Planning

Fundamentals of Artificial Intelligence

Slides are taken from AIMA,

Manuela Veloso (CMU)

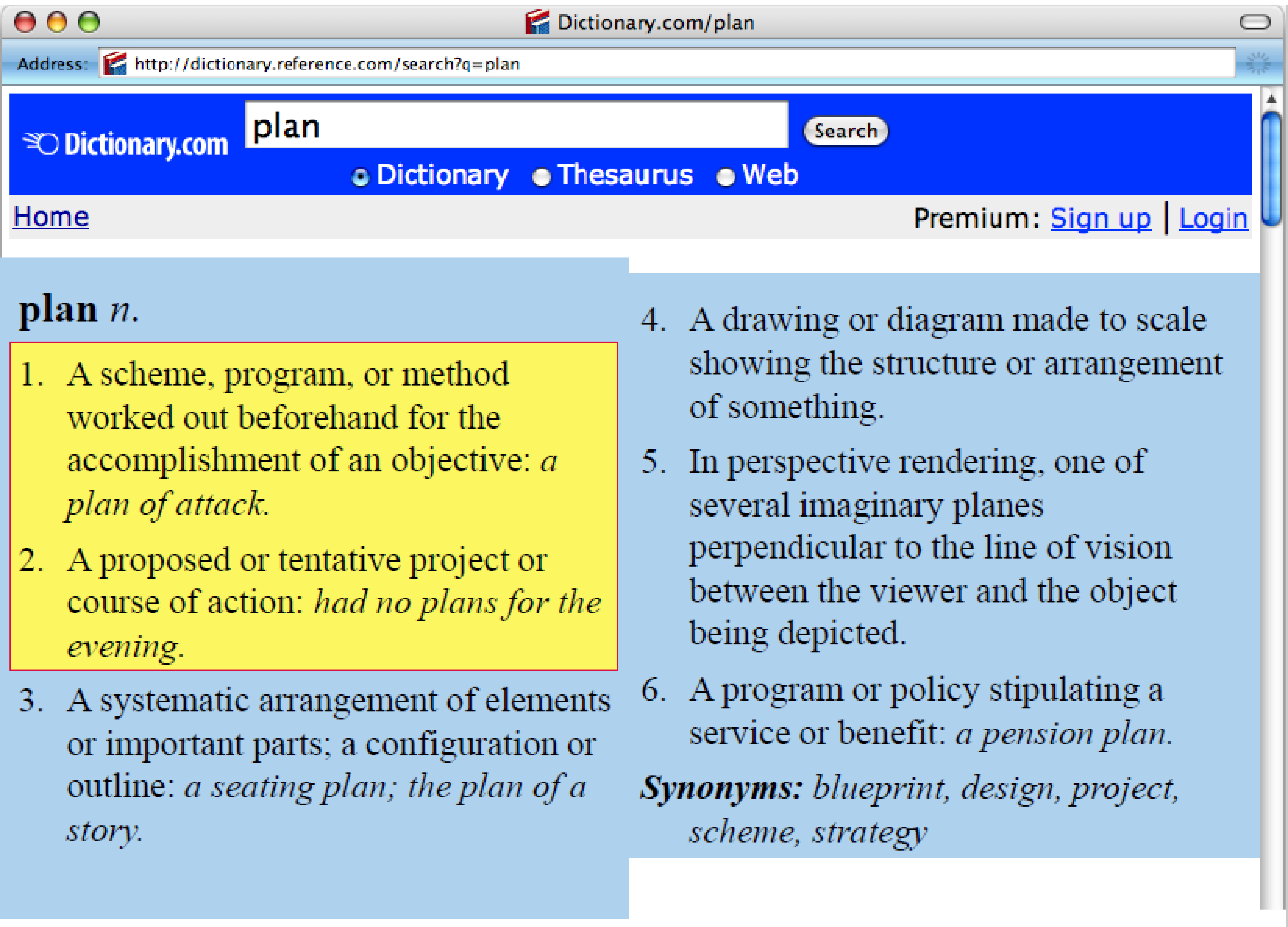
Sheila McIlraith, (University of Toronto)

Michael Scherger (Kent State University)

Rina Dechter (University of Irvine)

Berthe Y. Choueiry (University of Nebraska-Lincoln)

Bill Sverdlik (Uppsala University)


Dictionary.com/plan
Address: http://dictionary.reference.com/search?q=plan
Dictionary.com
plan
Search
Dictionary
Thesaurus
Web
Home
Premium: Sign up | Login
plan n.
1. A scheme, program, or method worked out beforehand for the accomplishment of an objective: a plan of attack.
2. A proposed or tentative project or course of action: had no plans for the evening.
3. A systematic arrangement of elements or important parts; a configuration or outline: a seating plan; the plan of a story.
4. A drawing or diagram made to scale showing the structure or arrangement of something.
5. In perspective rendering, one of several imaginary planes perpendicular to the line of vision between the viewer and the object being depicted.
6. A program or policy stipulating a service or benefit: a pension plan.
Synonyms: blueprint, design, project, scheme, strategy

[a representation] of future behavior … usually a set of actions, with temporal and other constraints on them, for execution by some agent or agents.

- Austin Tate

[*MIT Encyclopedia of the Cognitive Sciences*, 1999]

```
                              03   Establish datum point a
                                   Install 0.15-diameter s
                                   Rough side-mill pocket
                                   length 0.40, width 0.30
                                   Finish side-mill pocket
                                   length 0.40, width 0.30
                                   Rough side-mill pocket
                                   length 0.40, width 0.30
                                   Finish side-mill pocket
                                   length 0.40, width 0.30
                                   Install 0.08-diameter e
004 T  VMC1  2.50    4.87     01   Total time on VMC1

005 A   EC1  0.00   32.29     01   Pre-clean board (scrub
                              02   Dry board in oven at 85
005 B   EC1 30.00    0.48     01   Setup
                              02   Spread photoresist from
005 C   EC1 30.00    2.00     01   Setup
                              02   Photolithography of pho
                                   using phototool in "rea
005 D   EC1 30.00   20.00     01   Setup
                              02   Etching of copper
005 T   EC1 90.00   54.77     01   Total time on EC1

006 A   MC1 30.00    4.57     01   Setup
                              02   Prepare board for solde
006 B                                         solder stop
006 C   MC1 30.00    7.50     01   Setup
```

A portion of a manufacturing process plan

Human Planning and Acting

- acting without (explicit) planning:
  - when purpose is immediate e.g. turning on computer to start lecture
  - when performing well-trained behaviours e.g. biking, driving
  - when course of action can be freely adapted e.g. supermarket shop
- acting after planning:
  - when addressing a new situation e.g. moving a house
  - when tasks are complex e.g. preparing a course
  - when the environment imposes high risk/cost e.g. nuclear power
  - when collaborating with others e.g. coordination forbuilding a house
- people plan only when strictly necessary

Costly

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Defining AI Planning

- <u>planning</u>:
  - explicit deliberation process that chooses and organizes actions by anticipating their outcomes
  - aims at achieving some pre-stated objectives
- <u>AI planning</u>:
  - computational study of this deliberation process

We consciously think about planning to choose among different option. What will the world be like?

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Why Study Planning in AI?

- scientific goal of AI:
  understand intelligence
  - planning is an important component of rational (intelligent) behaviour

- engineering goal of AI:
  build intelligent entities
  - build planning software for choosing and organizing actions for autonomous intelligent machines

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh,
https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Domain-Specific vs. Domain-Independent Planning

- domain-specific planning: use specific representations and techniques adapted to each problem
  - important domains: path and motion planning, perception planning, manipulation planning, communication planning
- domain-independent planning: use generic representations and techniques
  - exploit commonalities to all forms of planning
  - leads to general understanding of planning
- domain-independent planning complements domain-specific planning

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Toy Problems vs. Real-World Problems

Toy Problems/Puzzles

- concise and exact description
- used for illustration purposes (e.g. here)
- used for performance comparisons

Real-World Problems

- no single, agreed-upon description
- people care about the solutions

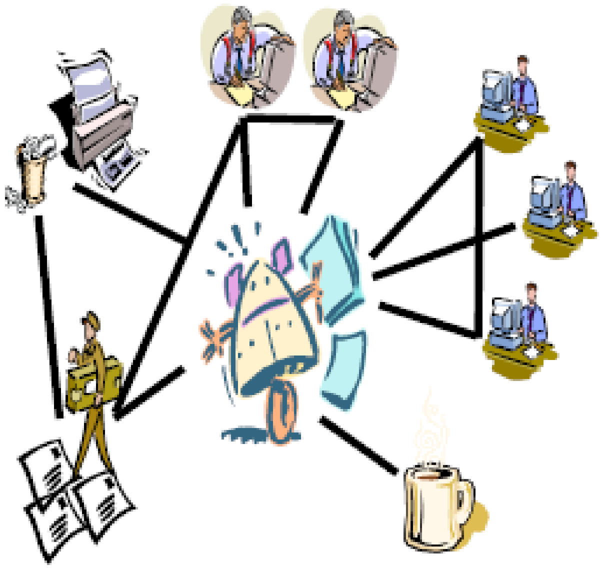
Why Planning

- Intelligent agents must operate in the world. They are not simply passive reasoners (Knowledge Representation, reasoning under uncertainty) or problem solvers (Search), they must also act on the world.

- We want intelligent agents to act in "intelligent ways". Taking purposeful actions, predicting the expected effect of such actions, composing actions together to achieve complex goals.

A Planning Problem

- How to *change* the world to suit our needs
- Critical issue: we need to reason about ***what the world will be like*** after doing a few actions, not *just* what it is like now



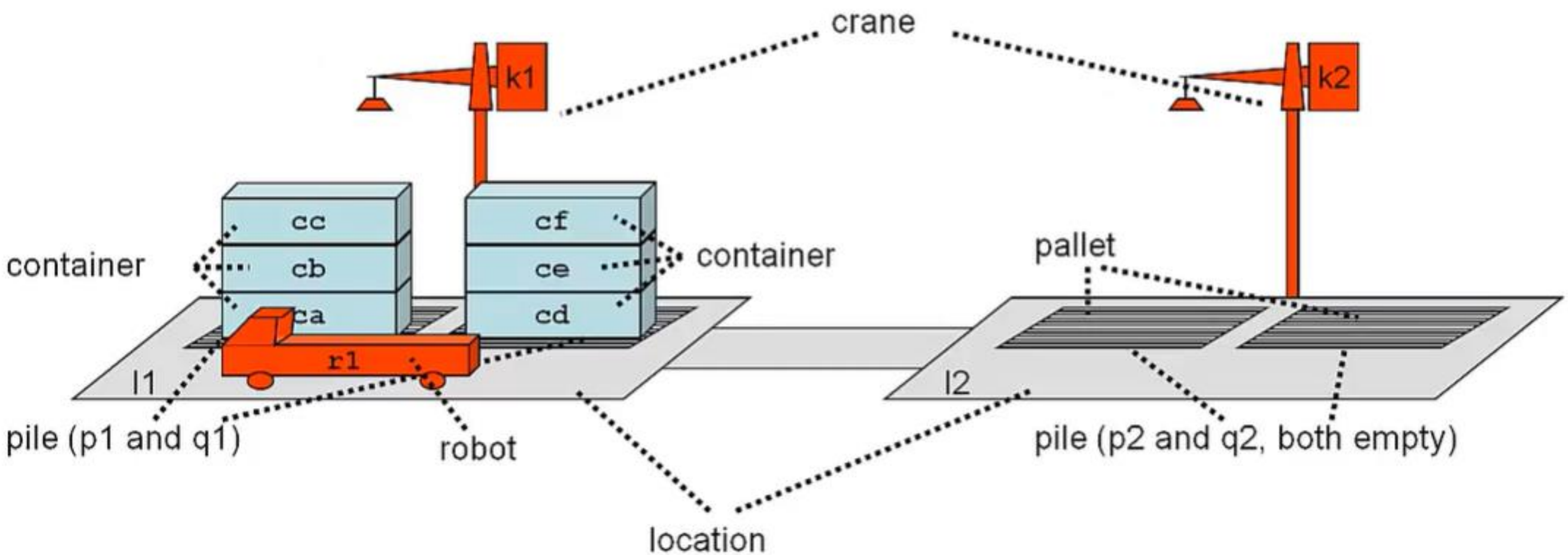
GOAL: Steven has coffee
CURRENTLY: robot in mailroom, has no coffee, coffee not made, Steven in office, etc.
TO DO: goto lounge, make coffee,...

The Dock-Worker Robots (DWR) Domain

- aim: have one example to illustrate planning procedures and techniques
- informal description:
  - harbour with several locations (docks), docked ships, storage areas for containers, and parking areas for trucks and trains
  - cranes to load and unload ships etc., and robot carts to move containers around

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

DWR Example State



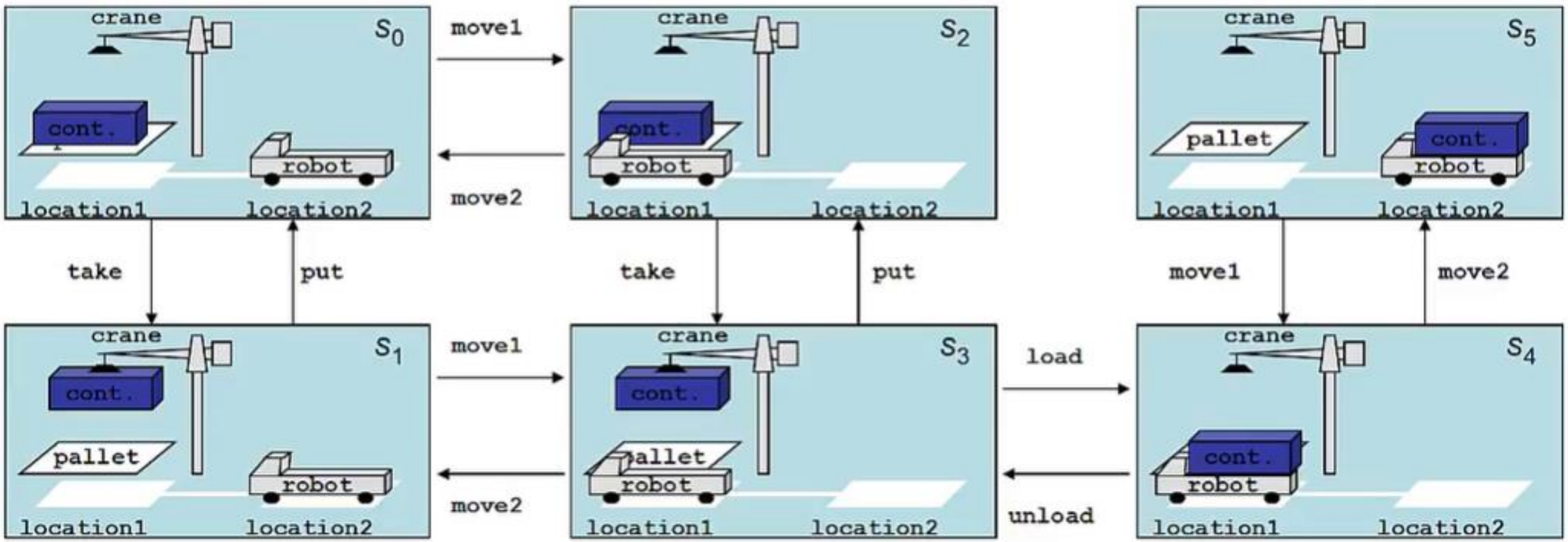
Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Actions in the DWR Domain

- **`move`** robot *r* from location *l* to some adjacent and unoccupied location *l'*
- **`take`** container *c* with empty crane *k* from the top of pile *p*, all located at the same location *l*
- **`put`** down container *c* held by crane *k* on top of pile *p*, all located at location *l*
- **`load`** container *c* held by crane *k* onto unloaded robot *r*, all located at location *l*
- **`unload`** container *c* with empty crane *k* from loaded robot *r*, all located at location *l*

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

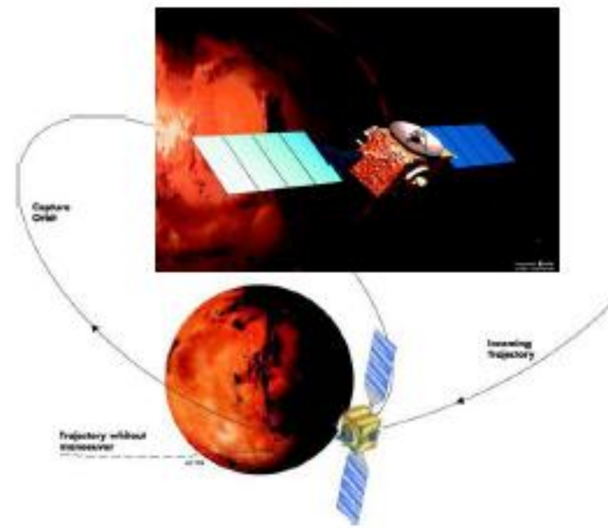
State-Transition Systems: Graph Example



Slide Credit: Artificial Intelligence Planning, The University of Edinburgh,
https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Autonomous Agents for Space Exploration

- Autonomous planning, scheduling, control
  - NASA: JPL and Ames
- Remote Agent Experiment (RAX)
  - Deep Space 1
- Mars Exploration Rover (MER)



Other Autonomous Systems

Manufacturing Automation

- Sheet-metal bending machines - Amada Corp
  - Software to plan the sequence of bends
    [Gupta and Bourne, *J. Manufacturing Sci. and Engr.*, 1999]

Other Applications (cont.)

Scheduling with Action Choices & Resource Requirements

- Problems in supply chain management
- HSTS (Hubble Space Telescope scheduler)
- Workflow management

Air Traffic Control

- Route aircraft between runways and terminals. Crafts must be kept safely separated. Safe distance depends on craft and mode of transport. Minimize taxi and wait time.

Character Animation

- Generate step-by-step character behaviour from high-level spec

Plan-based Interfaces

- E.g. NLP to database interfaces
- Plan recognition, Activity Recognition

Other Applications (cont.)

Web Service Composition

- Compose web services, and monitor their execution
- Many of the web standards have a lot of connections to plan representation languages
  - BPEL; BPEL-4WS allow workflow specifications
  - OWL-S allows process specifications

Grid Services/Scientific Workflow Management

Genome Rearrangement

- The relationship between different organisms can be measured by the number of “evolution events” (rearrangements) that separate their genomes
- Find shortest (or most likely) sequence of rearrangements between a pair of genomes

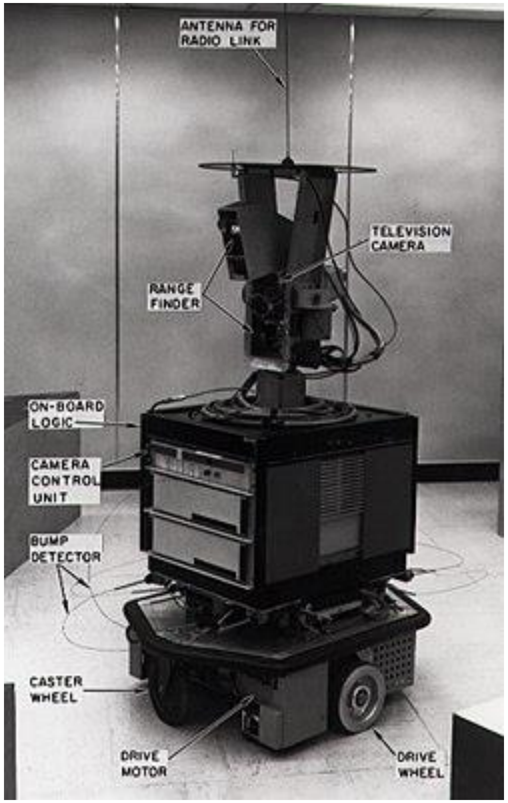
Practical AI Planners

Planner	*Reference*	*Applications*
STRIPS	Fikes & Nilsson 1971	Mobile Robot Control, etc.
HACKER	Sussman 1973	Simple Program Generation
NOAH	Sacerdoti 1977	Mechanical Engineers Apprentice Supervision
NONLIN	Tate 1977	Electricity Turbine Overhaul, etc.
NASL	McDermott 1978	Electronic Circuit Design
OPM	Hayes-Roth & Hayes-Roth 1979	Journey Planning
ISIS-II	Fox et. al. 1981	Job Shop Scheduling (Turbine Production)
MOLGEN	Stefik 1981	Experiment Planning in Molecular Genetics
DEVISER	Vere 1983	Spacecraft Mission Planning
FORBIN	Miller et al. 1985	Factory Control
SIPE/SIPE-2	Wilkins 1988	Crisis Action Planning, Oil Spill Management, etc.
SHOP/SHOP-2	Nau et al. 1999	Evacuation Planning, Forest Fires, Bridge Baron, etc.
I-X/I-Plan	Tate et al. 2000	Emergency Response, etc.

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Shakey made use of STRIPS

- **Shakey the Robot** was the first general-purpose mobile robot to be able to reason about its own actions.
- Shakey was developed at the Artificial Intelligence Center of Stanford Research Institute (now called SRI



https://www.youtube.com/watch?v=7bsEN8mwUB8

https://www.youtube.com/watch?v=GmU7SimFkpU

https://en.wikipedia.org/wiki/Shakey_the_robot https://media.ed.ac.uk/media/Artificial+Intelligence+Planning+-+Nils+Nilsson+-+A-Star+and+STRIPS/1_uhxvxo4a

Planning is Hard

- To be precise it's PSPACE hard.
- More intuitively, here's a problem:
  - Planning domain called logistics;
    - Actions called drive, load, unload…
  - How would you solve the problems?
    - Drive trucks around;
    - load packages in;
    - drive to package goal locations;
    - unload packages.

  –Easy? Yes, but you used a lot of intuition.

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Here's Another One

Initial: (cabbage monkey), (tasty pancake)

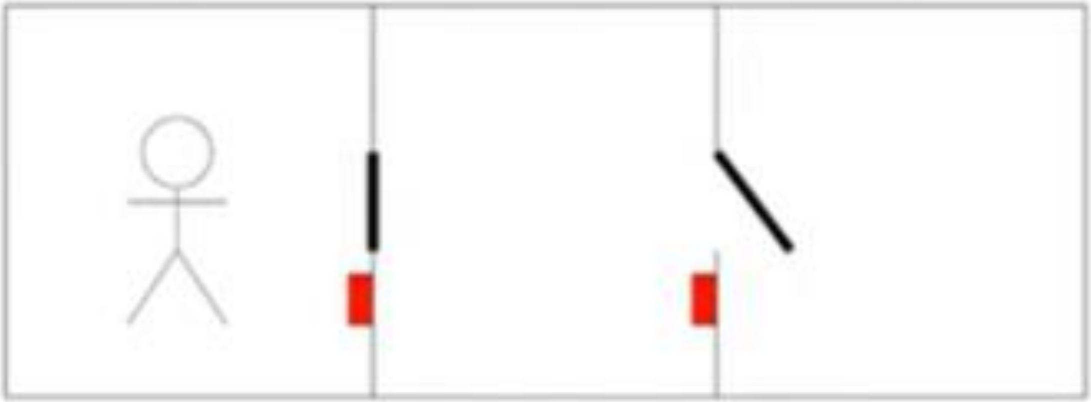
Goal: (jam doughnut)

Name	Preconditions	Effects
Liz	(alsatian kebab) (tasty pancake)	(not (alsatian kebab)) (jam doughnut)
Amanda	(tasty pancake)	(not (tasty pancake)) (alfresco dining)
Derek	(alfresco dining)	(not (alfresco dining)) (tasty pancake)
Andrew	(cabbage monkey) (alfresco dining)	(not (cabbage monkey)) (alsatian kebab)

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Back to the Easier Problems…

Pressing a red button opens one door and shuts the other



- How might one encode this in PDDL? Initial state: right door is open; standing in 1st room. Goal: standing in third room.
  - –Right door open (tasty pancake)
  - –In First room (cabbage monkey)
  - –Goal of standing in right room (jam doughnut)
  - –Actions for buttons: Amanda and Derek
  - –To move, Liz and Andrew

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Search Problems

- <u>initial state</u>
- set of possible <u>actions</u>/applicability conditions
  - successor function: *state* $\rightarrow$ set of <*action*, *state*>
  - successor function + initial state = state space
  - path (solution)
- <u>goal</u>
  - goal state or goal test function
- <u>path cost</u> function
  - for optimality
  - assumption: path cost = sum of step costs

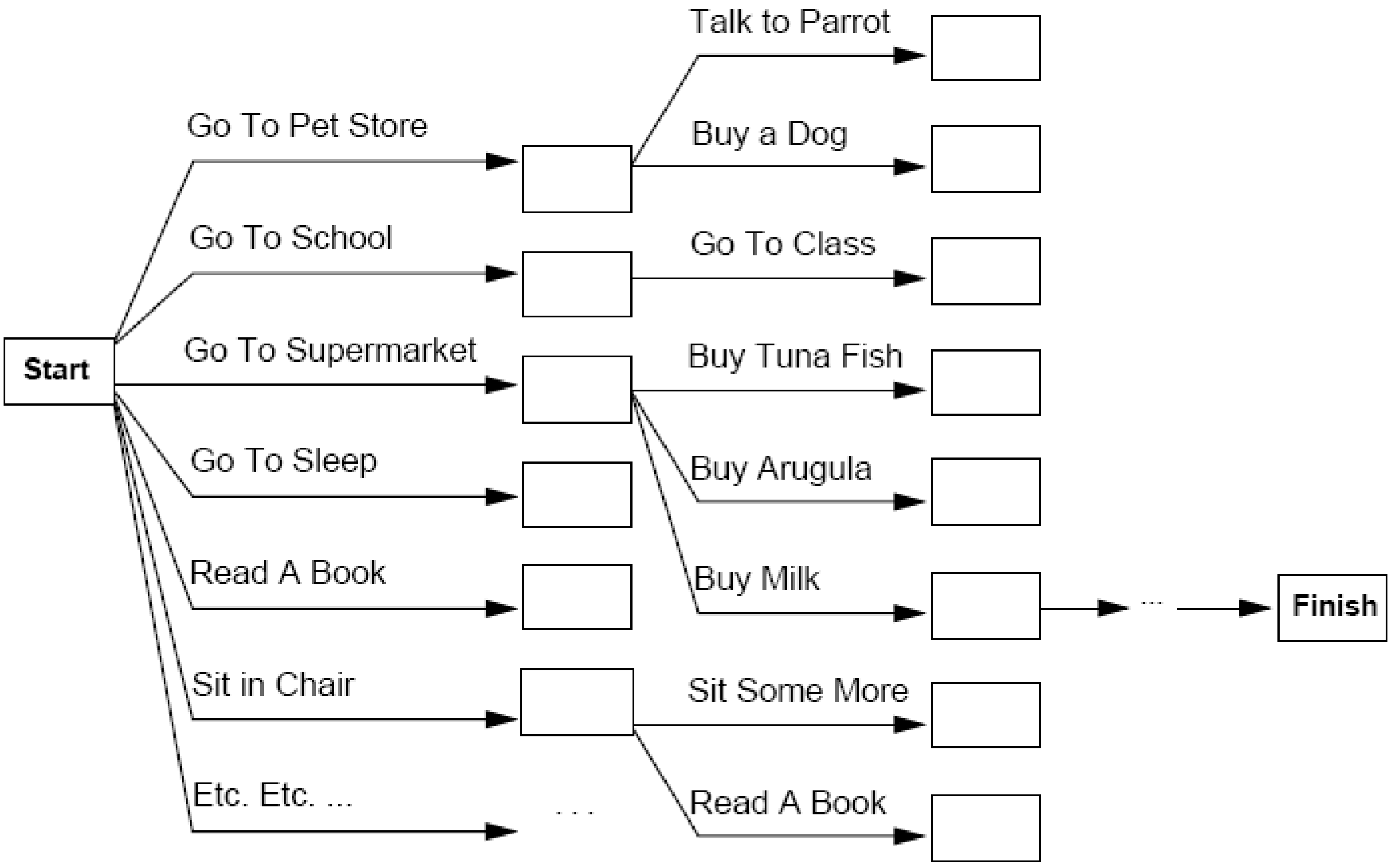
Slide Credit: Artificial Intelligence Planning, The University of Edinburgh,
https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Search vs Planning

Consider the task of getting milk, bananas, and a cordless drill

Really want to go to supermarket and then go to the hardware store

But we could get sidetracked! by irrelevant actions



Search vs. Planning

Planning Systems do the following:

Open up action and goal representation to allow selection

Divide-and-conquer by sub-goaling

Relax requirement for sequential construction of solutions

- Search
  - States: program data structures
  - Actions: program code
  - Goal: program code
  - Plan: sequence from S_0
- Planning
  - States: logical sentences
  - Actions: preconditions and outcomes
  - Goal: logical sentences (conjunction)
  - Plan: constraints on actions

Planning under Uncertainty

One of the major complexities in planning that we will deal with later is planning under uncertainty.

- Our knowledge of the world will almost certainly be incomplete. We may wish to model that probabilistic.

- Sensing is subject to noise (especially in robots).

- Actions and effectors are also subject to error (uncertainty in their effects).

Classical Planning

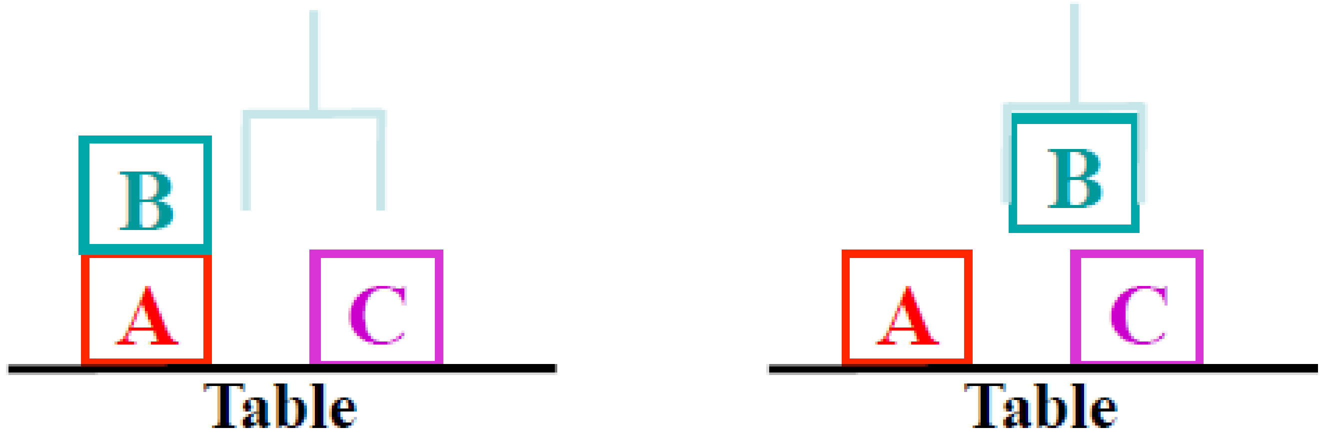
For now we restrict our attention to the deterministic case.

We will examine:

- Complete initial state specifications
- deterministic effects of actions.
- finding sequences of actions that can achieve a desired set of effects.

This will in some ways be a lot like search, but we will see that representation also plays an important role.

The Blocks World Definition

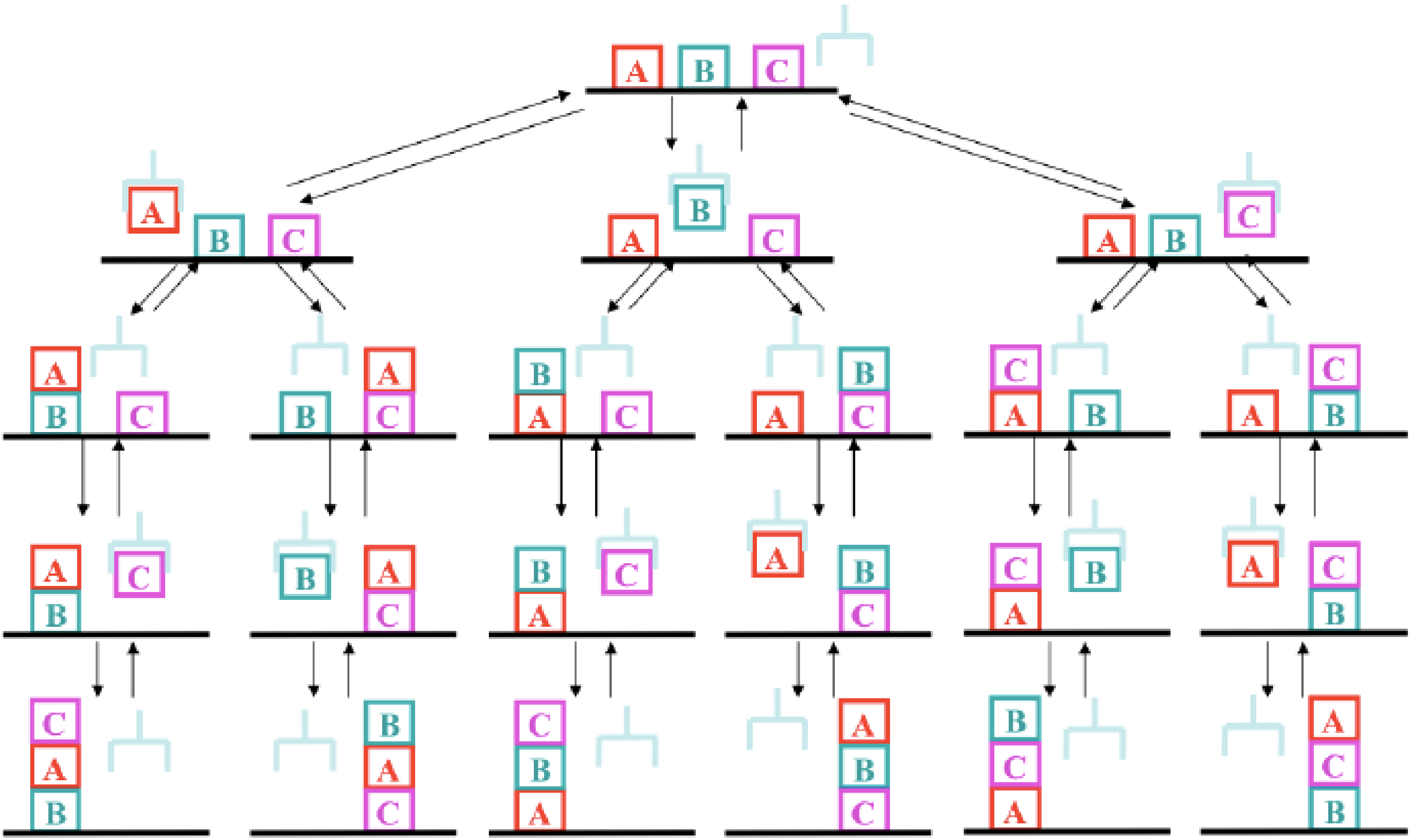


- Blocks are picked up and put down by the arm
- Blocks can be picked up only if they are clear, i.e., without any block on top
- The arm can pick up a block only if the arm is empty, i.e., if it is not holding another block, i.e., the arm can be pick up only one block at a time
- The arm can put down blocks on blocks or on the table

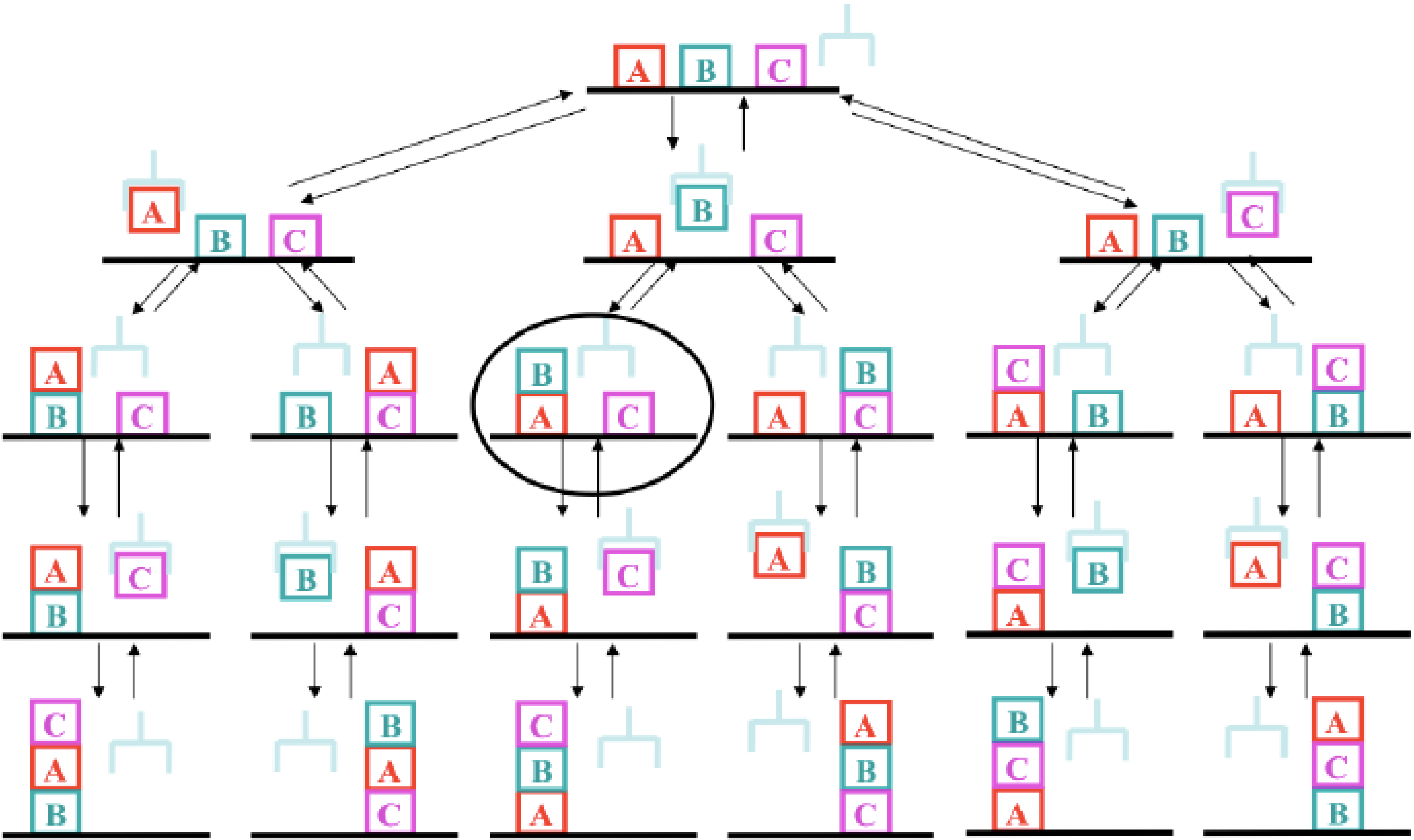
Planning by “Plain” State Search

- **Search from** an ***initial state*** of the world to a ***goal state***
- Enumerate all states of the world
- Connect states with legal actions
- Search for paths between initial and goal states

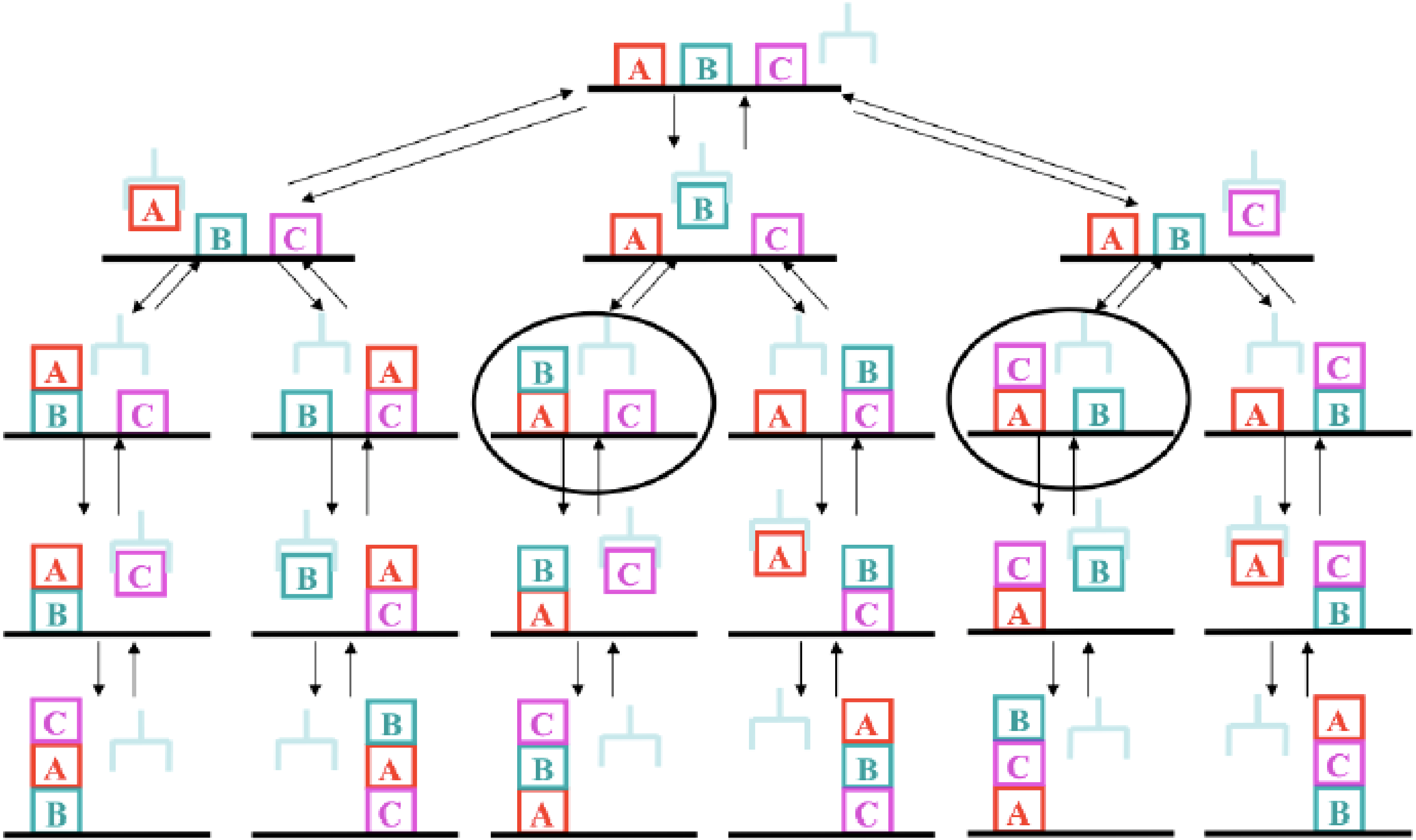
3-BlocksWorld State Transitions



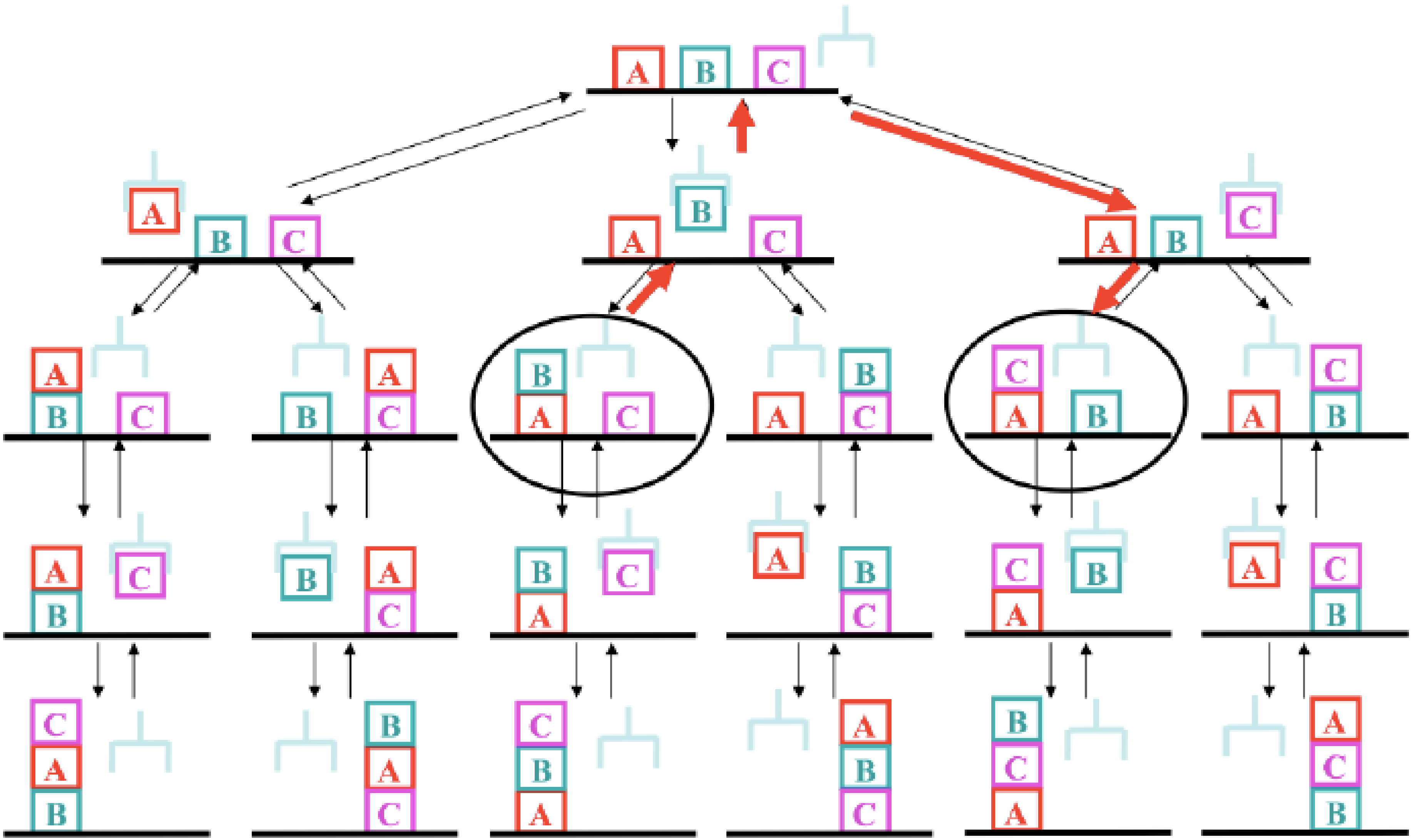
3-BlocksWorld State Transitions



3-BlocksWorld State Transitions



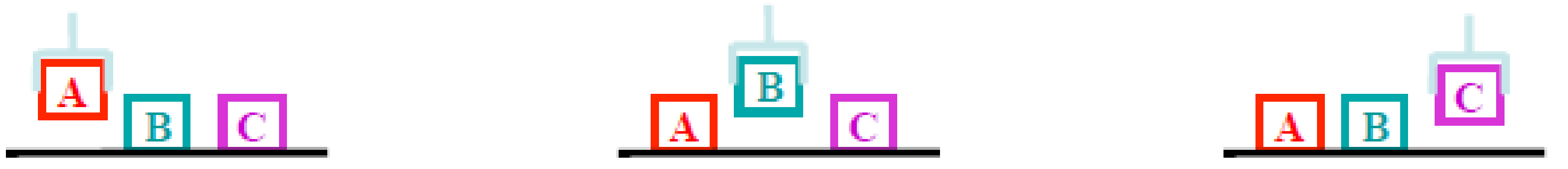
3-BlocksWorld State Transitions



Planning – Actions and States

- ***Model*** of an action
  - a description of *legal actions* in the domain
    - "move queen", "open door if unlocked", "unstack if top is clear",….
- ***Model*** of the state
  - **Numerical identification (s1, s2,...) – no information**
  - **"Symbolic" description**
    - **objects, predicates**

Planning – State Representation



on(B,table) and on(C, table) and holding(A)

...

Planning – The Problem

- In planning problems we have:
  - an initial state, I

```
(at mydvd amazon)
(at truck amazon)
(at driver home)
(path home amazon)
(link amazon london)
(link london myhouse)
```

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Planning – The Problem

- In planning problems we have:
  - an initial state, I
  - a goal state, G

```
(at mydvd myhouse)
```

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Planning – The Problem

- In planning problems we have:
  - an initial state, I
  - a goal state, G
  - some actions, A, defined according to a **domain**

```
(load mydvd truck depot)
(walk driver home amazon)
(board-truck driver truck amazon)
(drive-truck driver amazon london)
...
```

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

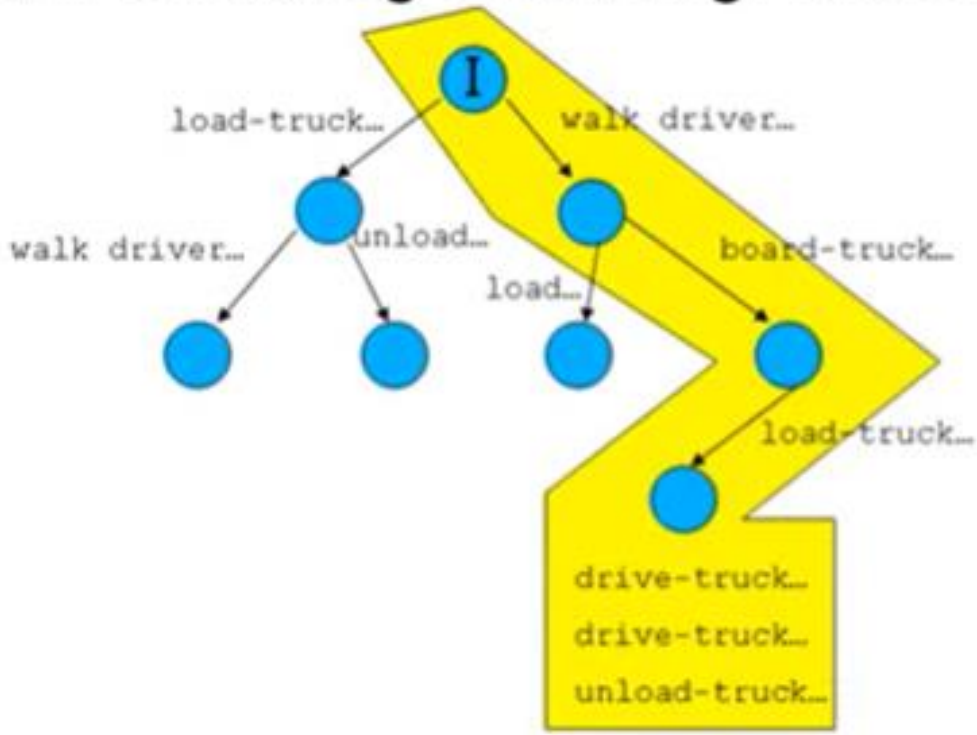
Planning – The Problem

- In planning problems we have:
  - an initial state, I
  - a goal state, G
  - some actions, A, defined according to a **domain**
    - these have **preconditions** and **effects**

```
(:action load
   (:parameters(?d - dvd ?t - truck ?dep - depot)
   (:precondition (and (at ?t ?dep) (at ?d ?dep))
   (:effect (and (not (at ?d ?dep))
                      (in ?d ?t))
    )
)
```

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Forward-Chaining Planning: Intuition



Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Planning Languages

- Languages must represent..
  - States
  - Goals
  - Actions
- Languages must be
  - Expressive for ease of representation
  - Flexible for manipulation by algorithms

Goal representation

- Goal is a <u>partially</u> specified state
  - Represented as a conjunction of ground literals
  - Examples
    - $At(\ Plane_1, LAS\ )$

Action representation

– Specified in terms of the preconditions that must hold before it can be executed and the effects that ensue when it is executed

– Action(Fly(p, from, to))
- Precond: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
- Effect: $\neg At(p, from) \wedge At(p, to)$

– This is also known as an ***action schema***

The Language of Planning Problems

- Suppose our current state is:
  - $At(P1, CLE) \wedge At(P2, LAS) \wedge Plane(P1) \wedge Plane(P2) \wedge Airport(CLE) \wedge Airport(LAS)$
- This state satisfies the precondition
  - $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
- Using the substitution
  - $\{p/P1, from/CLE, to/LAS\}$
- The following concrete action is applicable
  - $Fly(P1, CLE, LAS)$

Action Representation

- Action Schema
  - Action name
  - Preconditions
  - Effects

At(WHI,LNK),Plane(WHI),
Airport(LNK), Airport(OHA)

Fly(WHI,LNK,OHA)

At(WHI,OHA), ¬ At(WHI,LNK)

- Example

  Action(Fly(p,from,to),

  PRECOND: At(p,from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)

  EFFECT: ¬At(p,from) ∧ At(p,to))

- Sometimes, Effects are split into ADD list and DELETE list

Applying an Action

- Find a substitution list θ for the variables
  - of all the precondition literals
  - with (a subset of) the literals in the current state description
- Apply the substitution to the propositions in the effect list
- Add the result to the current state description to generate the new state
- Example:
  - Current state: At(P1,JFK) $\wedge$ At(P2,SFO) $\wedge$ Plane(P1) $\wedge$ Plane(P2) $\wedge$ Airport(JFK) $\wedge$ Airport(SFO)
  - It satisfies the precondition with θ={p/P1,from/JFK, to/SFO)
  - Thus the action Fly(P1,JFK,SFO) is applicable
  - The new current state is: At(P1,SFO) $\wedge$ At(P2,SFO) $\wedge$ Plane(P1) $\wedge$ Plane(P2) $\wedge$ Airport(JFK) $\wedge$ Airport(SFO)

Languages for Planning Problems

- STRIPS
  - <u>ST</u>anford <u>R</u>esearch <u>I</u>nstitute <u>P</u>roblem <u>S</u>olver
  - Historically important
- ADL
  - Action Description Languages
- PDDL
  - Planning Domain Definition Language
  - Revised & enhanced for the needs of the International Planning Competition

STRIPS (STanford Research Institute Problem Solver)

Tidily arranged actions descriptions, restricted language

ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p, x)$
EFFECT: $Have(x)$

[Note: this abstracts away many important details!]

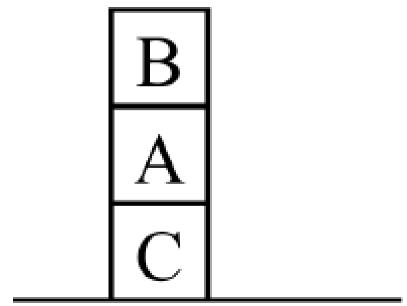
Restricted language $\Rightarrow$ efficient algorithm
Precondition: conjunction of positive literals
Effect: conjunction of literals

A complete set of STRIPS operators can be translated into a set of successor-state axioms

STRIPS: describing goals and state

```
On(B,A)
On(A,C)
On(C,Fl)
Clear(B)
Clear(Fl)
```

Factored representation of states



- On(B,A)
- On(A,C)
- On(C,F1)
- Clear(B)
- Clear(F1)
- The formula describes a set of world states
- Planning search for a formula satisfying a goal description
- **State descriptions**: conjunctions of ground literals.
- Also universal formulas: On(x,y) →(y=F1) or ~Clear(y)
- Goal wff: $\exists x.g(x) \wedge f(y)$
- Given a goal wff, the search algorithm looks for a sequence of **actions**

That transform into a state description that entails the goal wff.

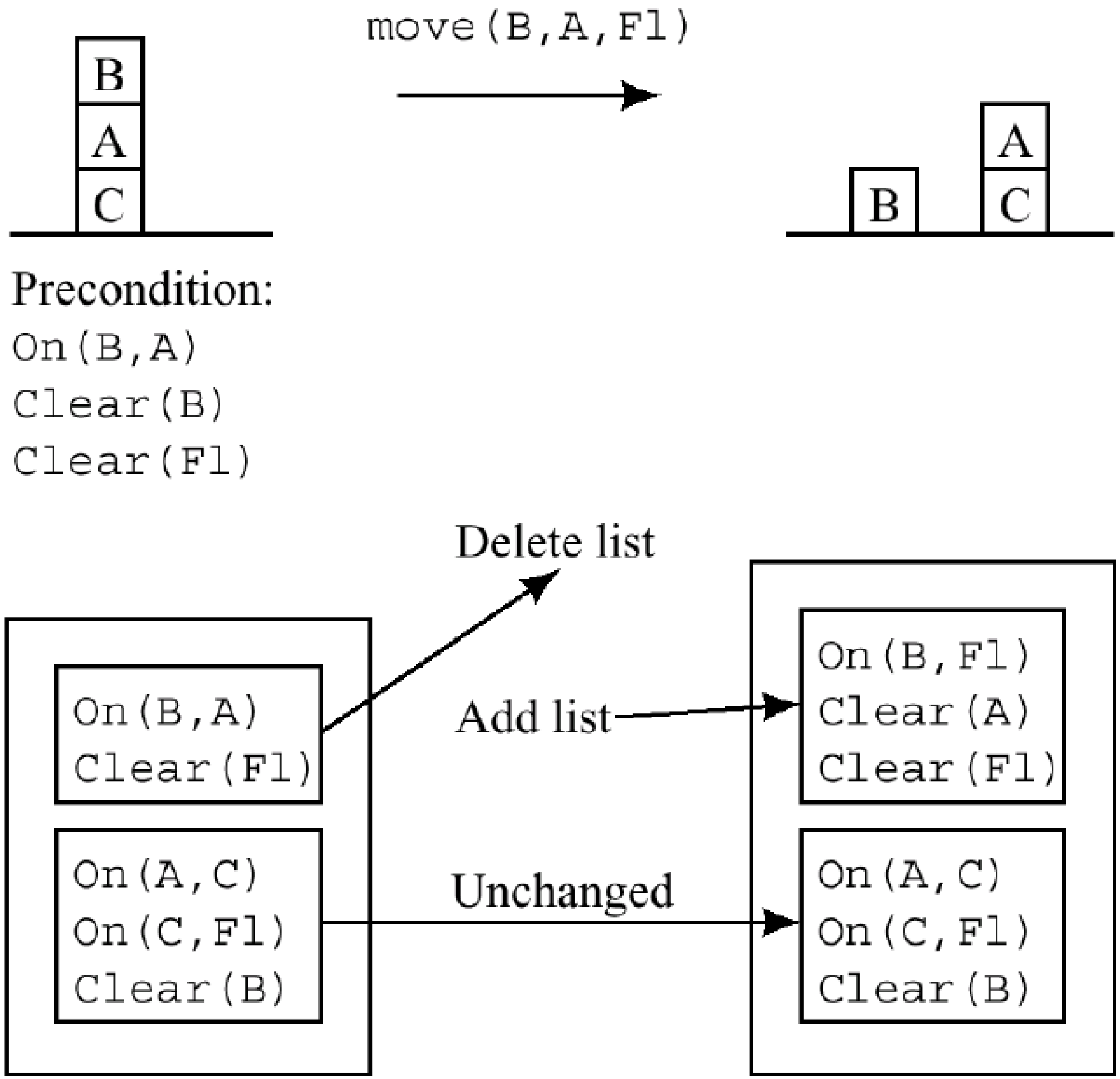
STRIPS Action Representation

- Actions - operators -- rules -- with:
  - **Precondition expression** -- must be satisfied before the operator is applied.
  - **Set of effects** -- describe how the application of the operator changes the state.
- Precondition expression: propositional, typed first order predicate logic, negation, conjunction, disjunction, existential and universal quantification, and functions.
- Effects: add-list and delete-list.
- Conditional effects -- dependent on condition on the state *when action takes place*.

STRIPS Description of Operators

- A STRIPS operator has 3 parts:
  - A set, PC (*preconditions*) of ground literals
  - A set D, of ground literals called the *delete list*
  - A set A, of ground literals called *add list*
- Usually described by Schema: *Move(x,y,z)*
  - *PC: On(x,y) and Clear(x) and Clear(z)*
  - *D: Clear(z) , On(x,y)*
  - *A: On(x,z), Clear(y), Clear(F1)*
- A state S1 is created applying operator O by adding A and deleting D from S1.

- MOVE(x,y,z) moves block x from the top of y to the top of z. y and z can be either the table or another block. MOVE is applicable only if x and z are clear, and x is on y.



Different Representation - Blocksworld

- MOVE(x,y,z) moves block x from the top of y to the top of z. y and z can be either the table or another block. MOVE is applicable only if x and z are clear, and x is on y.

```
(OPERATOR MOVE
 :preconds
   ?block BLOCK
   ?from OBJECT
   ?to OBJECT
  (and (clear ?block)
         (clear ?to)
         (on ?block ?from)
 :effects
  add (on ?block ?to)
  del (on ?block ?from)
  (if (block-p ?from)
      add (clear ?from))
  (if (block-p ?to)
      del (clear ?to)))
```

Action Representation - BlocksWorld

```
(OPERATOR PICK_FROM_TABLE
?ob BLOCK
:preconds
 (and (clear ?ob)
      (on-table ?ob)
      (arm-empty))
:effects
  del (on-table ?ob)
  del (clear ?ob)
  del (arm-empty)
  add (holding ?ob))
```

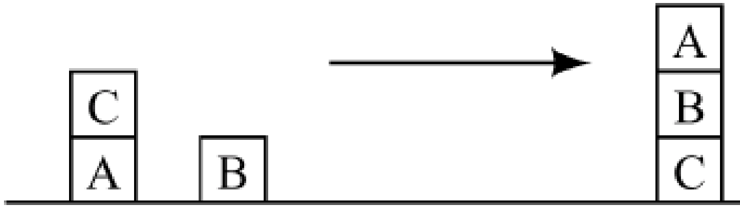
```
(OPERATOR
PICK_FROM_BLOCK
?ob BLOCK
?uob BLOCK
:preconds
 (and (on ?ob ?uob)
      (clear ?ob)
      (arm-empty))
:effects
   del (on ?ob ?uob)
   del (clear ?ob)
   del (arm-empty)
   add (holding ?ob)
   add (clear ?uob))
```

Action Representation - BlocksWorld

```
(OPERATOR PUT_ON_BLOCK
  ?ob BLOCK
  ?uob BLOCK
  :preconds
      (and (clear ?uob)
        (holding ?ob))
  :effects
   del  (holding ?ob)
   del  (clear ?uob)
   add  (clear ?ob )
   add  (arm-empty)
   add  (on ?ob ?uob))
```

```
(OPERATOR PUT_DOWN_ON_TABLE
?ob
BLOCK                    :precon
ds
          (holding ?ob)
 :effects
   del  (holding ?ob)
   add  (clear ?ob)
   add  (arm-empty)
   add  (on-table ?ob))
```

The block world



$Init(On(A, Table) \wedge On(B, Table) \wedge On(C, Table)$
$\quad \wedge Block(A) \wedge Block(B) \wedge Block(C)$
$\quad \wedge Clear(A) \wedge Clear(B) \wedge Clear(C))$
$Goal(On(A, B) \wedge On(B, C))$
$Action(Move(b, x, y),$
$\quad$ PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge$
$\qquad (b \neq x) \wedge (b \neq y) \wedge (x \neq y),$
$\quad$ EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$
$Action(MoveToTable(b, x),$
$\quad$ PRECOND: $On(b, x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x),$
$\quad$ EFFECT: $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

Figure 11.4 A planning problem in the blocks world: building a three-block tower. One solution is the sequence $[Move(B, Table, C), Move(A, Table, B)]$.

A STRIP/PDDL description of an aircargo transportation problem

Problem: flying cargo in planes from one location to another

$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\quad \land Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$
$Action(Load(c, p, a),$
$\quad$ PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
$\quad$ EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
$\quad$ PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
$\quad$ EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
$\quad$ PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
$\quad$ EFFECT: $\neg At(p, from) \land At(p, to))$

Figure 11.2 A STRIPS problem involving transportation of air cargo between airports.

In(c,p)- cargo c is inside plane p
At(x,a) – object x is at airport a

STRIP for spare tire problem

Problem: Changing a flat tire

$Init(At(Flat, Axle) \land At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
  PRECOND: $At(Spare, Trunk)$
  EFFECT: $\neg At(Spare, Trunk) \land At(Spare, Ground))$
$Action(Remove(Flat, Axle),$
  PRECOND: $At(Flat, Axle)$
  EFFECT: $\neg At(Flat, Axle) \land At(Flat, Ground))$
$Action(PutOn(Spare, Axle),$
  PRECOND: $At(Spare, Ground) \land \neg At(Flat, Axle)$
  EFFECT: $\neg At(Spare, Ground) \land At(Spare, Axle))$
$Action(LeaveOvernight,$
  PRECOND:
  EFFECT: $\neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)$
    $\land \neg At(Flat, Ground) \land \neg At(Flat, Axle))$

Figure 11.3 The simple spare tire problem.

State-Space Search

- Search the space of states
  - Initial state, goal test, step cost, etc.
  - Actions are the transitions between state
- Actions are invertible
  - Move forward from the initial state: Forward State-Space Search or <u>Progression Planning</u>
  - Move backward from goal state: Backward State-Space Search or <u>Regression Planning</u>

Planning forward and backward

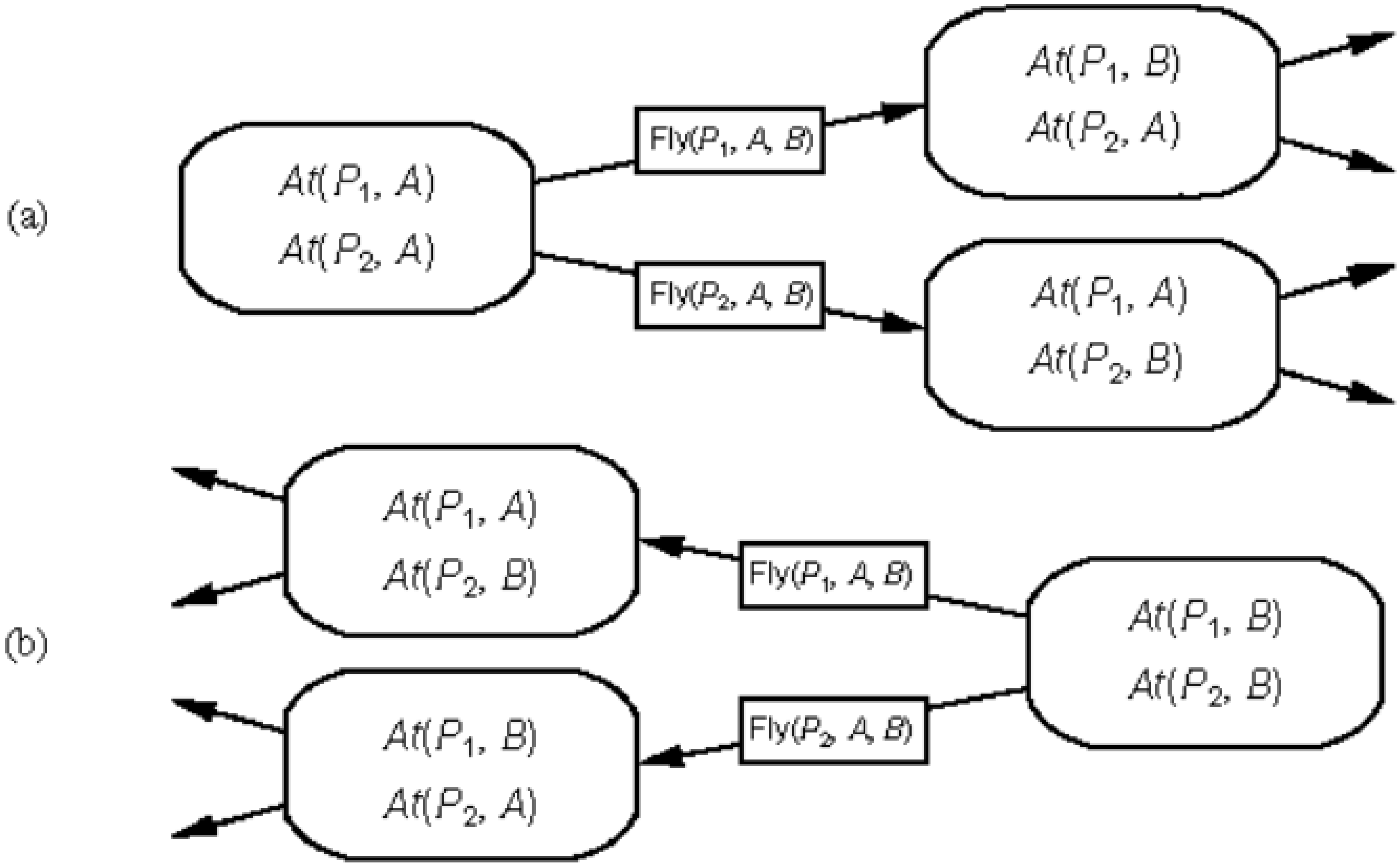
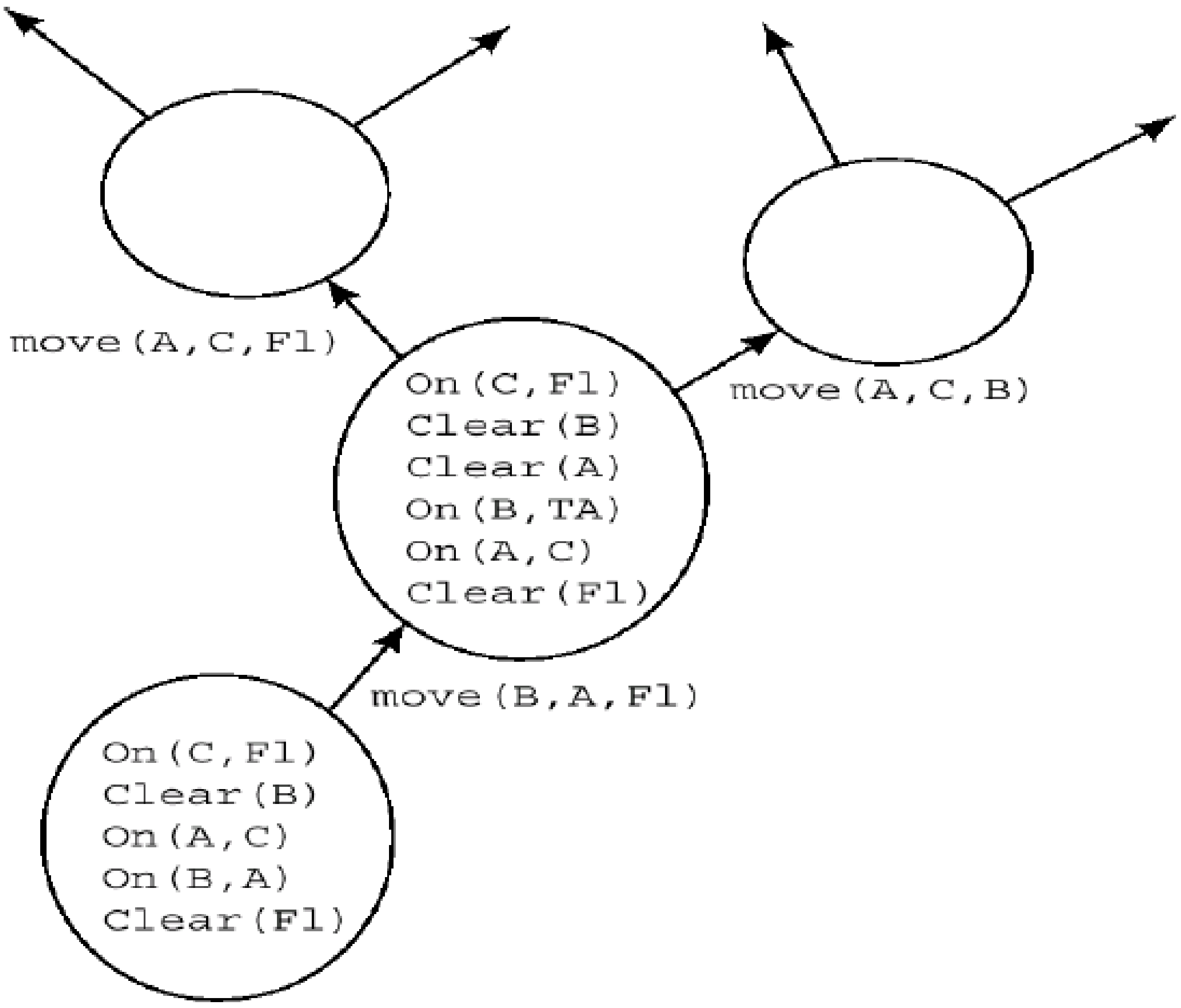


Figure 11.5 Two approaches to searching for a plan. (a) Forward (progression) state-space search, starting in the initial state and using the problem's actions to search forward for the goal state. (b) Backward (regression) state-space search: a belief-state search (see page 84)

Forward Search methods



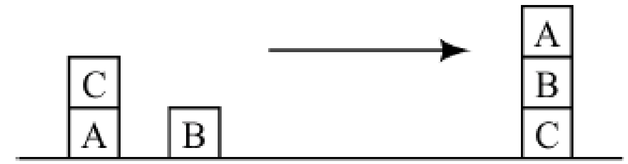
Backward: Recursive STRIPS

- **Forward** search with islands:
- Achieve one subgoal at a time. Achieve a new conjunct without ever violating already achieved conjuncts or maybe temporarily violating previous subgoals.
- **General Problem Solver** (GPS) by Newell Shaw and Simon (1959) uses **Means-Ends** analysis.
- Each subgoal is achieved via a matched rule, then its preconditions are subgoals and so on. This leads to a planner called STRIPS(gamma) when gamma is a goal formula.

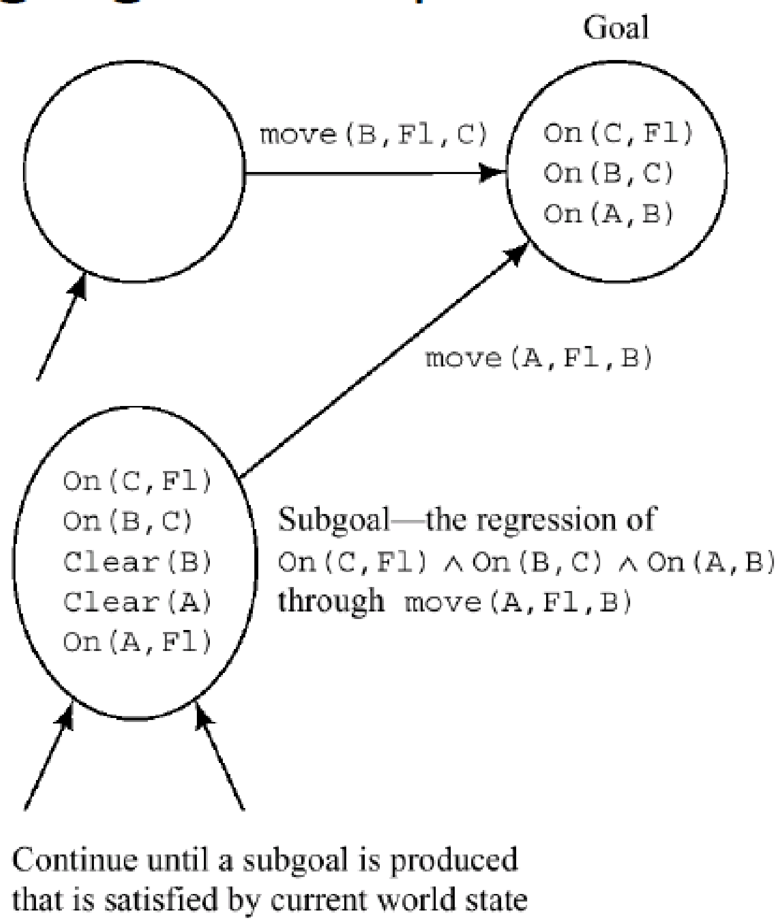
STRIPS algorithm

- Given a goal stack:
- 1. Consider the top goal
- 2. Find a sequence of actions satisfying the goal from the current state and apply them.
- 3. The next goal is considered from the new state.
- 4. Temination: stack empty
- 5. Check goals again.

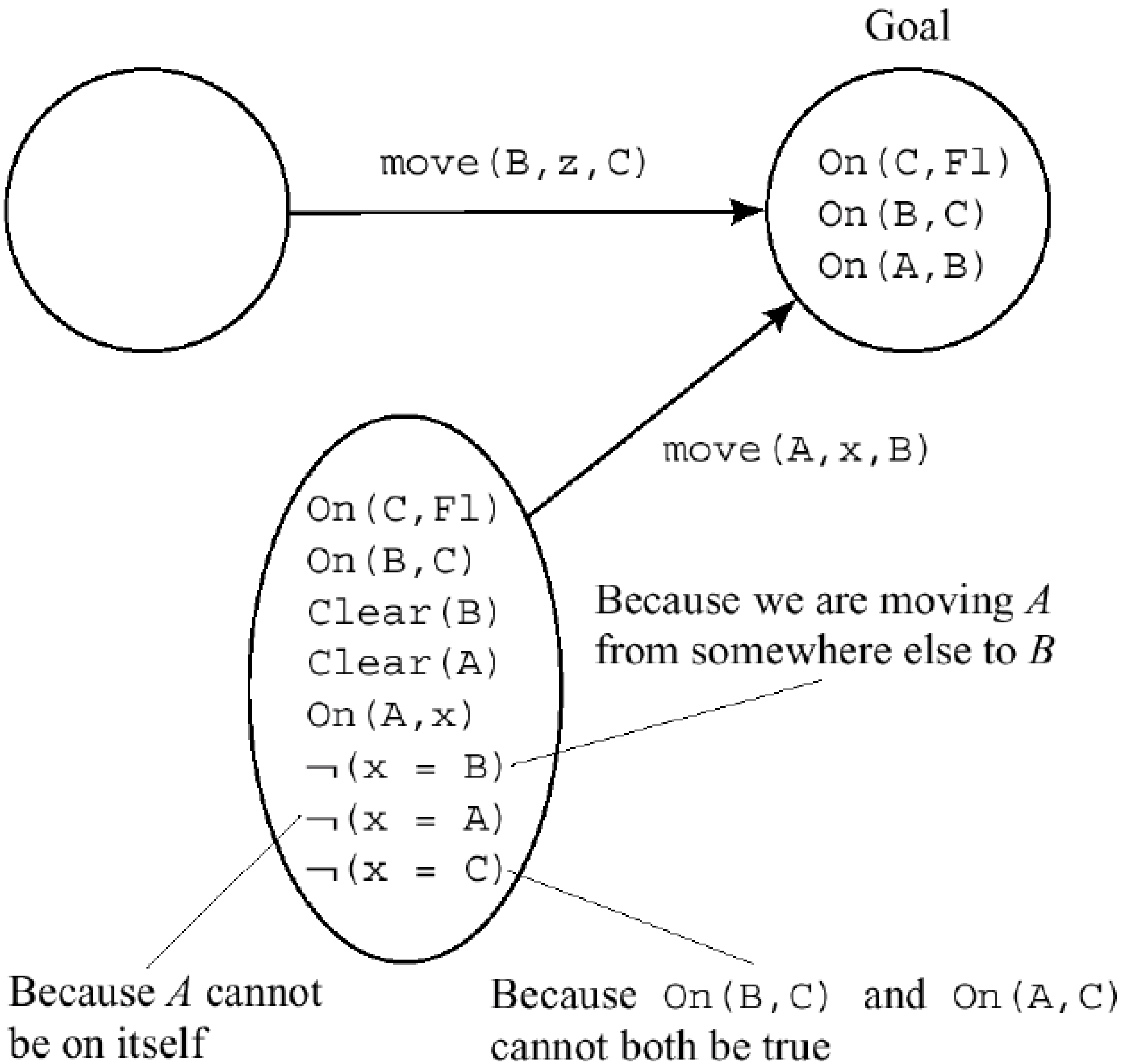
Backward search methods;

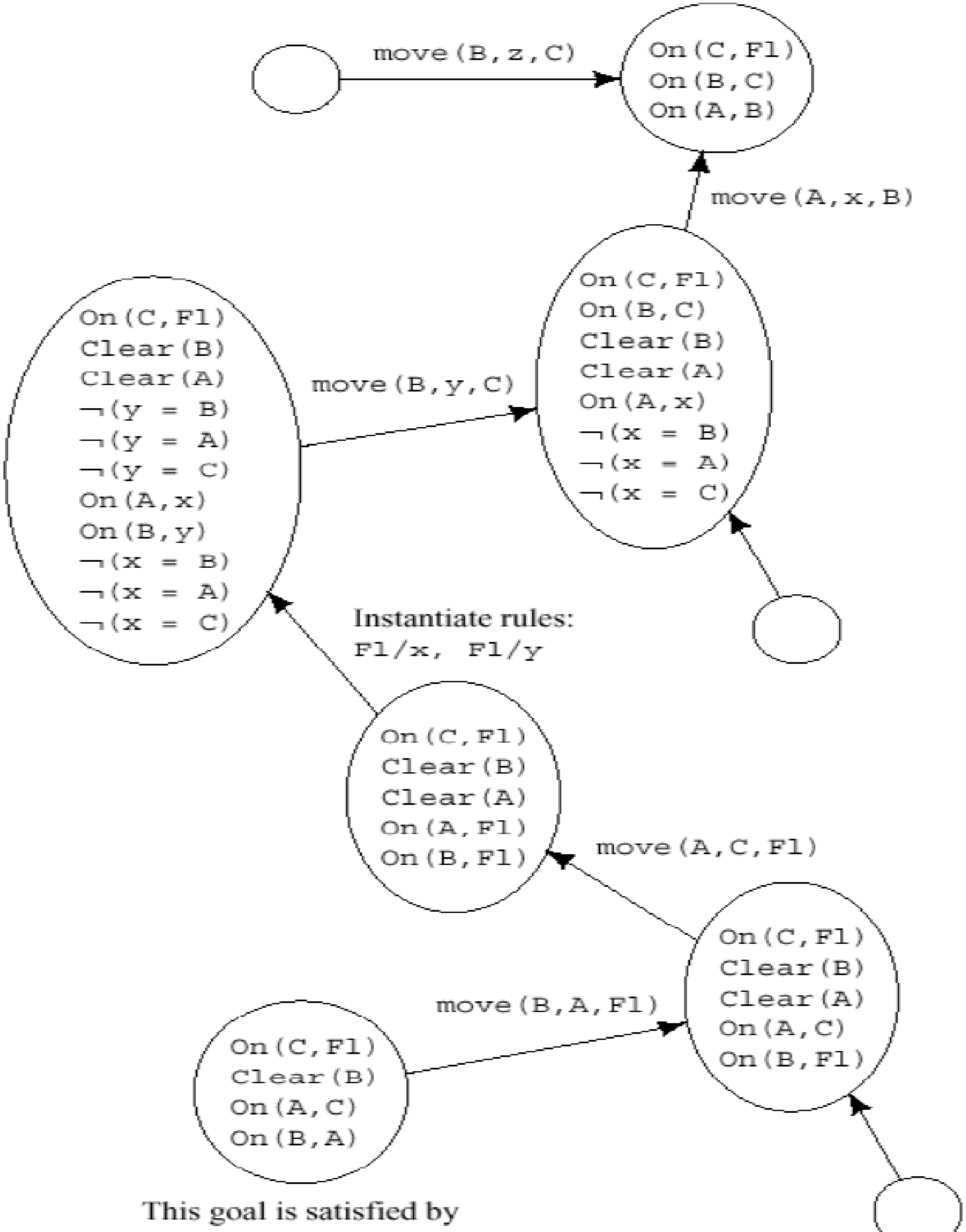


- Regressing a ground operator



Regressing an ungrounded operator

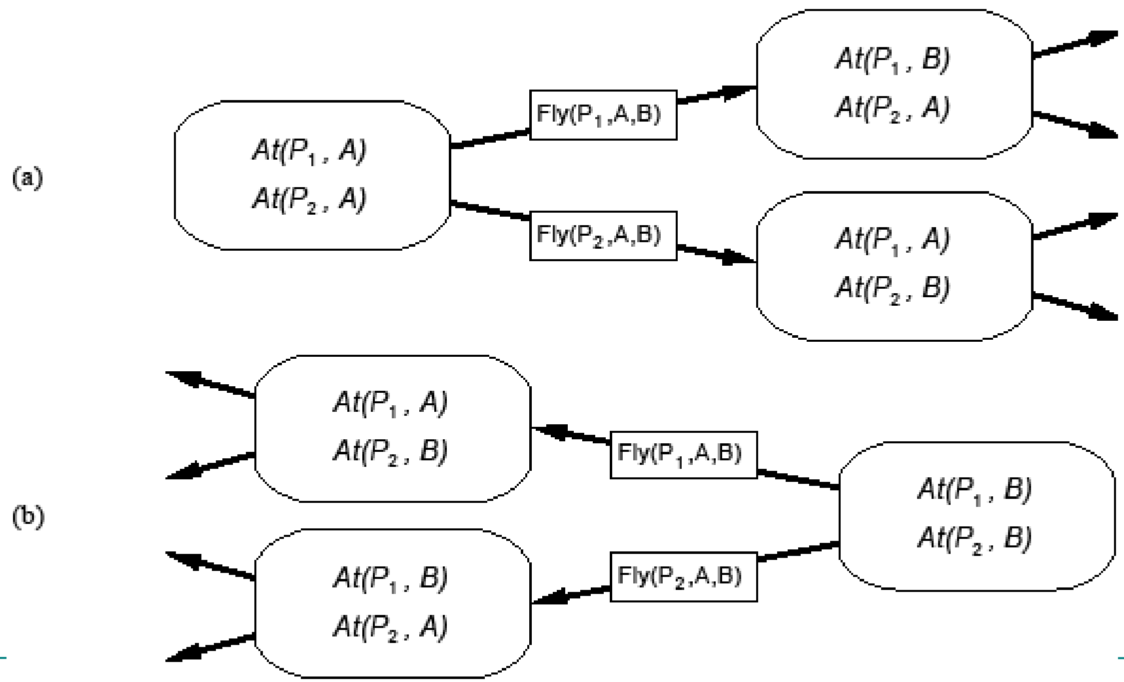




move(B,z,C)
On(C,Fl)
On(B,C)
On(A,B)
move(A,x,B)
On(C,Fl)
Clear(B)
Clear(A)
¬(y = B)
¬(y = A)
¬(y = C)
On(A,x)
On(B,y)
¬(x = B)
¬(x = A)
¬(x = C)
move(B,y,C)
On(C,Fl)
On(B,C)
Clear(B)
Clear(A)
On(A,x)
¬(x = B)
¬(x = A)
¬(x = C)
Instantiate rules:
Fl/x, Fl/y
On(C,Fl)
Clear(B)
Clear(A)
On(A,Fl)
On(B,Fl)
move(A,C,Fl)
On(C,Fl)
Clear(B)
Clear(A)
On(A,C)
On(B,Fl)
move(B,A,Fl)
On(C,Fl)
Clear(B)
On(A,C)
On(B,A)
This goal is satisfied by

SRIPS in State-Space Search

- STRIPS representation makes it easy to focus on 'relevant' propositions and
  - Work backward from goal (using EFFECTS)
  - Work forward from initial state (using PRECONDITIONS)
  - Facilitating bidirectional search



Relevant Action

- An action is relevant
  - In Progression planning when its preconditions match a subset of the current state
  - In Regression planning, when its effects match a subset of the current goal state

Consistent Action

- The purpose of applying an action is to ‘achieve a desired literal’
- We should be careful that the action does not undo a desired literal (as a side effect)
- A consistent action is an action that does not undo a desired literal

Backward State-Space Search

- Given
  - A goal *G* description
  - An action *A* that is relevant and consistent
- Generate a predecessor state where
  - Positive effects (literals) of *A* in *G* are deleted
  - Precondition literals of *A* are added unless they already appear
  - Substituting any variables in A's effects to match literals in G
  - Substituting any variables in A's preconditions to match substitutions in A's effects
- Repeat until predecessor description matches initial state

State-Space Search

- Remember that the language has no functions symbols
- Thus number of states is finite (but could be very large)
- And we can use any complete search algorithm (e.g., A*)
  - We need an admissible heuristic
  - The solution is a path, a sequence of actions: total-order planning
- Problem: Space and time complexity
  - STRIPS-style planning is PSPACE-complete
  - Becomes tractable when actions have
    - only positive preconditions and
    - only one literal effect

Relaxed Plans

- What makes planning difficult?
- **Delete effects** – if there are no 'bad moves', problem solving is easy.
- Ignoring delete effects gives us a **relaxed planning problem.**
- Solutions to the relaxed problem – **relaxed plans.**
- Length of relaxed plan approximates that of the non-relaxed plan.
- Can we use relaxed planning to make a heuristic?

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Building a Relaxed Planning Graph

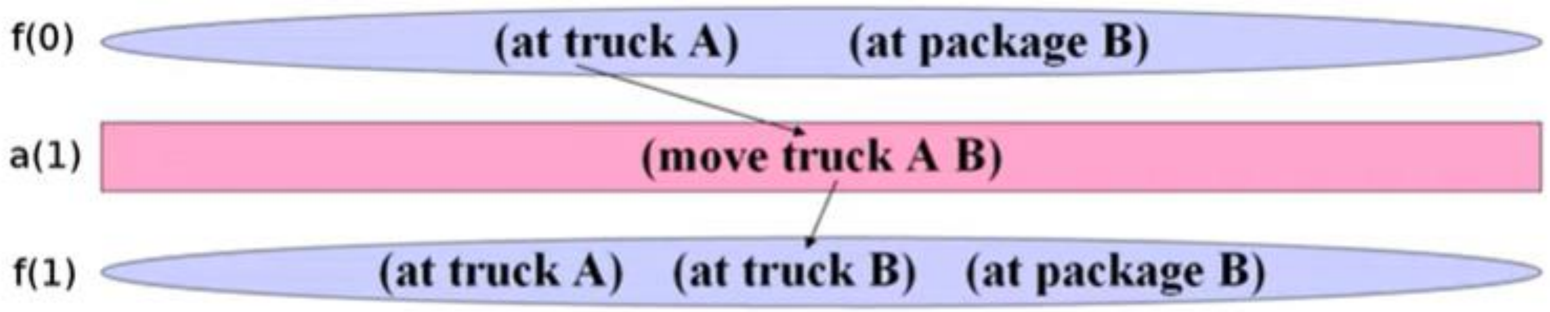
- The Relaxed Planning Graph (RPG) is made of alternate **fact layers** and **action layers**.
- Fact layer f(n) is used to determine which actions can appear in action layer a(n+1)
  - Those whose preconditions are satisfied in f(n)
- f(n+1) = f(n), plus all the add effects of the actions in a(n+1).
- Hence, **fact layers get bigger and bigger** as more actions become applicable.
- The first fact layer, f(0), is a **state S.**

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Building a Simple RPG

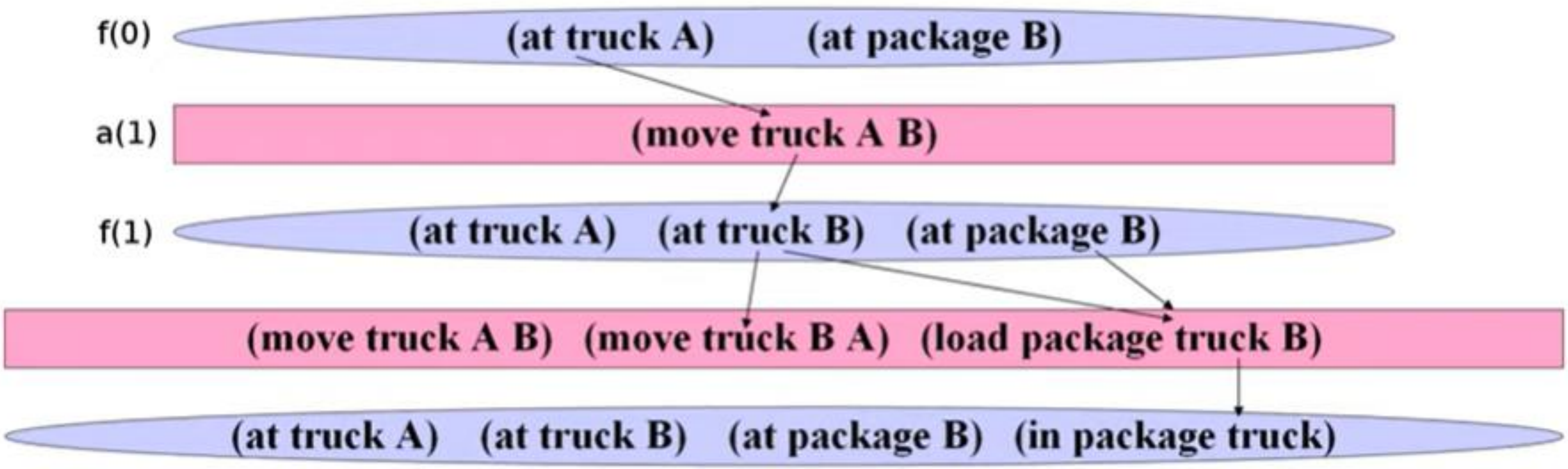
Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Building a Simple RPG

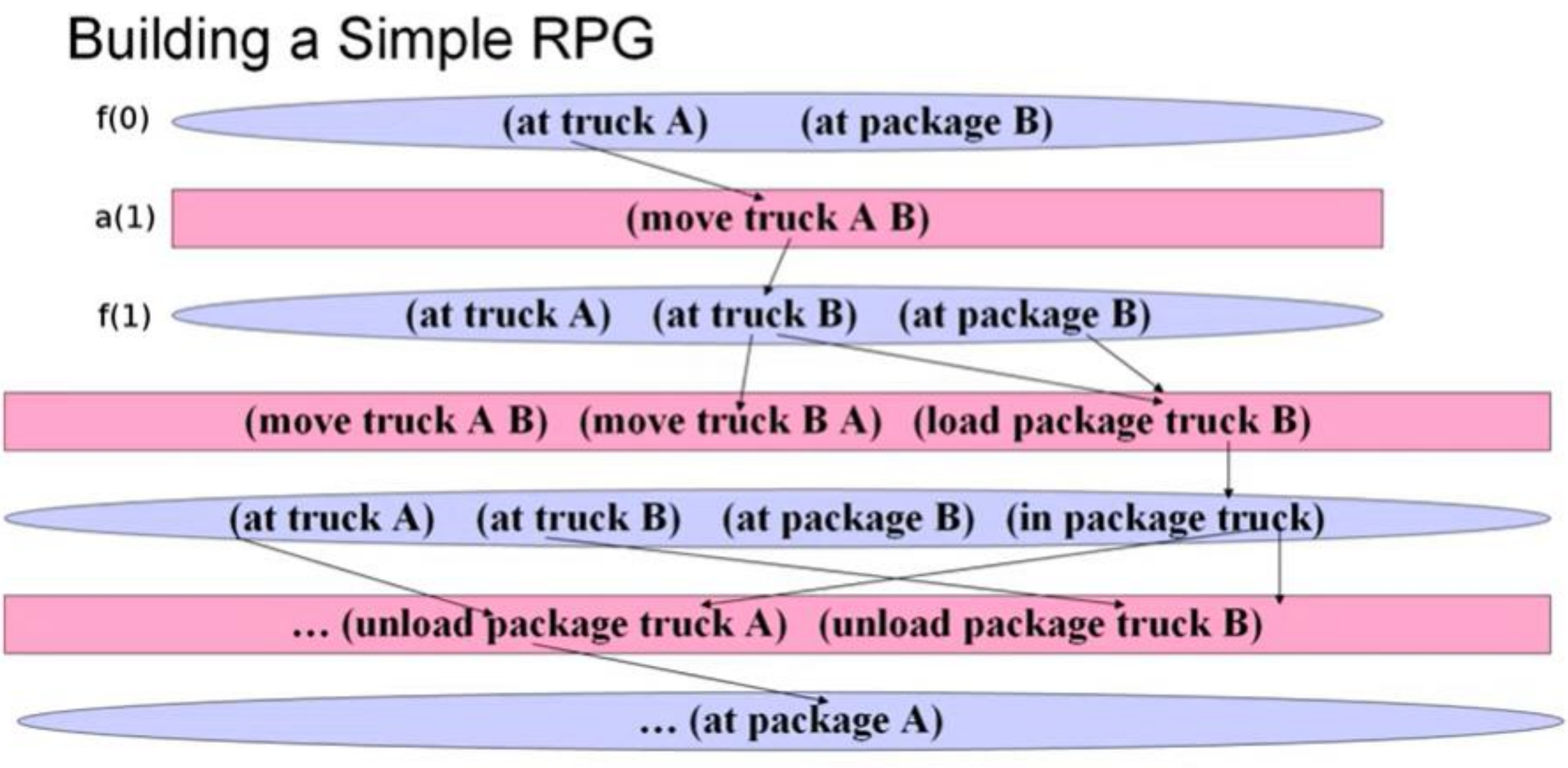


Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Building a Simple RPG



Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU



Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Termination Criterion

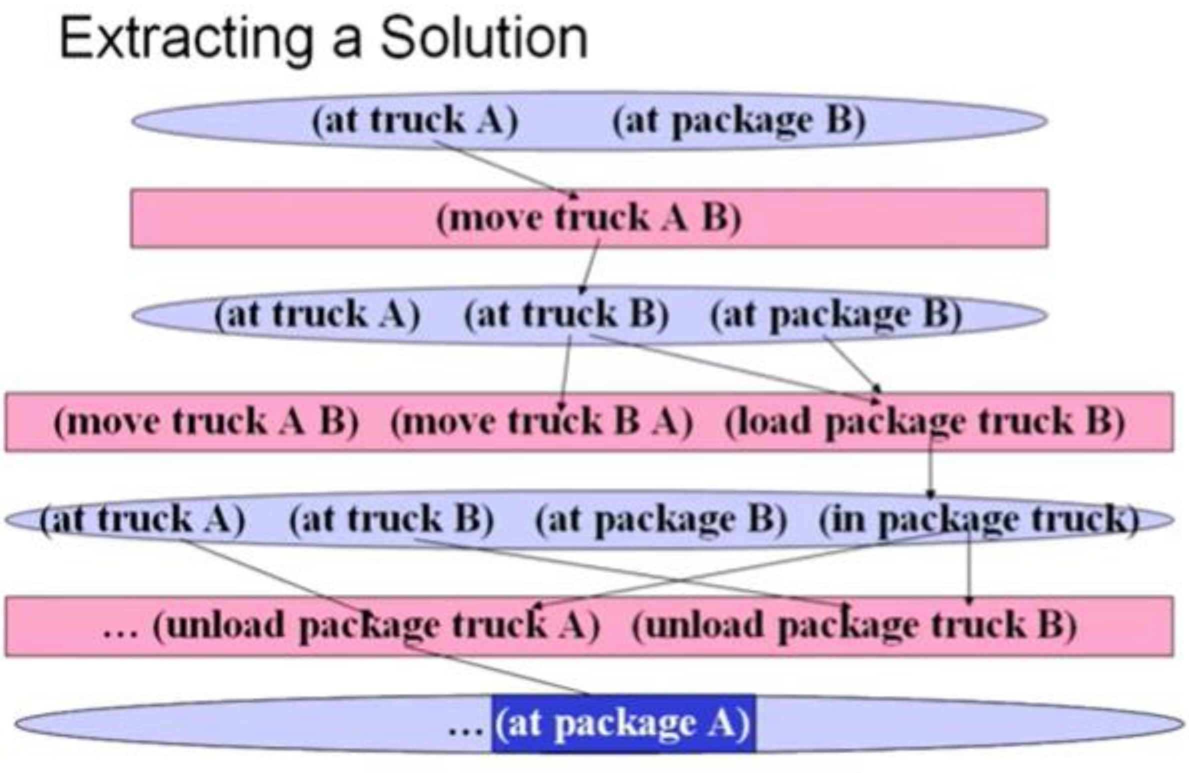
- If we are building a relaxed planning graph to find a relaxed plan to achieve the goals we can stop when we reach a fact layer in which all the goals appear.
- Otherwise we can stop when we generate a fact layer identical to the previous one (i.e. no new facts have appeared).
  - If nothing new appeared this time nothing new ever will.
- Can also use RPG reachability analysis to restrict action instantiation, only create action instances that are reachable from the initial state:
  - If it can't be reached in the relaxed problem it can't be reached in the real one.

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

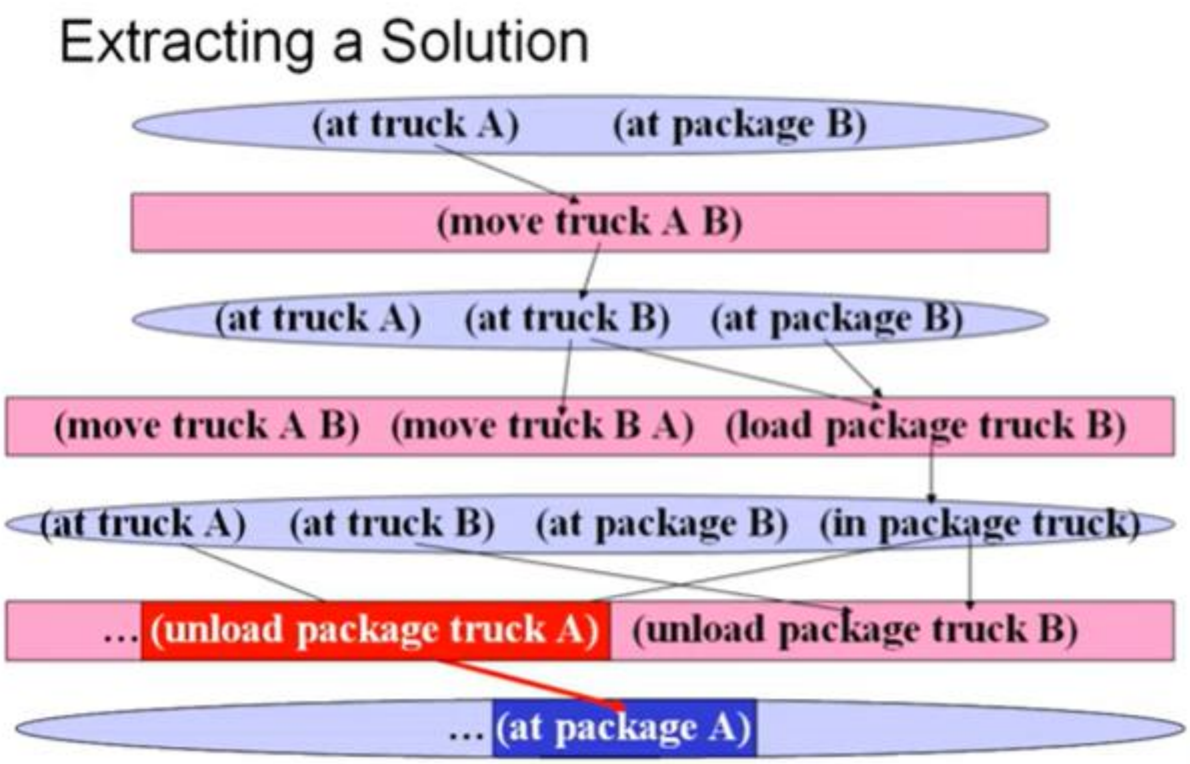
Extracting a Solution

- Planning graph gives us **facts** and **achievers**.
- To get a solution, work backwards through the RPG.
- At each fact layer f(n), we have goals to achieve g(n). We start with g(n) containing the problem goals.
- For each fact in g(n):
  - If it was in f(n-1), add it to g(n-1)
  - Otherwise, **choose an action** from a(n), and **add its preconditions** to g(n-1)
  - (FF has a greedy tie breaking criterion for this choice based on h_{add} values but for our purposes we'll say we choose arbitrarily.)
- Stop when at g(0).

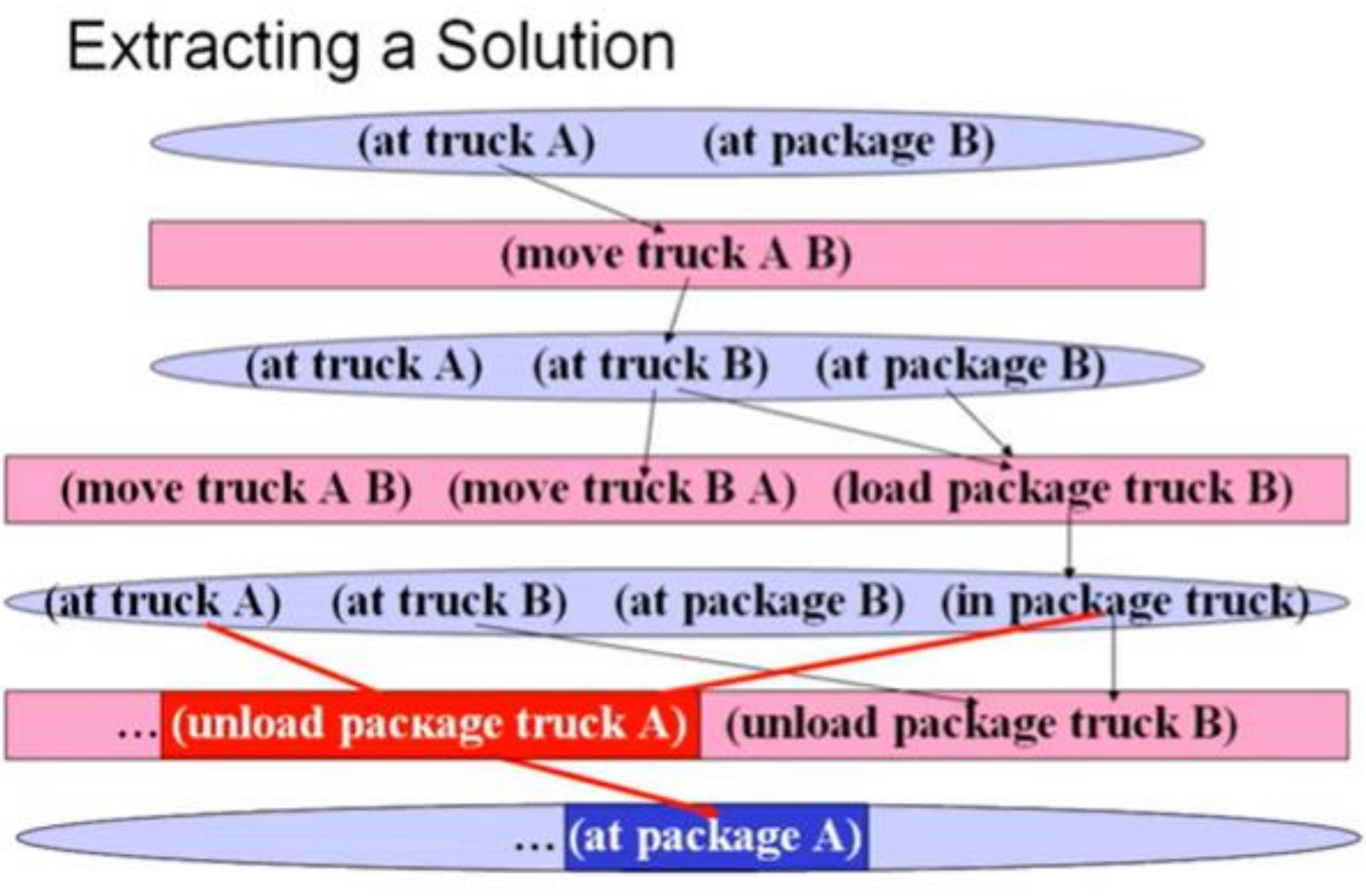
013
https://www.youtube.com/watch?v=EeQcCs9SnhU



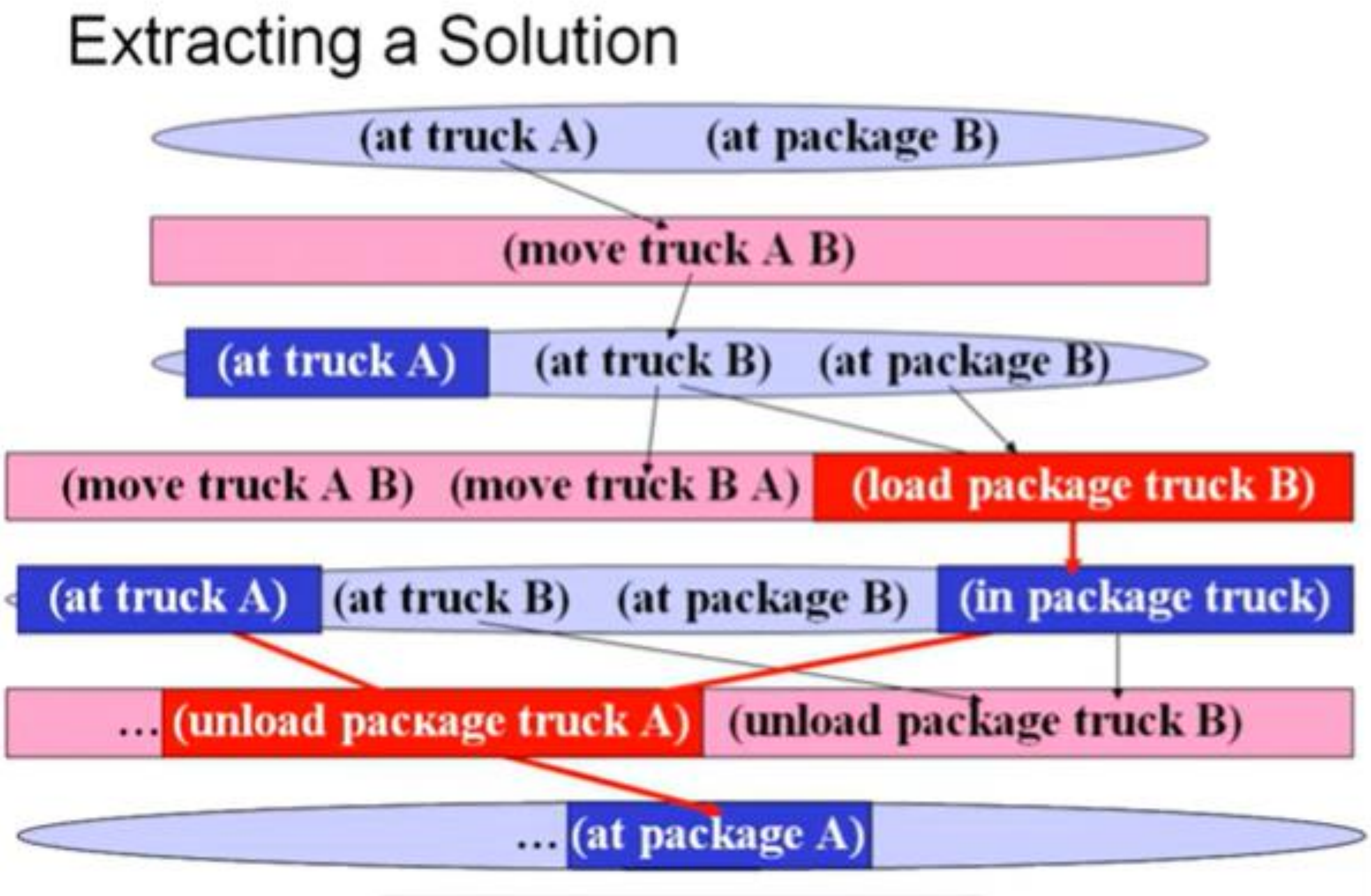
Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU



Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

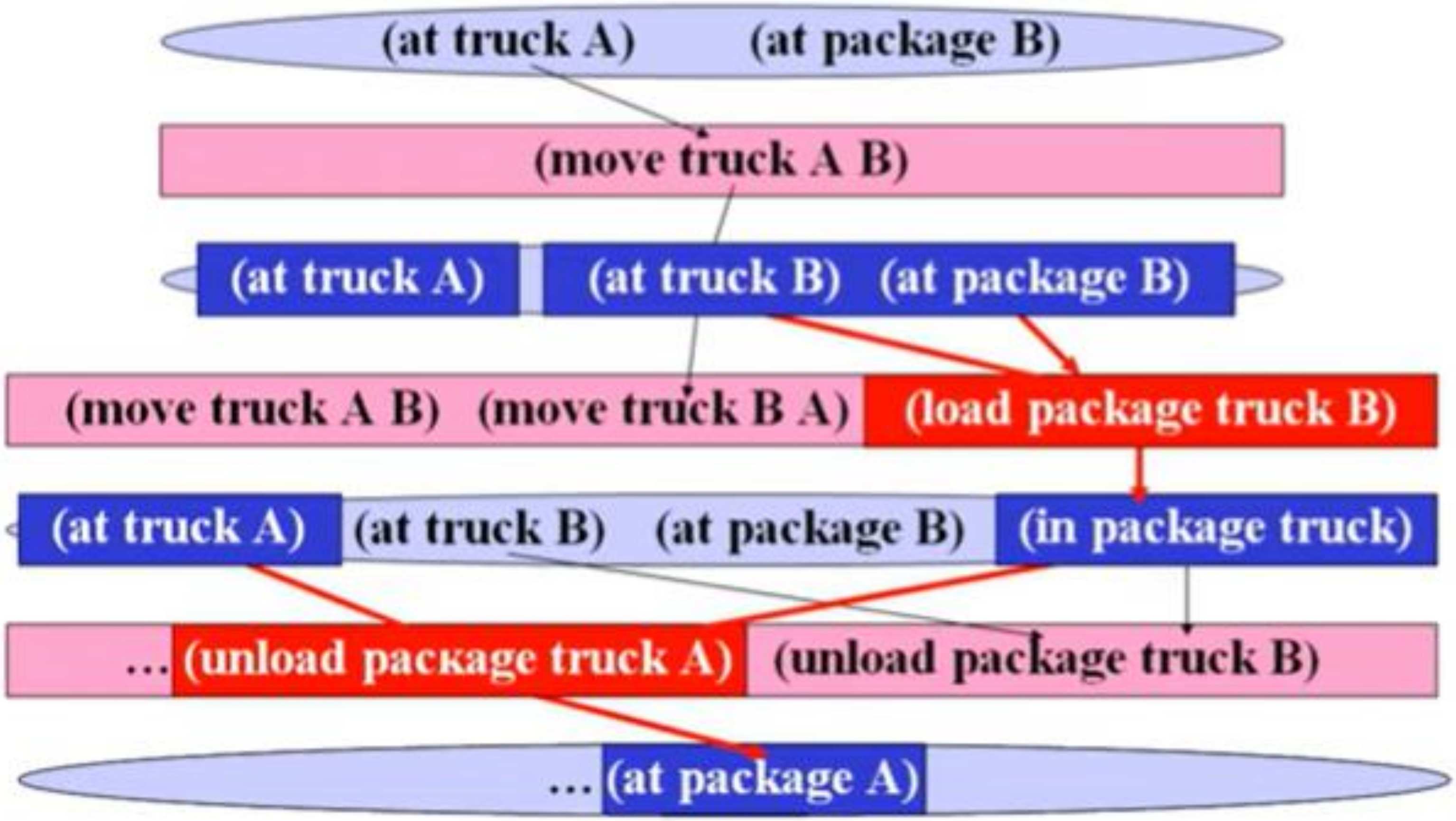


Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU



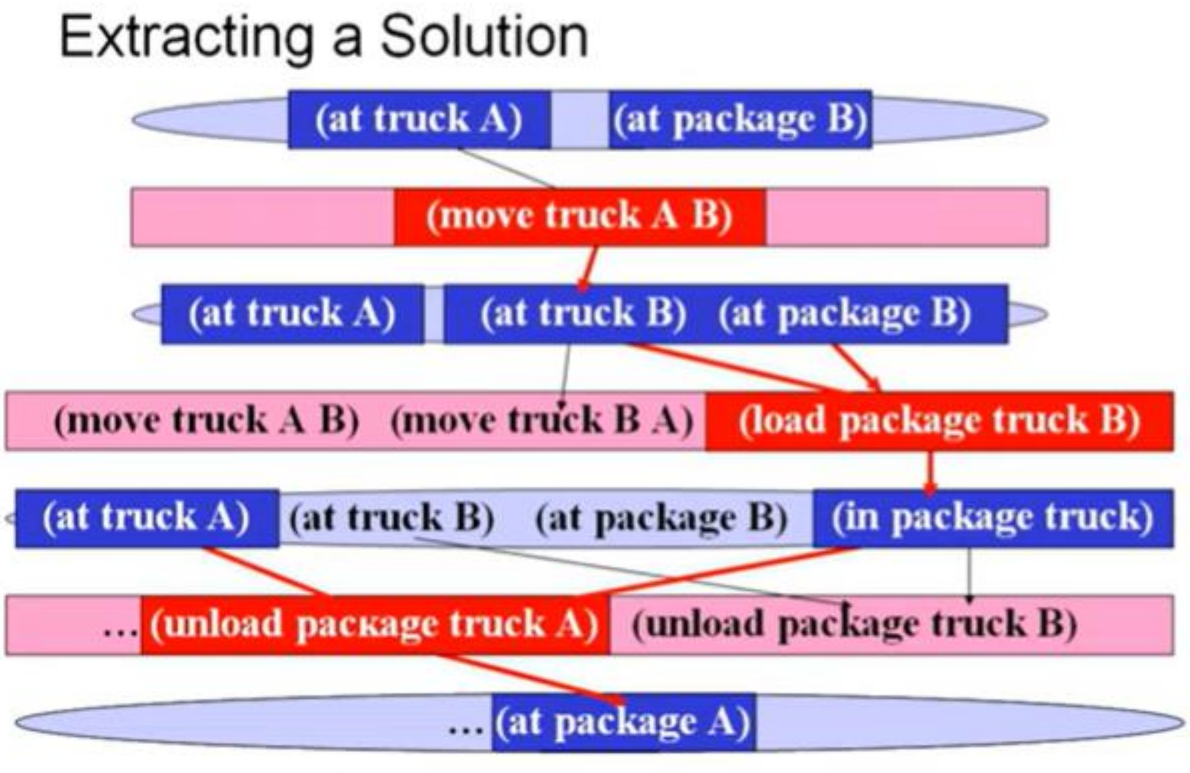
Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Extracting a Solution



Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013

https://www.youtube.com/watch?v=EeQcCs9SnhU



Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Extracting a Solution

(move truck A B)

(load package truck B)

(unload package truck A)

Heuristic value of S: 3

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Detecting Dead-End States

- What if the goal facts never appear in the planning graph?
- In this case, **no relaxed plan can be found** from the state to the goal.
- …and hence, the **state is a dead-end:**
  - if no relaxed plan can be found, adding delete effects won't make it easier.
  - We say the heuristic value of such states is infinite.
- Is a useful property of the heuristic – can discard such states during search.

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Or Not…

- Whilst the RPG can detect dead ends soundly, it is unfortunately not complete:
  - Sound: if it says there is a dead-end there is one
  - Complete: if there is a dead-end it will be found.
- There are some dead-ends that are undetectable by the RPG.
- Indeed domains with such dead-ends tend to be generally difficult for planners.

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Is the Relaxed-Plan Heuristic Admissible?

- Recall that in order to do optimal planning using A* we needed an admissible heuristic.
- In its standard for the RPG heuristic is ***not admissible***.
  - This is because of the greedy extraction procedure used;
  - We select the earliest achiever, breaking ties .
- The length of the ***optimal*** relaxed plan ($h+$) is an admissible heuristic;
  - Unfortunately finding an optimal solution to even the relaxed planning problem is NP-Hard (requires search).
  - We can't afford to do this at ***every*** state just to get an estimate.

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Can we Generate Admissible Heuristics?

- Yes!
- The easy one: count the number of action layers in the relaxed planning graph.
  - This is admissible, although clearly less informative (a price often paid by admissible heuristics).
- Can we do better?
  - Yes! But that's a whole other tutorial!
  - There's lots of exciting research going on now into heuristics for optimal planning.

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

Heuristic Computation: Trade Off

- Heuristic computation is expensive:
  - The planner FF spends around 80% of its search time performing heuristic computation.
- The heuristic must be sufficiently informative and prune enough of the search space to make calculating it worthwhile.
- In 2008 Blind (Breadth-First) Search was competitive with heuristic optimal planners.
  - This is not true any more...

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

FF: Fast Forward

- FF Written by Jörg Hoffmann circa 2000.
- Outstanding Performer IPC-2000.
- Top Performer IPC-2002.
- Still a very popular planner today.
  - It's fast;
  - It's robust.
- FF is a satisfycing planner that makes use of the relaxed plan length heuristic we just learnt about.

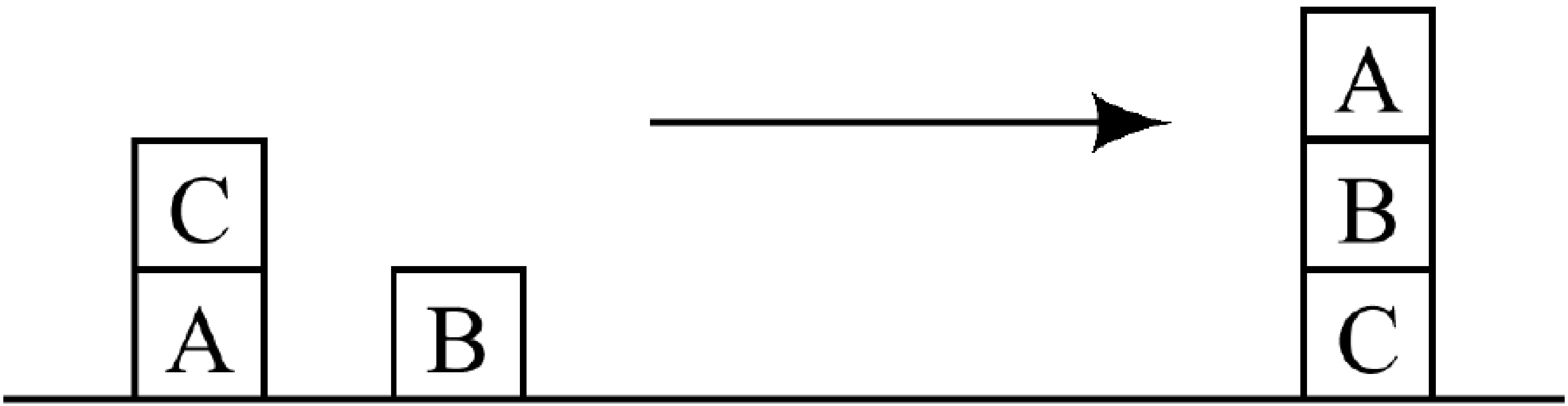
J. Hoffmann and B. Nebel (2001) "The FF Planning System: Fast Plan Generation Through Heuristic Search", Volume 14, pages 253-302

Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

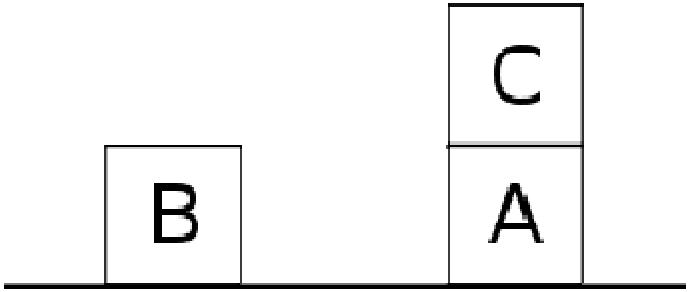
Slide credit: Intruction to AI planning, Amanda Coles, EASSS2013
https://www.youtube.com/watch?v=EeQcCs9SnhU

The Sussman annomaly

- RSTRIPS cannot achieve shortest plan
- Two possible orderings of subgoals:
  - On(A,B) and On(B,C) or On(B,C) and On(A,B)

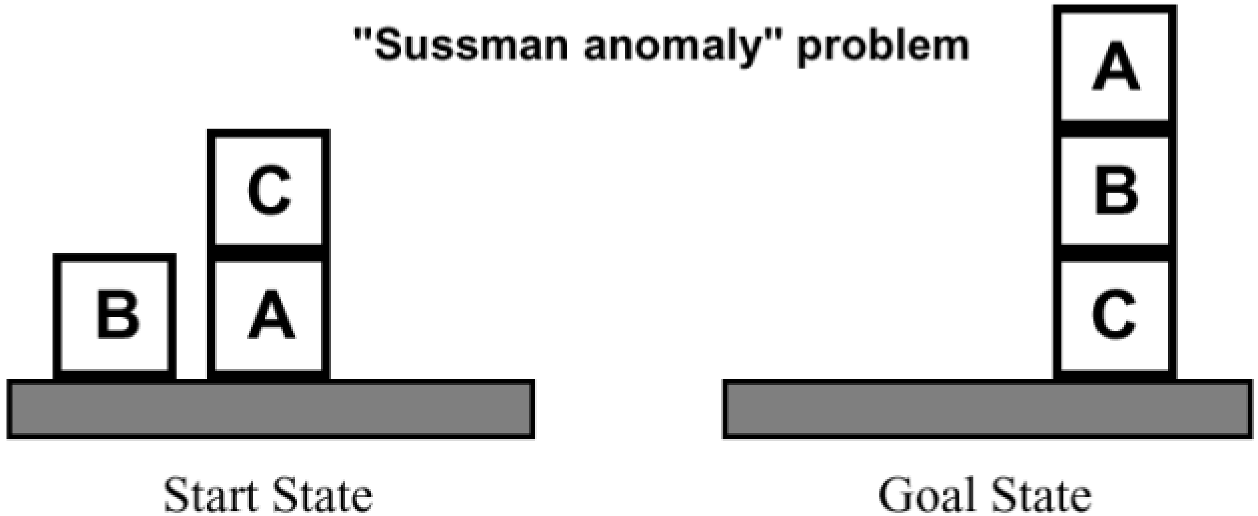


Sussman Anomaly



- The Sussman Anomaly shows the limitations of non-interleaved planning methods
- Before this was described, people used to do planning by considering different subgoals in SEQUENCE
- The Anomaly will show that naively pursuing one subgoal X after you satisfy the other subgoal Y may not work because steps required to accomplish X might undo things subgoal Y

Sussman Anomaly in the block world



Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)
Clear(z) On(x,y)

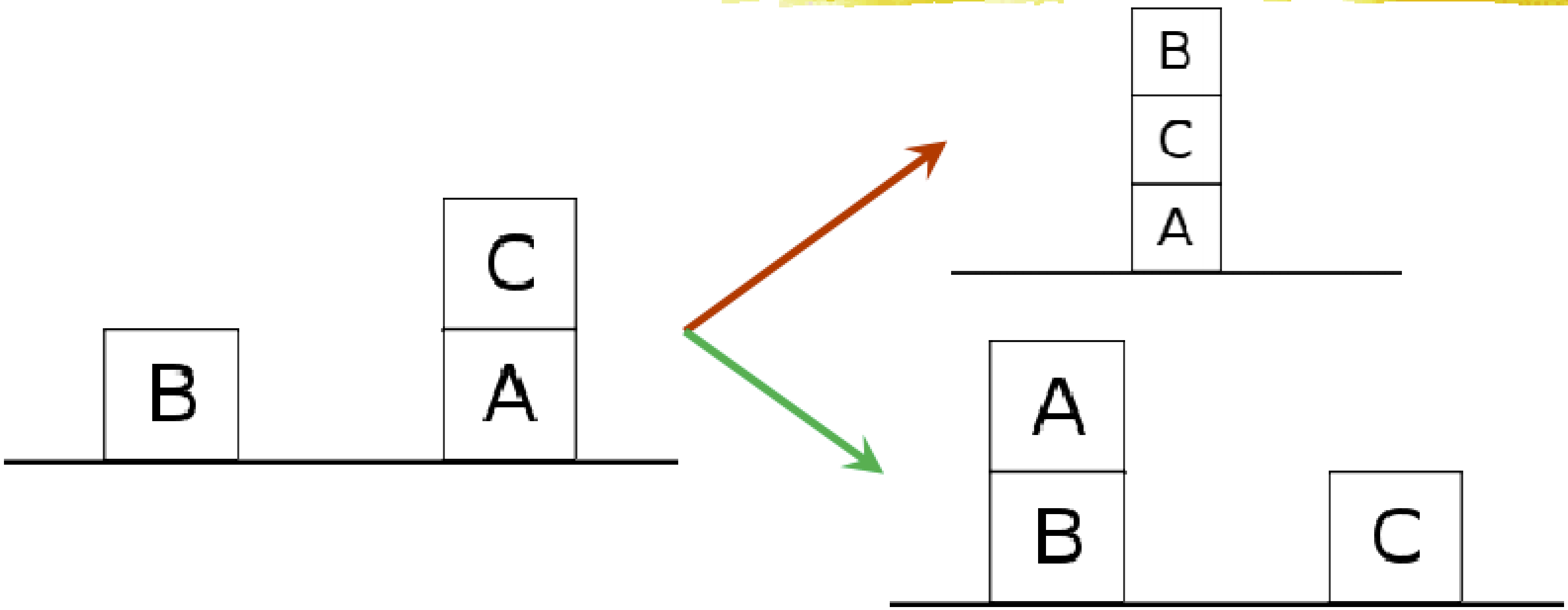
Clear(x) On(x,z)

PutOnTable(x)

~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Sussman Anomaly

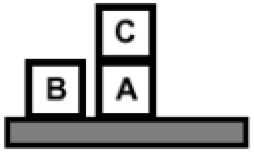


- Final state requires On(A,B) and On(B, C)
- Top diagram tries to focus on subgoal: On(B,C) -- Now trying to put A on top of B cannot be done without undoing On(B, C)
- Bottom diagram tries to focus on subgoal: On(A, B) first; but now trying to put B on top of C would cause On(A,B) to be undone!

Anomaly Illustrates the Need for Interleaved Plans

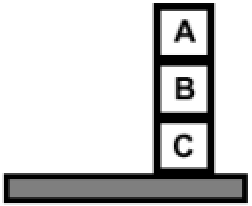
START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

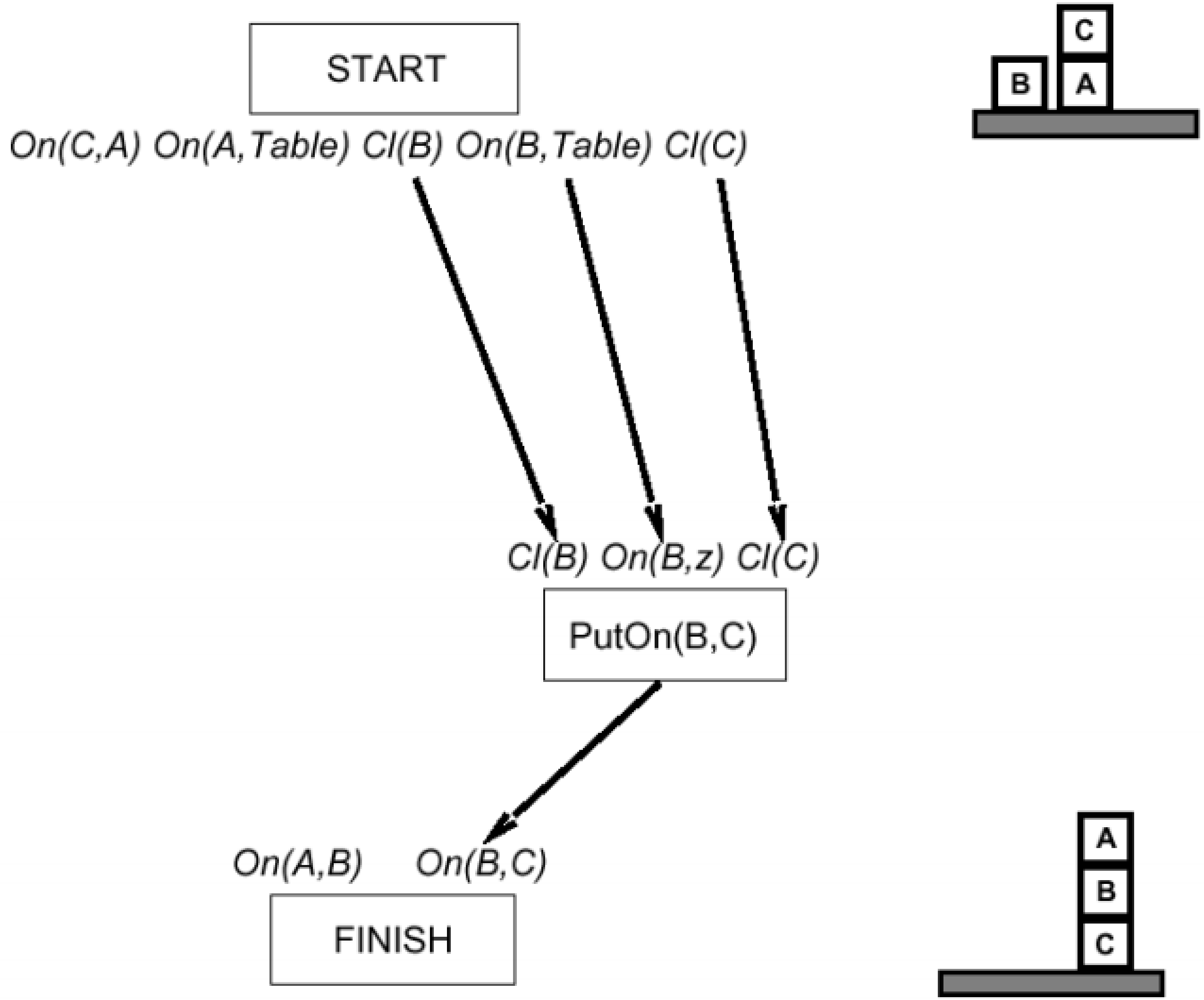


On(A,B) *On(B,C)*

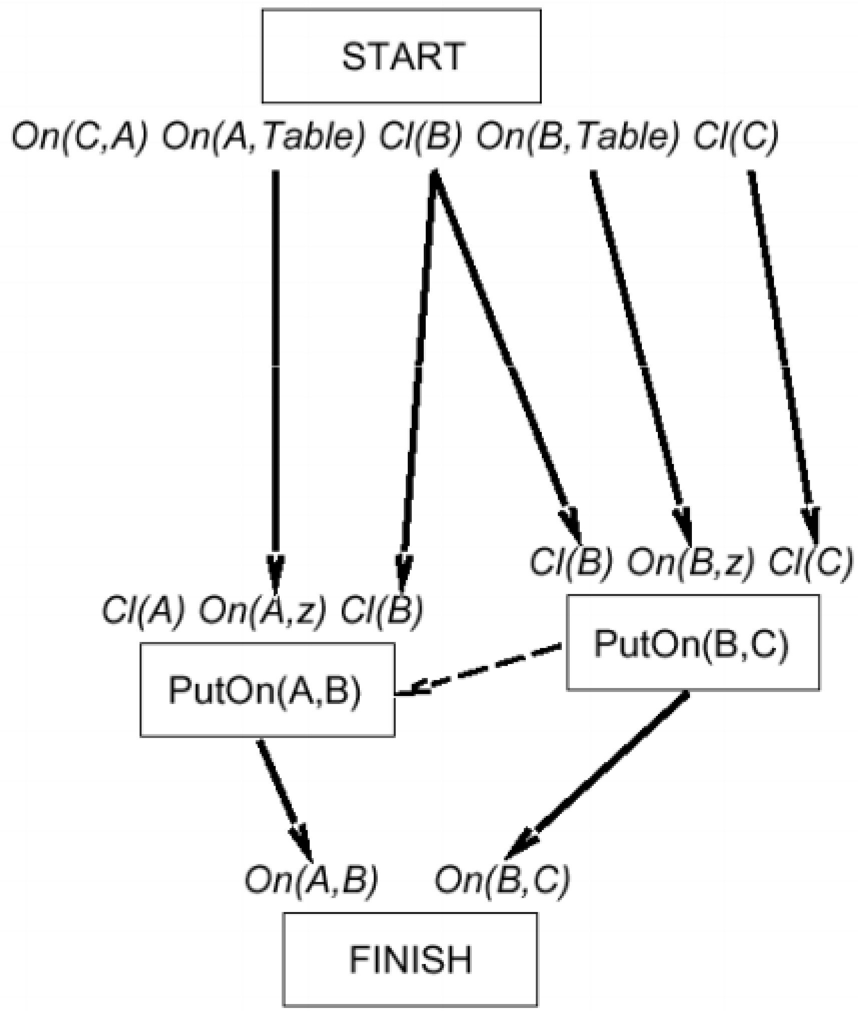
FINISH

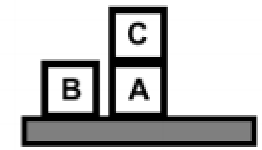


Example: continued

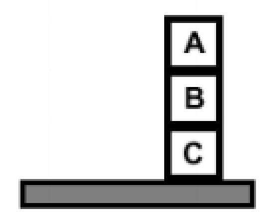


Need to Re-Order Plan Steps Dynamically

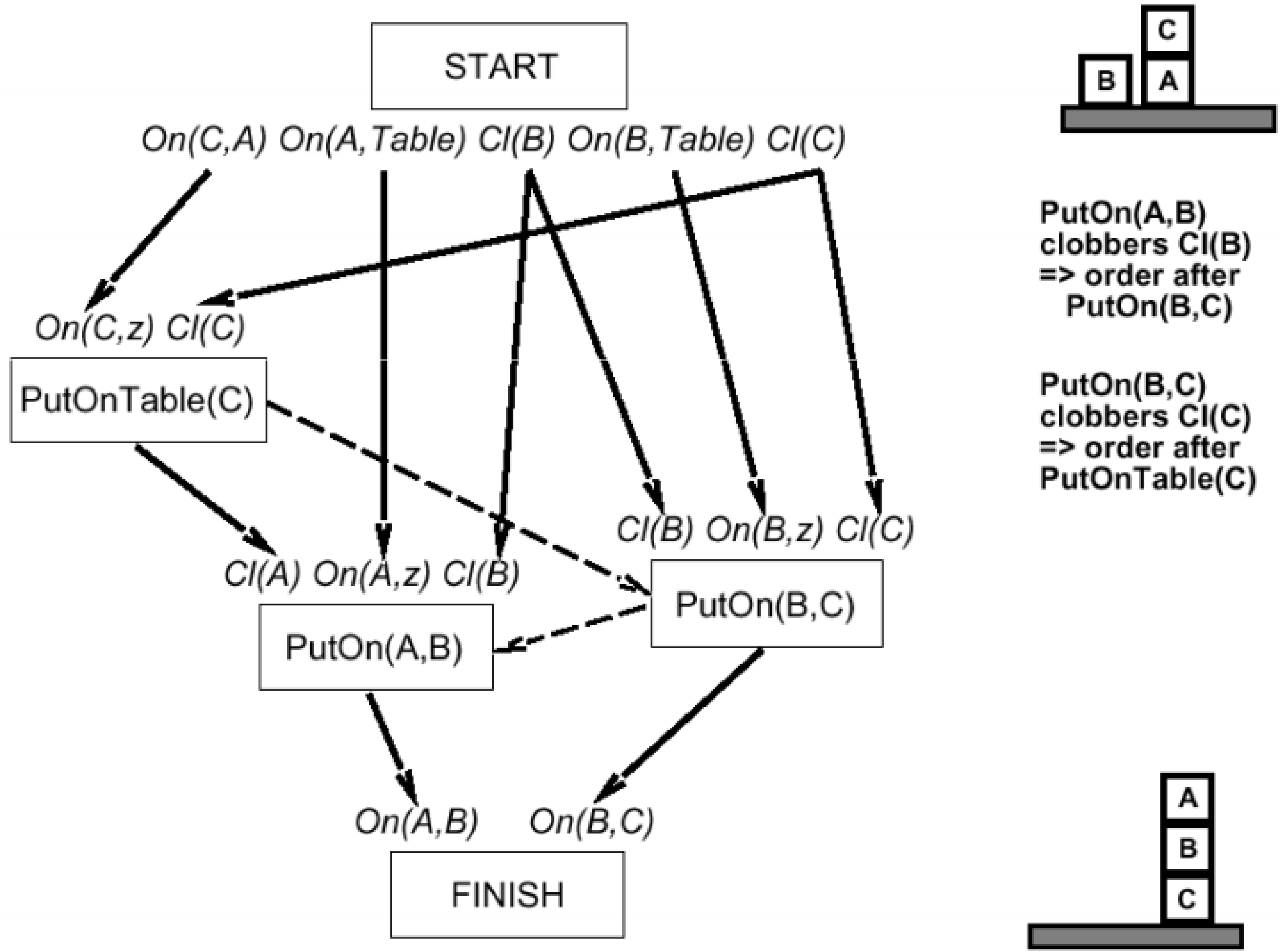




PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)



Example (cont.)



Partial order planning

- Least commitment planning
- Nonlinear planning
- Search in the space of partial plans
- A state is a partial incomplete partially ordered plan
- Operators transform plans to other plans by:
  - Adding steps
  - Reordering
  - Grounding variables
- SNLP: Systematic Nonlinear Planning (McAllester and Rosenblitt 1991)
- NONLIN (Tate 1977)

A partial order plan for putting shoes and socks

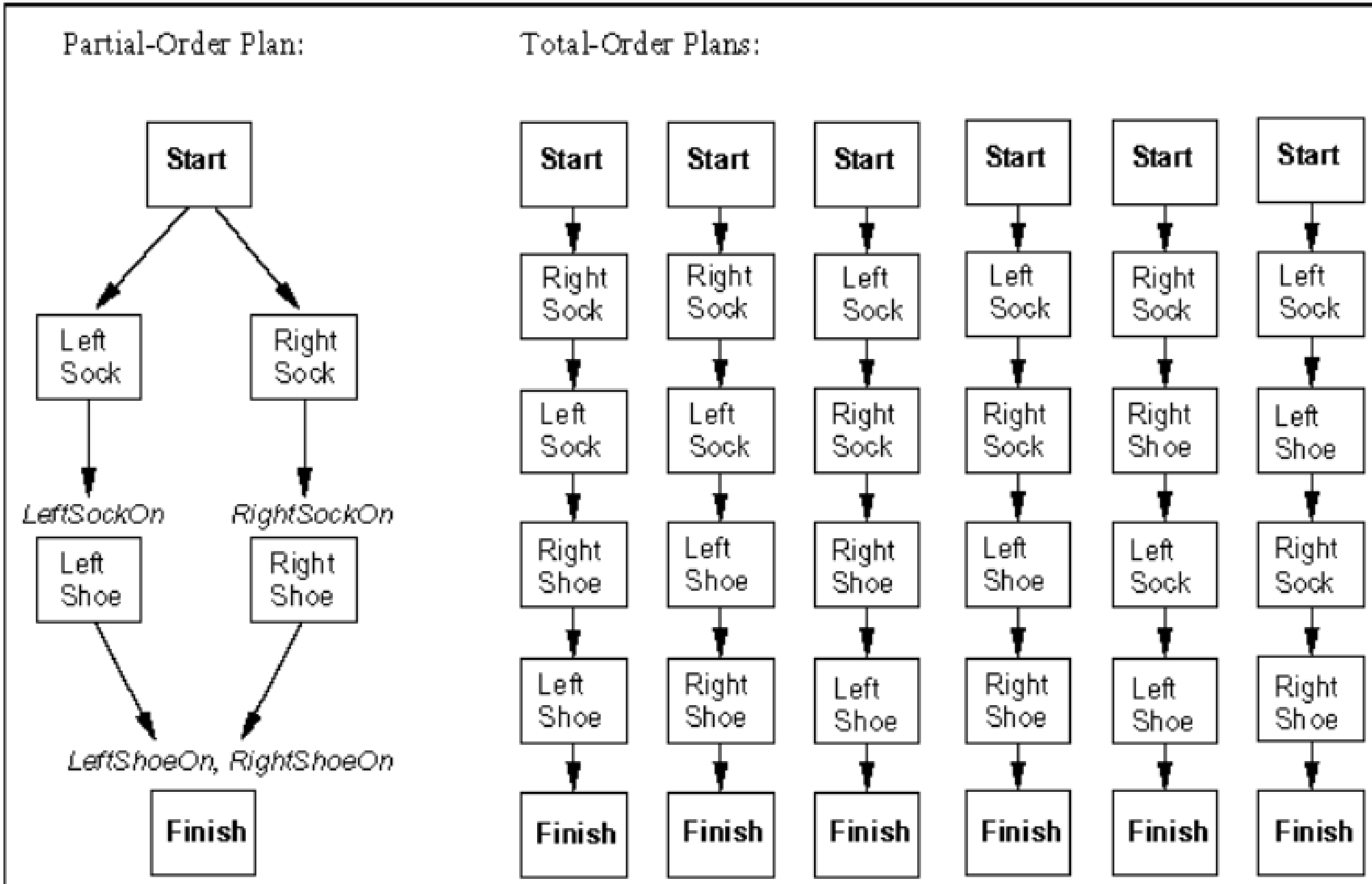


Figure 11.6 A partial-order plan for putting on shoes and socks, and the six corresponding linearizations into total-order plans.

Partial Order Planning (POP)

- State-space search
  - Yields totally ordered plans (linear plans)
- POP
  - Works on subproblems independently, then combines subplans
  - Example
    - **Goal state**: (RightShoeOn $\wedge$ LeftShoeOn)
    - **Initial state**: Init()
    - **Actions**:

      Action(RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)

      Action(RightSock, EFFECT: RightSockOn)

      Action(LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)

      Action(LeftSock, EFFECT: LeftSockOn)

Partially Ordered Plans

- Partially Ordered Plan
  - A partially ordered collection of steps
    - ***Start step*** has the initial state description and its effect
    - ***Finish step*** has the goal description as its precondition
    - ***Causal links*** from outcome of one step to precondition of another step
    - ***Temporal ordering*** between pairs of steps

Components of a Plan

1. A set of actions
2. A set of ordering constraints
   - A $\prec$ B reads "A before B" but not necessarily immediately before B
   - Alert: caution to cycles A $\prec$ B and B $\prec$ A
3. A set of causal links (protection intervals) between actions
   - A $\xrightarrow{p}$ B reads "A achieves p for B" and p must remain true from the time A is applied to the time B is applied
   - Example "RightSock $\xrightarrow{RightSockOn}$ RightShoe
4. A set of open preconditions
   - Planners work to reduce the set of open preconditions to the empty set w/o introducing contradictions

Partial Ordered Plans

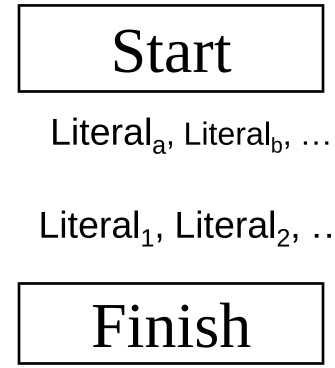
- An open condition is a precondition of a step not yet causally linked
- A plan is ***complete*** iff every precondition is achieved
- A precondition is ***achieved*** iff it is the effect if an earlier step and no possibly intervening step undoes it

Consistent Plan (POP)

- Consistent plan is a plan that has
  - No cycle in the ordering constraints
  - No conflicts with the causal links
- Solution
  - Is a consistent plan with no open preconditions
- To solve a conflict between a causal link $A \xrightarrow{p} B$ and an action C (that clobbers, threatens the causal link), we force C to occur outside the "protection interval" by adding
  - the constraint $C \prec A$ (demoting C) or
  - the constraint $B \prec C$ (promoting C)

Setting up the PoP

- Add dummy states
  - Start
    - Has no preconditions
    - Its effects are the literals of the initial state
  - Finish
    - Its preconditions are the literals of the goal state
    - Has no effects

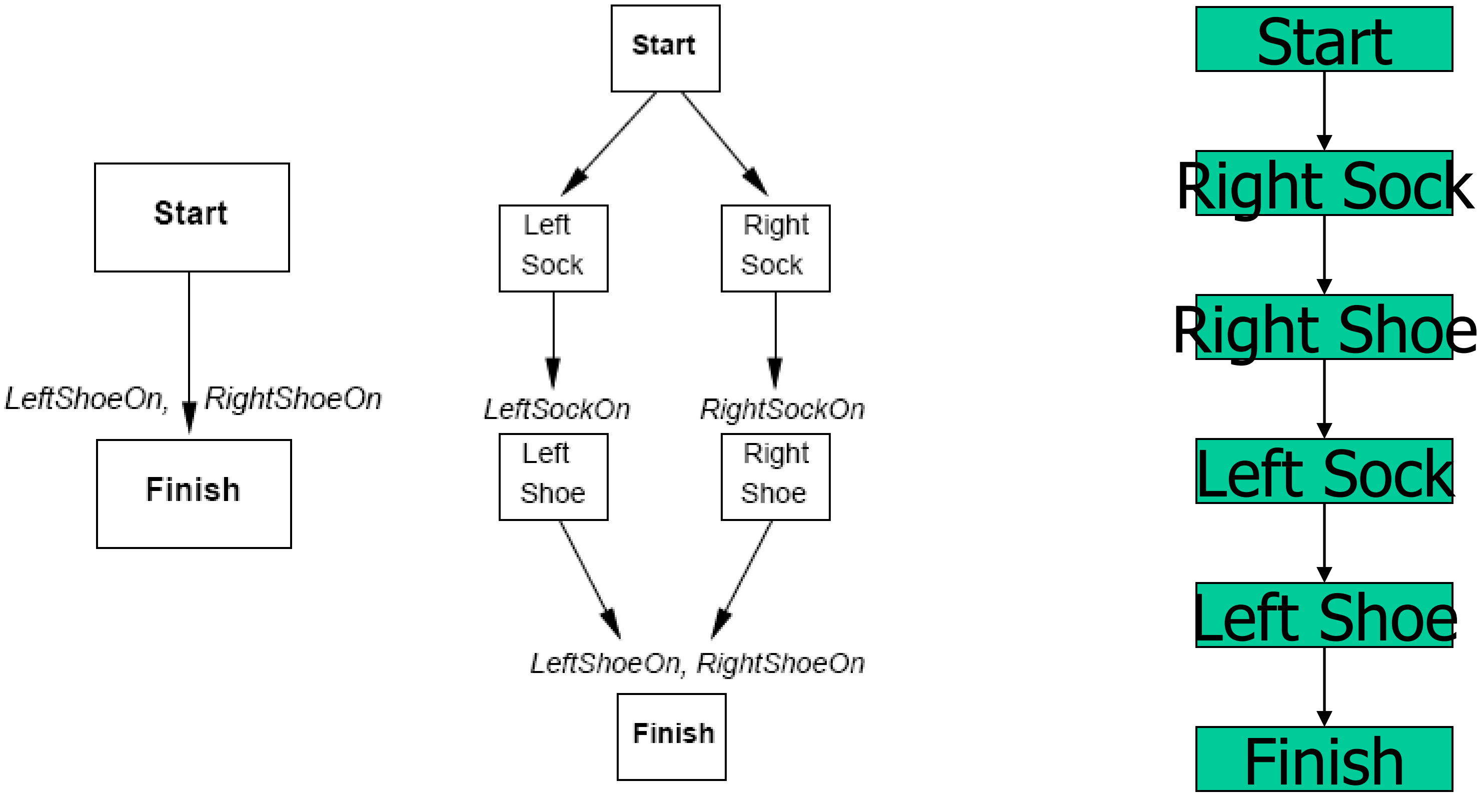


- Initial Plan:
  - Actions: {Start, Finish}
  - Ordering constraints: {Start $\prec$ Finish}
  - Causal links: {}
  - Open Preconditions: {LeftShoeOn,RightShoeOn}

POP as a Search Problem

- The successor function arbitrarily picks one open precondition *p* on an action B
- For every possible consistent action A that achieves *p*
  - It generates a successor plan adding the causal link $A \xrightarrow{p} B$ and the ordering constraint $A \prec B$
  - If A was not in the plan, it adds $Start \prec A$ and $A \prec Finish$
  - It resolves all conflicts between
    - the new causal link and all existing actions
    - between A and all existing causal links
  - Then it adds the successor states for combination of resolved conflicts
- It repeats until no open precondition exists

Partially Ordered Plans

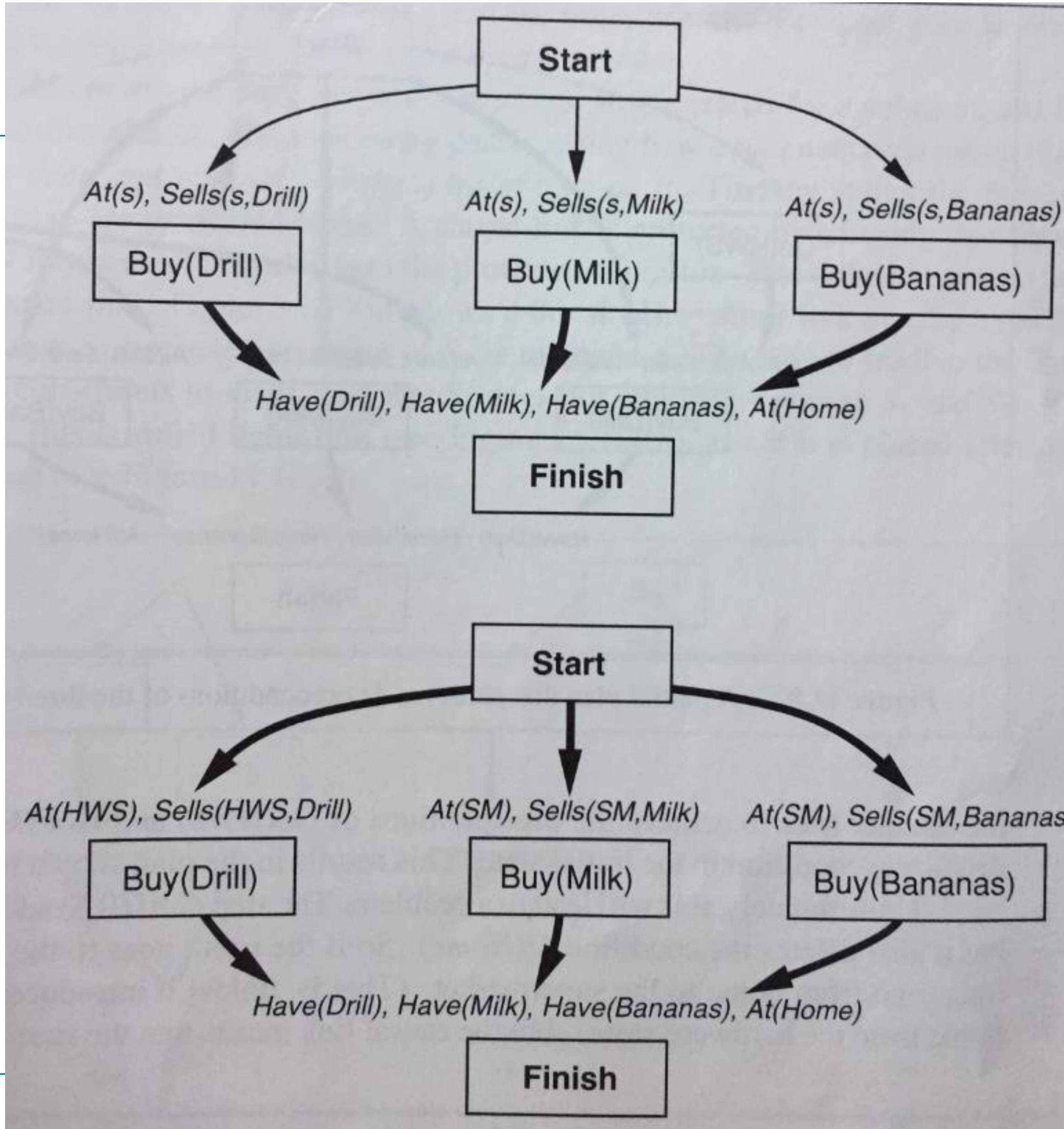


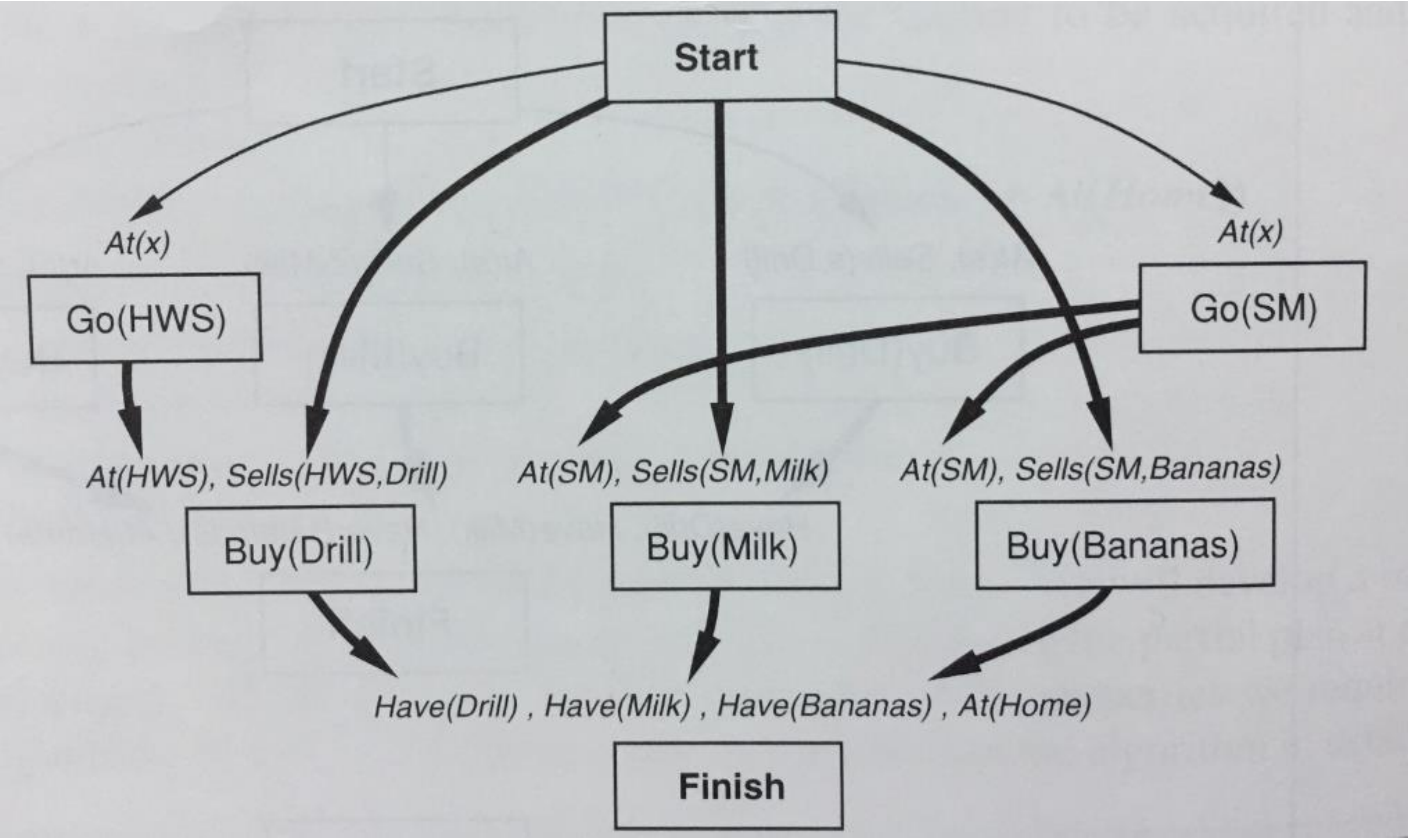
Start

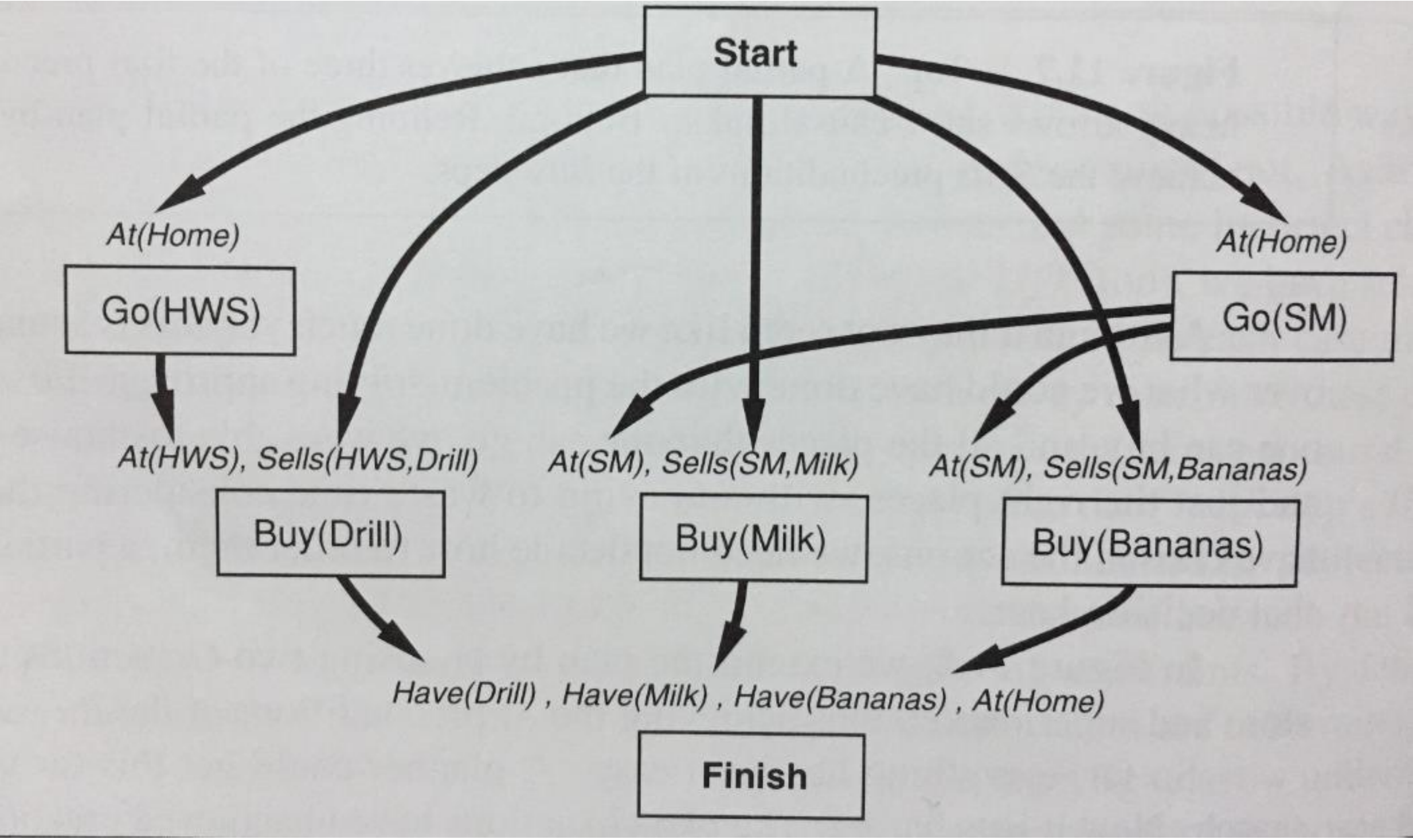
At(Home) *Sells(HWS,Drill)* *Sells(SM,Milk)* *Sells(SM,Ban.)*

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

Finish


Start
At(s), Sells(s,Drill)
At(s), Sells(s,Milk)
At(s), Sells(s,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill), Have(Milk), Have(Bananas), At(Home)
Finish
Start
At(HWS), Sells(HWS,Drill)
At(SM), Sells(SM,Milk)
At(SM), Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill), Have(Milk), Have(Bananas), At(Home)
Finish

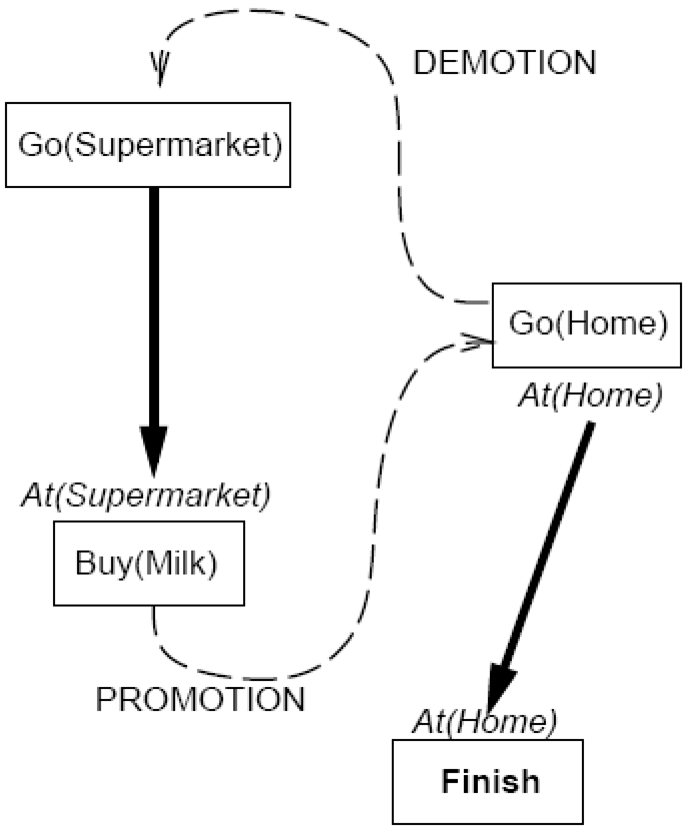

Start
At(x)
At(x)
Go(HWS)
Go(SM)
At(HWS), Sells(HWS,Drill)
At(SM), Sells(SM,Milk)
At(SM), Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill) , Have(Milk) , Have(Bananas) , At(Home)
Finish


Start
At(Home)
Go(HWS)
At(Home)
Go(SM)
At(HWS), Sells(HWS,Drill)
At(SM), Sells(SM,Milk)
At(SM), Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill) , Have(Milk) , Have(Bananas) , At(Home)
Finish

Clobbering

- A ***clobberer*** is a potentially intervening step that destroys the condition achieved by a causal link
  - Example Go(Home) clobbers At(Supermarket)

- Demotion
  - Put before Go(Supermarket)
- Promotion
  - Put after Buy(Milk)



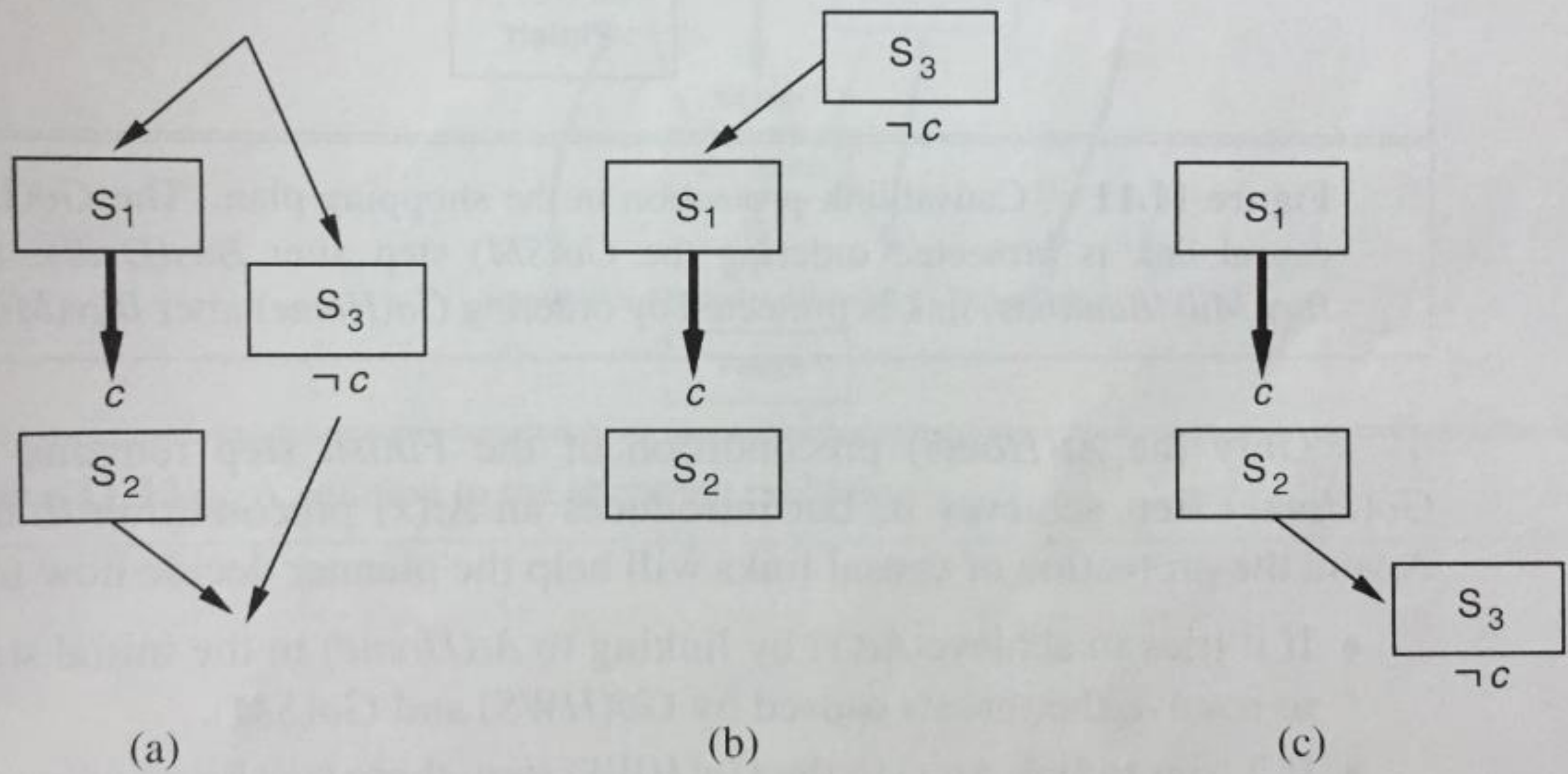
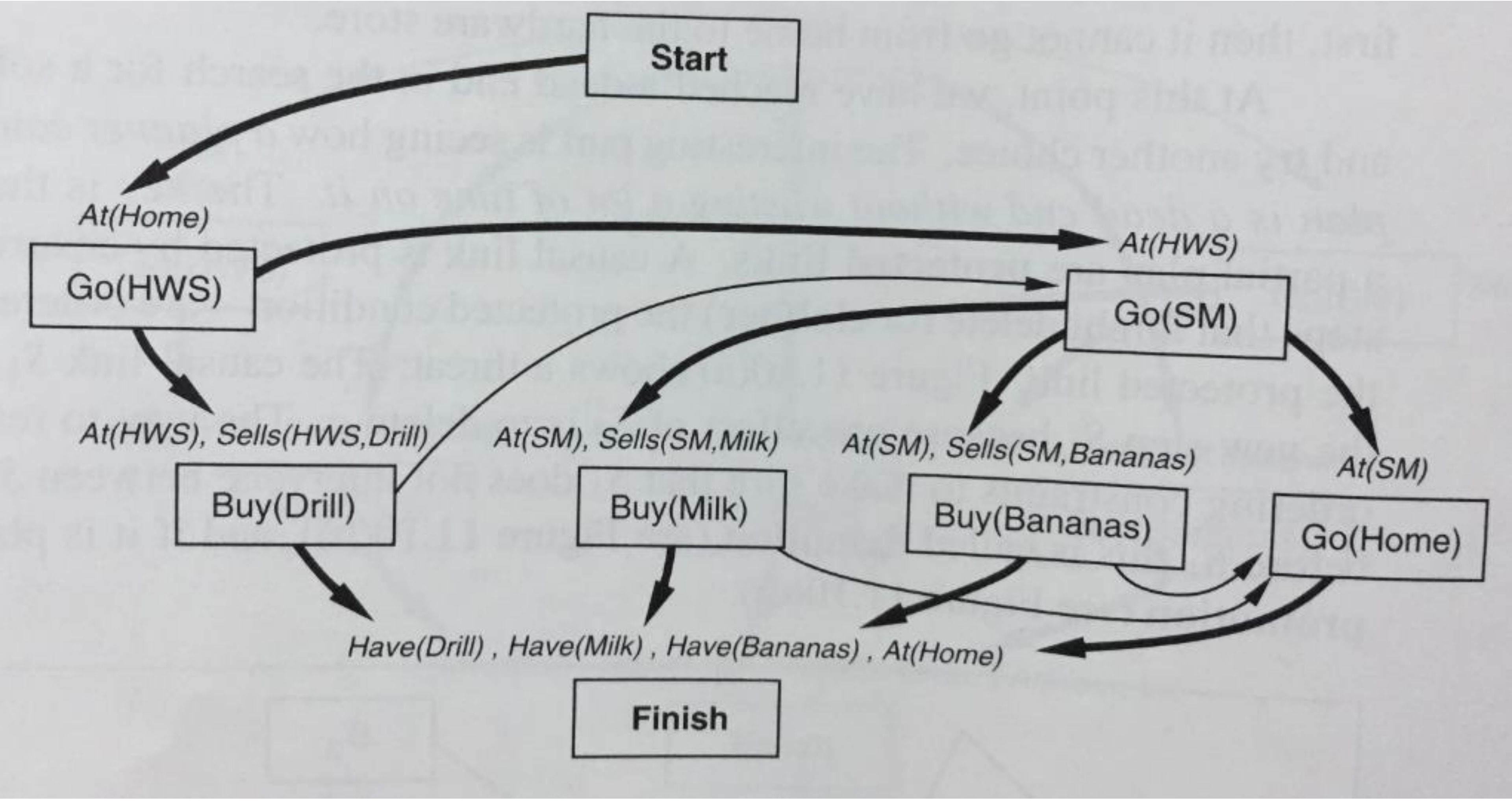
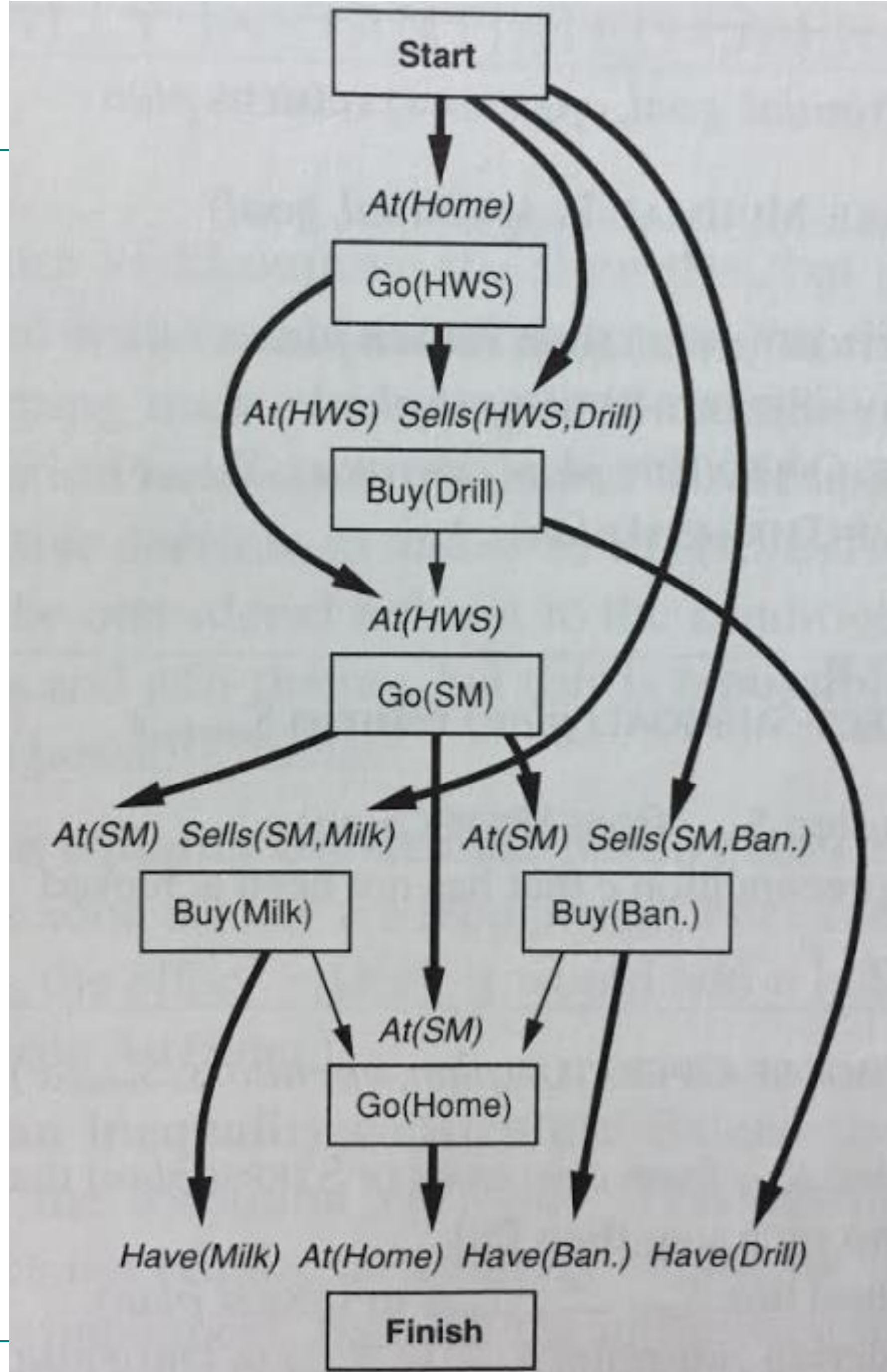
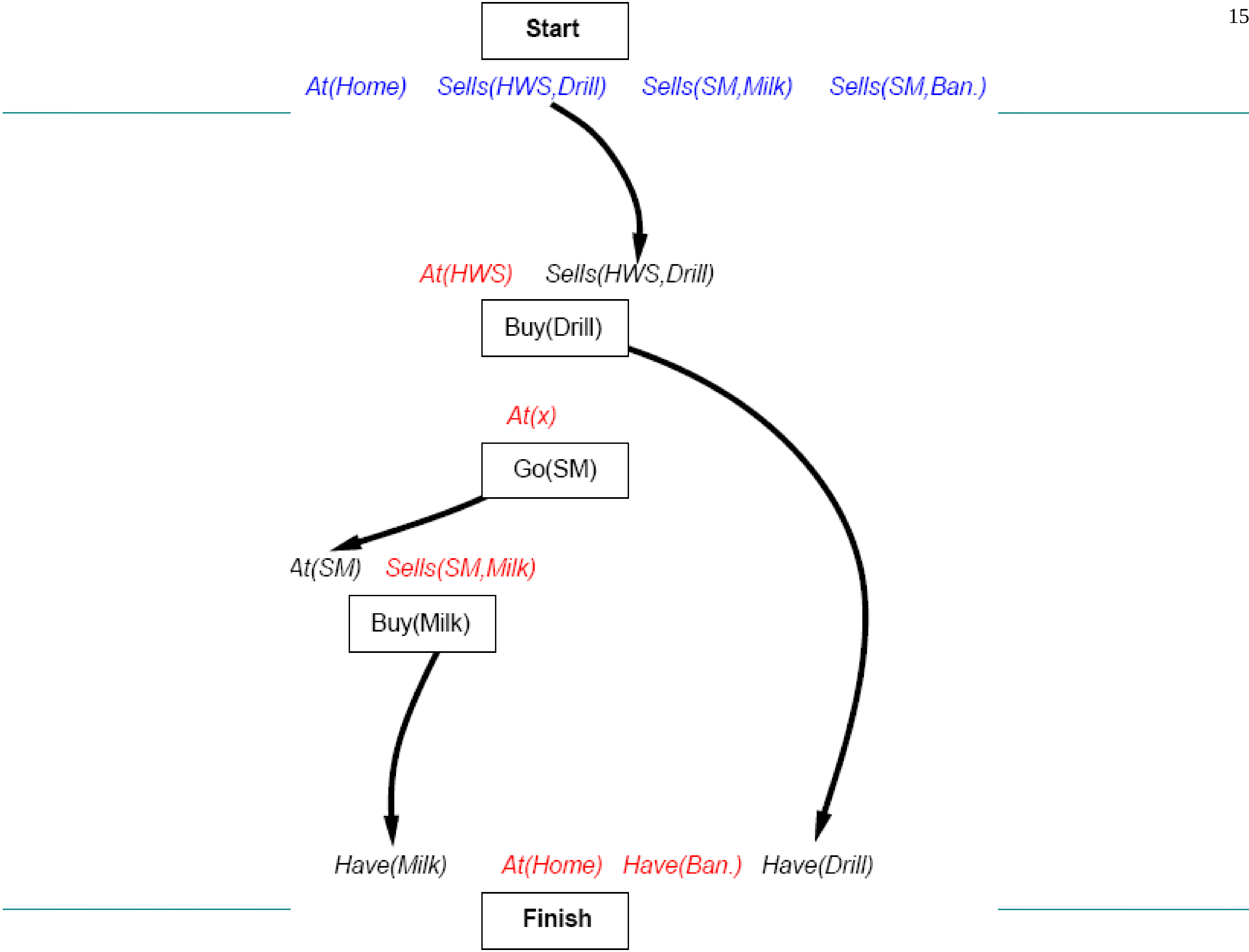
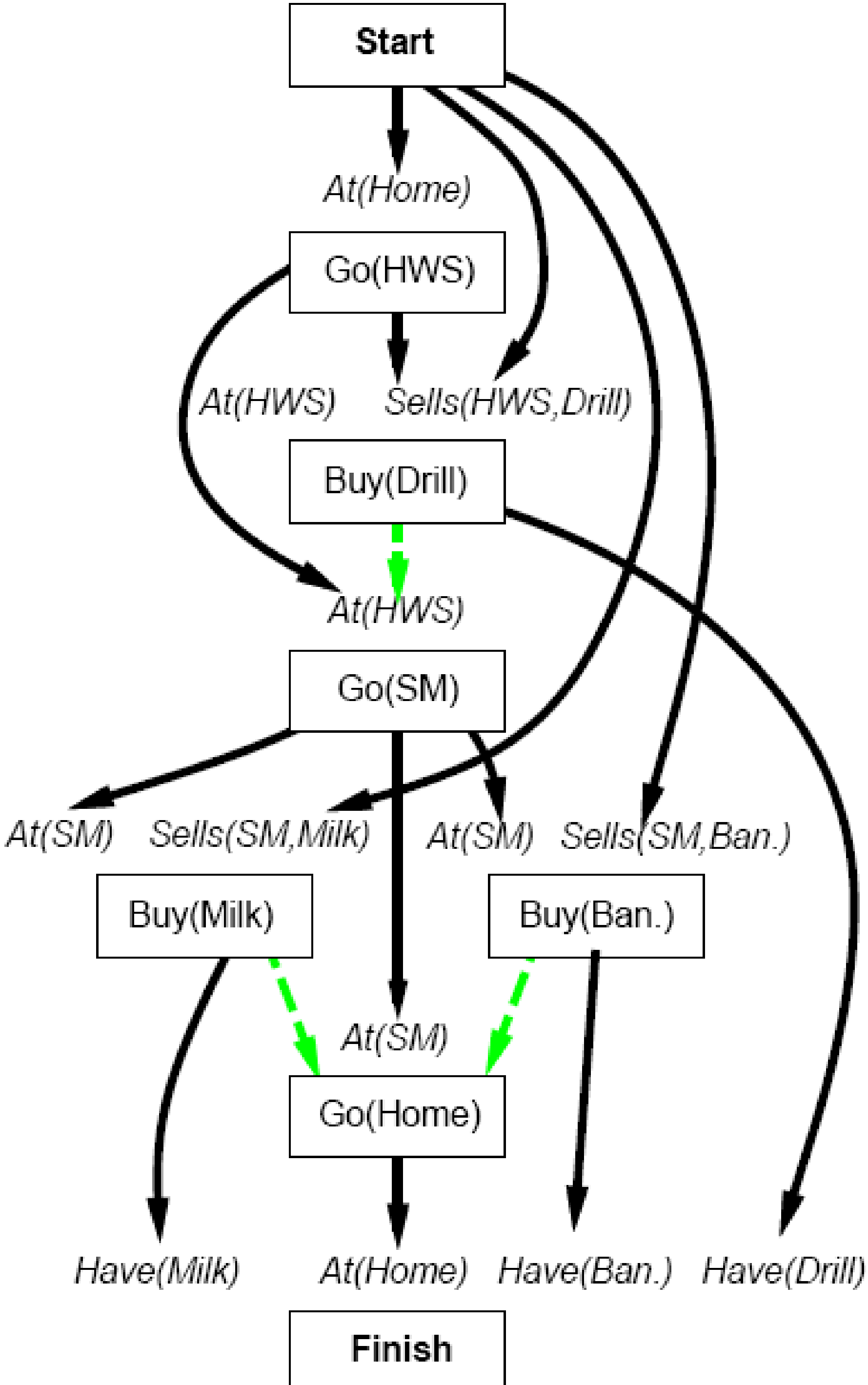


Figure 11.10 Protecting causal links. In (a), the step S_3 threatens a condition c that is established by S_1 and protected by the causal link from S_1 to S_2. In (b), S_3 has been demoted to come before S_1, and in (c) it has been promoted to come after S_2.


Start
At(Home)
Go(HWS)
At(HWS)
Go(SM)
At(HWS), Sells(HWS,Drill)
At(SM), Sells(SM,Milk)
At(SM), Sells(SM,Bananas)
At(SM)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Go(Home)
Have(Drill) , Have(Milk) , Have(Bananas) , At(Home)
Finish

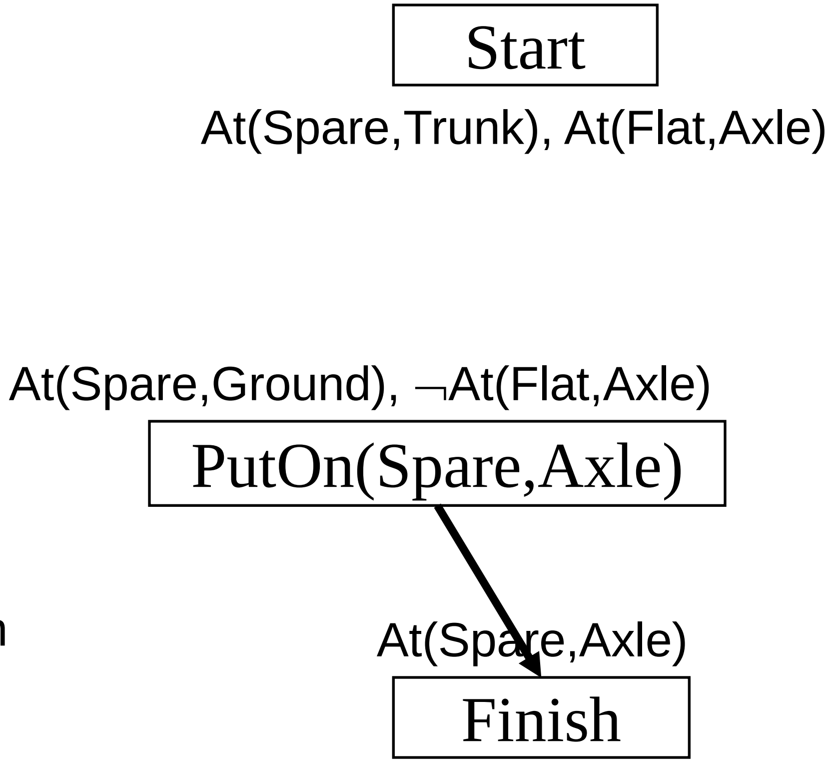

Start
At(Home)
Go(HWS)
At(HWS) Sells(HWS,Drill)
Buy(Drill)
At(HWS)
Go(SM)
At(SM) Sells(SM,Milk)
At(SM) Sells(SM,Ban.)
Buy(Milk)
Buy(Ban.)
At(SM)
Go(Home)
Have(Milk) At(Home) Have(Ban.) Have(Drill)
Finish


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


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

Example of POP: Flat tire problem

- Only one open precondition
- Only 1 applicable action



- Pick up At(Spare,Ground)
- Choose only applicable action Remove(Spare,Trunk)

Example: Spare tire problem

$Init(At(Flat,\ Axle) \wedge At(Spare,trunk))$

$Goal(At(Spare,Axle))$

$Action(Remove(Spare,Trunk)$

PRECOND: $At(Spare,Trunk)$

EFFECT: $\neg At(Spare,Trunk) \wedge At(Spare,Ground))$

$Action(Remove(Flat,Axle)$

PRECOND: $At(Flat,Axle)$

EFFECT: $\neg At(Flat,Axle) \wedge At(Flat,Ground))$

$Action(PutOn(Spare,Axle)$

PRECOND: $At(Spare,Groundp) \wedge \neg At(Flat,Axle)$

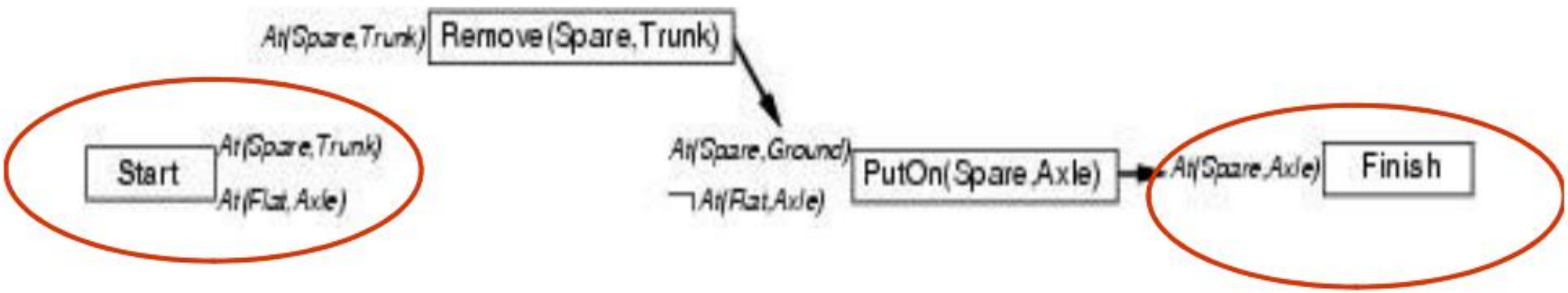
EFFECT: $At(Spare,Axle) \wedge \neg Ar(Spare,Ground))$

$Action(LeaveOvernight$

PRECOND:

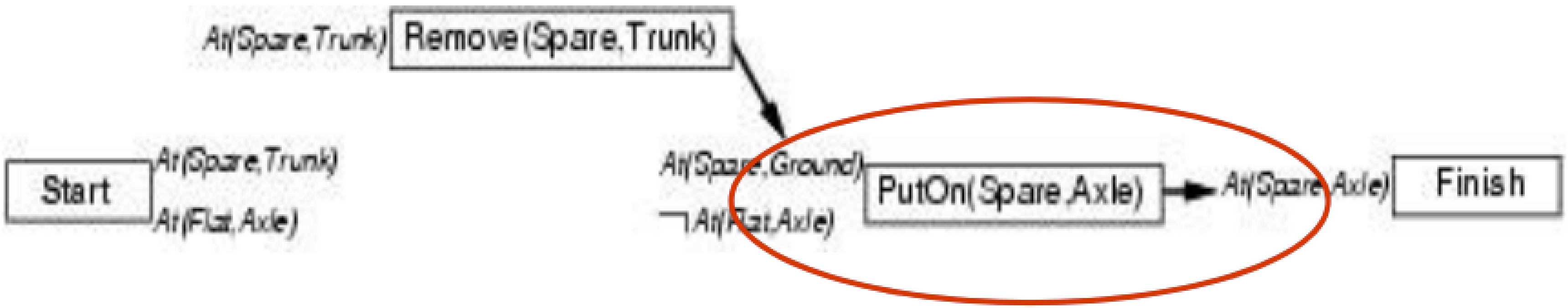
EFFECT: $\neg At(Spare,Ground) \wedge \neg At(Spare,Axle) \wedge \neg At(Spare,trunk) \wedge \neg At(Flat,Ground) \wedge \neg At(Flat,Axle))$

Solving the problem



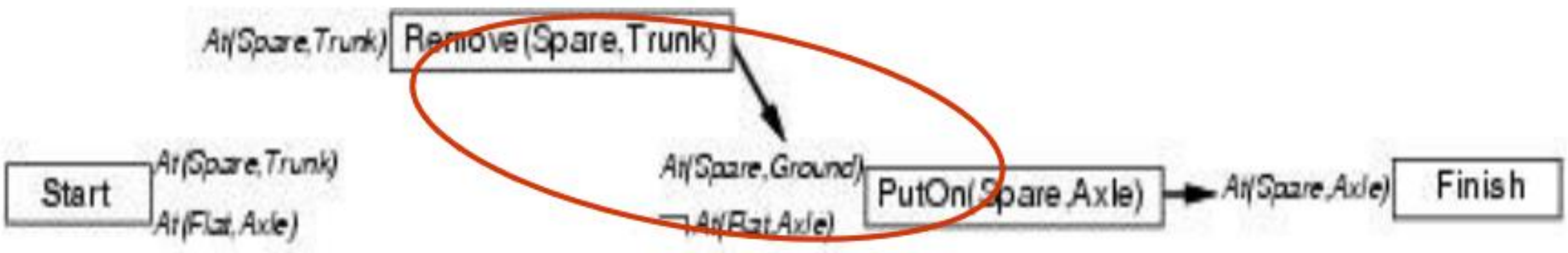
- Intial plan: Start with EFFECTS and Finish with PRECOND.

Solving the problem



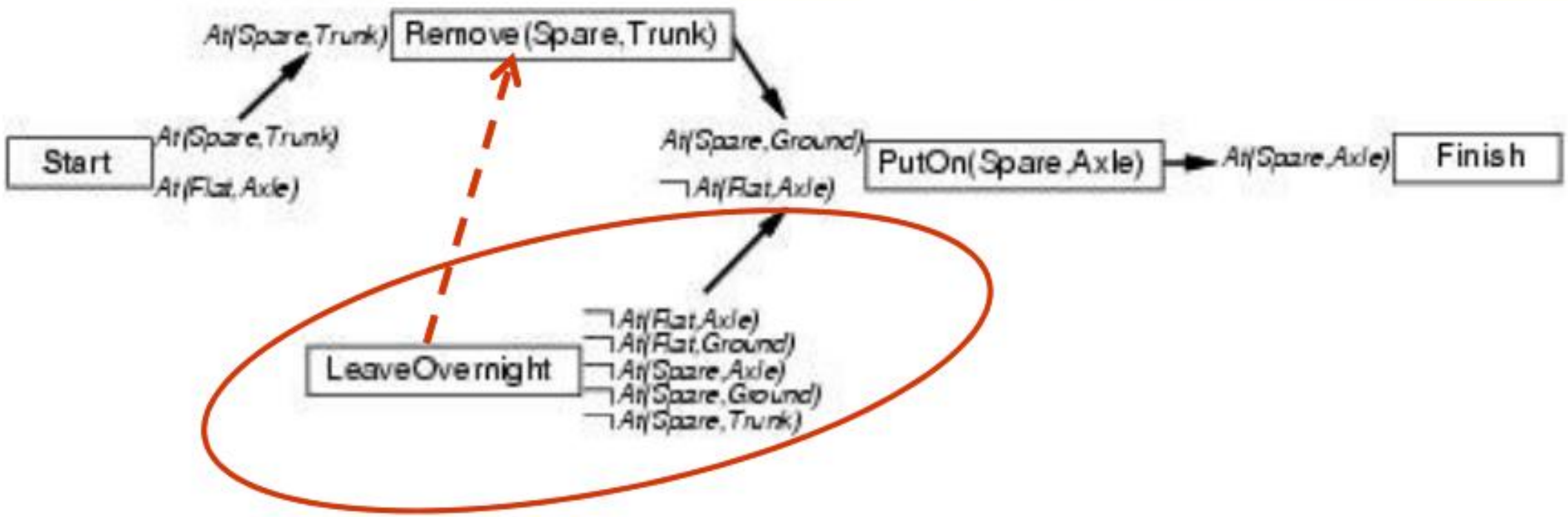
- Intial plan: Start with EFFECTS and Finish with PRECOND.
- Pick an open precondition: *At(Spare, Axle)*
- Only *PutOn(Spare, Axle)* is applicable
- Add causal link: $PutOn(Spare, Axle) \xrightarrow{At(Spare,Axle)} Finish$
- Add constraint : *PutOn(Spare, Axle) < Finish*

Solving the problem



- Pick an open precondition: *At(Spare, Ground)*
- Only *Remove(Spare, Trunk)* is applicable
- Add causal link: $Remove(Spare,Trunk) \xrightarrow{At(Spare,Ground)} PutOn(Spare,Axle)$
- Add constraint : *Remove(Spare, Trunk) < PutOn(Spare,Axle)*

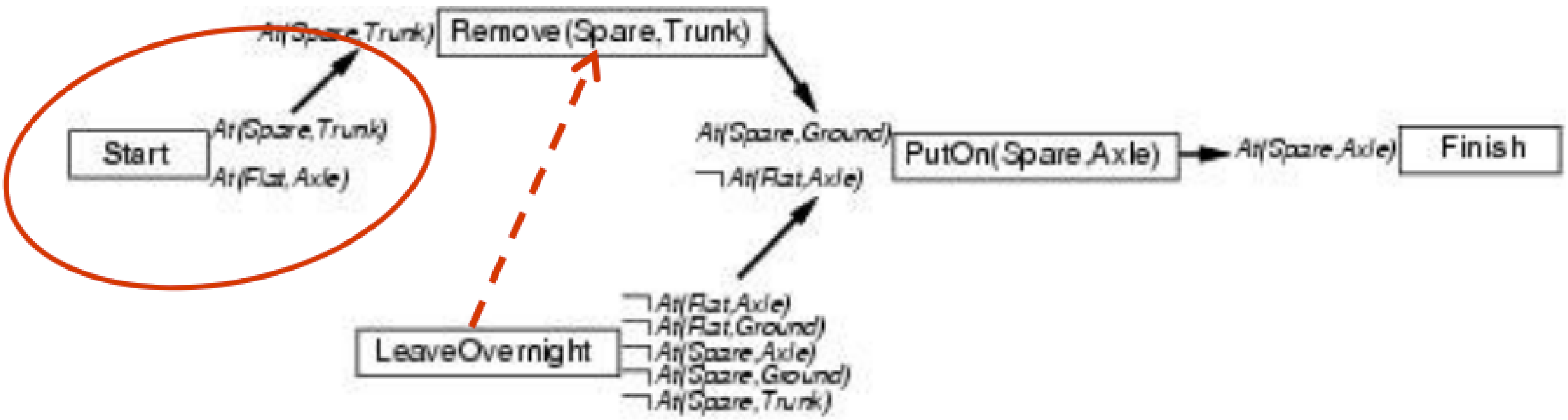
Solving the problem



- Pick an open precondition: ¬ *At(Flat, Axle)*
- *LeaveOverNight* is applicable
- conflict: $Remove(Spare,Trunk) \xrightarrow{At(Spare,Ground)} PutOn(Spare,Axle)$

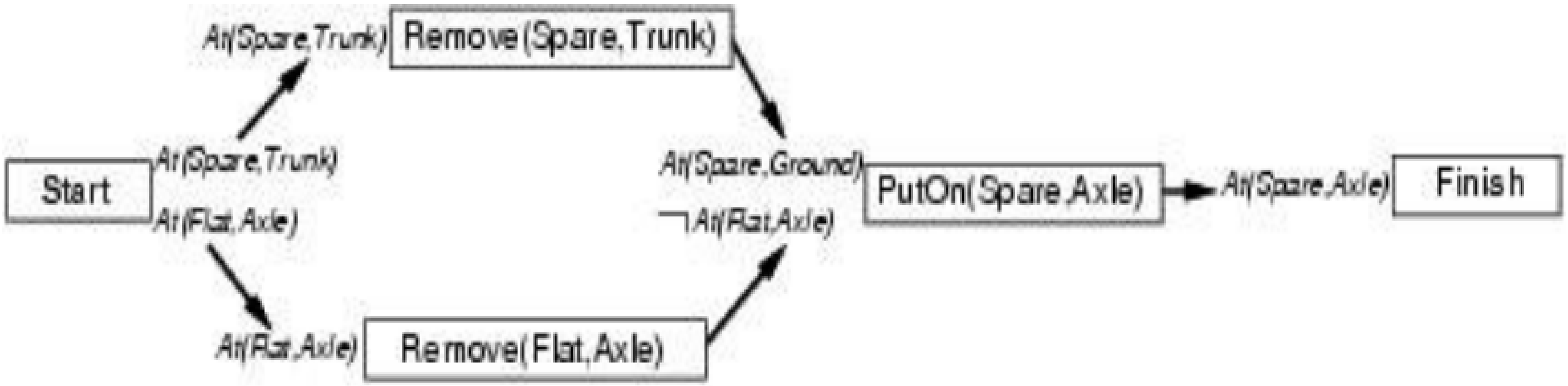
- Because *LeaveOverNight* also makes ¬ *At(Spare, Ground)*
- To resolve, add constraint : *LeaveOverNight < Remove(Spare, Trunk)*

Solving the problem



- Pick an open precondition: *At(Spare, Trunk)*
- Only *Start* is applicable
- Add causal link: $Start \xrightarrow{At(Spare,Trunk)} Remove(Spare,Trunk)$
- Conflict: of causal link with effect ¬ *At(Spare,Trunk)* in *LeaveOverNight*
  - *No re-ordering solution possible.*
- Backtrack to a prior move since there is no way to fix this

Solving the problem



- Backtracking step: Remove *LeaveOverNight* and its causal links
- Now try *Remove(Flat, Axle)* as a way to satisfy ¬ *At(Flat, Axle)*
- That one works… and the partial plan can be completed as above

POP Algorithm (1)

- Backtrack when fails to resolve a threat or find an operator
- Causal links
  - Recognize when to abandon a doomed plan without wasting time expanding irrelevant part of the plan
  - allow early pruning of inconsistent combination of actions
- When actions include variables, we need to find appropriate substitutions
  - Typically we try to delay commitments to instantiating a variable until we have no other choice (least commitment)
- POP is sound, complete, and systematic (no repetition)

POP Algorithm (2)

- Decomposes the problem (advantage)
- But does not represent states explicitly: it is hard to design heuristic to estimate distance from goal
  - Example: Number of open preconditions – those that match the effects of the start node. Not perfect (same problems as before)
- A heuristic can be used to choose which plan to refine (which precondition to pick-up):
  - Choose the most-constrained precondition, the one satisfied by the least number of actions. Like in CSPs!
  - When no action satisfies a precondition, backtrack!
  - When only one action satisfies a precondition, pick up the precondiction.

POP Algorithm

function POP(*initial, goal, operators*) **returns** *plan*

$plan \leftarrow$ MAKE-MINIMAL-PLAN(*initial, goal*)
loop do
  if SOLUTION?(*plan*) **then return** *plan*
  S_{need}, $c \leftarrow$ SELECT-SUBGOAL(*plan*)
  CHOOSE-OPERATOR(*plan, operators*, S_{need}, c)
  RESOLVE-THREATS(*plan*)
end

---

function SELECT-SUBGOAL(*plan*) **returns** S_{need}, c

pick a plan step S_{need} from STEPS(*plan*)
  with a precondition c that has not been achieved
return S_{need}, c

POP Algorithm

procedure CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, c)

choose a step S_{add} from *operators* or STEPS(*plan*) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step from *operators* **then**
  add S_{add} to STEPS(*plan*)
  add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

---

procedure RESOLVE-THREATS(*plan*)

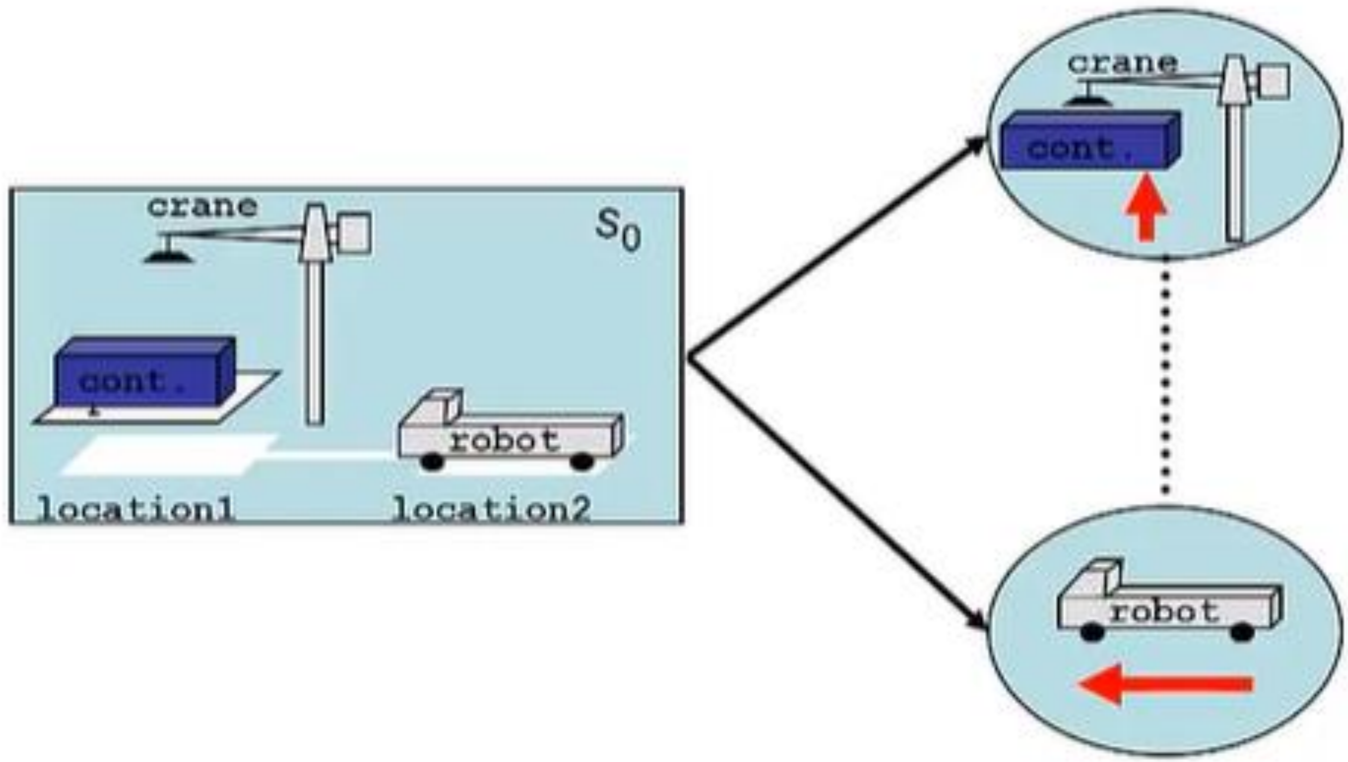
for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
  choose either
    Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
    Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
  if not CONSISTENT(*plan*) **then fail**
end

Case Study: DWR domain

Overview

- States with Internal Structure
- Operators with Structure
- Planning Domains and Problems
- Forward State-Space Search
- Backward State-Space Search

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/



Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Objects in the DWR Domain

- robots {`robot1`, `robot2`, …}:
  - container carrier carts for one container
  - can move between adjacent locations
- cranes {`crane1`, `crane2`, …}:
  - belongs to a single location
  - can move containers between robots and piles at same location
- containers {`cont1`, `cont2`, …}:
  - stacked in some pile on some pallet, loaded onto robot, or held by crane
- locations {`loc1`, `loc2`, …}:
  - storage area, dock, docked ship, or parking or passing area
- piles {`pile1`, `pile2`, …}:
  - attached to a single location
  - pallet at the bottom, possibly with containers stacked on top of it
- `pallet`:
  - at the bottom of a pile

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh,
https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Example: DWR Types in PDDL Syntax

```
(define (domain dock-worker-robot)

  (:requirements :strips :typing )

  (:types
    location    ;there are several connected locations
    pile        ;is attached to a location,
                ;it holds a pallet and a stack of containers
    robot       ;holds at most 1 container,
                ;only 1 robot per location
    crane       ;belongs to a location to pickup containers
    container )

…)
```

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh,
https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Example: DWR Predicates (PDDL)

```
(:predicates
    (adjacent ?l1 ?l2 - location)            ;location ?l1 is adjacent to ?l2
    (attached ?p - pile ?l - location)       ;pile ?p attached to location ?l
    (belong ?k - crane ?l - location)        ;crane ?k belongs to location ?l

    (at ?r - robot ?l - location)            ;robot ?r is at location ?l
    (occupied ?l - location)                 ;there is a robot at location ?l
    (loaded ?r - robot ?c - container )      ;robot ?r is loaded with container ?c
    (unloaded ?r - robot)                    ;robot ?r is empty

    (holding ?k - crane ?c - container)      ;crane ?k is holding a container ?c
    (empty ?k - crane)                       ;crane ?k is empty

    (in ?c - container ?p - pile)            ;container ?c is within pile ?p
    (top ?c - container ?p - pile)           ;container ?c is on top of pile ?p
    (on ?c1 - container ?c2 - container)     ;container ?c1 is on container ?c2
)
```

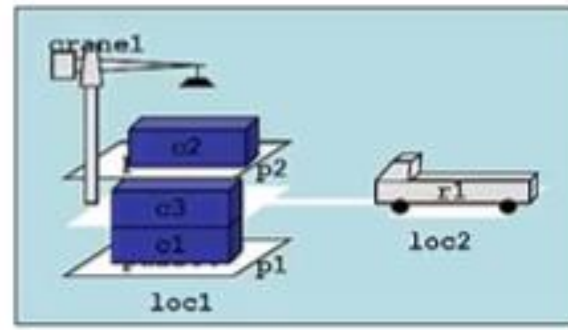
States in the STRIPS Representation

- Let $\mathcal{L}$ be a first-order language with finitely many predicate symbols, finitely many constant symbols, and no function symbols.
- A <u>state in a STRIPS planning domain</u> is a set of ground atoms of $\mathcal{L}$.
  - (ground) atom p <u>holds</u> in state s iff $p \in s$
  - s <u>satisfies</u> a set of (ground) literals g (denoted $s \models g$) if:
    - every positive literal in g is in s and
    - every negative literal in g is not in s.

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

DWR Example: STRIPS States

state = {
adjacent(loc1,loc2), adjacent(loc2, loc1),
attached(p1,loc1), attached(p2,loc1),
belong(crane1,loc1),
occupied(loc2),
empty(crane1),
at(r1,loc2),
unloaded(r1),
in(c1,p1),in(c3,p1),
on(c3,c1), on(c1,pallet),
top(c3,p1),
in(c2,p2),
on(c2,pallet),
top(c2,p2)}



Slide Credit: Artificial Intelligence Planning, The University of Edinburgh,
https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Operators and Actions in STRIPS Planning Domains

- A planning operator in a STRIPS planning domain is a triple o = (name(o), precond(o), effects(o)) where:
  - the name of the operator name(o) is a syntactic expression of the form $n(x_1,\dots,x_k)$ where n is a (unique) symbol and $x_1,\dots,x_k$ are all the variables that appear in o, and
  - the preconditions precond(o) and the effects effects(o) of the operator are sets of literals.
- An action in a STRIPS planning domain is a ground instance of a planning operator.

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

- move(r,l,m)
  - precond: adjacent(l,m), at(r,l), ¬occupied(m)
  - effects: at(r,m), occupied(m), ¬occupied(l), ¬at(r,l)

- load(k,l,c,r)
  - precond: belong(k,l), holding(k,c), at(r,l), unloaded(r)
  - effects: empty(k), ¬holding(k,c), loaded(r,c), ¬unloaded(r)

- put(k,l,c,d,p)
  - precond: belong(k,l), attached(p,l), holding(k,c), top(d,p)
  - effects: ¬holding(k,c), empty(k), in(c,p), top(c,p), on(c,d), ¬top(d,p)

Example: DWR Operator (PDDL)

```
;; moves a robot between two adjacent locations
(:action move
  :parameters (?r - robot ?from ?to - location)
  :precondition (and
    (adjacent ?from ?to) (at ?r ?from)
    (not (occupied ?to)))
  :effect (and
    (at ?r ?to) (occupied ?to)
    (not (occupied ?from)) (not (at ?r ?from)) ))
```

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Applicability and State Transitions

- Let L be a set of literals.
  - L^+ is the set of atoms that are positive literals in L and
  - L^- is the set of all atoms whose negations are in L.
- Let a be an action and s a state. Then a is applicable in s iff:
  - $\text{precond}^+(a) \subseteq s$; and
  - $\text{precond}^-(a) \cap s = \{\}$.
- The state transition function γ for an applicable action a in state s is defined as:
  - $\gamma(s,a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$

Finding Applicable Actions: Algorithm

```
function addApplicables(A, op, precs, σ, s)
  if precs+.isEmpty() then
     for every np in precs- do
        if s.falsifies(σ(np)) then return
     A.add(σ(op))
  else
     pp ← precs+.chooseOne()
     for every sp in s do
        σ' ← σ.extend(sp, pp)
        if σ'.isValid() then
           addApplicables(A, op, (precs - pp), σ', s)
```

Example: Applicable Actions

State

{adjacent(loc1,loc2),
adjacent(loc2, loc1),
attached(p1,loc1),
attached(p2,loc1),
belong(crane1,loc1),
occupied(loc2),
empty(crane1),
at(r1,loc2),
unloaded(r1),
in(c1,p1),in(c3,p1),
on(c3,c1), on(c1,pallet),
top(c3,p1),
in(c2,p2),
on(c2,pallet),
top(c2,p2)}

```
(:action move :parameters (?r - robot ?from ?to - location)
  :precondition (and (adjacent ?from ?to) (at ?r ?from)
    (not (occupied ?to)))
  :effect (and (at ?r ?to) (occupied ?to)
    (not (occupied ?from)) (not (at ?r ?from)) ))
```

pp ← (adjacent ?from ?to)

sp ← (adjacent loc1 loc2)	*sp* ← (adjacent loc2 loc1)
σ' ← {?from=loc1, ?to=loc2}	σ' ← {?from=loc2, ?to=loc1}
pp ← (at ?r ?from)	*pp* ← (at ?r ?from)
sp ← (at r1 loc2)	*sp* ← (at r1 loc2)
σ' not valid	σ' ← {?from=loc2, ?to=loc1, ?r=r1}
	σ(*np*) = (not (occupied loc1))
	A.add((move r1 loc2 loc1))

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh,
https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Classical Planning

- task: find solution for planning problem
- planning <u>problem</u>
  - initial <u>state</u>
    - atoms (relations, objects)
  - planning <u>domain</u>
    - operators (name, preconditions, effects)
  - goal
- <u>solution</u> (plan)

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

States in the STRIPS Representation

- Let $\mathcal{L}$ be a first-order language with finitely many predicate symbols, finitely many constant symbols, and no function symbols.
- A state in a STRIPS planning domain is a set of ground atoms of $\mathcal{L}$.
  - (ground) atom p holds in state s iff $p \in s$
  - s satisfies a set of (ground) literals g (denoted $s \models g$) if:
    - every positive literal in g is in s and
    - every negative literal in g is not in s.

Example: Domain (PDDL)

```
(define (domain dock-worker-robot)
    (:requirements :strips :typing )
    (:types location pile robot crane container)
    (:constants  pallet - container)
    (:predicates
        (adjacent ?l1  ?l2 - location)
        (attached ?p - pile ?l - location)
        (belong ?k - crane ?l - location)
        (at ?r - robot ?l - location)
        (occupied ?l - location)
        (loaded ?r - robot ?c - container )
        (unloaded ?r - robot)
        (holding ?k - crane ?c - container)
        (empty ?k - crane)
        (in ?c - container ?p - pile)
        (top ?c - container ?p - pile)
        (on ?k1 - container ?k2 - container));;

(:action move :parameters (?r - robot ?from ?to - location)
    :precondition (and (adjacent ?from ?to) (at ?r ?from) (not (occupied ?to)))
    :effect (and (at ?r ?to) (not (occupied ?from)) (occupied ?to) (not (at ?r ?from)) ))

(:action load :parameters (?k - crane ?l - location ?c - container ?r - robot)
    :precondition (and (at ?r ?l) (belong ?k ?l) (holding ?k ?c) (unloaded ?r))
    :effect (and  (loaded ?r ?c) (not (unloaded ?r)) (empty ?k) (not (holding ?k ?c))))

(:action unload :parameters (?k - crane ?l - location ?c - container ?r - robot)
    :precondition (and (belong ?k ?l) (at ?r ?l) (loaded ?r ?c) (empty ?k))
    :effect (and (unloaded ?r) (holding ?k ?c) (not (loaded ?r ?c))(not (empty ?k))))

(:action take :parameters (?k - crane ?l - location ?c ?else - container ?p - pile)
    :precondition (and (belong ?k ?l)(attached ?p ?l) (empty ?k) (in ?c ?p) (top ?c ?p)
    (on ?c ?else))
    :effect (and (holding ?k ?c) (top ?else ?p) (not (in ?c ?p)) (not (top ?c ?p)) (not (on ?c ?else))
    (not (empty ?k))))

(:action put :parameters (?k - crane ?l - location ?c ?else - container ?p - pile)
    :precondition (and (belong ?k ?l) (attached ?p ?l) (holding ?k ?c) (top ?else ?p))
    :effect (and (in ?c ?p) (top ?c ?p) (on ?c ?else) (not (top ?else ?p)) (not (holding ?k ?c))
    (empty ?k))))
```

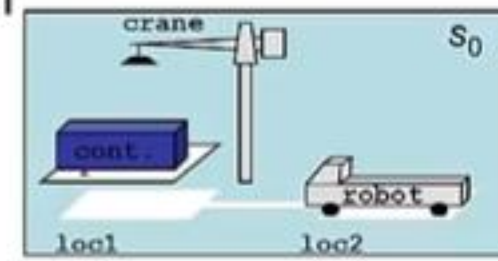
Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

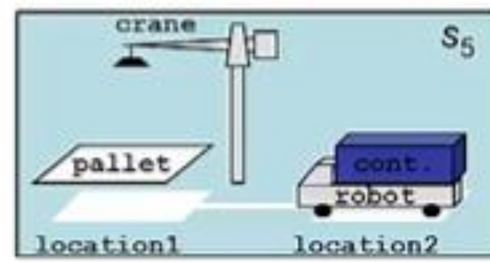
STRIPS Planning Problems

- A STRIPS planning problem is a triple $\mathcal{P}=(\Sigma,s_i,g)$ where:
  - $\Sigma=(S,A,\gamma)$ is a STRIPS planning domain on some first-order language $\mathcal{L}$
  - $s_i \in S$ is the initial state
  - g is a set of ground literals describing the goal such that the set of goal states is: $S_g=\{s \in S \mid s \text{ satisfies } g\}$

DWR Example: STRIPS Planning Problem

- Σ: STRIPS planning domain for DWR domain
- s_i: any state
  - example: s_0 = {attached(pile,loc1), in(cont,pile), top(cont,pile), on(cont,pallet), belong(crane,loc1), empty(crane), adjacent(loc1,loc2), adjacent(loc2,loc1), at(robot,loc2), occupied(loc2), unloaded(robot)}
- g: any subset of L
  - example: g = {¬unloaded(robot), at(robot,loc2)}, i.e. S_g={s_5}





Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Example: DWR Problem (PDDL)

```
;; a simple DWR problem with 1 robot and 2 locations
(define (problem dwrpb1)
    (:domain dock-worker-robot)
    (:objects
        r1 - robot
        l1 l2 - location
        k1 k2 - crane
        p1 q1 p2 q2 - pile
        ca cb cc cd ce cf pallet - container)
    (:init
        (adjacent l1 l2)
        (adjacent l2 l1)
        (attached p1 l1)
        (attached q1 l1)
        (attached p2 l2)
        (attached q2 l2)
        (belong k1 l1)
        (belong k2 l2)

        (in ca p1) (in cb p1) (in cc p1)
        (on ca pallet) (on cb ca) (on cc cb)
        (top cc p1)

        (in cd q1) (in ce q1) (in cf q1)
        (on cd pallet) (on ce cd) (on cf ce)
        (top cf q1)

        (top pallet p2)
        (top pallet q2)

        (at r1 l1)
        (unloaded r1)
        (occupied l1)

        (empty k1)
        (empty k2))

    ;; task is to move all containers to locations l2
    ;; ca and cc in pile p2, the rest in q2
    (:goal (and
        (in ca p2) (in cc p2)
        (in cb q2) (in cd q2) (in ce q2) (in cf q2))))
```

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Classical Plans

- A plan is any sequence of actions $\pi=\langle a_1,\ldots,a_k\rangle$, where $k\geq 0$.
  - The length of plan π is $|\pi|=k$, the number of actions.
  - If $\pi_1=\langle a_1,\ldots,a_k\rangle$ and $\pi_2=\langle a'_1,\ldots,a'_j\rangle$ are plans, then their concatenation is the plan $\pi_1\bullet\pi_2=\langle a_1,\ldots,a_k,a'_1,\ldots,a'_j\rangle$.
  - The extended state transition function for plans is defined as follows:
    - $\gamma(s,\pi)=s$ if $k=0$ (π is empty)
    - $\gamma(s,\pi)=\gamma(\gamma(s,a_1),\langle a_2,\ldots,a_k\rangle)$ if $k>0$ and a_1 applicable in s
    - $\gamma(s,\pi)=$undefined otherwise

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Classical Solutions

- Let $\mathcal{P}=(\Sigma,s_i,g)$ be a planning problem. A plan π is a solution for $\mathcal{P}$ if $\gamma(s_i,\pi)$ satisfies g.
  - A solution π is redundant if there is a proper subsequence of π is also a solution for $\mathcal{P}$.
  - π is minimal if no other solution for $\mathcal{P}$ contains fewer actions than π.

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh,
https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Classical Representations

- <u>propositional representation</u>
  - world state is set of propositions
  - action consists of precondition propositions, propositions to be added and removed
- <u>STRIPS representation</u>
  - like propositional representation, but first-order literals instead of propositions
- <u>state-variable representation</u>
  - state is tuple of state variables $\{x_1,\ldots,x_n\}$
  - action is partial function over states

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

State-Space Search

- idea: apply standard search algorithms (breadth-first, depth-first, A*, etc.) to planning problem:
  - search space is subset of state space
  - nodes correspond to world states
  - arcs correspond to state transitions
  - path in the search space corresponds to plan

State-Space Planning as a Search Problem

- given: statement of a planning problem $P=(O,s_i,g)$
- define the search problem as follows:
  - initial state: s_i
  - goal test for state s: s satisfies g
  - path cost function for plan π: $|\pi|$
  - successor function for state s: $\Gamma(s)$

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Reachable Successor States

- The successor function $\Gamma^m: 2^S \rightarrow 2^S$ for a STRIPS domain $\Sigma=(S,A,\gamma)$ is defined as:
  - $\Gamma(s)=\{\gamma(s,a) \mid a \in A \text{ and } a \text{ applicable in } s\}$ for $s \in S$
  - $\left.\begin{array}{l} \Gamma(\{s_1,\ldots,s_n\}) = \cup_{(k\in[1,n])}\Gamma(s_k) \\ \Gamma^0(\{s_1,\ldots,s_n\}) = \{s_1,\ldots,s_n\} \\ \Gamma^m(\{s_1,\ldots,s_n\}) = \Gamma(\Gamma^{m-1}(\{s_1,\ldots,s_n\})) \end{array}\right\} s_1,\ldots,s_n \in S$
- The transitive closure of Γ defines the set of all reachable states:
  - $\Gamma^>(s) = \cup_{(k\in[0,\infty])}\Gamma^k(\{s\})$ for $s \in S$

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

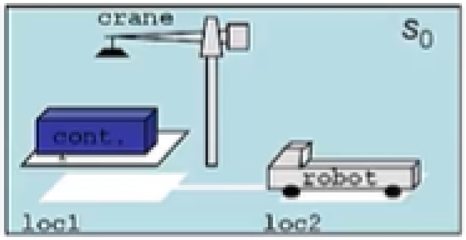
Forward State-Space Search Algorithm

function fwdSearch(O,s_i,g)
    state ← s_i
    plan ← ⟨⟩
    loop
        if *state*.satisfies(g) **then return** *plan*
        applicables ← {ground instances from O applicable in *state*}
        if *applicables*.isEmpty() **then return** failure
        action ← *applicables*.chooseOne()
        state ← γ(*state*,*action*)
        plan ← plan • ⟨*action*⟩

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

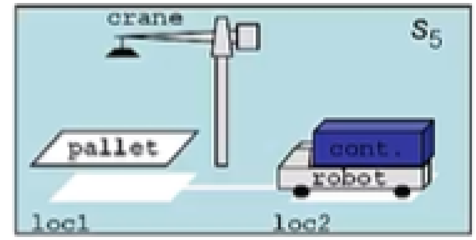
DWR Example: Forward Search

initial state:



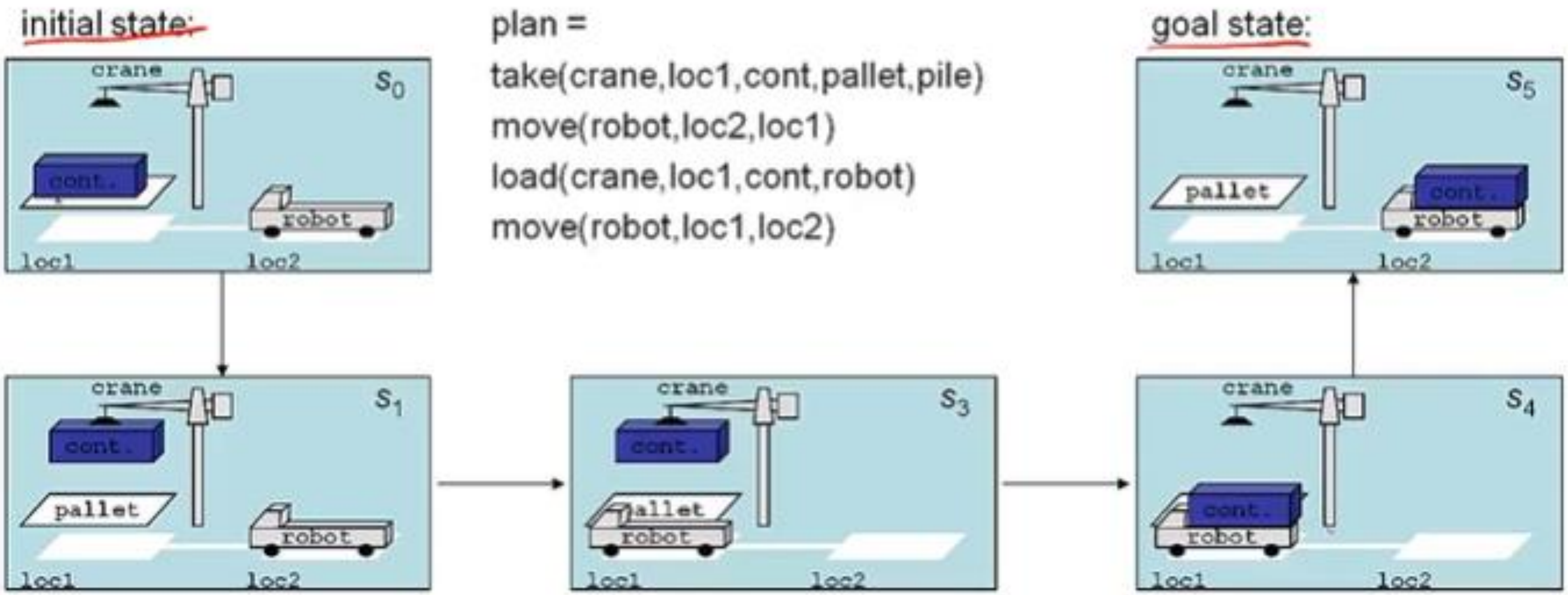
plan =

goal state:



Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

DWR Example: Forward Search



Slide Credit: Artificial Intelligence Planning, The University of Edinburgh,
https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Properties of Forward Search

- **Proposition**: fwdSearch is sound, i.e. if the function returns a plan as a solution then this plan is indeed a solution.
  - proof idea: show (by induction) $state=\gamma(s_i,plan)$ at the beginning of each iteration of the loop

- **Proposition**: fwdSearch is complete, i.e. if there exists solution plan then there is an execution trace of the function that will return this solution plan.
  - proof idea: show (by induction) there is an execution trace for which *plan* is a prefix of the sought plan

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Relevance and Regression Sets

- Let $\mathcal{P}=(\Sigma,s_i,g)$ be a STRIPS planning problem. An action $a \in A$ is <u>relevant for g</u> if
  - $g \cap \text{effects}(a) \neq \{\}$ and
  - $g^+ \cap \text{effects}^-(a) = \{\}$ and $g^- \cap \text{effects}^+(a) = \{\}$.
- The <u>regression set</u> of g for a relevant action $a \in A$ is:
  - $\gamma^{-1}(g,a)=(g - \text{effects}(a)) \cup \text{precond}(a)$

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Regression Function

- The regression function Γ^{-m} for a STRIPS domain $\Sigma=(S,A,\gamma)$ on L is defined as:
  - $\Gamma^{-1}(g)=\{\gamma^{-1}(g,a) \mid a\in A \text{ is relevant for } g\}$ for $g\in 2^L$
  - $\Gamma^{-0}(\{g_1,\ldots,g_n\})= \{g_1,\ldots,g_n\}$
  - $\Gamma^{-1}(\{g_1,\ldots,g_n\})= \cup_{(k\in[1,n])}\Gamma^{-1}(g_k)$
  - $\Gamma^{-m}(\{g_1,\ldots,g_n\})= \Gamma^{-1}(\Gamma^{-(m-1)}(\{g_1,\ldots,g_n\}))$

  $g_1,\ldots,g_n\in 2^L$
- The transitive closure of Γ^{-1} defines the set of all regression sets:
  - $\Gamma^{<}(g)= \cup_{(k\in[0,\infty])}\Gamma^{-k}(\{g\})$ for $g\in 2^L$

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

State-Space Planning as a Search Problem

- given: statement of a planning problem $P=(O,s_i,g)$
- define the search problem as follows:
  - initial search state: g
  - goal test for state s: s_i satisfies s
  - path cost function for plan π: $|\pi|$
  - successor function for state s: $\Gamma^{-1}(s)$

Example: Regression with Operators

- goal: at(robot,loc1)
- operator: move(*r*,*l*,*m*)
  - precond: adjacent(*l*,*m*), at(*r*,*l*), ¬occupied(*m*)
  - effects: at(*r*,*m*), occupied(*m*), ¬occupied(*l*), ¬at(*r*,*l*)
- actions: move(robot,*l*,loc1)
  - *l*=?
  - many options increase branching factor

- lifted backward search: use partially instantiated operators instead of actions

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Overview

- Search States: Partial Plans
- Plan Refinement Operations
- The Plan-Space Search Problem
- Flawless Partial Plans
- The PSP Algorithm
- PSP Implementation Details
- Partial-Order Planning

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

State-Space vs. Plan-Space Search

- state-space search:
  search through graph of nodes representing world states
- plan-space search:
  search through graph of partial plans
  - nodes: partially specified plans
  - arcs: plan refinement operations
  - solutions: partial-order plans

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Partial Plans

- plan: set of actions organized into some structure
- partial plan:
  - subset of the actions
  - subset of the organizational structure
    - temporal ordering of actions
    - rationale: what the action achieves in the plan
  - subset of variable bindings

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh,
https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Definition of Partial Plans

- A partial plan is a tuple $\pi = (A,\prec,B,L)$, where:
  - $A = \{a_1,\ldots,a_k\}$ is a set of partially instantiated planning operators;
  - $\prec$ is a set of ordering constraints on A of the form $(a_i \prec a_j)$;
  - B is a set of binding constraints on the variables of actions in A of the form $x=y$, $x \neq y$, or $x \in D_x$;
  - L is a set of causal links of the form $\langle a_i -[p] \rightarrow a_j \rangle$ such that:
    - a_i and a_j are actions in A;
    - the constraint $(a_i \prec a_j)$ is in $\prec$;
    - proposition p is an effect of a_i and a precondition of a_j; and
    - the binding constraints for variables in a_i and a_j appearing in p are in B.

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh,
https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Adding Actions

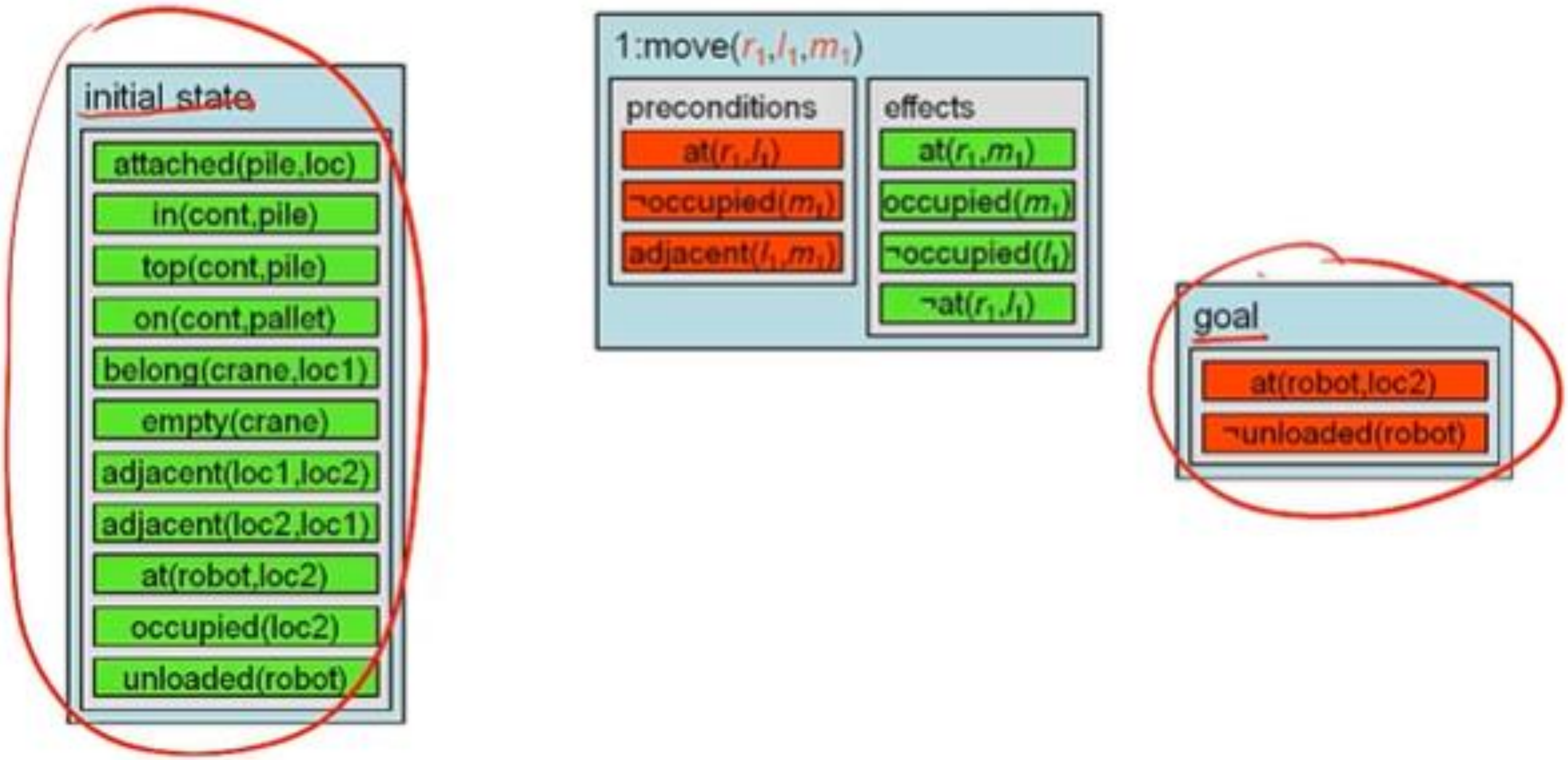
- partial plan contains actions
  - initial state
  - goal conditions
  - set of operators with different variables
- reason for adding new actions
  - to achieve unsatisfied preconditions
  - to achieve unsatisfied goal conditions

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh,
https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Adding Actions: Example

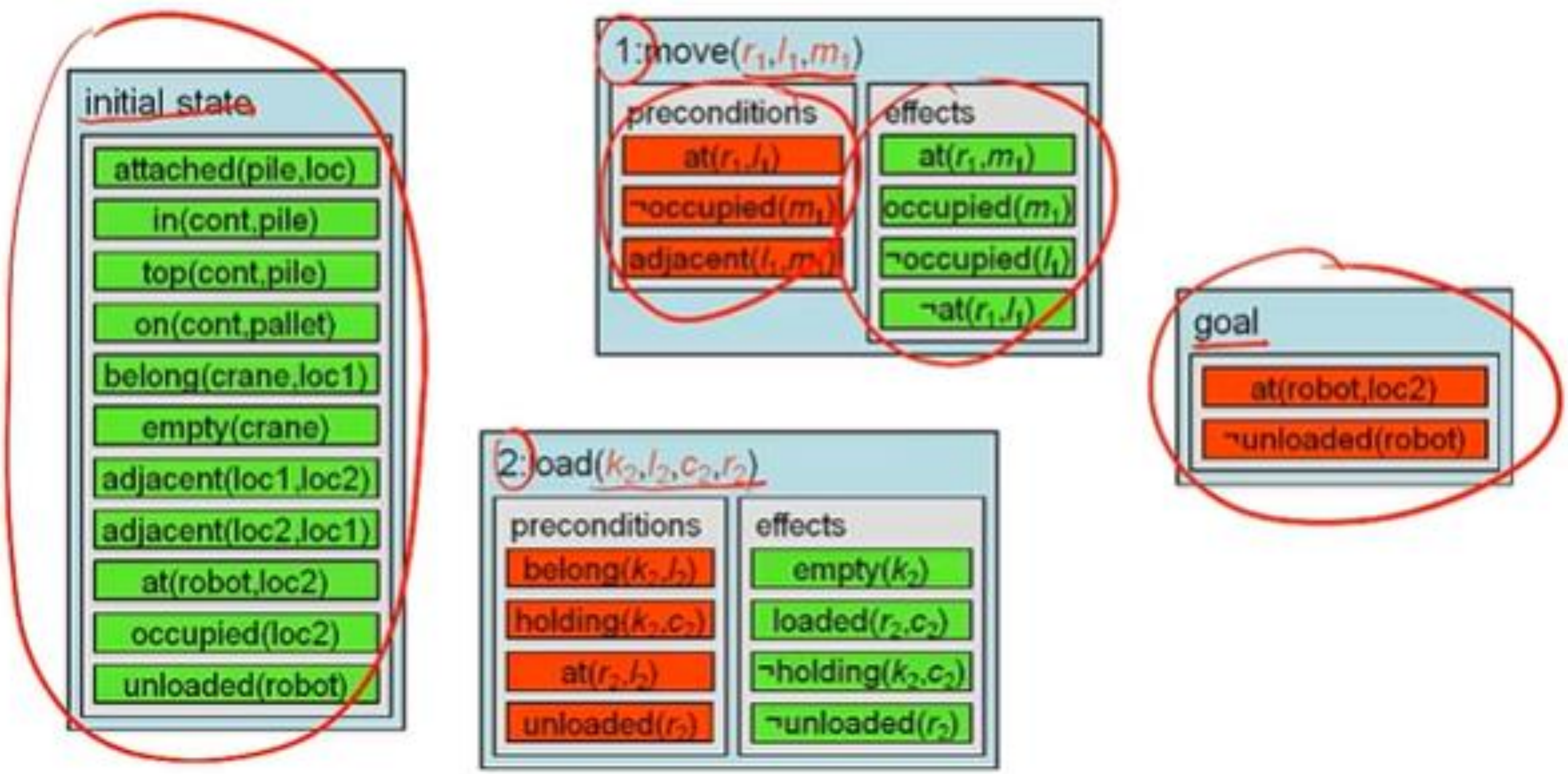
Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Adding Actions: Example



Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Adding Actions: Example



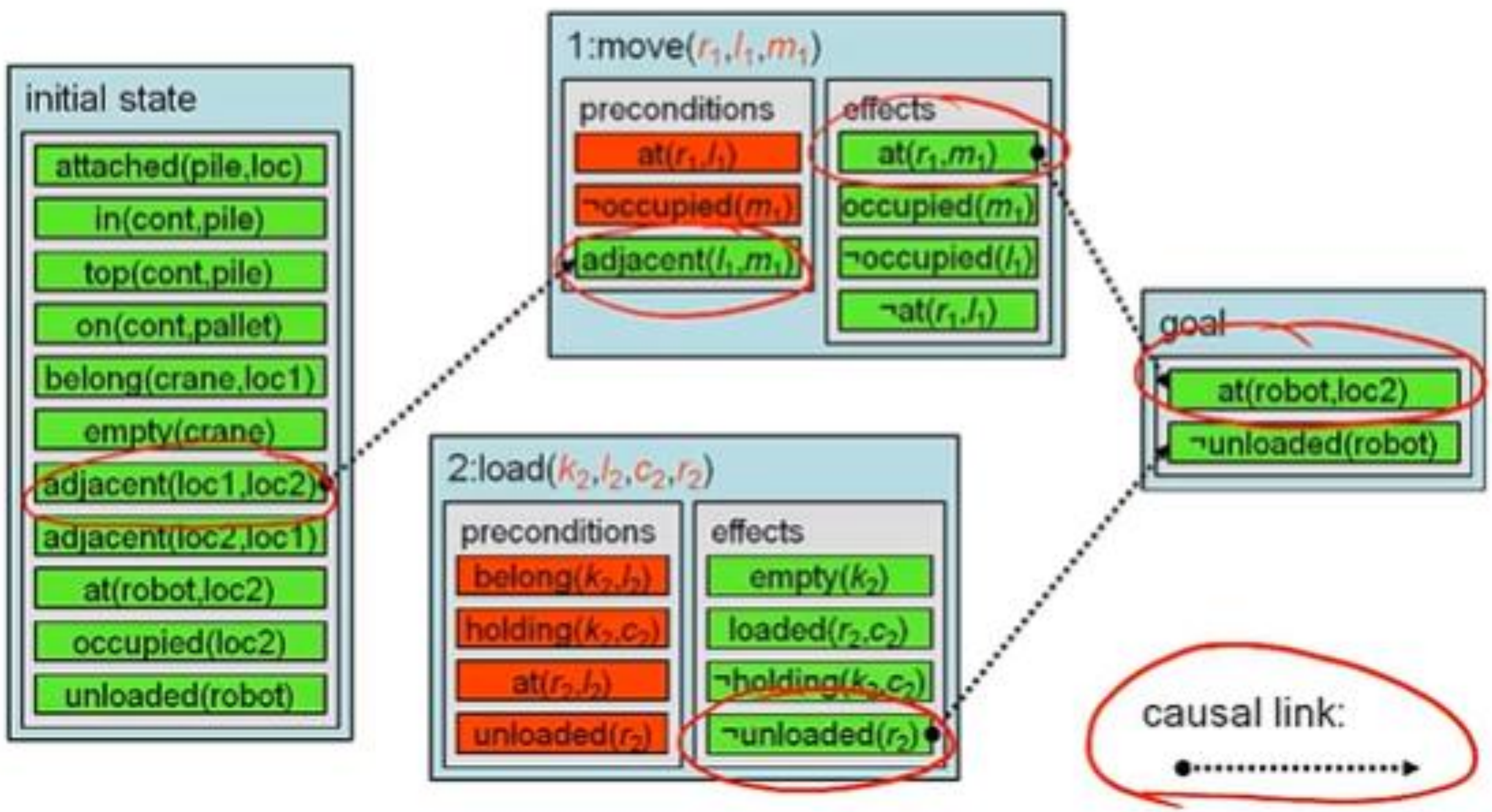
Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Adding Causal Links

- partial plan contains causal links
  - links from the provider
    - an effect of an action or
    - an atom that holds in the initial state
  - to the consumer
    - a precondition of an action or
    - a goal condition
- reasons for adding causal links
  - prevent interference with other actions

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Adding Causal Links: Example



Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Slide Credit: Artificial Intelligence Planning, The University of Edinburgh, https://media.ed.ac.uk/channel/Artificial-Intelligence-Planning/

Planning Graphs

- A planning graph consists of a sequence of levels that correspond to time-steps in the plan
- Level 0 is the initial state.
- Each level contains a set of literals and a set of actions
- Literals are those that could be true at the time step.
- Actions are those that their preconditions could be satisfied at the time step.
- Works only for propositional planning.

Example:Have cake and eat it too

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg Have(Cake)$
  EFFECT: $Have(Cake))$

Figure 11.11 The "have cake and eat cake too" problem.

The Planning graphs for "have cake",

- Persistence actions: Represent "inactions" by boxes: frame axiom
- Mutual exclusions (mutex) are represented between literals and actions.
- S1 represents multiple states
- Continue until two levels are identical. The graph levels off.
- The graph records the impossibility of certain choices using mutex links.
- Complexity of graph generation: polynomial in number of literals.

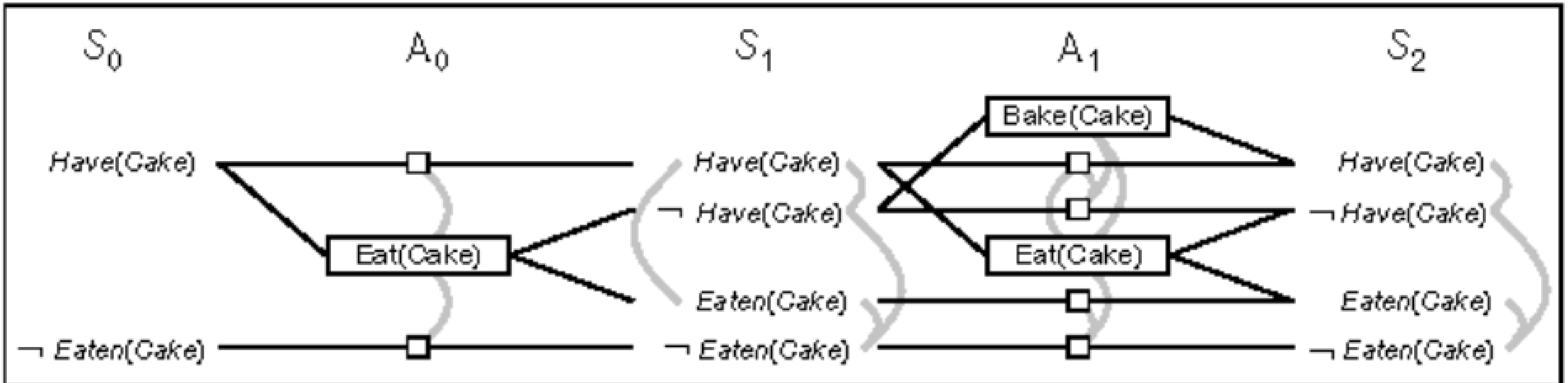


Figure 11.12 The planning graph for the "have cake and eat cake too" problem up to level S_2. Rectangles indicate actions (small squares indicate persistence actions) and straight lines indicate preconditions and effects. Mutex links are shown as curved gray lines.

Focus

- Building the Planning Graph
- Using it for Heuristic Estimation
- Using it for generating the plan
  - GraphPlan algorithm [Blum & Furst, 95]

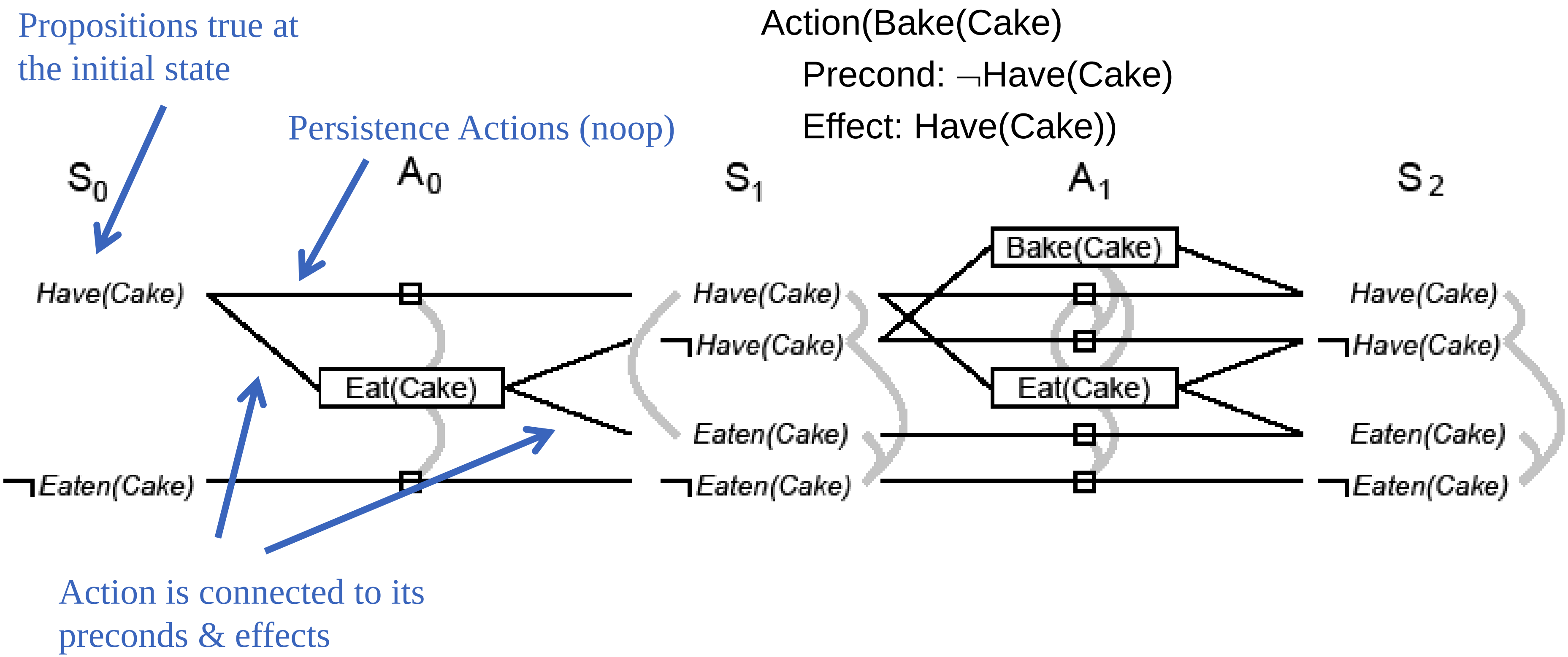
Example of a Planning Graph (1)

Init(Have(Cake))

Goal(Have(Cake)∧Eaten(Cake))

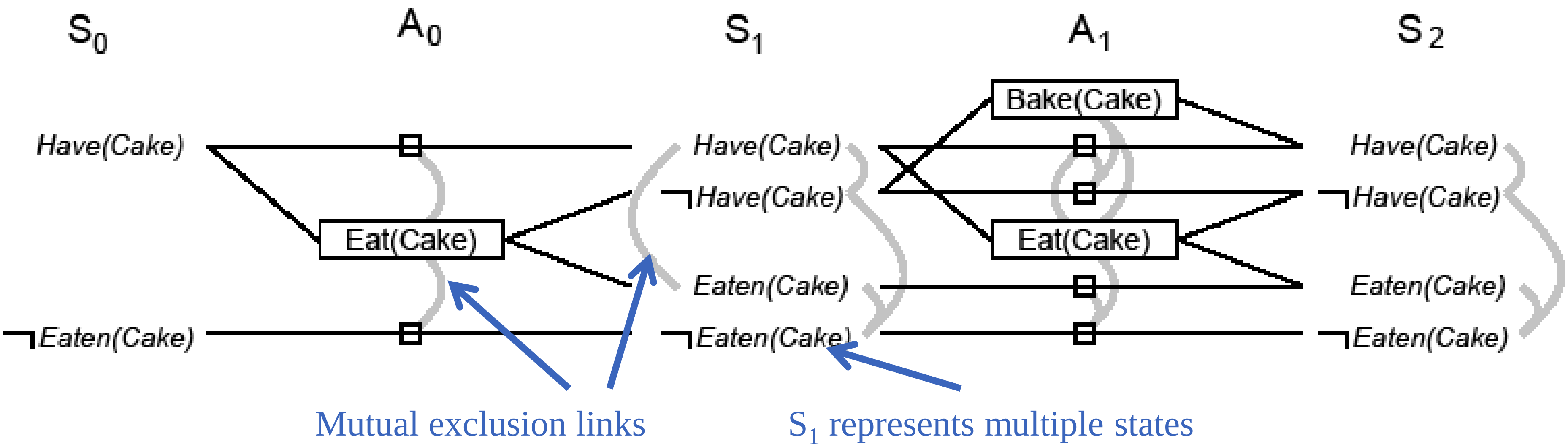
Action(Eat(Cake)
  Precond: Have(Cake)
  Effect: ¬Have(Cake)∧Eaten(Cake))

Action(Bake(Cake)
  Precond: ¬Have(Cake)
  Effect: Have(Cake))



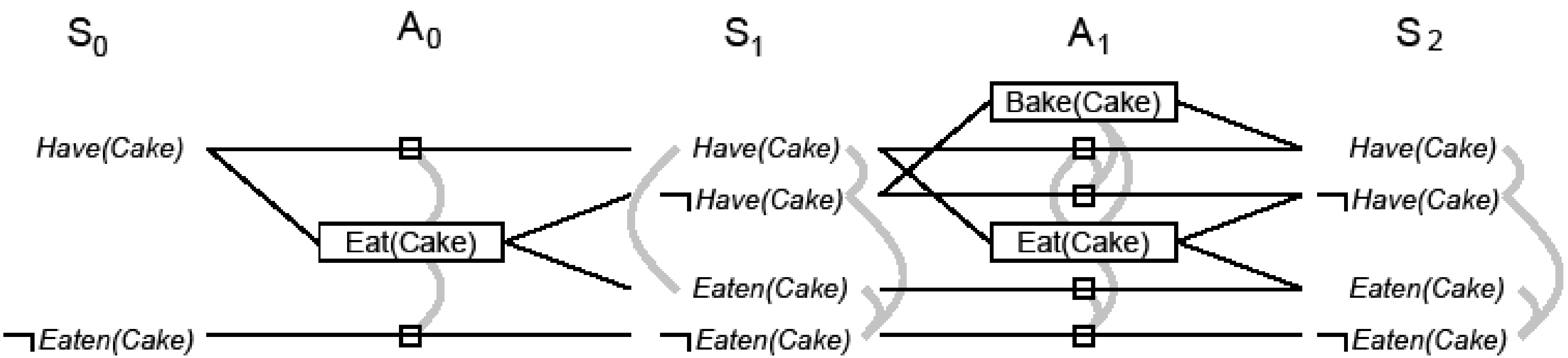
Example of a Planning Graph (2)

- At each state level, list all literals that may hold at that level
- At each action level, list all noops & all actions whose preconditions may hold at previous levels
- Repeat until plan 'levels off,' no new literals appears ($S_i = S_{i+1}$)
- Building the Planning Graph is a polynomial process
- Add (binary) mutual exclusion (mutex) links between conflicting actions and between conflicting literals



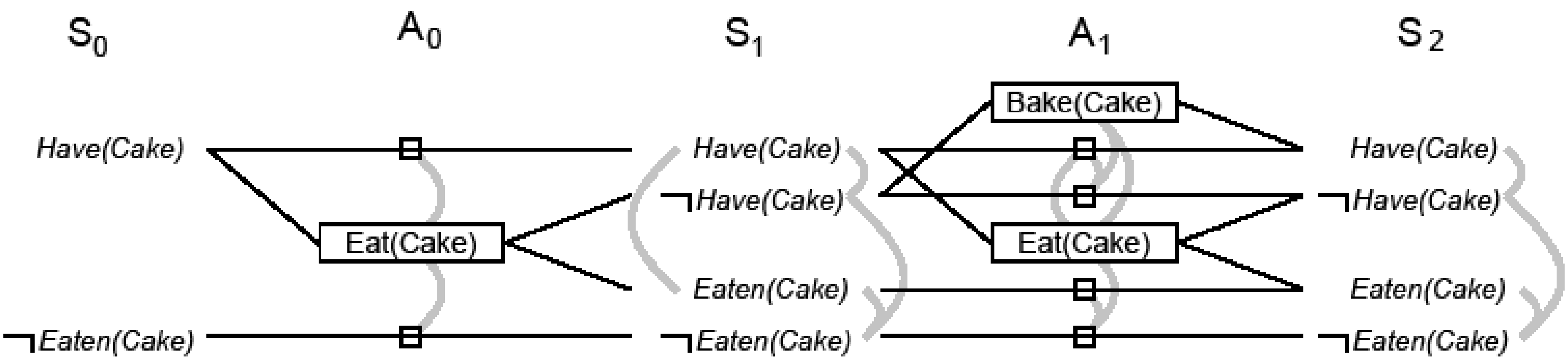
Mutex Links between Actions

1. **Inconsistent effects**: one action negates an effect of another
   – Eat(Cake) & noop of Have(Cake) disagree on effect Have(Cake)
2. **Interference**: An action effect negates the precondition of another
   – Eat(Cake) negates precondition of the noop of Have(Cake):
3. **Competing needs**: A precondition on an action is mutex with the precondition of another
   – Bake(Cake) & Eat(Cake): compete on Have(Cake) precondition



Mutex Links between Literals

1. Two literals are negation of each other
2. **Inconsistent support**: Each pair of actions that can achieve the two literals is mutex. Examples:
   - In S1, Have(Cake) & Eaten(Cake) are mutex
   - In S2, they are not because Bake(Cake) & the noop of Eaten(Cake) are not mutex



Defining Mutex relations

- A mutex relation holds between two actions on the same level iff any of the following holds:
  - **Inconsistency effect**: one action negates the effect of another. Example "eat cake and persistence of have cake"
  - **Interference**: One of the effect of one action is the negation of the precondition of the other. Example: eat cake and persistence of Have cake
  - **Competing needs:** one of the preconditions of one action is mutually exclusive with a precondition of another. Example: Bake(cake) and Eat(Cake).
  - **A mutex relation holds between 2 literals** at the same level iff one is the negation of the other or if each possible pair of actions that can achieve the 2 literals is mutually exclusive.

Planning Graph

- Is a sequence $\langle S_0, A_0, S_1, A_1, \ldots, S_i \rangle$ of levels
  - Alternating state levels & action levels
  - Levels correspond to time stamps
  - Starting at initial state
  - State level is a set of (propositional) literals
    - All those literals that could be true at that level
  - Action level is a set of (propositionalized) actions
    - All those actions whose preconditions appear in the state level (ignoring all negative interactions, etc.)
- Is special data structure used for
  1. Deriving better heuristic estimates
  2. Extract a solution to the planning problem: GRAPHPLAN algorithm

- Propositionalization may yield combinatorial explosition in the presence of a large number of objects

Planning graphs for heuristic estimation

- Estimate the cost of achieving a goal by the level in the planning graph where it appears.
- To estimate the cost of a conjunction of goals use one of the following:
- Max-level: take the maximum level of any goal (admissible)
- Sum-cost: Take the sum of levels (inadmissible)
- Set-level: find the level where they all appear without Mutex (admissible). Dominates max-level

- Graph plans are relaxation of the problem. Representing more than pair-wise mutex is not cost-effective

Planning Graph for Heuristic Estimation

- A literal that does not appear in the final level cannot be achieved by any plan
  - State-space search: Any state containing an unachievable literal has cost $h(n)=\propto$
  - POP: Any plan with an unachievable open condition has cost $h(n)=\propto$
- The estimate cost of any goal literal is the first level at which it appears
  - Estimate is admissible for individual literals
  - Estimate can be improved by serializing the graph (serial planning graph: one action per level) by adding mutex between all actions in a given level
- The estimate of a conjunction of goal literals
  - Three heuristics: max level, level sum, set level

Estimate of Conjunction of Goal Literals

- Max-level
  - The largest level of a literal in the conjunction
  - Admissible, not very accurate
- Level sum
  - Under subgoal independence assumption, sums the level costs of the literals
  - Inadmissible, works well for largely decomposable problems
- Set level
  - Finds the level at which all literals appear w/o any pair of them being mutex
  - Dominates max-level, works extremely well on problems where there is a great deal of interaction among subplans

The graphplan algorithm

```
function GRAPHPLAN(problem) returns solution or failure

  graph ← INITIAL-PLANNING-GRAPH(problem)
  goals ← GOALS[problem]
  loop do
     if goals all non-mutex in last level of graph then do
        solution ← EXTRACT-SOLUTION(graph, goals, LENGTH(graph))
        if solution ≠ failure then return solution
        else if NO-SOLUTION-POSSIBLE(graph) then return failure
     graph ← EXPAND-GRAPH(graph, problem)
```

Figure 11.13 The GRAPHPLAN algorithm. GRAPHPLAN alternates between a solution extraction step and a graph expansion step. EXTRACT-SOLUTION looks for whether a plan can be found, starting at the end and searching backwards. EXPAND-GRAPH adds the actions for the current level and the state literals for the next level.

Planning graph for spare tire

goal: at(spare,axle)

- S2 has all goals and no mutex so we can try to extract solutions
- Use either CSP algorithm with actions as variables
- Or search backwards

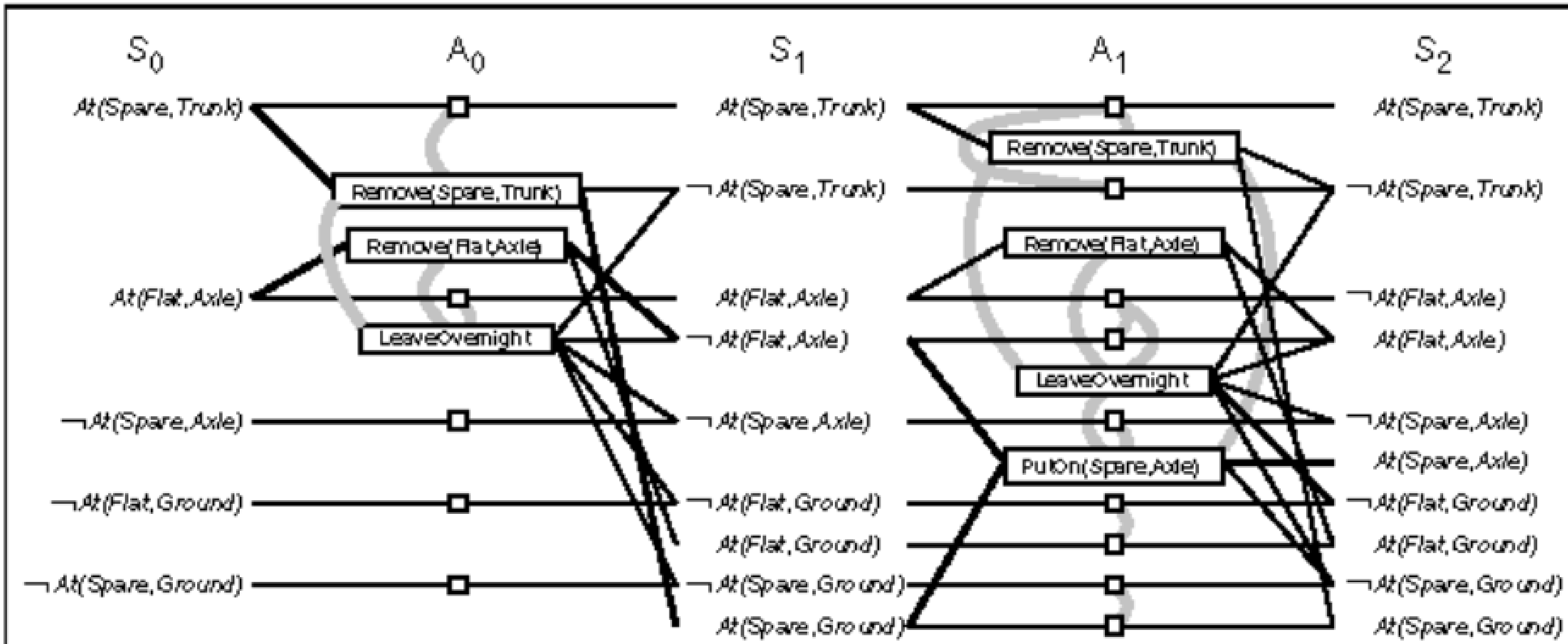


Figure 11.14 The planning graph for the spare tire problem after expansion to level S_2. Mutex links are shown as gray lines. Only some representative mutexes are shown, because the graph would be too cluttered if we showed them all. The solution is indicated by bold

Search planning-graph backwards with heuristics

- How to choose an action during backwards search:
  - Use greedy algorithm based on the level cost of the literals.
- For any set of goals:
- 1. Pick first the literal with the highest level cost.
- 2. To achieve the literal, choose the action with the easiest preconditions first (based on sum or max level of precond literals).

Properties of planning graphs; termination

- Literals increase monotonically
  - Once a literal is in a level it will persist to the next level
- Actions increase monotonically
  - Since the precondition of an action was satisfied at a level and literals persist the action's precond will be satisfied from now on
- Mutexes decrease monotonically:
  - If two actions are mutex at level Si, they will be mutex at all previous levels at which they both appear
- Because literals increase and mutex decrease it is guaranteed that we will have a level where all goals are non-mutex

GRAPHPLAN Algorithm

GRAPHPLAN (*problem*) **returns** *solution* or *failure*

graph ← InitPlanningGRAPH (*problem*)

goals ← GOALS[*problem*]

loop do

  if *goals* all non-mutex in last level of graph **then do**

    solution ← EXTRACTSOLUTION(*graph*,*goals*,LENGTH(*graph*))

    if *solution* ≠ *failure* **then return** *solution*

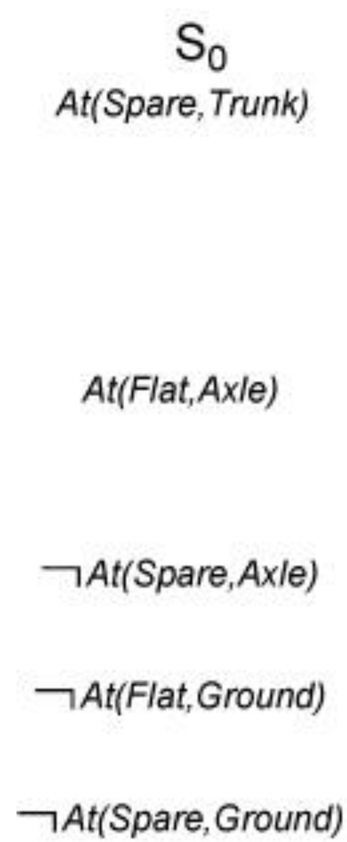
    else if NoSolutionPossible(*graph)* **then return** *failure*

  graph ← ExpandGarph(*graph*,*problem*)

- Two main stages
  1. Extract solution
  2. Expand the graph

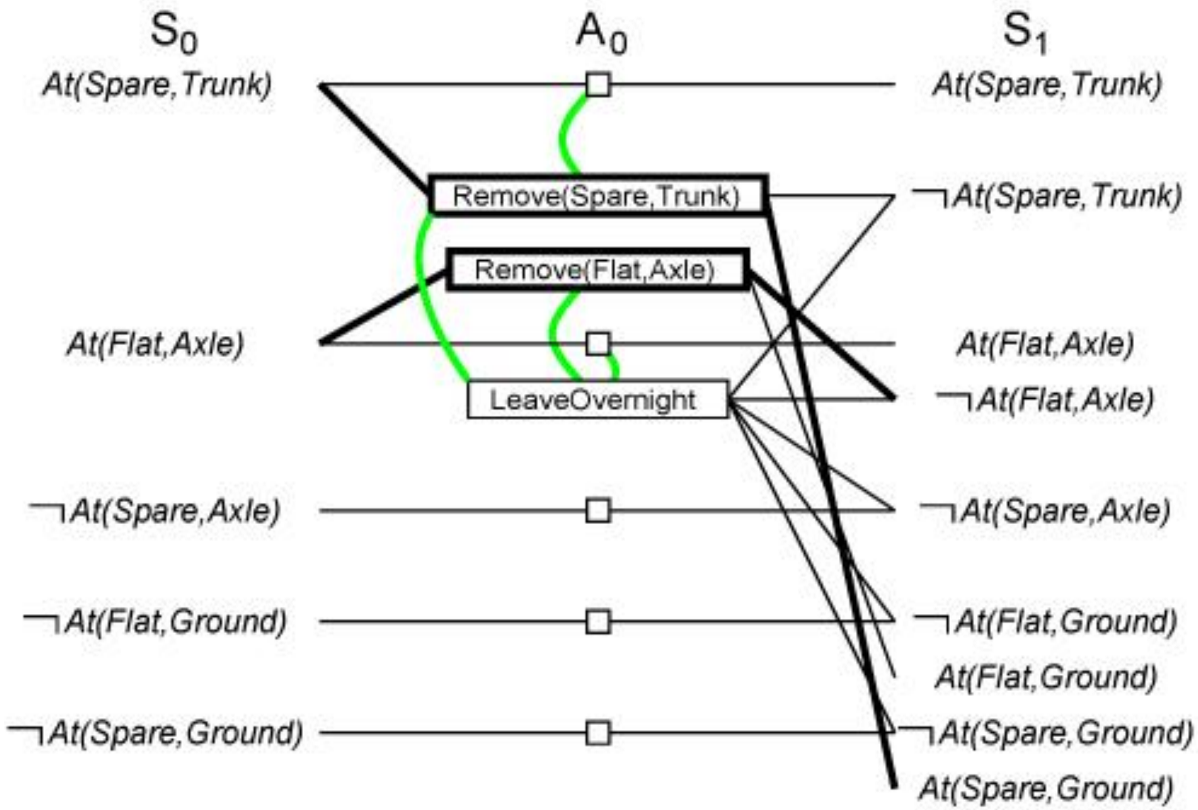
Example: GRAPHPLAN Execution (1)

- At(Spare,Axle) is not in S_0
- No need to extract solution
- Expand the plan



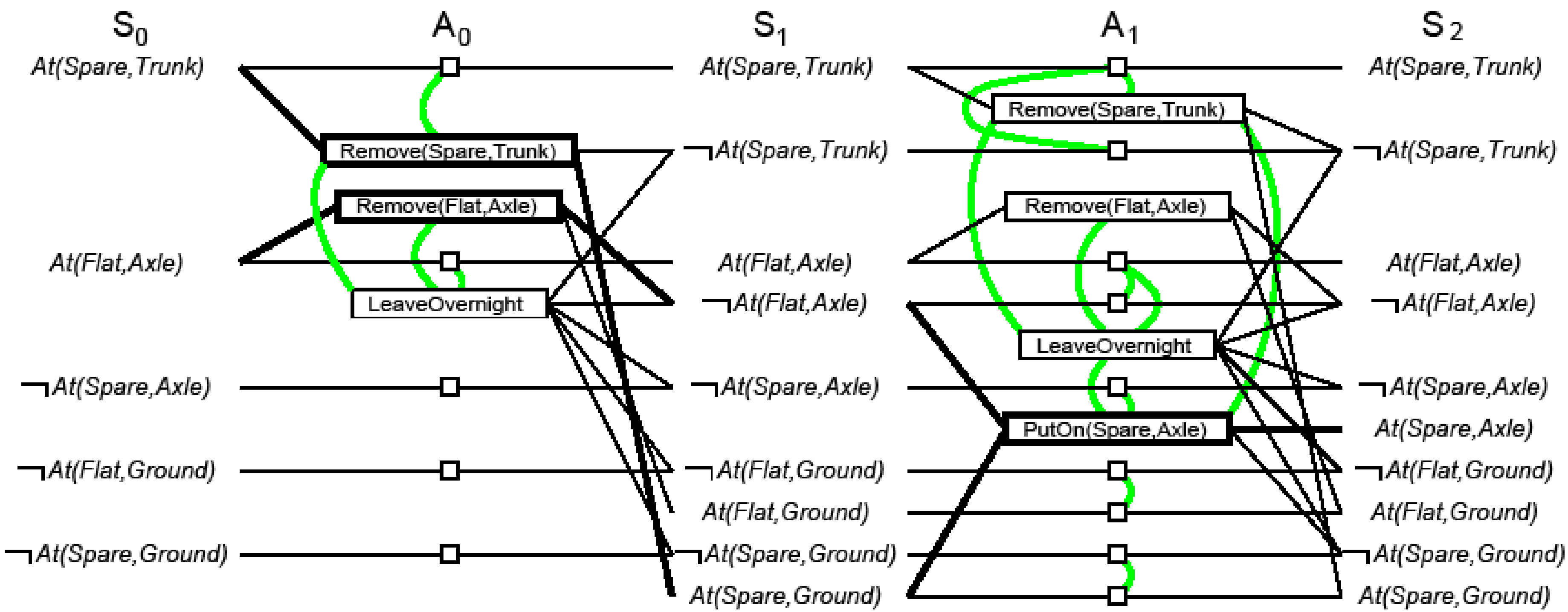
Example: GRAPHPLAN Execution (2)

- Three actions are applicable
- 3 actions and 5 noops are added
- Mutex links are added
- At(Spare,Axle) still not in S_1
- Plan is expanded



Example: GRAPHPLAN Execution (3)

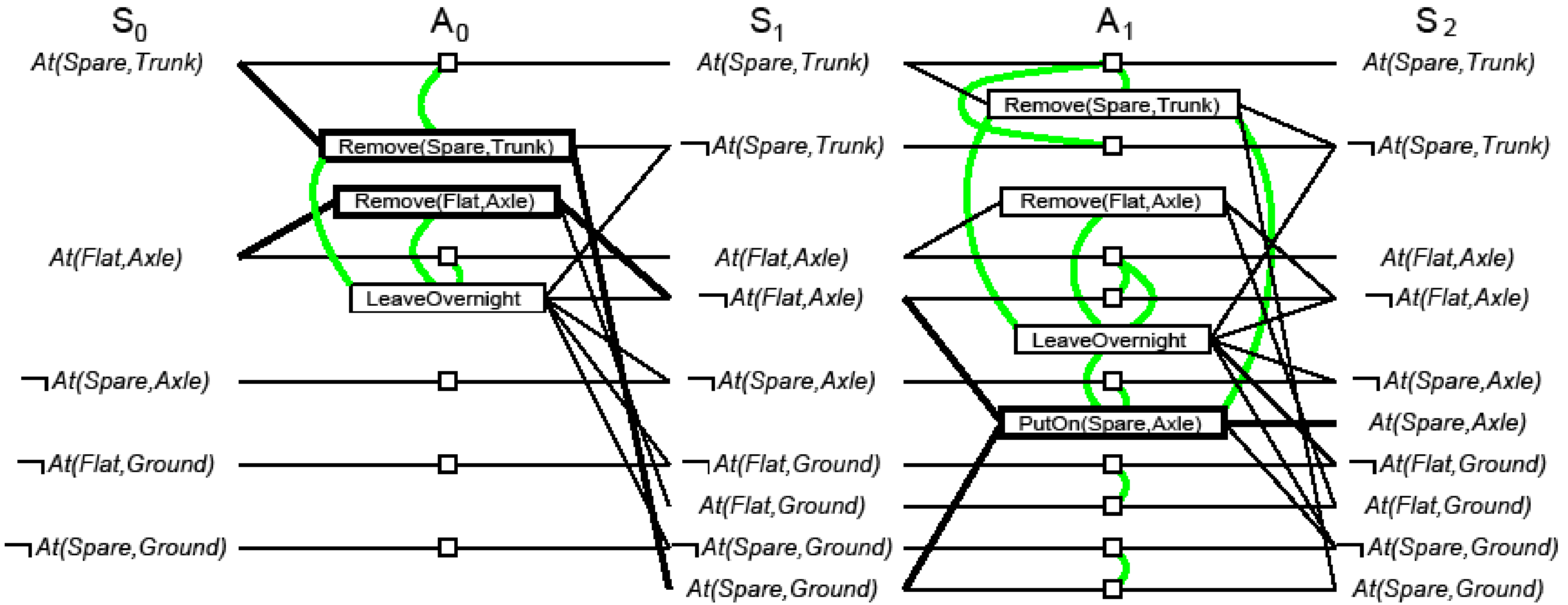
-



Solution Extraction (Backward)

1.

2.



Backtrack Search for Solution Extraction

- Starting at the highest fact level
  - Each goal is put in a goal list for the current fact layer
  - Search iterates thru each fact in the goal list trying to find an action to support it which is not mutex with any other chosen action
  - When an action is chosen, its preconditions are added to the goal list of the lower level
  - When all facts in the goal list of the current level have a consistent assignment of actions, the search moves to the next level
- Search backtracks to the previous level when it fails to assign an action to each fact in the goal list at a given level
- Search succeeds when the first level is reached.

Termination of GRAPHPLAN

- GRAPHPLAN is guaranteed to terminate
  - Literal increase monotonically
  - Actions increase monotonically
  - Mutexes decrease monotinically
- A solution is guaranteed not to exist when
  - The graph levels off with all goals present & non-mutex, and
  - EXTRACTSOLUTION fails to find solution

Optimality of GRAPHPLAN

- The plans generated by GRAPHPLAN
  - Are optimal in the number of steps needed to execute the plan
  - Not necessarily optimal in the number of actions in the plan (GRAPHPLAN produces partially ordered plans)

Other classical planning approaches

- The most effective approached to planning currently are:
  - Translating to Boolean Satisfiability
  - Forward state-space search with carefully crafted heuristics
  - Search using planning graphs (covered already)

Planning as Satisfiability

- Express propositional planning as a set of propositions.
- Index propositions with time steps:
- On(A,B)_0, ON(B,C)_0
- Goal conditions: the goal conjuncts at time T, T is determined arbitrarily.
- Unknown propositions are not stated.
- Propositions known not to be true are stated negatively.
- Actions: a proposition for each action for each time slot.
- Succesor state axioms need to be expressed for each action (like in the situation calculus but it is propositional)

Planning with propositional logic (continued)

- We write the formula:
  - Initial state and succesor state axioms and goal
- We search for a model to the formula. Those actions that are assigned true consititute a plan.
- To have a single plan we may have a mutual exclusion for all actions in the same time slot.
- We can also choose to allow partial order plans and only write exclusions between actions that interfere with each other.

- Planning: iteratively try to find longer and longer plans.

SATplan algorithm

function SATPLAN(*problem*, T_{max}) **returns** solution or failure
  inputs: *problem*, a planning problem
    T_{max}, an upper limit for plan length

  for $T = 0$ **to** T_{max} **do**
    cnf, *mapping* ← TRANSLATE-TO-SAT(*problem*, T)
    assignment ← SAT-SOLVER(*cnf*)
    if *assignment* is not null **then**
      return EXTRACT-SOLUTION(*assignment*, *mapping*)
  return *failure*

Figure 11.15 The SATPLAN algorithm. The planning problem is translated into a CNF sentence in which the goal is asserted to hold at a fixed time step T and axioms are included for each time step up to T. (Details of the translation are given in the text.) If the satisfiability algorithm finds a model, then a plan is extracted by looking at those proposition symbols that refer to actions and are assigned *true* in the model. If no model exists, then the process is repeated with the goal moved one step later.

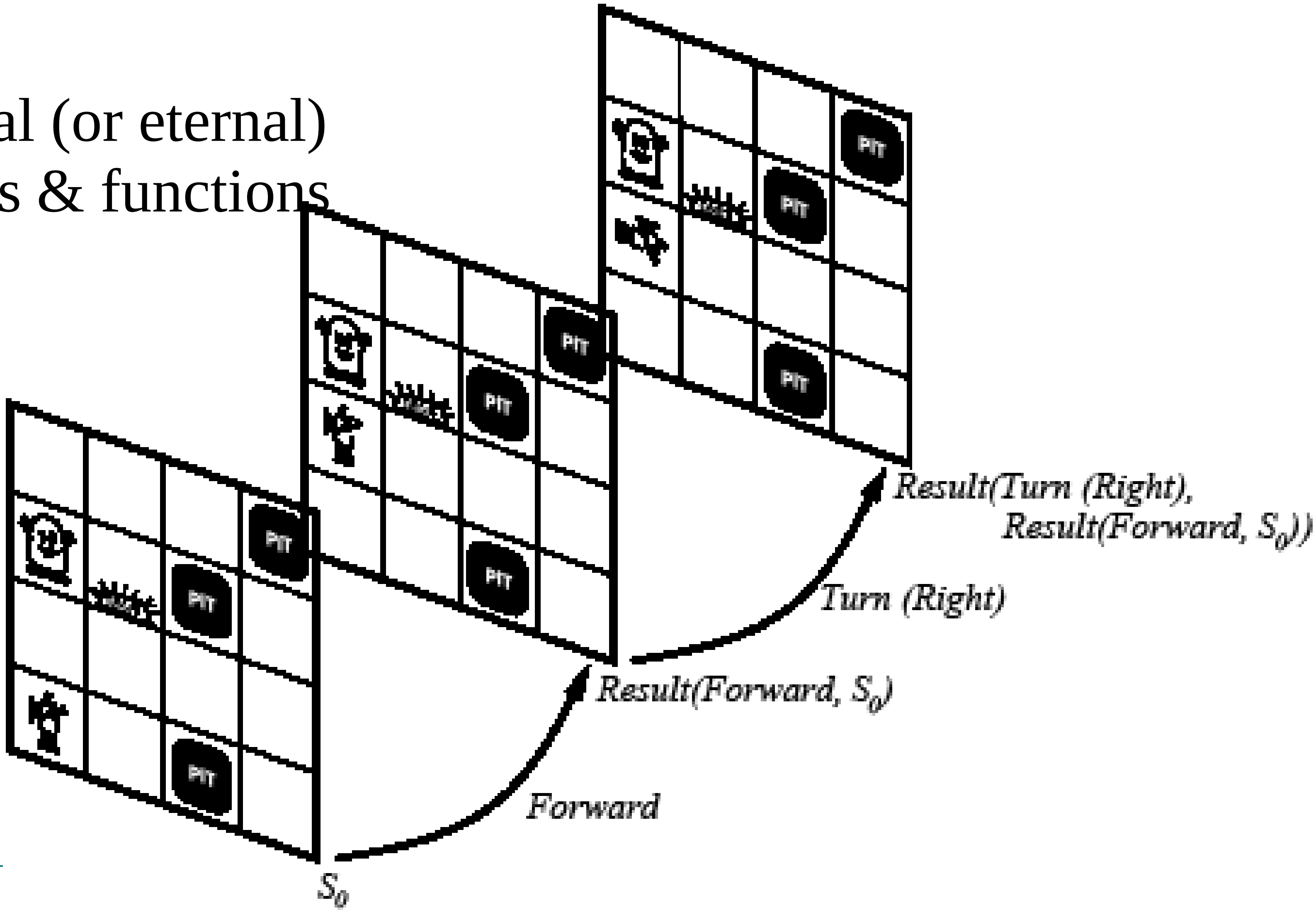
Further reading

Situation Calculus

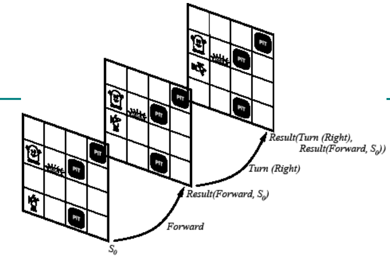
- First we look at how to model dynamic worlds within first-order logic.
- The situation calculus is an important formalism developed for this purpose.
- Situation Calculus is a (mostly) first-order language.
- Include in the domain of individuals a special set of objects called situations. Of these S_0 is a special distinguished constant which denotes the "initial" situation.

Situation Calculus: Ontology

- Situations
- Fluents
- Atemporal (or eternal) predicates & functions



Situation Calculus: Ontology



- Situations
  - Initial state: S_0
  - A function *Result*(*a*,*s*) gives the situation resulting from applying action *a* in situation *s*
- Fluents
  - Functions & predicates whose truth values can change from one situation to the other
  - Example: $\neg Holding(G_1,S_0)$
- Atemporal (or eternal) predicates and functions
  - Example: $Gold(G_1)$, $LeftLegOf(Wumpus)$

Situation Calculus

- Sequence of actions
  - Result([],s)=s
  - Result([a | seq],s)=Result(seq,Result(a,s))
- Projection task
  - Deducing the outcome of a sequence of actions
- Planning task
  - Find a sequence of actions that achieves a desired effect

Example: Wumpus World

- Fluents
  - At(o,p,s), Holding(o,s)
- Agent is in [1,1], gold is in [1,2]
  - $At(Agent,[1,1],S_0) \wedge At(G_1,[1,2],S_0)$
- In S_0, we also need to have:
  - $At(o,x,S_0) \Leftrightarrow [(o=Agent) \wedge x=[1,1]] \vee [(o=G_1) \wedge x=[1,2]]$
  - $\neg Holding(o,S_0)$
  - $Gold(G_1) \wedge Adjacent([1,1],[1,2]) \wedge Adjacent([1,2],[1,1])$
- The query is:
  - $\exists$ seq $At(G_1,[1,1],Result(seq,S_0))$
- The answer is
  - $At(G1,[1,1],Result(Go([1,1],[1,2]),Grab(G_1),Go([1,2],[1,1]),S_0))$

Importance of Situation Calculus

- Historical note
  - Situation Calculus was the first attempt to formalizing planning in FOL
  - Other formalisms include Event Calculus
  - The area of using logic for planning is informally called in the literature “Reasoning About Action & Change”
- Highlighted three important problems
  1. Frame problem
  2. Qualification problem
  3. Ramification problem

'Famous' Problems

- Frame problem
  - Representing all things that stay the same from one situation to the next
  - Inferential and representational
- Qualification problem
  - Defining the circumstances under which an action is guaranteed to work
  - Example: what if the gold is slippery or nailed down, etc.
- Ramification problem
  - Proliferation of implicit consequences of actions as actions may have secondary consequences
  - Examples: How about the dust on the gold?

Situation Calculus Building Blocks

- Situations
- Fluents
- Actions

Situations

- Situations are the history of actions from s_0. You can think of them as indexing “states” of the world, but two different situations can have the same state. (E.g., “scratch, eat” may lead to the same state of the world as “eat, scratch” When dealing with dynamic environments, the world has different properties at different points in time.

  E.g.,

  $in(robby, room1, s_0)$, $\neg in(robby, room3, s_0)$

  $\neg in(robby, room3, s_1)$, $in(robby, room1, s_1)$.

- Different things are true in situation s_1 than in the initial situation s_0.

- Contrast this with the previous kinds of knowledge we examined.

Fluents

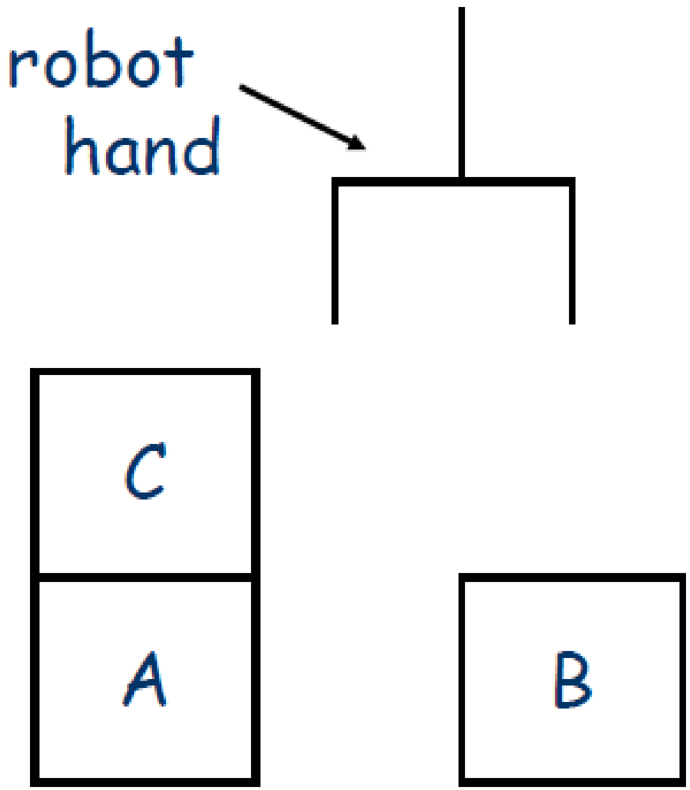
- Previously, we were encoding a property of a term as a relation in first-order logic. The distinction here is that properties that change from situation to situation (called fluents) take an extra situation argument.

  E.g.,

  $$\text{clear}(b) \rightarrow \text{clear}(b,s)$$

  "clear(b)" is no longer statically true, it is true contingent on what situation we are talking about

Blocks World Example.



clear(c,s_0)

on(c,a,s_0)

clear(b,s_0)

handempty(s_0)

Actions

- Actions are also part of language
  - A set of “primitive” action objects in the (semantic) domain of individuals.
  - In the syntax they are represented as functions mapping objects to primitive action objects.

Examples:

  - pickup(X) function mapping blocks to actions
    - pickup(c) = “the primitive action object corresponding to ‘picking up block c’
  - stack(X,Y)
    - stack(a,b) = “the primitive action object corresponding to ‘stacking a on top of b’

Actions applied to situation → new situation

- Remember that actions are terms in the language.
- In order to talk about the situation that results from executing an action in a particular situation, there is a “generic” action application function do(A,S).

  do maps a primitive action A and a situation S to a new situation.
  - The new situation is the situation that results from applying A to S.

Example:

do(pickup(c), s_0) = the new situation that is the result of applying action “pickup(c)” to the initial situation s_0.

What do Actions do?

- Actions affect the situation by changing what is true.
  - on(c,a,s_0); clear(a,do(pickup(c),s_0))
- We want to represent the effects of actions, this is done in the situation calculus with two components:
  - Action Precondition Axioms
  - Action Effect Axioms

Specifying the effects of actions

Action preconditions:

Certain things must hold for actions to have a predictable effect.

- pickup(c) this action is only applicable to situations S when "clear(c,S) $\wedge$ handempty(S)" is true.

Action effects:

- Actions make certain things true and certain things false.
  - holding(c, do(pickup(c), S))
  - $\forall$ X.$\neg$handempty(do(pickup(X),S))

Specifying the effects of actions

Action effects are conditional on their precondition being true.

$$
\begin{aligned}
&\forall S,X.\\
&\quad ontable(X,S) \wedge clear(X,S) \wedge handempty(S)\\
&\quad \rightarrow \quad holding(X, do(pickup(X),S))\\
&\qquad \wedge \neg handempty(do(pickup(X),S))\\
&\qquad \wedge \neg ontable(X, do(pickup(X,S))\\
&\qquad \wedge \neg clear(X, do(pickup(X,S)).
\end{aligned}
$$

Green indicates a situation term

Plan Generation

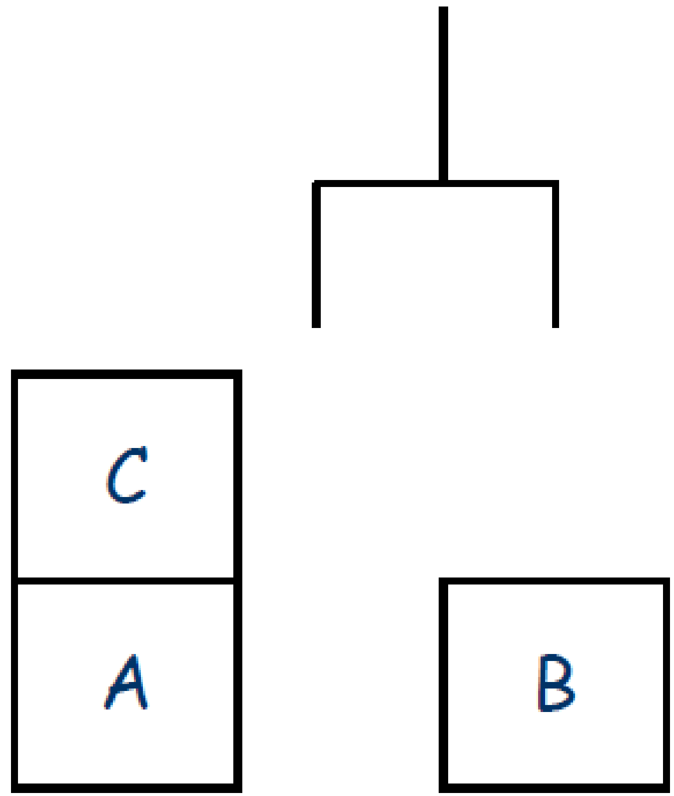
There are many ways to generate plans. Here we show how to do it by representing actions in the situation calculus (as you have just seen) and generating a plan via **deductive plan synthesis**.

This is *not* the approach taken by state-of-the-art planners, as we will see later, but it is where the field started and is still used for specifying, studying and advancing research for more complex tasks in reasoning about action and change

…so for now, back to resolution!

Reasoning with the Situation Calculus.

1. $clear(c,s_0)$
2. $on(c,a,s_0)$
3. $clear(b,s_0)$
4. $ontable(a,s_0)$
5. $ontable(b,s_0)$
6. $handempty(s_0)$



Query:

$\exists Z.holding(b,Z)$

7. $(\neg holding(b,Z), ans(Z))$

Does there exists a situation in which we are holding b? And if so what is the name of that situation.

Resolution

Convert "pickup" action axiom into clause form:

$\forall$S,Y. ontable(Y,S) $\wedge$ clear(Y,S) $\wedge$ handempty(S) $\rightarrow$

holding(Y, do(pickup(Y),S))
$\wedge$ $\neg$handempty(do(pickup(Y),S))
$\wedge$ $\neg$ontable(Y,do(pickup(Y,S))
$\wedge$ $\neg$clear(Y,do(pickup(Y,S)).

8. ($\neg$ontable(Y,S), $\neg$clear(Y,S), $\neg$handempty(S), holding(Y,do(pickup(Y),S))
9. ($\neg$ontable(Y,S), $\neg$clear(Y,S), $\neg$handempty(S), $\neg$handempty(do(pickup(X),S)))
10. ($\neg$ontable(Y,S), $\neg$clear(Y,S), $\neg$handempty(S), $\neg$ontable(Y,do(pickup(Y,S)))
11. ($\neg$ontable(Y,S), $\neg$clear(Y,S), $\neg$handempty(S), $\neg$clear(Y,do(pickup(Y,S)))

Resolution

12. R[8d, 7]{Y=b,Z=do(pickup(b),S)}
(¬ontable(b,S), ¬clear(b,S), ¬handempty(S),
ans(do(pickup(b),S)))

13. R[12a,5] {S=s_0}
(¬clear(b,s_0), ¬handempty(s_0),
ans(do(pickup(b),s_0)))

14. R[13a,3] {}
(¬handempty(s_0), ans(do(pickup(b),s_0)))

15. R[14a,6] {}
ans(do(pickup(b),s_0))

The answer?

- $ans(do(pickup(b),s_0))$
- This says that a situation in which you are holding b is called "$do(pickup(b),s_0)$"
- This tells you what actions to execute to achieve "holding(b)".

Two types of reasoning.

Two common types of queries :

1. Predicting the effects of a *given* sequence of action
   E.g., on(b,c, do(stack(b,c), do(pickup(b), s_0)))

2. Computing a sequence of actions that achieve a goal conditions E.g.,

   $\exists S.\ on(b,c,S) \wedge on(c,a,S)$

The Frame Problem

Unfortunately, logical reasoning won't immediately yield the answer to these kinds of questions.

e.g., query: $on(c,a,do(pickup(b),s_0))$?

- is c still on a after we pickup b?
- Intuitively it should be
- Can logical reasoning reach this conclusion given the representation of actions that we have proposed thus far?

The Frame Problem

1. $clear(c,s_0)$
2. $on(c,a,s_0)$
3. $clear(b,s_0)$
4. $ontable(a,s_0)$
5. $ontable(b,s_0)$
6. $handempty(s_0)$

8. $(\neg ontable(Y,S), \neg clear(Y,S), \neg handempty(S), holding(Y,do(pickup(Y),S))$

9. $(\neg ontable(Y,S), \neg clear(Y,S), \neg handempty(S), \neg handempty(do(pickup(X),S)))$

10. $(\neg ontable(Y,S), \neg clear(Y,S), \neg handempty(S), \neg ontable(Y,do(pickup(Y,S)))$

11. $(\neg ontable(Y,S), \neg clear(Y,S), \neg handempty(S), \neg clear(Y,do(pickup(Y,S)))$

12. $\neg on(c,a,do(pickup(b),s_0))$ {QUERY)

Nothing can resolve with 12!

Logical Consequence

- Remember that resolution only computes logical consequences.
- We stated the effects of pickup(b), but did not state that it doesn't affect on(c,a).
- Hence there are models in which on(c,a) no longer holds after pickup(b) (as well as models where it does hold).

- The problem is that representing the non-effects of actions is very tedious and in general is not possible.
  - Think of all of the things that pickup(b) does not affect!

The Frame Problem

- Finding an effective way of specifying the non-effects of actions, without having to explicitly write them all down is the frame problem.

- Good solutions have been proposed, and the situation calculus has been a powerful way of dealing with dynamic worlds:
  - Logic-based high-level robotic programming languages

Computation Problems

- Although the situation calculus is a very powerful representation. It is not always efficient enough to use to compute sequences of actions.
- The problem of computing a sequence of actions to achieve a goal is “planning”
- Next we will study some less expressive representations that support more efficient planning.

From Situation Calculus to STRIPS

Simplifying the Planning Problem

- Assume **complete information about the initial state** through the closed-world assumption (**CWA**).
- Assume a **finite domain** of objects
- Assume that **action effects** are restricted to making **(conjunctions of) atomic formulae** true or false. No conditional effects, no disjunctive effects, etc.
- Assume **action preconditions** are restricted to **conjunctions of ground atoms**.

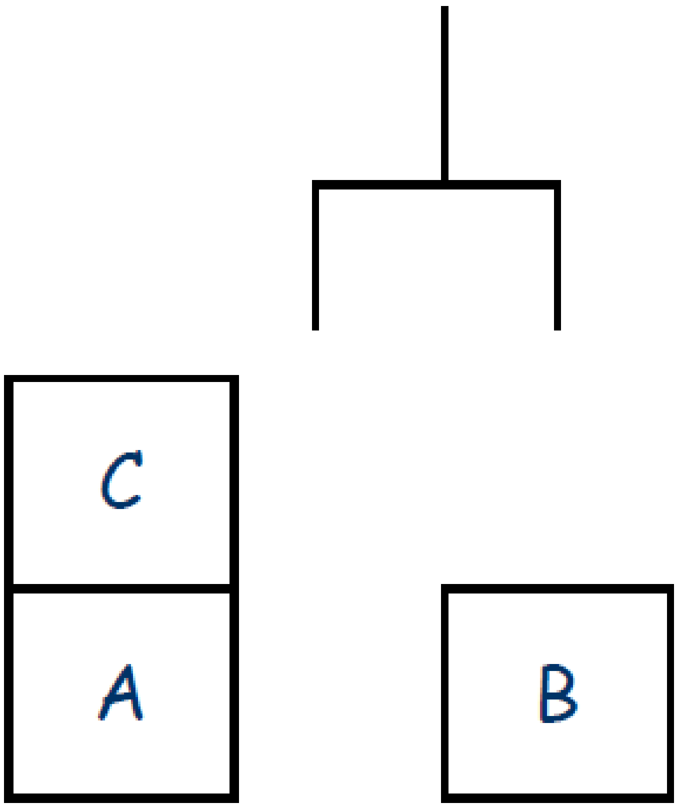
Closed World Assumption (CWA)

- "Classical Planning". No incomplete or uncertain knowledge.
- Use the "Closed World Assumption" in our knowledge representation and reasoning.
  - The knowledge base used to represent a state of the world is a **list of positive ground atomic facts**.
  - CWA is the assumption that
    a) if a ground atomic fact is not in our list of "known" facts, its negation must be true.
    b) the constants mentioned in KB are all the domain objects.

CWA

- CWA makes our knowledge base much like a database: if employed(John,CIBC) is not in the database, we conclude that $\neg$employed(John, CIBC) is true.

CWA Example



KB = {handempty
clear(c), clear(b),
on(c,a),
ontable(a), ontable(b)}

1. $clear(c) \wedge clear(b)$?
2. $\neg on(b,c)$?
3. $on(a,c) \vee on(b,c)$?
4. $\exists X.on(X,c)$? (D = {a,b,c})
5. $\forall X.ontable(X) \rightarrow X = a \vee X = b$?

Querying a Closed World KB

- With the CWA, **we can evaluate the truth or falsity of arbitrarily complex first-order formulas**.
- This process is very similar to query evaluation in databases.
- Just as databases are useful, so are CW KB's.

"CW KB" or "CW-KB" = Closed-world knowledge base

"CWA" = Closed World Assumption

Querying A Closed-World KB

Query(F, KB) /*return whether or not KB |= F */

if F is atomic
  return(F $\in$ KB)

Querying A CW KB

if $F = F_1 \wedge F_2$

return(Query(F_1) && Query(F_2))

if $F = F_1 \vee F_2$

return(Query(F_1) || Query(F_2))

if $F = \neg F_1$

return(! Query(F_1))

if $F = F_1 \rightarrow F_2$

return(!Query(F_1) || Query(F_2))

Querying A CW KB

```
if F = ∃X.F1
  for each constant c ∈ KB
    if (Query(F1{X=c}))
      return(true)
  return(false).

if F = ∀X.F1
  for each constant c ∈ KB
    if (!Query(F1{X=c}))
      return(false)
  return(true).
```

Querying A CW KB

Guarded quantification (for efficiency).

if F = $\forall$X.F_1

for each constant c $\in$ KB

if (!Query(F_1{X=c}))

return(false)

return(true).

E.g., consider checking

$\forall$ X. apple(x) $\rightarrow$ sweet(x)

we already know that the formula is true for all “non-apples”

Querying A CW KB

Guarded quantification (for efficiency).

$\forall X{:}[p(X)]\ F_1 \quad \leftrightarrow \quad \forall X{:}\ p(X) \rightarrow F_1$

```
for each constant c s.t. p(c)
   if (!Query(F1{X=c}))
      return(false)
return(true).
```

$\exists X{:}[p(X)]F_1 \quad \leftrightarrow \quad \exists X{:}\ p(X) \wedge F_1$

```
for each constant c s.t. p(c)
   if (Query(F1{X=c}))
      return(true)
return(false).
```

STRIPS representation.

- STRIPS (Stanford Research Institute Problem Solver.) is a way of representing actions.

- Actions are modeled as ways of modifying the world.
  - since the world is represented as a CW-KB, a STRIPS action represents a way of updating the CW-KB.
  - Now actions yield new KB's, describing the new world—the world as it is once the action has been executed.

Sequences of Worlds

- In the situation calculus where in one logical sentence we could refer to two different situations at the same time.
  - $on(a,b,s_0) \wedge \neg on(a,b,s_1)$
- In STRIPS, we would have two separate CW-KB's. One representing the initial state, and another one representing the next state (much like search where each state was represented in a separate data structure).

STRIPS Actions

- STRIPS represents actions using 3 lists.
  1. A list of action preconditions.
  2. A list of action add effects.
  3. A list of action delete effects.
- These lists contain variables, so that we can represent a whole class of actions with one specification.
- Each ground instantiation of the variables yields a specific action.

STRIPS Actions: Example

pickup(X):

Pre: {handempty, clear(X), ontable(X)}

Adds: {holding(X)}

Dels: {handempty, clear(X), ontable(X)}

"pickup(X)" is called a STRIPS operator.

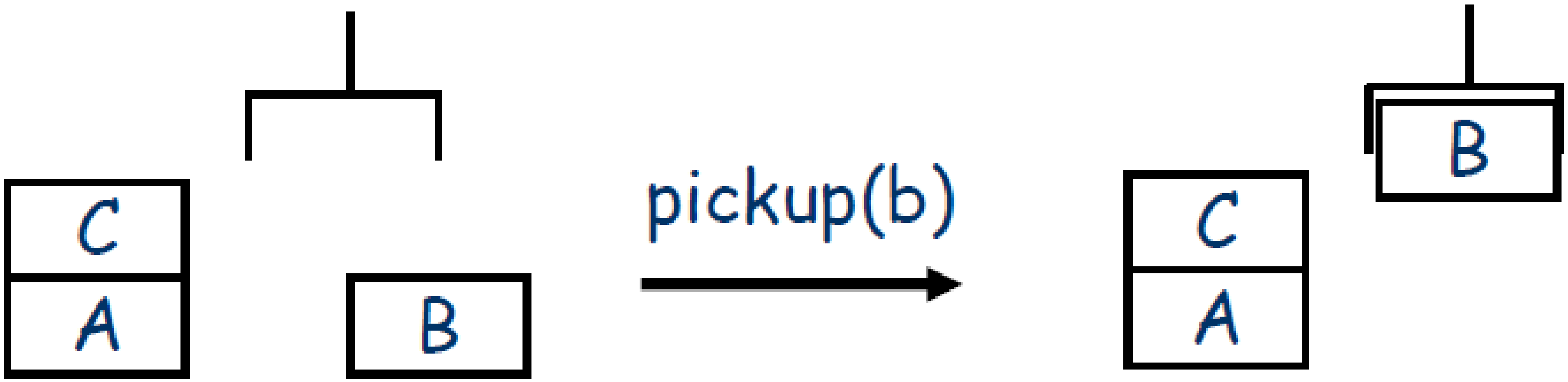
a particular instance e.g.

"pickup(a)" is called an action.

Operation of a STRIPS action.

- For a particular STRIPS action (ground instance) to be applicable to a state (a CW-KB)
  - every fact in its precondition list must be true in KB.
    - This amounts to testing membership since we have only atomic facts in the precondition list.
- If the action is applicable, the new state is generated by
  - removing all facts in Dels from KB, then
  - adding all facts in Adds to KB.

Operation of a Strips Action: Example



pre = {handmpty,
clear(b),
ontable(b)}

add = {holding(b)}

del = {handmpty,
clear(b),
ontable(b)}

KB = {handempty
clear(c), clear(b),
on(c,a),
ontable(a), ontable(b)}

KB = { holding(b),
clear(c),
on(c,a),
ontable(a)}

STRIPS Blocks World Operators.

- pickup(X)
  Pre: {clear(X), ontable(X), handempty}
  Add: {holding(X)}
  Del: {clear(X), ontable(X), handempty}
- putdown(X)
  Pre: {holding(X)}
  Add: {clear(X), ontable(X), handempty}
  Del: {holding(X)}

STRIPS Blocks World Operators.

- unstack(X,Y)
  Pre: {clear(X), on(X,Y), handempty}
  Add: {holding(X), clear(Y)}
  Del: {clear(X), on(X,Y), handempty}
- stack(X,Y)
  Pre: {holding(X),clear(Y)}
  Add: {on(X,Y), handempty, clear(X)}
  Del: {holding(X),clear(Y)}

STRIPS has no Conditional Effects

- putdown(X)
  Pre: {holding(X)}
  Add: {clear(X), ontable(X), handempty}
  Del: {holding(X)}
- stack(X,Y)
  Pre: {holding(X),clear(Y)}
  Add: {on(X,Y), handempty, clear(X)}
  Del: {holding(X),clear(Y)}

- The table has infinite space, so it is always clear. If we "stack(X,Y)" if Y=Table we cannot delete clear(Table), but if Y is an ordinary block "c" we must delete clear(c).

Conditional Effects

- Since STRIPS has no conditional effects, we must sometimes utilize extra actions: one for each type of condition.
  - We embed the condition in the precondition, and then alter the effects accordingly.

Other Example Domains

8 Puzzle as a planning problem

The Constants

- A constant representing each position, P1,…,P9

P1	P2	P3
P4	P5	P6
P7	P8	P9

- A constant for each tile (and blank) B,T1, …, T8.

8-Puzzle

The Relations/Predicates/Properties

- at(T,P) tile T is at position P.

1	2	5
7	8	
6	4	3

at(T1,P1), at(T2,P2), at(T5,P3), ...

- adjacent(P1,P2) P1 is next to P2 (i.e., we can slide the blank from P1 to P2 in one move.
  - adjacent(P5,P2), adjacent(P5,P8), …

8-Puzzle

The Operators

slide(T,X,Y)

Pre: {at(T,X), at(B,Y), adjacent(X,Y)}

Add: {at(B,X), at(T,Y)}

Del: {at(T,X), at(B,Y)}

at(T1,P1), at(T5,P3),
at(T8,P5), at(B,P6), ...,

at(T1,P1), at(T5,P3),
at(B,P5), at(T8,P6), ...,

1	2	5
7	8	
6	4	3

$\xrightarrow{\text{slide(T8,P5,P6)}}$

1	2	5
7		8
6	4	3

Elevator Control

Figure 1: A Miconic-10™ keypad allows passengers to enter their destination before they enter the elevator. A display informs the passenger about the elevator that will offer the most suitable transport.

Elevator Control

Schindler Lifts.

- Central panel to enter your elevator request.
- Your request is scheduled and an elevator assigned to you.
- You can’t travel with someone going to a secure floor, emergency travel has priority, etc.

- Modeled as a planning problem and fielded in one of Schindler’s high end elevators.

Planning as a Search Problem

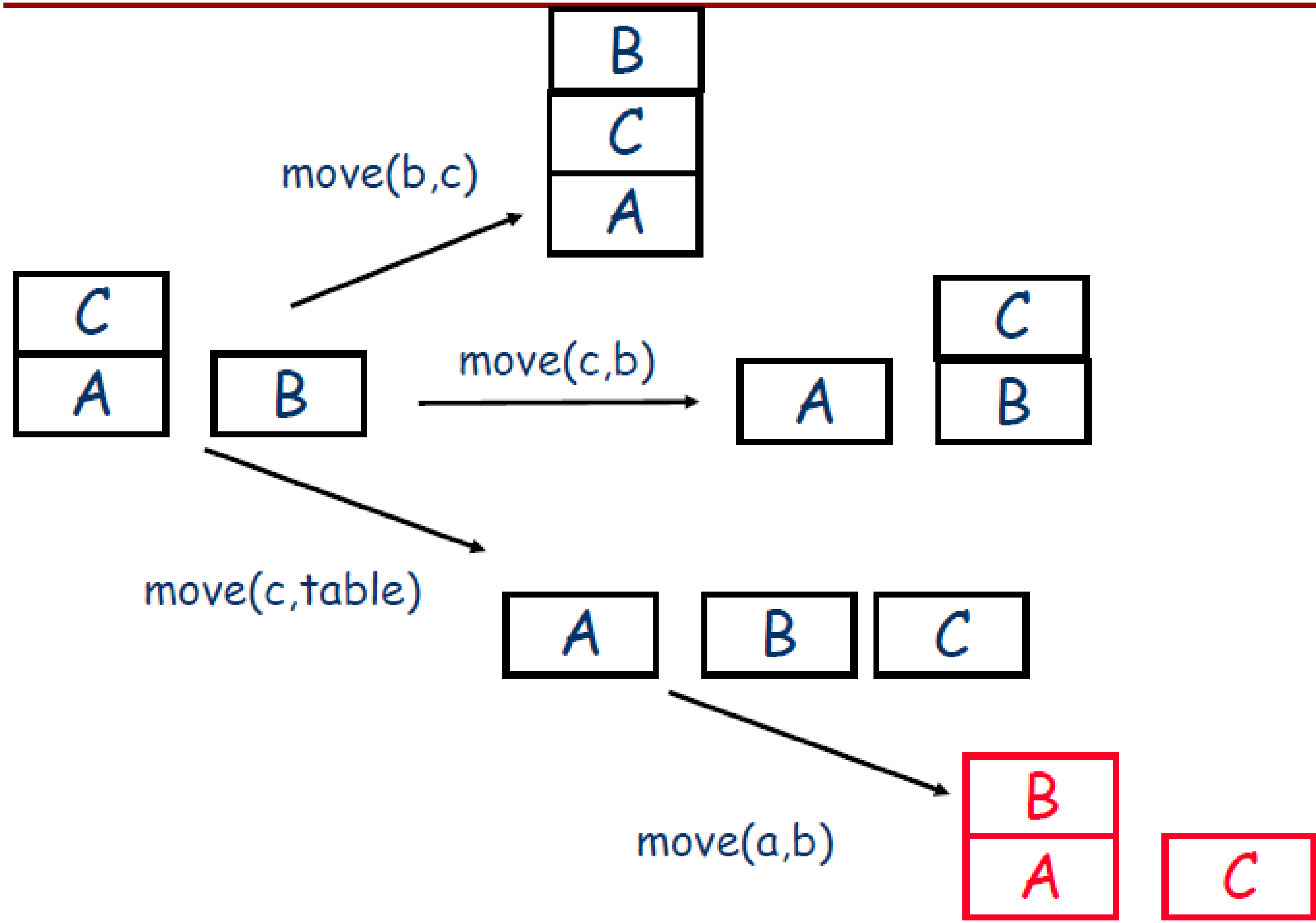
- Given a CW-KB representing the initial state, a set of STRIPS or ADL (Action Description Language) operators, and a goal condition we want to achieve (specified either as a conjunction of facts, or as a formula)
  - The planning problem is to determine a sequence of actions that when applied to the initial CW-KB yield an updated CW-KB which satisfies the goal.

This is known as the classical planning task.

Planning As Search

- This can be treated as a search problem.
  - The initial CW-KB is the initial state.
  - The actions are operators mapping a state (a CW-KB) to a new state (an updated CW-KB).
  - The goal is satisfied by any state (CW-KB) that satisfies the goal.

Example.



Problems

- Search tree is generally quite large
  - randomly reconfiguring 9 blocks takes thousands of CPU seconds.
- The representation suggests some structure. Each action only affects a small set of facts, actions depend on each other via their preconditions.
- Planning algorithms are designed to take advantage of the special nature of the representation.

Planning

- We will look at one technique:

  Relaxed Plan heuristics used with heuristic search.

  The heuristics are domain independent. As such they are part of a class of so-called

  domain-independent heuristic search for planning

Reachability Analysis.

- The idea is to consider what happens if we ignore the delete lists of actions.
- This is yields a "relaxed problem" that can produce a useful heuristic estimate.

Reachability Analysis

- In the relaxed problem actions add new facts, but never delete facts.
- Then we can do reachability analysis, which is much simpler than searching for a solution.

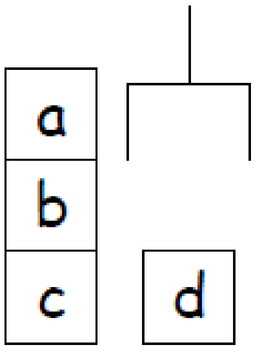
Reachability

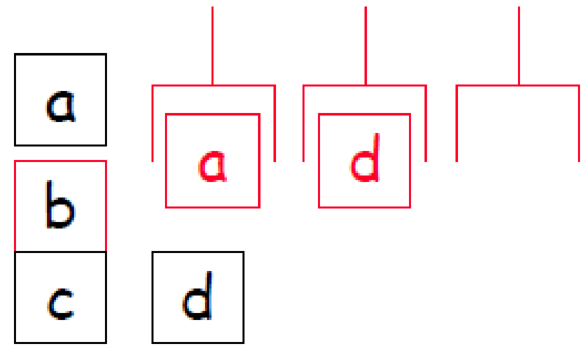
- We start with the initial state S_0.
- We alternate between state and action layers.
- We find all actions whose preconditions are contained in S_0. These actions comprise the first action layer A_0.
- The next state layer contains:
  - S_0 U all states added by the actions in A_0.
- In general:
  - A_i … set of actions whose preconditions are in S_i.
  - $S_i = S_{i-1}$ U the add lists of all of the actions in A_i

STRIPS Blocks World Operators.

- pickup(X)
  Pre: {handempty, ontable(X), clear(X)}
  Add: {holding(X)}
  ~~Del: {handempty, ontable(X), clear(X)}~~
- putdown(X)
  Pre: {holding(X)}
  Add: {handempty, ontable(X), clear(X)}
  ~~Del: {holding(X)}~~
- unstack(X,Y)
  Pre: {handempty, clear(X), on(X,Y)}
  Add: {holding(X), clear(Y)}
  ~~Del: {handempty, clear(X), on(X,Y)}~~
- stack(X,Y)
  Pre: {holding(X),clear(Y)}
  Add: {handempty, clear(X), on(X,Y)}
  ~~Del: {holding(X),clear(Y)}~~

Example





on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty

S_0

unstack(a,b)
pickup(d)

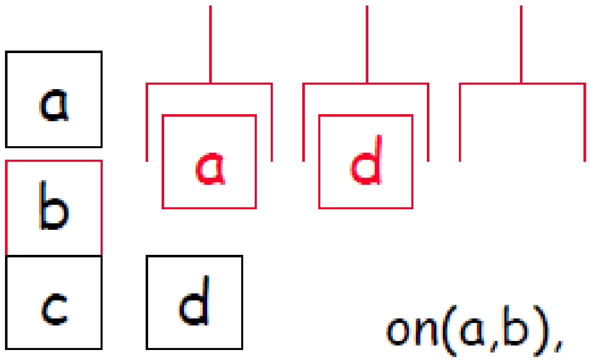
A_0

on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
handempty,
clear(d),
holding(a),
clear(b),
holding(d)

S_1

this is not a state as some of these facts cannot be true at the same time!

Example



on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty,
holding(a),
clear(b),
holding(d)

S_1

unstack(a,b)
pickup(d)

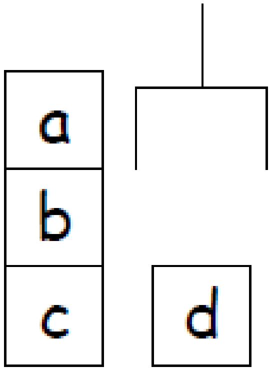
from $\boxed{A_0}$

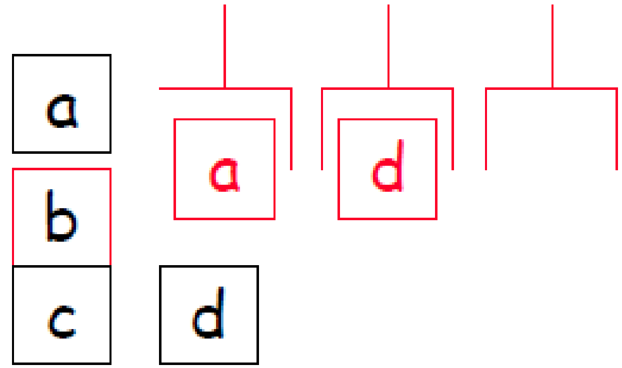
putdown(a),
putdown(d),
stack(a,b),
stack(a,a),
stack(d,a),
stack(d,b),
stack(d,d),
unstack(b,c)
...

Impossible, but we don't know because we ignore dels.

$\boxed{A_1}$

Example





on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty

S_0

unstack(a,b)
pickup(d)

A_0

on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
handempty,
clear(d),
holding(a),
clear(b),
holding(d)

S_1

this is not a state!

Example

on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty,
holding(a),
clear(b),
holding(d)

S_1

putdown(a),
putdown(d),
stack(a,b),
stack(a,a),
stack(d,b),
stack(d,a),
pickup(d),
...
unstack(b,c)
...

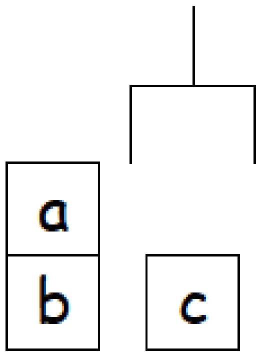
$\boxed{A_1}$

Reachabilty

- We continue until:
  - the goal G is contained in the state layer, or
  - until the state layer no longer changes (reached fix point).
- Intuitively:
  - the actions at level A_i are the actions that could be executed at the i-th step of some plan, and
  - the facts in level S_i are the facts that could be made true within a plan of length i.
- Some of the actions/facts have this property.
  But not all!

Reachability

to reach on(c,b) requires 4 actions



S_0	A_0	S_1	A_1
on(a,b), ontable(c), ontable(b), clear(a), clear(c), handempty	**unstack(a,b)** **pickup(c)**	on(a,b), ontable(c), ontable(b), clear(a), clear(c), handempty, holding(a), clear(b), holding(c)	**stack(c,b)** ...

on(c,b),
...

but stack(c,b) cannot be executed after one step

Heuristics from Reachability Analysis

Grow the levels until the goal is contained in the final state level S_K.

- If the state level stops changing and the goal is not present: The goal is unachievable under the assumption that (a) the goal is a set of positive facts, and (b) all preconditions are positive facts.

- Then do the following

Heuristics from Reachability Analysis

CountActions(G,S_K):

/ Compute the number of actions contained in a relaxed plan achieving the goal. */*

- Split G into facts in S_{K-1} and elements in S_K only.
  - G_P contains the previously achieved (in S_{K-1}) and
  - G_N contains the just achieved parts of G (only in S_K).
- Find a minimal set of actions A whose add effects cover G_N.
  - may contain no redundant actions,
  - but may not be the minimum sized set (computing the minimum sized set of actions is the set cover problem and is NP-Hard)
- NewG := S_{K-1} U preconditions of A.
- return CountAction(NewG,S_{K-1}) + size(A)

Heuristics from Reachability Analysis

CountActions(G,S_K):

/ Compute the number of actions contained in a relaxed plan achieving the goal. */*

- Split G into facts in S_{K-1} and elements in S_K only.
  - G_P contains the previously achieved (in S_{K-1}) and
  - G_N contains the just achieved parts of G (only in S_K).
- Find a minimal set of actions A whose add effects cover G_N.
  - may contain no redundant actions,
  - but may not be the minimum sized set (computing the minimum sized set of actions is the set cover problem and is NP-Hard)
- NewG := G_P U preconditions of A.
- return CountAction(NewG,S_{K-1}) + size(A)

Example

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4],\ [f_2]a_2[f_5]\}$

Goal: f_6, f_5, f_1

Actions:

$[f_1]a_1[f_4]$

$[f_2]a_2[f_5]$

$[f_2, f_4, f_5]a_3[f_6$

Example

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], \ [f_2]a_2[f_5]\}$

$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$

Goal: f_6, f_5, f_1

Actions:

$[f_1]a_1[f_4]$

$[f_2]a_2[f_5]$

$[f_2, f_4, f_5]a_3[f_6$

Example

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], \ [f_2]a_2[f_5]\}$

$S_1 = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

Goal: f_6,f_5,f_1

Actions:

$[f_1]a_1[f_4]$

$[f_2]a_2[f_5]$

$[f_2,f_4,f_5]a_3[f_6$

Example

legend: [pre]act[add]

S_0 = {f_1, f_2, f_3}

A_0 = {[f_1]a_1[f_4], [f_2]a_2[f_5]}

S_1 = {f_1,f_2,f_3,f_4,f_5}

A_1 = {[f_2,f_4,f_5]a_3[f_6]}

S_2 ={f_1,f_2,f_3,f_4,f_5,f_6}

Goal: f_6,f_5,f_1

Actions:

[f_1]a_1[f_4]

[f_2]a_2[f_5]

[f_2,f_4,f_5]a_3[f_6

Example

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], \; [f_2]a_2[f_5]\}$

$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$

$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$

$\mathbf{S_2} = \{f_1, f_2, f_3, f_4, f_5, f_6\}$

$\mathbf{G = \{f_6, f_5, f_1\}}$

Example

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4],\ [f_2]a_2[f_5]\}$

$S_1 = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$\mathbf{S_2} = \{f_1,f_2,f_3,f_4,f_5,f_6\}$

Goal: f_6,f_5,f_1

Actions:

$[f_1]a_1[f_4]$

$[f_2]a_2[f_5]$

$[f_2,f_4,f_5]a_3[f_6$

$G = \{f_6,f_5, f_1\}$

We split G into G_P and G_N:

Example

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], \ [f_2]a_2[f_5]\}$

$S_1 = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$\mathbf{S_2} = \{f_1,f_2,f_3,f_4,f_5,f_6\}$

$G = \{f_6,f_5,f_1\}$

$G_N = \{f_6\}$ (newly achieved)

$G_p = \{f_5, f_1\}$ (achieved before)

Example

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4],\ [f_2]a_2[f_5]\}$

$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$

$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$

$\mathbf{S_2} = \{f_1, f_2, f_3, f_4, f_5, f_6\}$

$\mathbf{G = \{f_6, f_5, f_1\}}$

We split G into G_P and G_N:

CountActs(G, S_2)

$G_P = \{f_5, f_1\}$ //already in S1

$G_N = \{f_6\}$ //New in S2

$A = \{a_3\}$ //adds all in G_N

//the new goal: $G_P \cup$ Pre(A)

$G_1 = \{f_5, f_1, f_2, f_4\}$

Return

1 + CountActs(G_1, S_1)

Example

Now, we are at level S1

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], \ [f_2]a_2[f_5]\}$

$\mathbf{S_1} = \{f_1, f_2, f_3, f_4, f_5\}$

$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$

$S_2 = \{f_1, f_2, f_3, f_4, f_5, f_6\}$

$G_1 = \{f_5, f_1, f_2, f_4\}$

CountActs(G_1,S_1)

Example

Now, we are at level S1

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], \; [f_2]a_2[f_5]\}$

$\mathbf{S_1} = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$S_2 = \{f_1,f_2,f_3,f_4,f_5,f_6\}$

$G_1 = \{f_5,f_1,f_2,f_4\}$

We split G_1 into G_P and G_N:

CountActs(G_1,S_1)

Example

Now, we are at level S1

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], [f_2]a_2[f_5]\}$

$\mathbf{S_1} = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$S_2 = \{f_1,f_2,f_3,f_4,f_5,f_6\}$

$G_1 = \{f_5,f_1,f_2,f_4\}$

We split G_1 into G_P and G_N:

$\mathbf{G_N} = \{f_5,f_4\}$

$\mathbf{G_P} = \{f_1,f_2\}$

CountActs(G_1,S_1)

$G_P = \{f_1,f_2\}$ **//already in S0**

$G_N = \{f_4,f_5\}$ **//New in S1**

$A = \{a_1,a_2\}$ **//adds all in G_N**

//the new goal: $G_P \cup$ Pre(A)

$G_2 = \{f_1,f_2\}$

Return

2 + CountActs(G_2,S_0)

Example

Now, we are at level S1

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], \ [f_2]a_2[f_5]\}$

$S_1 = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$S_2 = \{f_1,f_2,f_3,f_4,f_5,f_6\}$

$G_2 = \{f_1,f_2\}$

We split G_2 into G_P and G_N:

$G_N = \{f_1,f_2\}$

$G_P = \{\}$

CountActs(G_2,S_0)

$G_N = \{f_1,f_2\}$ **//already in S0**

$G_P = \{\}$ **//New in S1**

$A = \{\}$ **//No actions needed.**

Return

0

Example

Now, we are at level S1

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], \ [f_2]a_2[f_5]\}$

$S_1 = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$S_2 =\{f_1,f_2,f_3,f_4,f_5,f_6\}$

$G_2 = \{f_1,f_2\}$

We split G_2 into G_P and G_N:

$\mathbf{G_N} = \{f_1,f_2\}$

$\mathbf{G_P} = \{\}$

$CountActs(G_2,S_0)$

$G_N =\{f_1,f_2\}$ **//already in S0**

$G_P = \{\}$ **//New in S1**

$A = \{\}$ **//No actions needed.**

Return

0

So, in total CountActs(G,S2)=1+2+0=3

Using the Heuristic

- First, build a layered structure from a state S that reaches a goal state.

- CountActions: counts how many actions are required in a relaxed plan.
  - Use this as our heuristic estimate of the distance of S to the goal.
  - This heuristic tends to work better with greedy best-first search rather than A* search
    - That is when we ignore the cost of getting to the current state.

Admissibility

- A minimum sized plan in the delete relaxed problem would be a lower bound on the optimal size of a plan in the real problem. And could serve as an admissible heuristic for A*.
- However, CountActions **does NOT compute** the length of the optimal relaxed plan.
  - The choice of which *action set* to use to achieve G_P ("just achieved part of G") is not necessarily optimal – it is minimal, but not necessary a minimum.
  - Furthermore even if we picked a true minimum set A at each stage of CountActions, we might not obtain a minimum set of actions for the entire plan---the set A picked at each state influences what set can be used at the next stage!

Admissibility

- It is NP-Hard to compute the optimal length plan even in the relaxed plan space.
  - So CountActions cannot be made into an admissible heuristic without making it much harder to compute.
  - Empirically, refinements of CountActions performs very well on a number of sample planning domains.

Beyond STRIPS

STRIPS operators are not very expressive and as a consequence not as compact as they might be.

ADL (Action Description Language) extends the expressivity of STRIPS

ADL Operators.

ADL operators add a number of features to STRIPS.

1. Their preconditions can be arbitrary formulas, not just a conjunction of facts.
2. They can have conditional and universal effects.
3. Open world assumption:
   1. States can have negative literals
   2. The effect $(P \wedge \neg Q)$ means add P and $\neg Q$ but delete $\neg$ P and Q.

But ADL operators must still specify atomic changes to the knowledge base (add or delete ground atomic facts).

ADL Operators Examples.

move(X,Y,Z)

Pre: on(X,Y) $\wedge$ clear(Z)

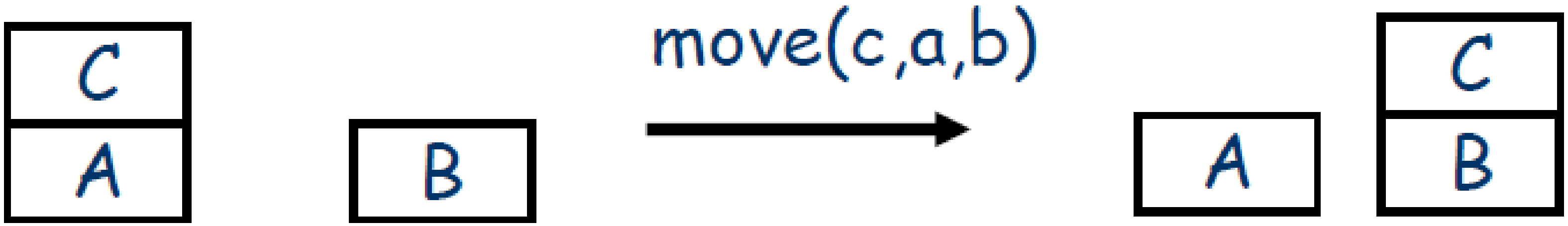
Effs: ADD[on(X,Z)]

DEL[on(X,Y)]

Z $\neq$ table $\rightarrow$ DEL[clear(Z)]

Y $\neq$ table $\rightarrow$ ADD[clear(Y)]

ADL Operators, example



move(c,a,b)
  Pre: on(c,a) $\wedge$ clear(b)
  Effs: ADD[on(c,b)]
      DEL[on(c,a)]
  b $\neq$ table $\rightarrow$ DEL[clear(b)]
  a $\neq$ table $\rightarrow$ ADD[clear(a)]

KB = { clear(c), clear(b),
on(c,a),
on(a,table),
on(b,table)}

KB = { on(c,b)
clear(c), clear(a)
on(a,table),
on(b,table)}

ADL Operators Examples.

clearTable()

Pre:

Effs: $\forall X.\ on(X,table) \rightarrow DEL[on(X,table)]$

ADL Operators.

1. Arbitrary formulas as preconditions.
   - in a CW-KB we can evaluate whether or not the preconditions hold for an arbitrary precondition.
2. They can have conditional and universal effects.
   - Similarly we can evaluate the condition to see if the effect should be applied, and find all bindings for which it should be applied.

Specify atomic changes to the knowledge base.

- CW-KB can be updated just as before.

Example

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4],\ [f_2]a_2[f_5]\}$

$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$

$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$

$\mathbf{S_2} = \{f_1, f_2, f_3, f_4, f_5, f_6\}$

$\mathbf{G = \{f_6, f_5, f_1\}}$

We split G into G_P and G_N:

CountActs(G,S_2)

$G_P = \{f_5, f_1\}$ //already in S1

$G_N = \{f_6\}$ //New in S2

$A = \{a_3\}$ //adds all in G_N

//the new goal: $G_P \cup Pre(A)$

$G_1 = \{f_5, f_1, f_2, f_4\}$

Return

1 + CountActs(G_1,S_1)

ADL Operators.

ADL operators add a number of features to STRIPS.

1. Their preconditions can be arbitrary formulas, not just a conjunction of facts.
2. They can have conditional and universal effects.
3. Open world assumption:
   1. States can have negative literals
   2. The effect $(P \wedge \neg Q)$ means add P and $\neg Q$ but delete $\neg P$ and Q.

But they must still specify atomic changes to the knowledge base (add or delete ground atomic facts).

ADL Operators Examples.

move(X,Y,Z)

Pre: $on(X,Y) \wedge clear(Z)$

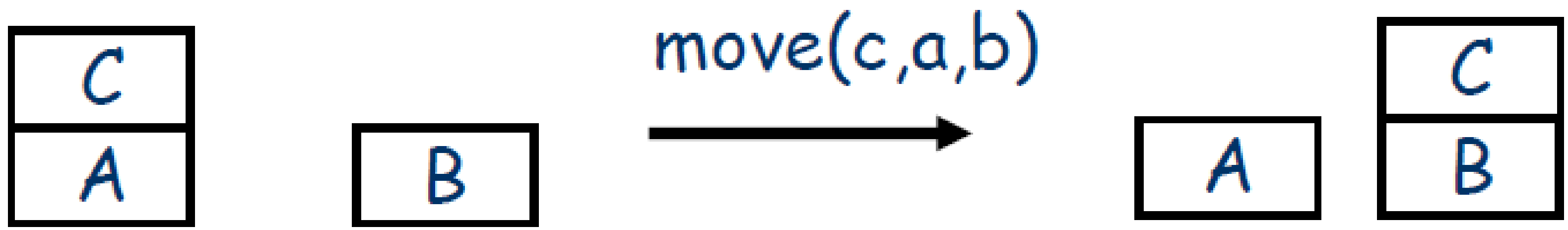
Effs: ADD[on(X,Z)]

DEL[on(X,Y)]

$Z \neq table \rightarrow$ DEL[clear(Z)]

$Y \neq table \rightarrow$ ADD[clear(Y)]

ADL Operators, example



move(c,a,b)
  Pre: on(c,a) $\wedge$ clear(b)
  Effs: ADD[on(c,b)]
      DEL[on(c,a)]
  b $\neq$ table $\rightarrow$ DEL[clear(b)]
  a $\neq$ table $\rightarrow$ ADD[clear(a)]

KB = { clear(c), clear(b),
  on(c,a),
  on(a,table),
  on(b,table)}

KB = { on(c,b)
  clear(c), clear(a)
  on(a,table),
  on(b,table)}

ADL Operators Examples.

clearTable()

Pre:

Effs: $\forall X.\ on(X,table) \rightarrow DEL[on(X,table)]$

ADL Operators.

1. Arbitrary formulas as preconditions.
   - in a CW-KB we can evaluate whether or not the preconditions hold for an arbitrary precondition.
2. They can have conditional and universal effects.
   - Similarly we can evaluate the condition to see if the effect should be applied, and find all bindings for which it should be applied.

Specify atomic changes to the knowledge base.

- CW-KB can be updated just as before.