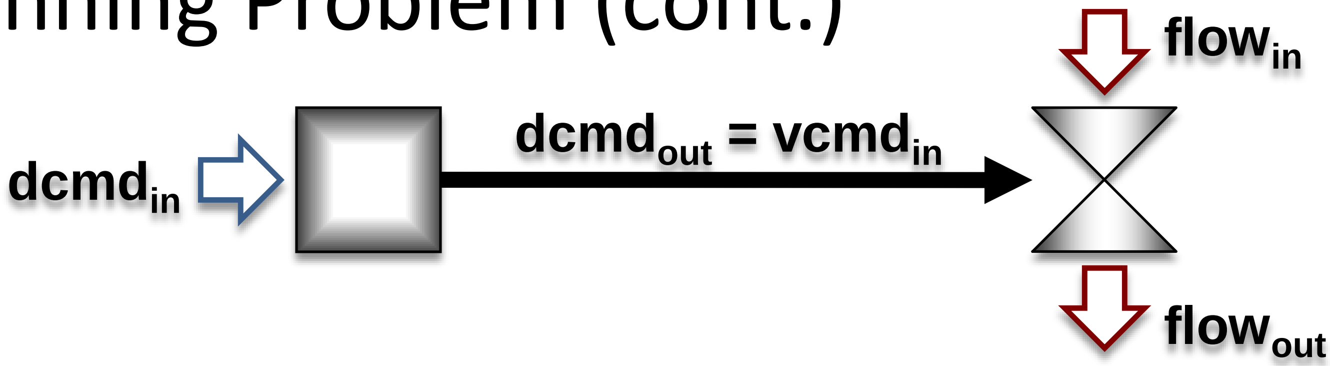
- Review: programs on state
- Planning as goal regression (SNLP/UCPOP)
- Goal regression planning with causal graphs (Burton)
  - Constraint automata planning problem
  - Causal graphs
  - Using causal graphs to order goals
  - Computing policies for selecting actions
  - Appendix: Planning for cyclic causal graphs
- Appendix: HFS planning with the causal graph heuristic (Fast Downward)

Concurrent Constraint Automata Planning Problem

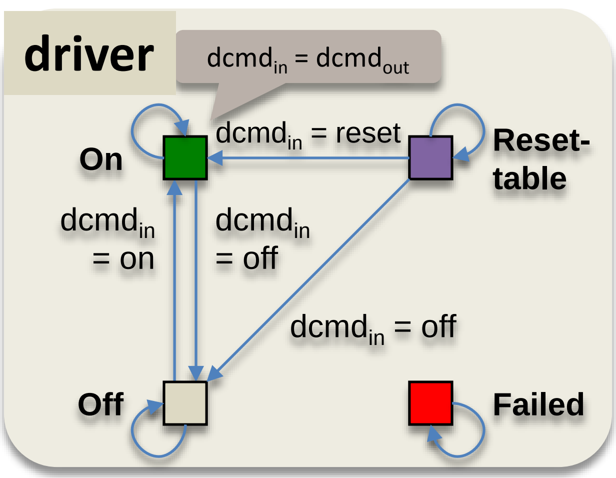


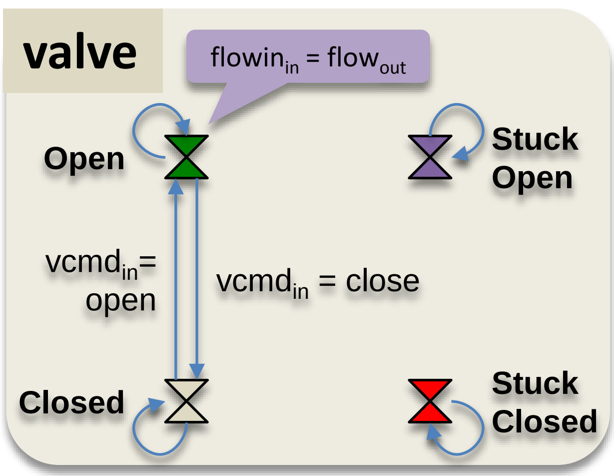
- Variables and Domains:
  - State: driver in {on, off, resettable, failed}, valve in {open, closed, stuck-open, stuck-closed}.
  - Control: $dcmd_{in}$ in {idle, on, off, reset, open, close}
  - Dependent: $flow_{in}$, $flow_{out}$ in {pos, neg, zero}
    $dcmd_{out}$, $vcmd_{in}$ in Domain{$dcmd_{in}$}
- Initial assignment: {driver = on, valve = closed}
- Goal assignment: {driver = off, valve = open}

Concurrent Constraint Automata Planning Problem (cont.)



- Constraint automata (one per state variable):



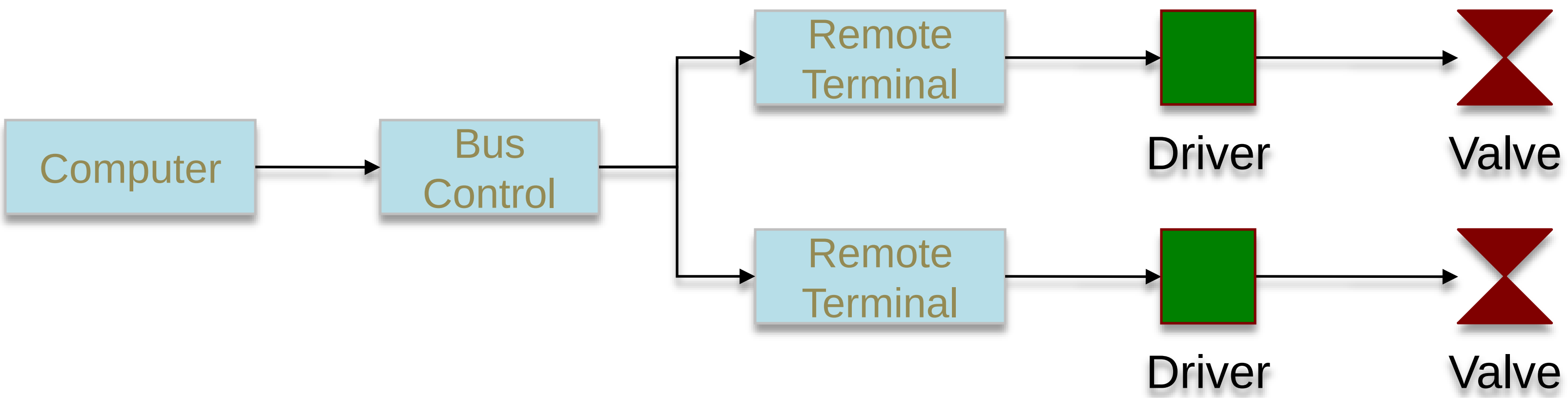


  - Assume: transitions independently controlled, each location can idle.
- State constraints: $\{dcmd_{out} = vcmd_{in}\}$

Outline

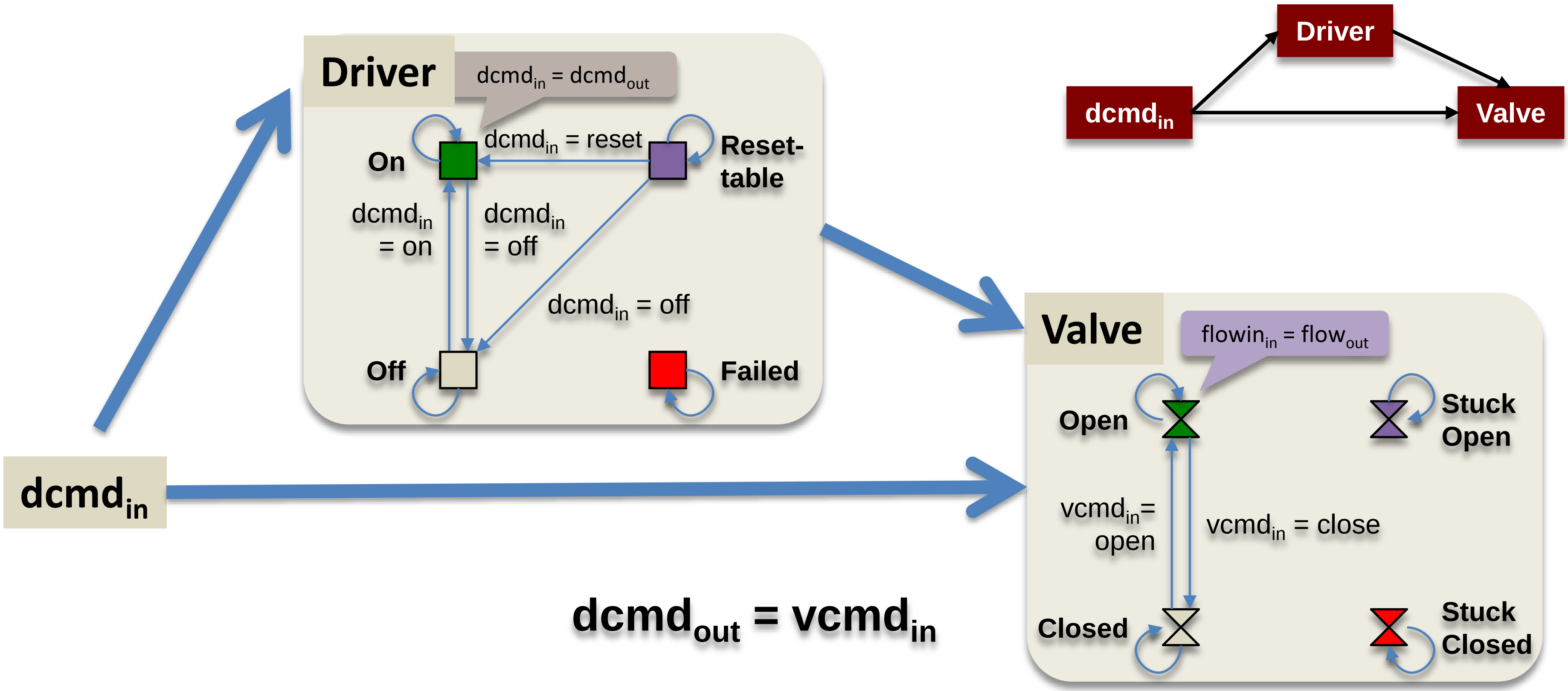
- Review: programs on state
- Planning as goal regression (SNLP/UCPOP)
- Goal regression planning with causal graphs (Burton)
  - Constraint automata planning problem
  - Causal graphs
  - Using causal graphs to order goals
  - Computing policies for selecting actions
  - Appendix: Planning for cyclic causal graphs
- Appendix: HFS planning with the causal graph heuristic (Fast Downward)

Observation:
Engineered systems are largely loop free.



Causal Graph *G* of concurrent automata S:

- **Vertices** are control and state **variables of automata**.
- **Edge** from v_i to v_j if v_j's **transition** is **conditioned** on v_i.



Outline

- Review: programs on state
- Planning as goal regression (SNLP/UCPOP)
- Goal regression planning
  with causal graphs (Burton)
  - Constraint automata planning problem
  - Causal graphs
  - Using causal graphs to order goals
  - Computing policies for selecting actions
  - Appendix: Planning for cyclic causal graphs
- Appendix: HFS planning with the
  causal graph heuristic (Fast Downward)

Idea: use causal graph analysis to eliminate ALL forms of search (Burton)

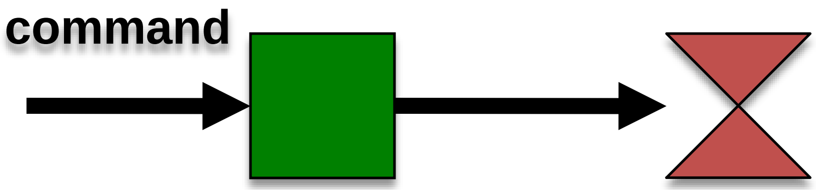
POP(<A,O,L>, agenda, actions):

1. **Termination**: If agenda is empty, return plan <A,O,L>.
2. **Goal Selection**: Select and remove open condition $\langle p, a_{need} \rangle$ from agenda.
3. **Action Selection**: **Choose new or existing action** a_{add} that can precede a_{need} and whose effects include p.
   Link and order actions.
4. **Update Agenda**: If a_{add} is new, add its preconditions to agenda.
5. **Threat Detection**: For **every action a_{threat} that might threaten** some causal link from $a_{produce}$ to $a_{consume}$, choose a consistent ordering:
   a) **Demote**: Add $a_{threat} < a_{produce}$
   b) **Promote**: Add $a_{consume} < a_{threat}$
6. **Recurse:** on modified plan and agenda

Burton [Williams & Nayak, IJCAI 1997]

Why do goal orderings matter?

1. An achieved goal can be clobbered by a subsequent goal.



- Example:

  Current State: **driver = on**, **valve = closed.**

  Goal State: **driver = off**, **valve = open.**

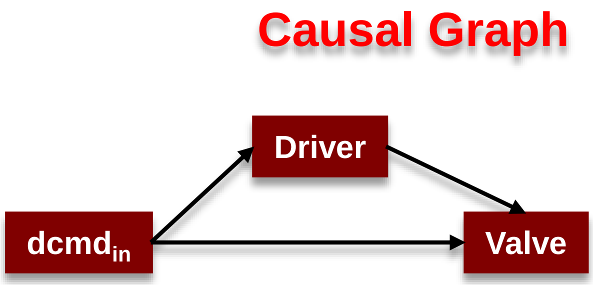
  – Achieving (**driver = off**), followed by (**valve = open**) clobbers (**driver = off**).

Achieve valve goal before driver goal.

Effect Cause

Goal Ordering for Causal Graph *Planning*

Require: The CCA causal graph to be acyclic.

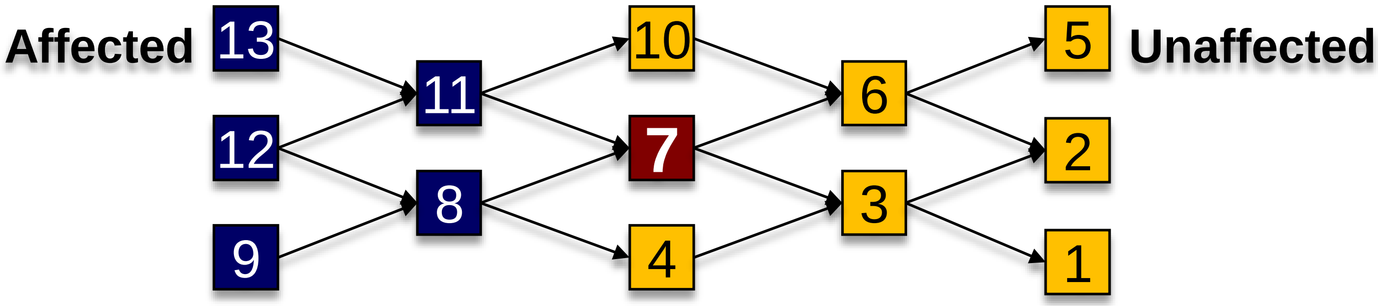


Idea: Achieve conjunctive goals upstream within the causal graph, from "effects" to "causes"

(i.e., children to parents).

Property: Safe to **achieve** goals in an **upstream order**

- The only variables used to set some variable v_7 is its ancestors.
- Variable v_7 can be changed without affecting its descendants.

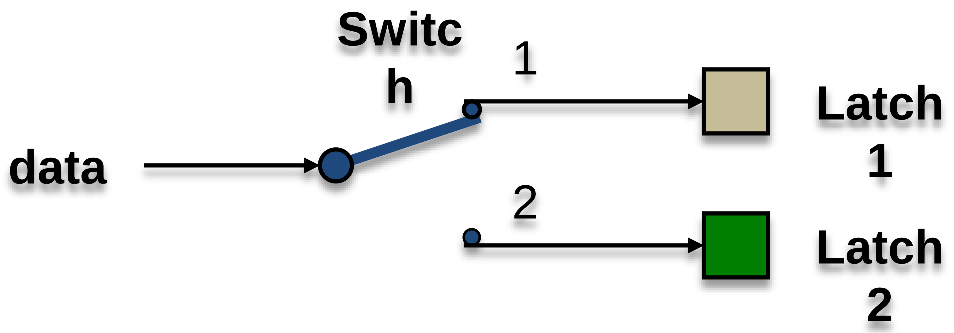


  – Exploits: Each transitions independently controlled, each location can idle.

- Simple check:
  1. Number the causal graph depth-first from leaves.
     - Child has lower number than parents
  2. Achieve goals in the order of increasing depth-first number.

Why do goal orderings matter?

2. Two goals can compete for the same variable associated with their sub-goals.

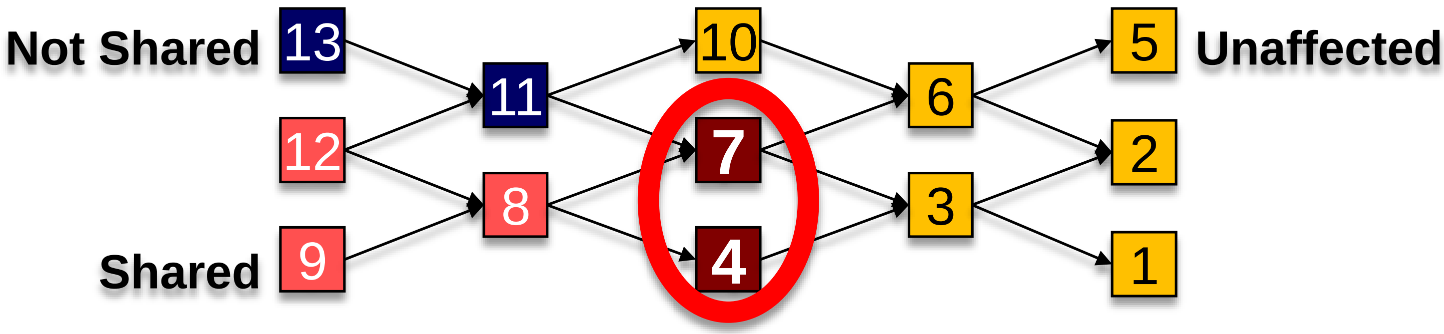


- Example: Latch Data at Latches 1 and 2
  - If Latch1 and Latch2 goals achieved at same time, Latch1 and Latch2 compete for Switch position.

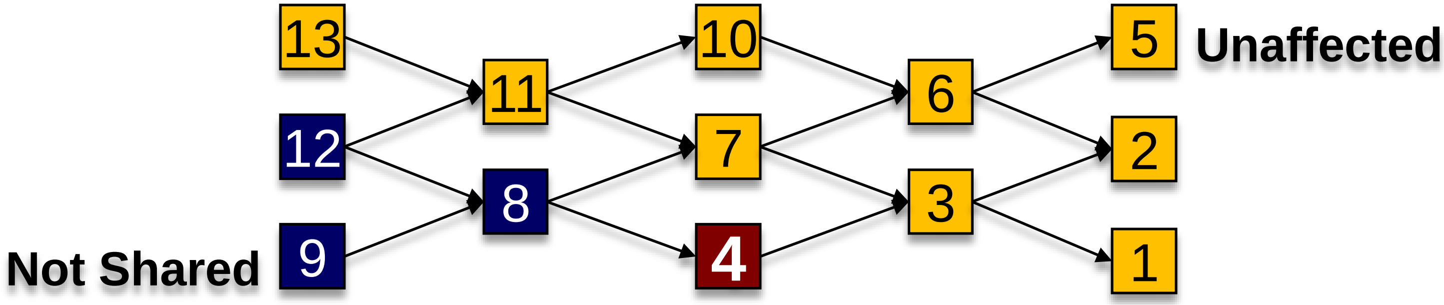
➡ Solve one goal completely before the other (serially).

Property: Safe to achieve one goal **before** starting next sibling (**serialization**).

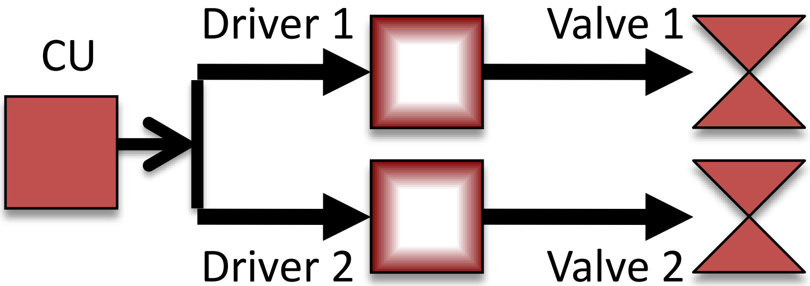
- Sibling goals v_7 and v_4 may both need shared ancestors.



- Competition gone once sibling goal v_7 is satisfied.



Goal regression, causal graph planning example



Burton's goal ordering rules:

1. Achieve effect goals before their causes, with control actiions last.
2. Achieve goals serially

(like Hacker[Sussman PhD]).

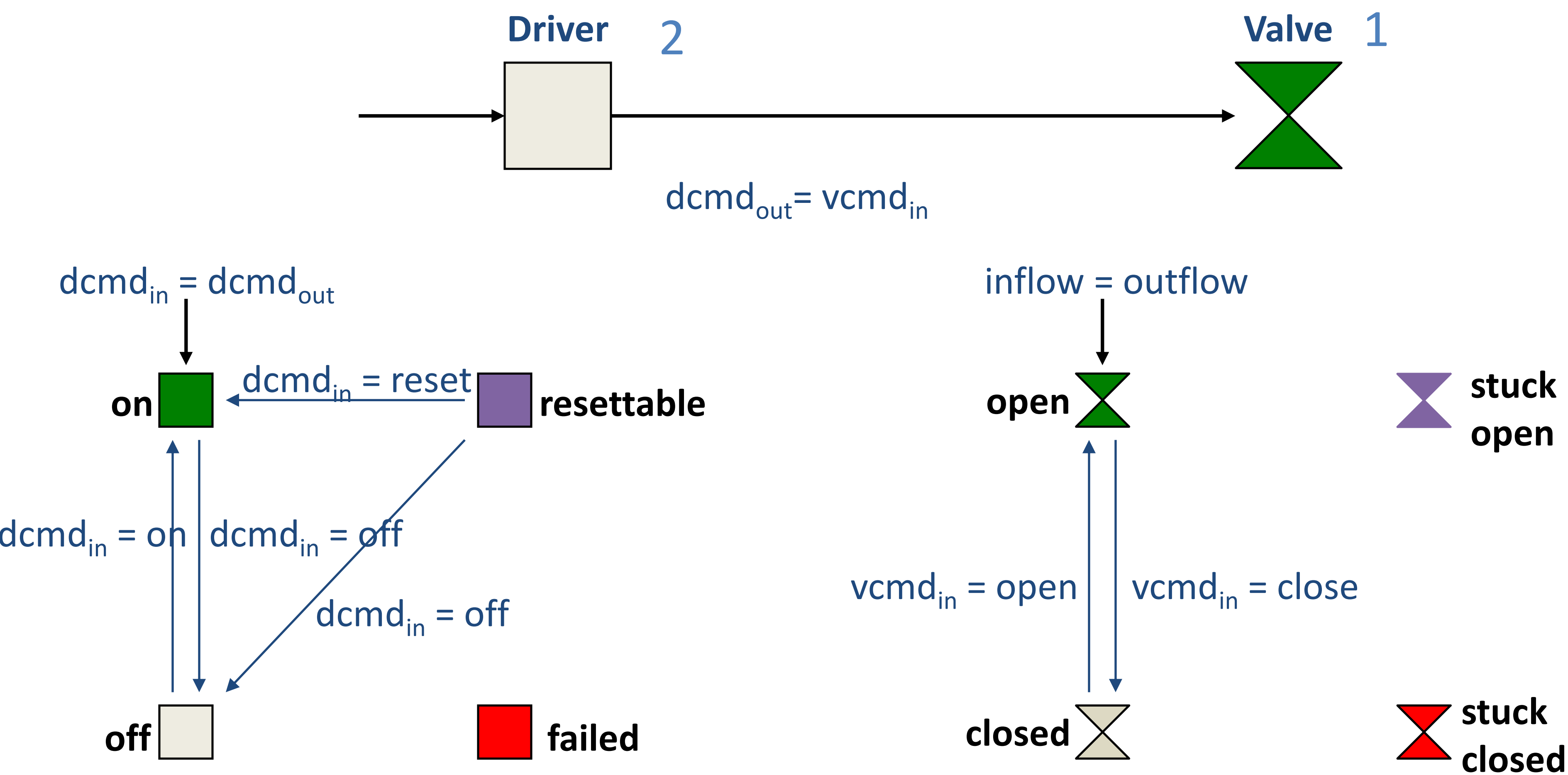
- *Valve$_1$ = open* — Open
  - *Driver$_1$ = on* — Turn-On
    - *CU = on* — Turn-On
      - CU = off
  - *CU = on*
- *Valve$_2$ = open* — Open
  - *Driver$_2$ = on* — Turn-On
    - *CU = on*
  - *CU = on*
- *Driver$_1$ = off* — Turn-Off
  - *CU = on*
- *Driver$_2$ = off* — Turn-Off
  - *CU = on*

First action generated first!

No Search!

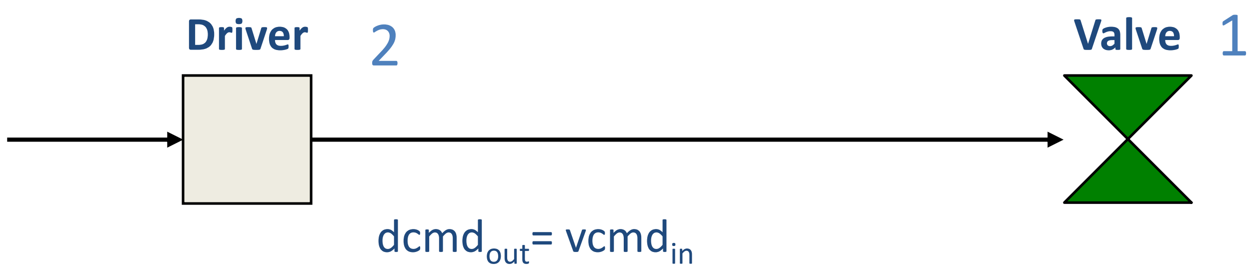
- Worst Case per action: Depth * Sub-goal branch factor.
- Average Cost per action: Sub-goal branch factor.

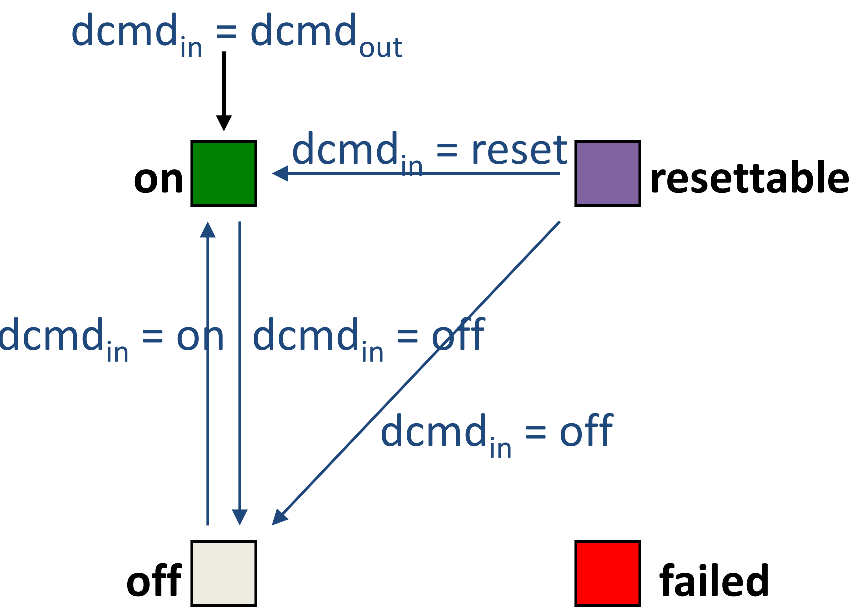
To select actions reactively, convert constraint automata to lookup policies



To select actions reactively, convert constraint automata to lookup policies

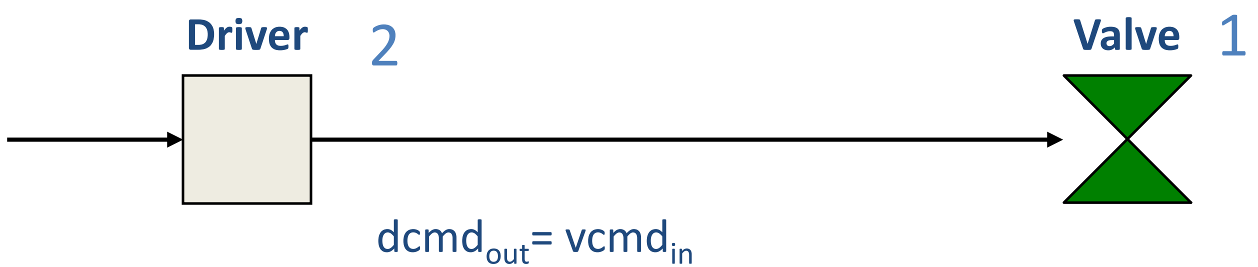
Algorithm: Instance of APSP



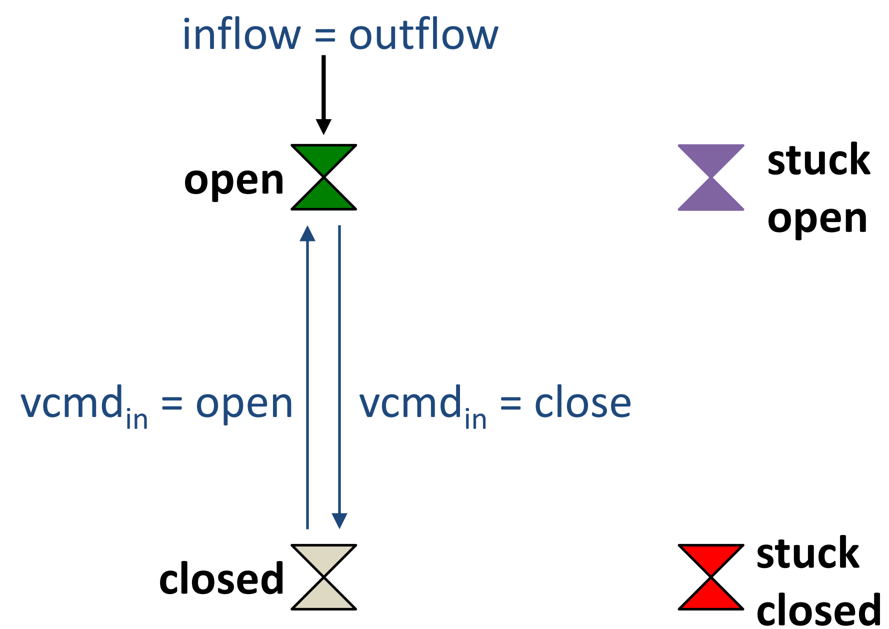


Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

To select actions reactively, convert constraint automata to lookup policies



Current \ Goal	**Open**	**Closed**
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

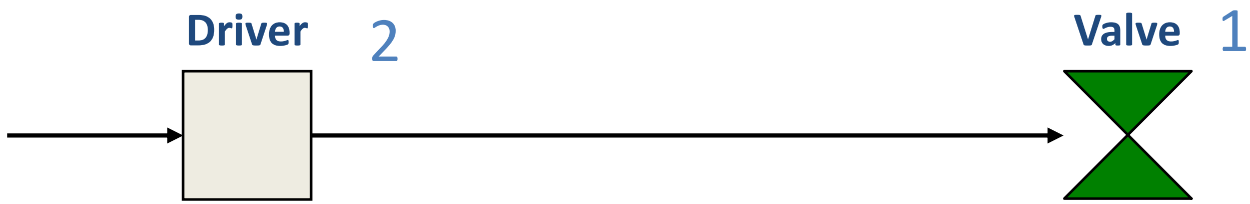


Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = off, Valve = open

Algorithm:
see [williams and nayak, IJCAI97]



Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

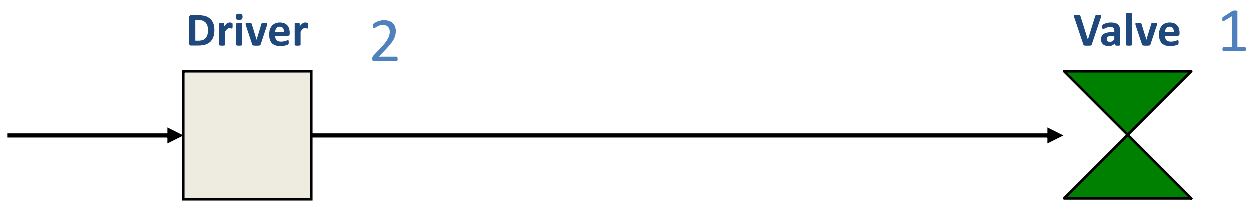
Current \ Goal	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = off, Valve = open

Algorithm:
see [williams and nayak, IJCAI97]



Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

Current \ Goal	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

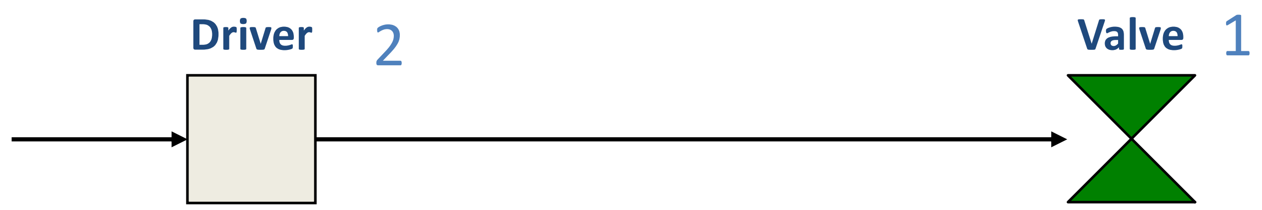
Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = off, Valve = open

Algorithm:
see [williams and nayak, IJCAI97]

Send:
cmd = on



Driver:

Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

Valve:

Current \ Goal	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = resettable, Valve = open

Algorithm:
see [williams and nayak, IJCAI97]

Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	cmd = reset	cmd = off

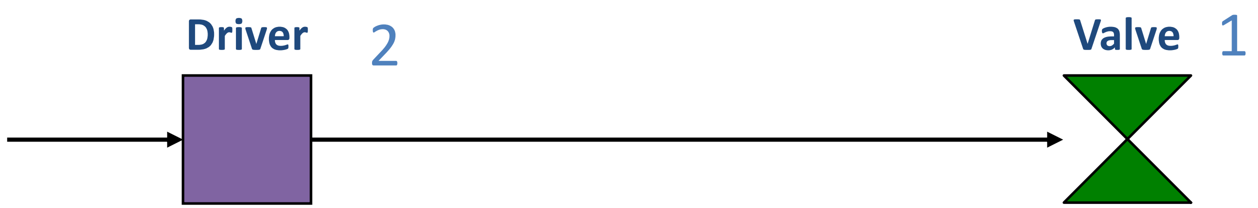
Current \ Goal	Open	Closed
Open	idle	driver = on cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = resettable, Valve = open

Algorithm:
see [williams and nayak, IJCAI97]



Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	**cmd = reset**	cmd = off

Current \ Goal	Open	Closed
Open	idle	**driver = on** cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

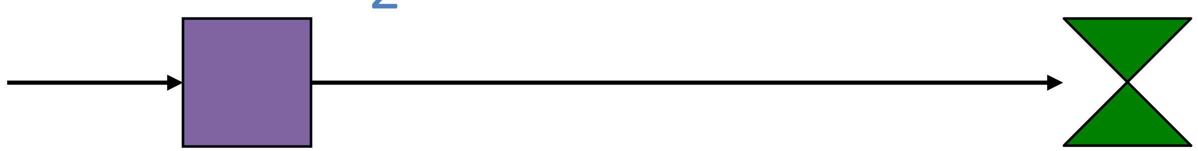
Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed

Current: Driver = resettable, Valve = open

Algorithm:
see [williams and nayak, IJCAI97]

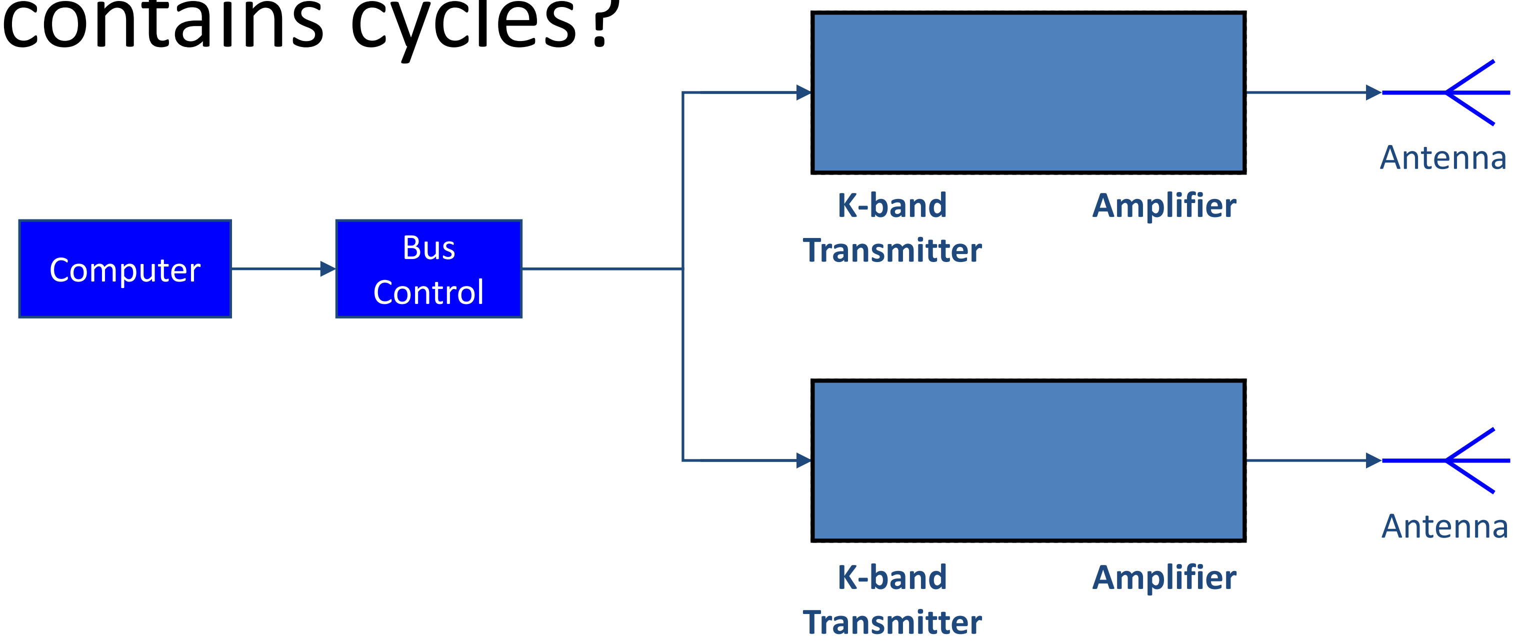
Send
cmd = reset



Current \ Goal	On	Off
On	idle	cmd = off
Off	cmd = on	idle
Resettable	**cmd = reset**	cmd = off

Current \ Goal	Open	Closed
Open	idle	**driver = on** cmd = close
Closed	driver = on cmd = open	idle
Stuck	fail	fail

What if the causal graph G contains cycles?



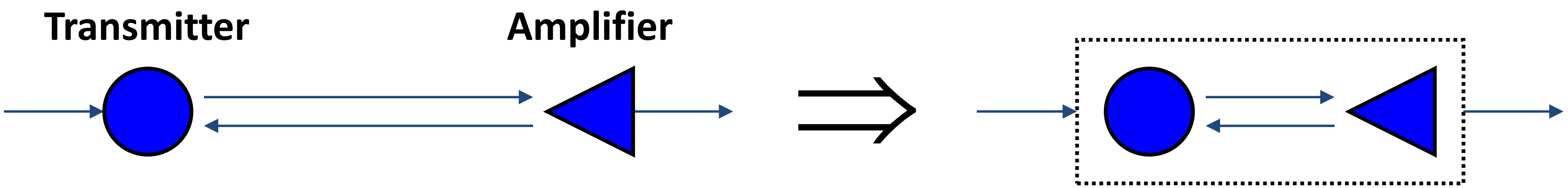
Problem: Plan is no longer serializable.

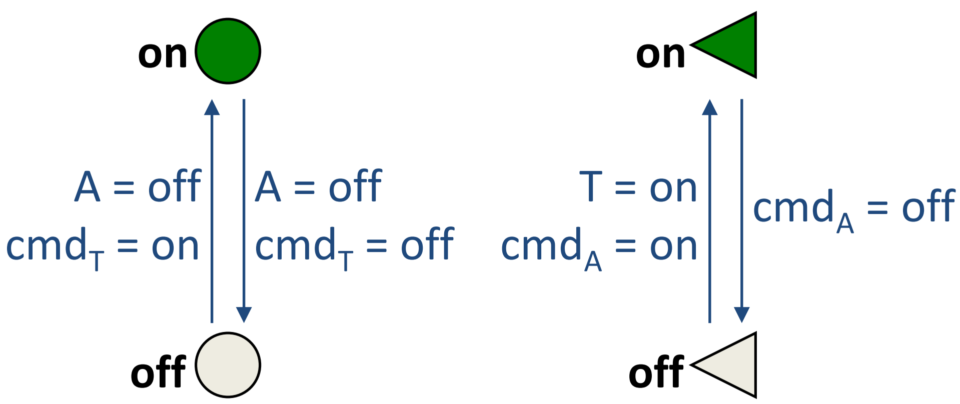
Solution:

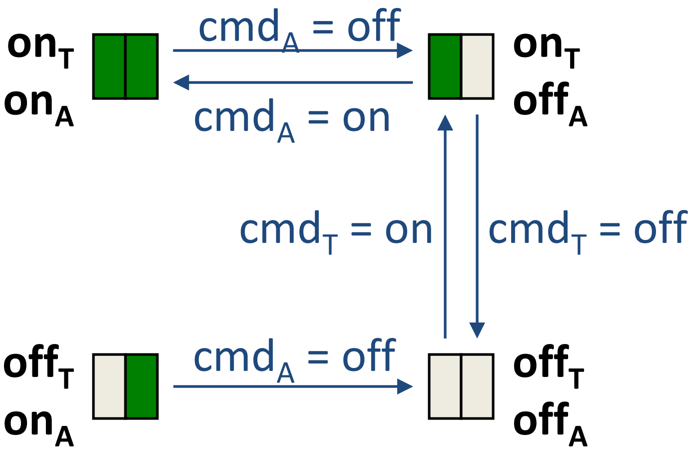
- Isolate the cyclic components (compute Strongly Connected Components).
- compose each cycle into a single component.

- New causal graph *G'* is acyclic.
- Goals of G' are serializable.

Action Policy for Composed Components

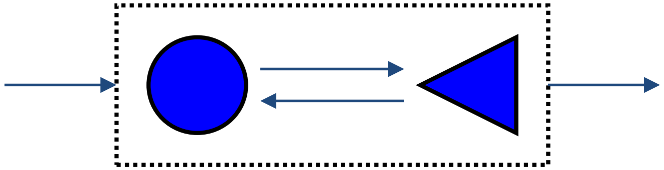




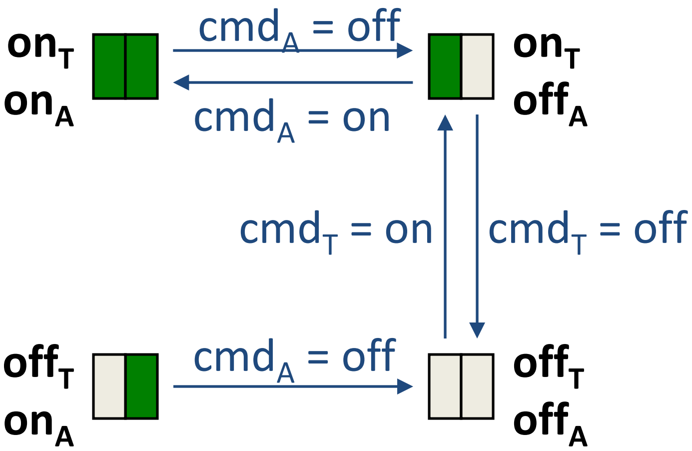


Action Policy for Composed Components

- Problem: Composition grows exponential in space usage.



- Solution: Use BDD encoding.
  [Chung and Williams, Self-adaptive SW 03]



Current \ Goal	On_T, On_A	On_T, Off_A	Off_T, Off_A	Off_T, On_A
On_T, On_A	idle	cmd_A = off	cmd_A = off	fail
On_T, Off_A	cmd_A = on	idle	cmd_T = off	fail
Off_T, Off_A	cmd_T = on	cmd_T = on	idle	fail
Off_T, On_A	fail	fail	cmd_A = off	idle

Outline

- Review: programs on state
- Planning as goal regression (SNLP/UCPOP)
- Goal regression planning
  with causal graphs (Burton)
- Appendix: HFS planning with the
  causal graph heuristic (Fast Downward)

Causal Graph Heuristic for PDDL

- Recall: The ***Fast Forward (FF) Heuristic*** is computed over a ***Relaxed Planning Graph***.

- Likewise: The ***Causal Graph (CG) Heuristic*** is computed over a ***Causal Graph***.
  - Map PDDL to concurrent automata, and extract causal graph (called domain transition graph (DTG).

Problem Reformulation

Original Representation:
STRIPS or PDDL

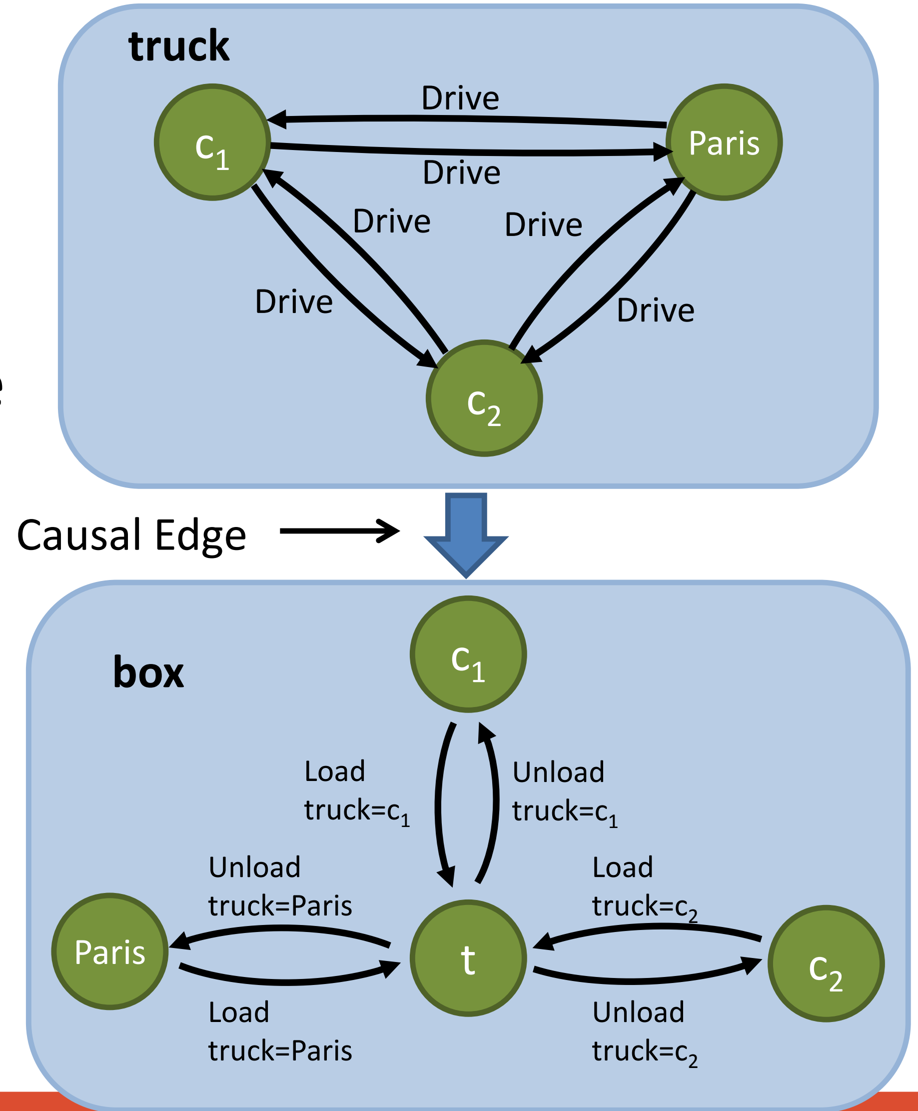
- Init
  - TruckIn(t, c_2)
  - BoxIn(b,c_1)
- Goal
  - BoxIn(b, Paris)
- Operators, e.g.
  - Drive(t, $c_{1,}$ c_2)
    - Pre: TruckIn(c_1)
    - Add: TruckIn(c_2)
    - Del: TruckIn(c_1)

New Representation:
Multi-valued Planning Task

- Variables:
  - truck := {$c_{1,}$ $c_{2,}$ Paris}
  - box := {onTruck, $c_{1,}$ $c_{2,}$ Paris}
- Init
  - truck = c_2
  - box = c_1
- Goal
  - box = Paris
- Operators, e.g.
  - Drive(t, $c_{1,}$ c_2)
    - Pre: truck = c_1
    - Post: truck = c_2

Domain Transition Graphs (DTG)

- One DTG per variable.
- Edges represent possible transitions (actions) and are guarded by preconditions
- A causal edge between the DTG represents that the "box" DTG has preconditions that depend on the "truck" DTG.



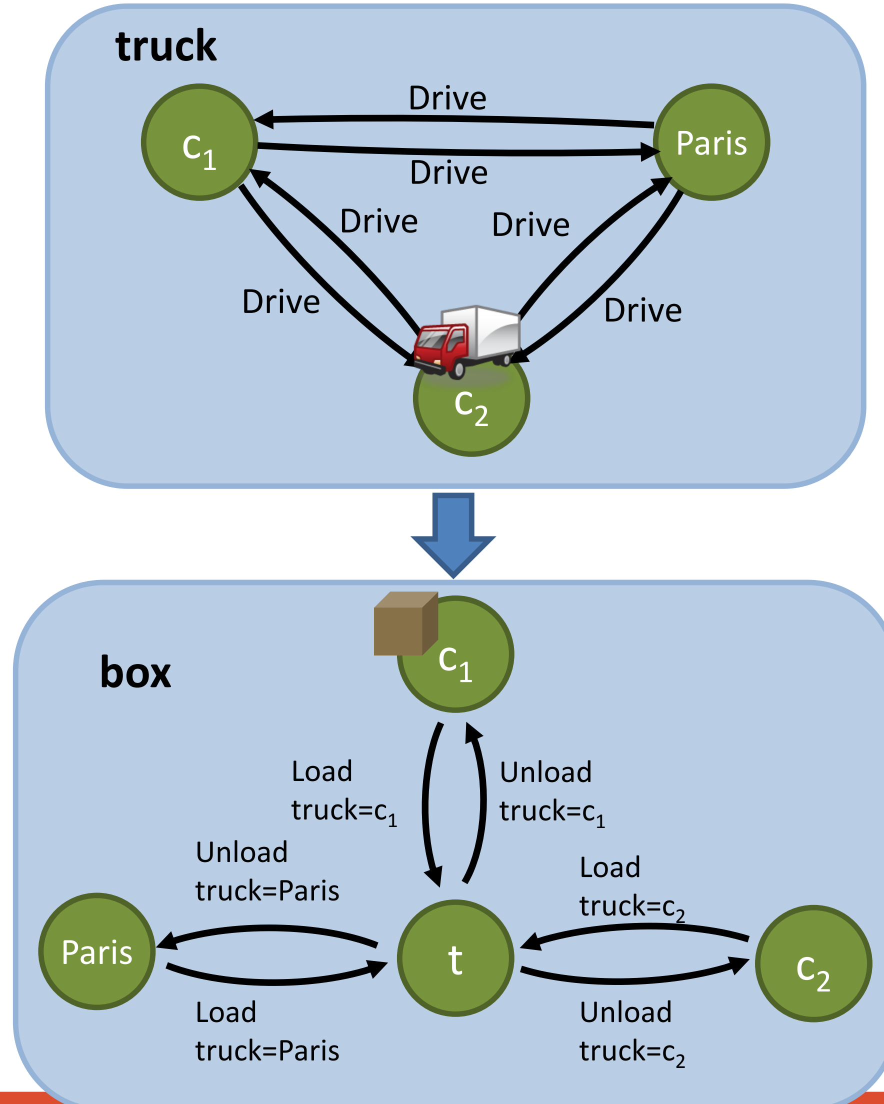
Calculating the Heuristic

Initial:

BoxIn(b, c_1)
TruckIn(t, c_2)

Goal:

BoxIn(b, Paris)



Calculating the Heuristic

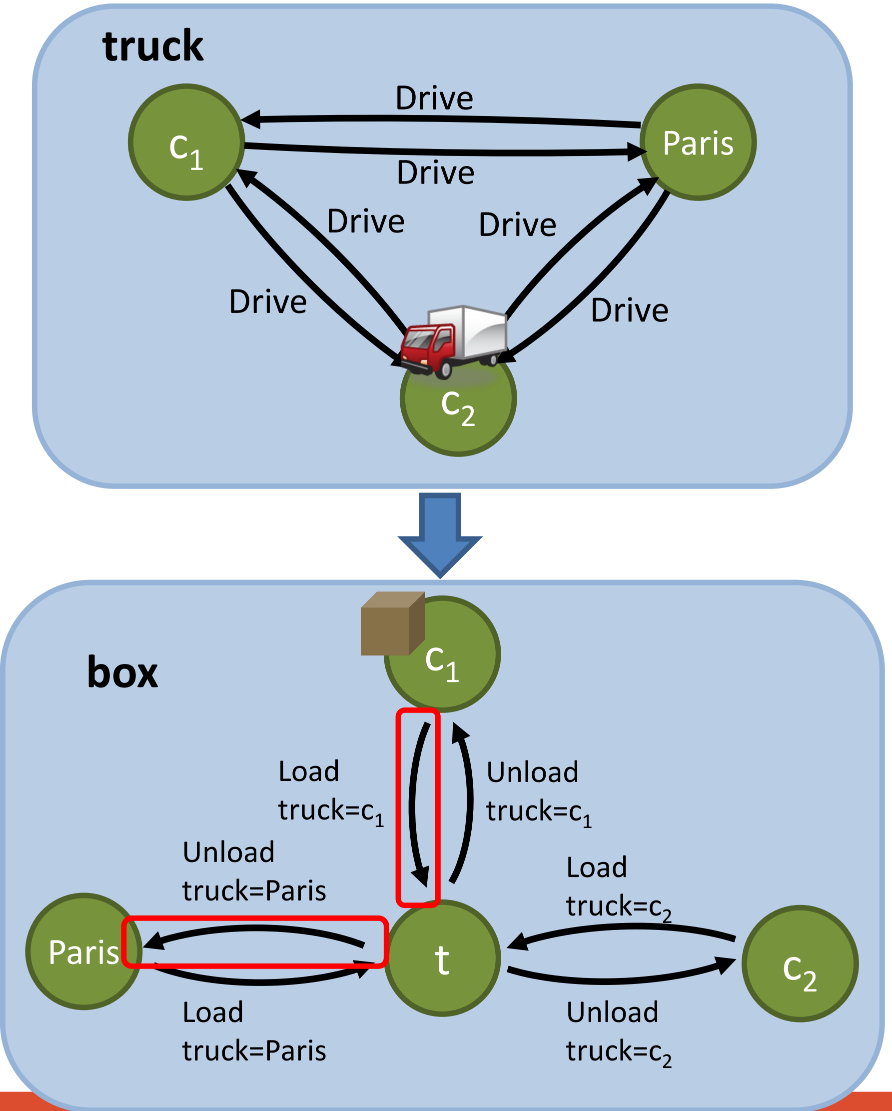
Initial:

BoxIn(b, c_1)
TruckIn(t, c_2)

Goal:

BoxIn(b, Paris)

# of transitions to get the box to Paris: 2



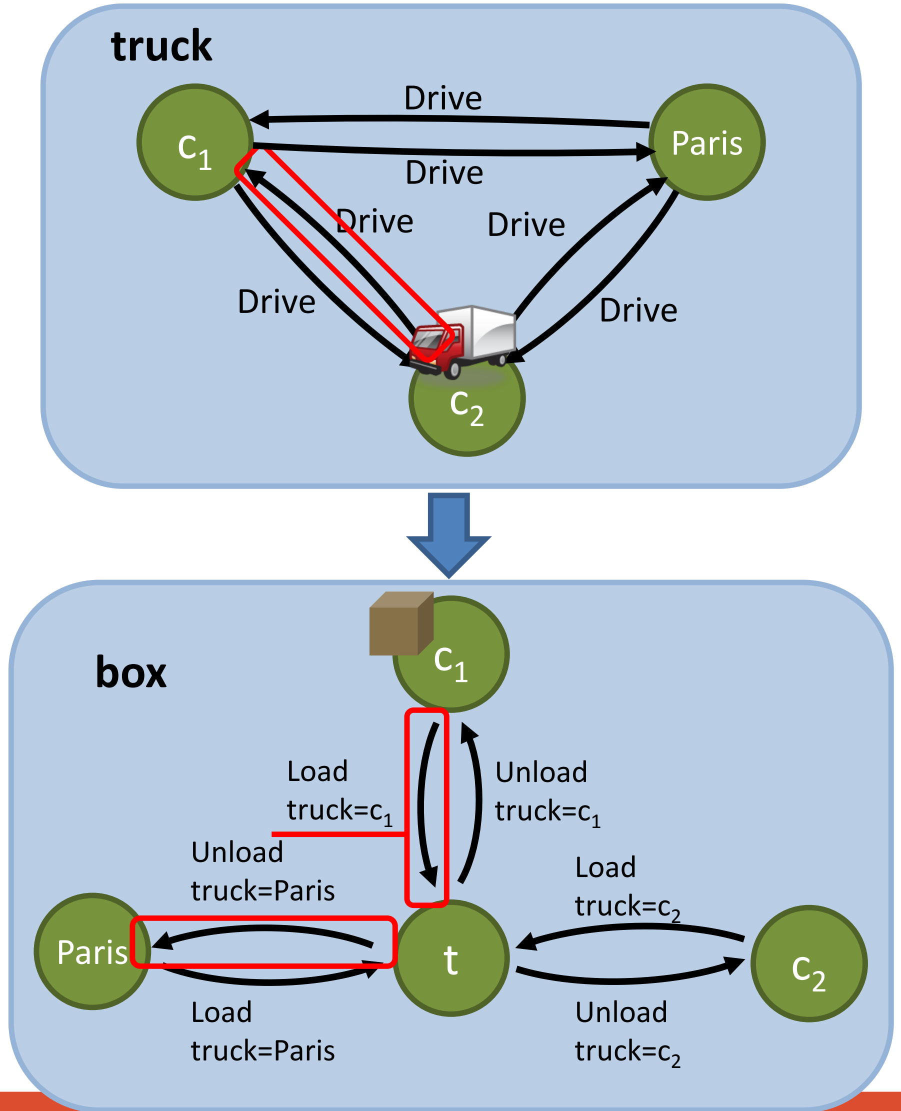
Calculating the Heuristic

Initial:

BoxIn(b, c_1)
TruckIn(t, c_2)

Goal:

BoxIn(b, Paris)



# of transitions to get the box to Paris: 2

# of transitions to get the truck to c_1: 1

Calculating the Heuristic

Initial:

BoxIn(b, c_1)
TruckIn(t, c_2)

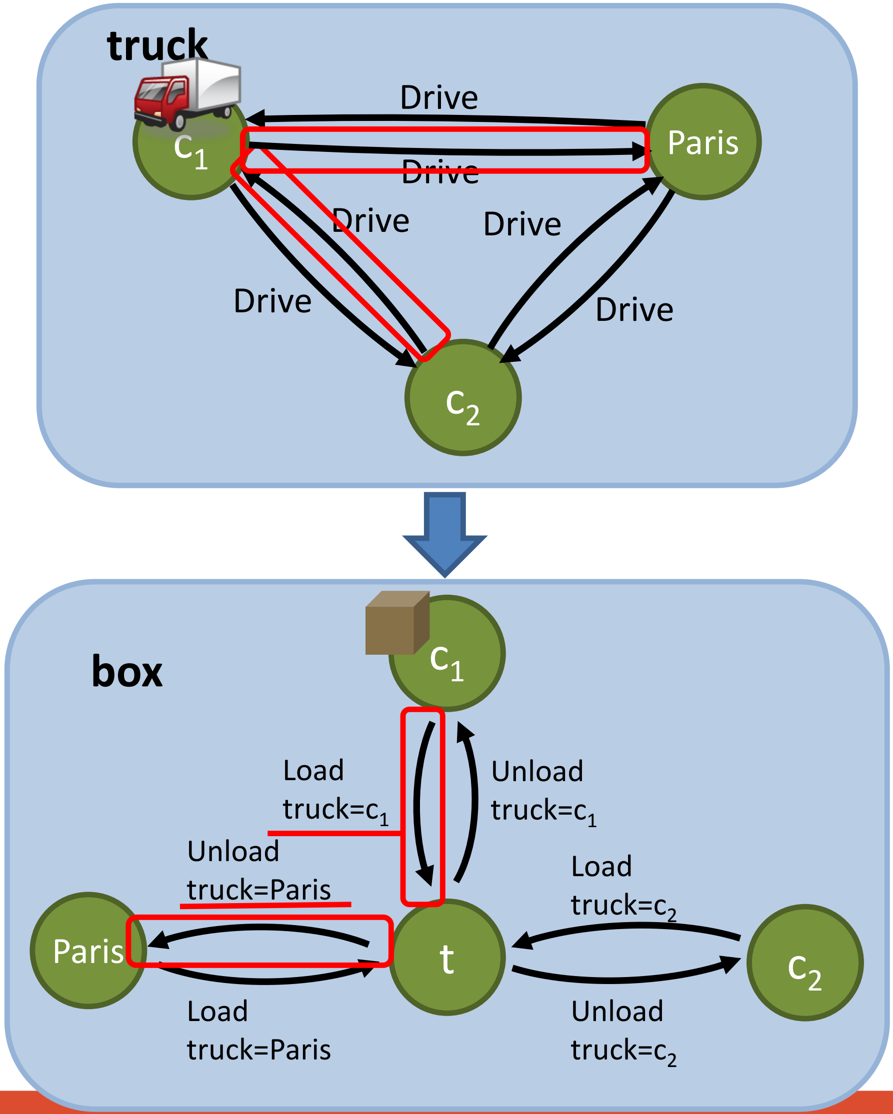
Goal:

BoxIn(b, Paris)

# of transitions to get the box to Paris: 2

# of transitions to get the truck to c_1: 1

# of transitions to get the truck to Paris: 1



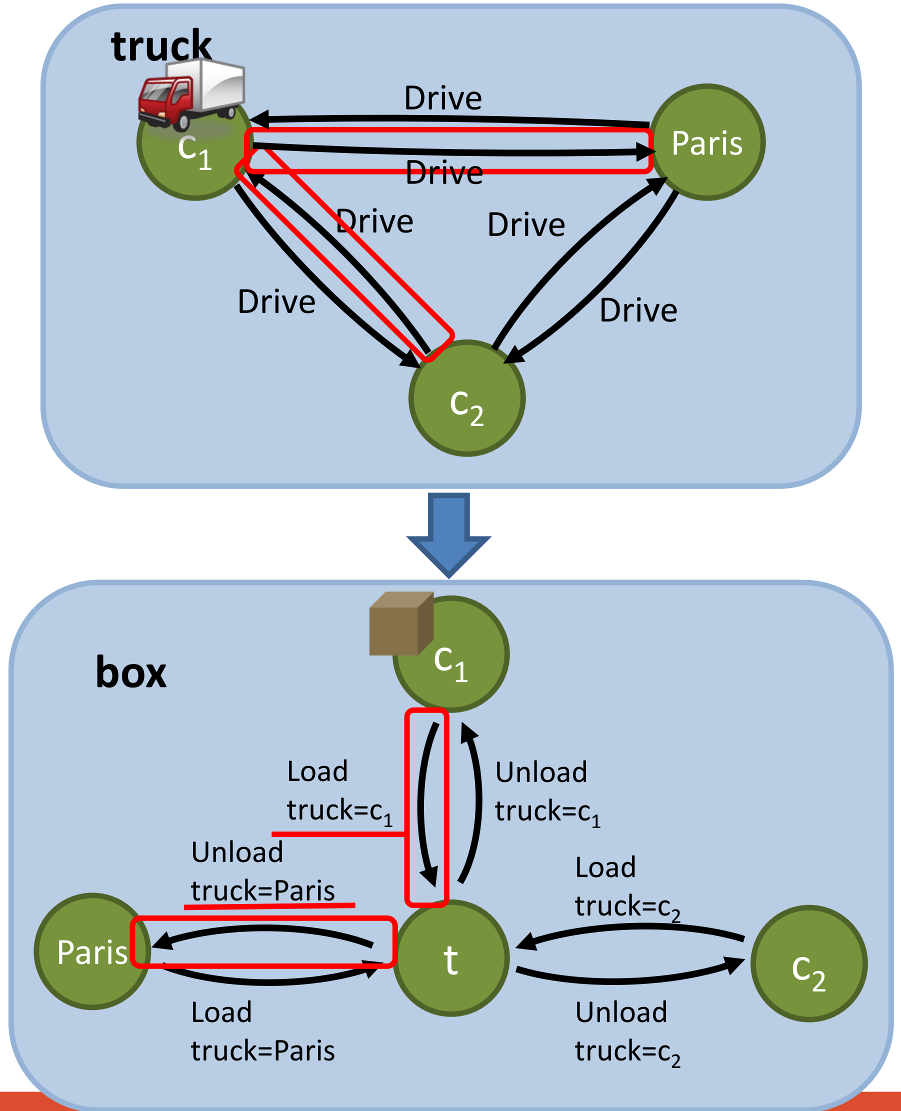
Calculating the Heuristic

Initial:

BoxIn(b, c_1)
TruckIn(t, c_2)

Goal:

BoxIn(b, Paris)



# of transitions to get the box to Paris: 2

# of transitions to get the truck to c_1: 1

# of transitions to get the truck to Paris: 1

Sum the transitions to get the heuristic value for the initial state…

$$h_{CG} = 4$$

Causal Graph Heuristic

$$h_{CG}(s) = \sum_{v \in dom(s_*)} \underbrace{\text{cost}_v(s(v), s_*(v))}_{\text{The cost of getting from the current state, s, to the goal state, s*}}$$

Idea: Sum the domain transition costs to get to the goal.

1. Identify each variable involved in the goal.
2. Recurse from child to parent in the causal graph.
   – For each variable, sum the cost of along the shortest path to change the current value to the goal value.
   – If changing that variable's value has preconditions, also add the cost of changing its parent variable.

Causal Graph Heuristic Notes

- Can not handle cyclic causal graphs
  - Relax some links until the graph is acyclic
- Calculation performed differently in practice
  - Modified Dijkstra algorithm
- Each cost calculation can over estimate.
  - Not admissible
  - Assumes no helpful interactions between sub-goals

Breaking Cycles

- Break action with multiple effects into multiple unary effect actions
- If there are still cycles, ignore some preconditions of actions

<u>Pickup(d1, cream, w1,5)</u>

Pre: d1_location=w1, w1_cream=5

Eff: w1_cream=4, d1_payload=cream

<u>Pickup1(d1, cream, w1,5)</u>

Pre: d1_location=w1, w1_cream=5

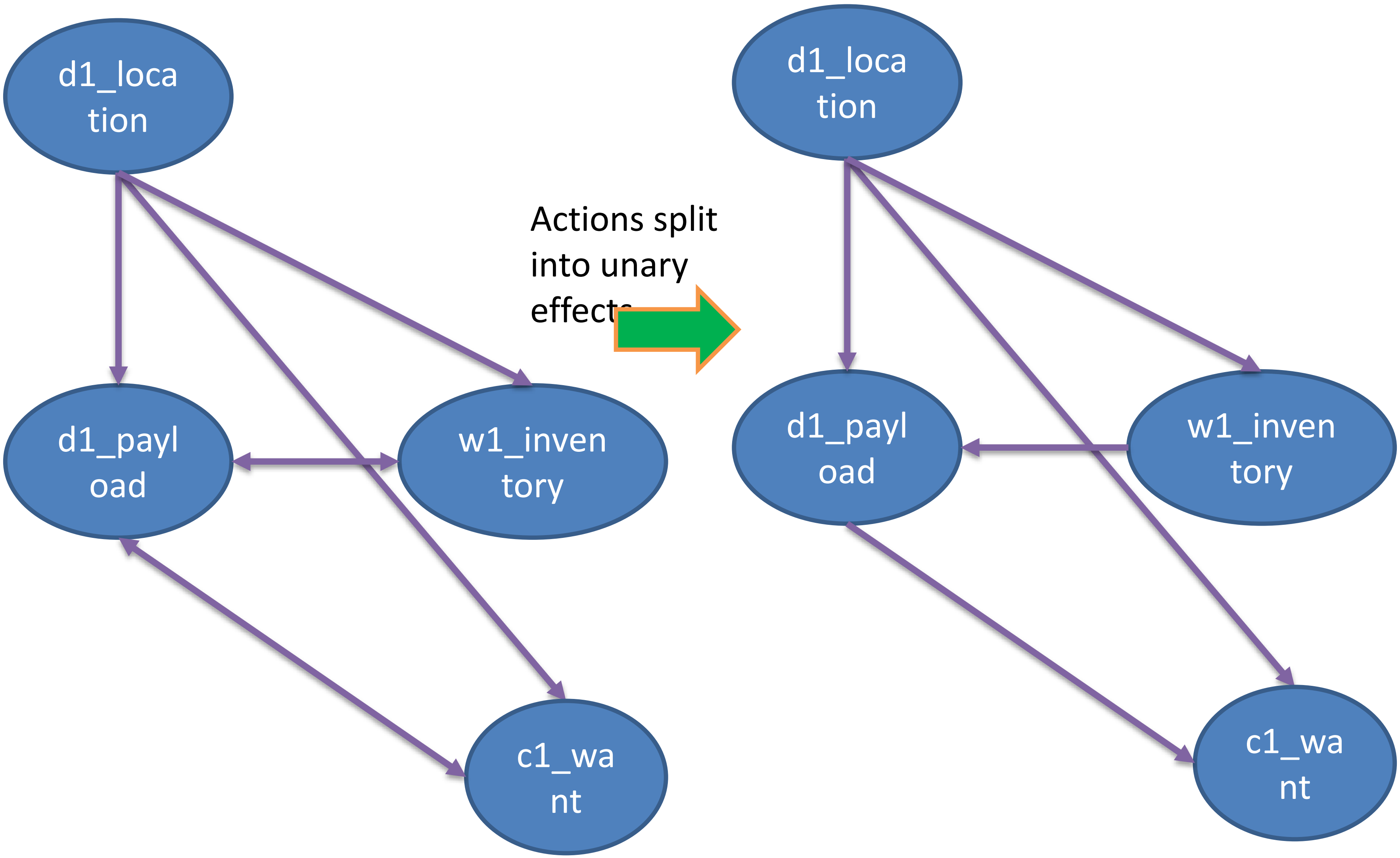
Eff: d1_payload=cream

<u>Pickup2(d1, cream, w1,5)</u>

Pre: d1_location=w1, w1_cream=5

Eff: w1_cream=4

Breaking Cycles



Causal Graph Heuristic Summary

- Graph capture dependence between variables
- Requires re-achieving conditions
- Multi-valued formulation reveals structure
- h_{CEA} heuristic extends to cyclic graphs
- Performs extremely well on classical problems

Heuristic Families

There are many other heuristics, we've only covered two…

Type	Relaxation	Heuristics	Estimates
Relaxed planning graph	Ignore deletes	h_{max}, h_{add}, $\boxed{h_{ff}}$, h_{cs}, h_m, ADHG	h+ (cost of relaxed plan)
Causal Graph	Limited interactions	$\boxed{h^{CG}}$, h^{cea}	h+ (cost of relaxed plan)
Projection Abstractions	Projection	h_{PDB}, h_{STAN}, h_{FORK}	h (cost of optimal)

Key Ideas

- Heuristic forward search is one of the fastest current planning techniques

- Domain independent heuristic design still problematic

- Fast Forward Heuristic
  - Solve a relaxed problem in a Planning Graph

- Causal Graph Heuristic
  - Consider problem's structure in a Causal Graph

MIT OpenCourseWare
https://ocw.mit.edu

16.412J / 6.834J Cognitive Robotics
Spring 2016

For information about citing these materials or our Terms of Use, visit: https://ocw.mit.edu/terms.