Plans are worthless, but planning is everything.

There is a very great distinction because when you are planning for an emergency you must start with this one thing: **the very definition of "emergency" is that it is unexpected, therefore it is not going to happen the way you are planning.**

Dwight D. Eisenhower

Sidebar – Architectures

- The "**Game Loop**"

  while game is running

    process inputs

    update game world

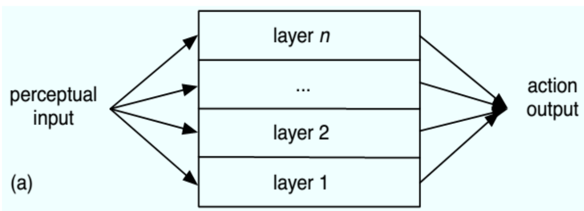
    generate outputs

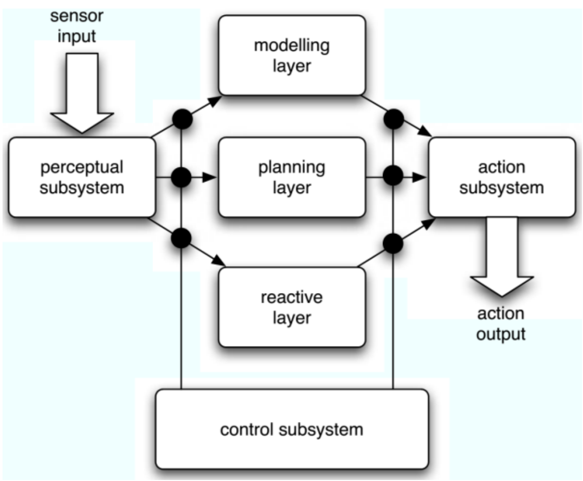
```
while( user doesn't exit )
  check for user input
  run AI
  move enemies
  resolve collisions
  draw graphics
  play sounds
end while
```

https://en.wikipedia.org/wiki/Game_programming

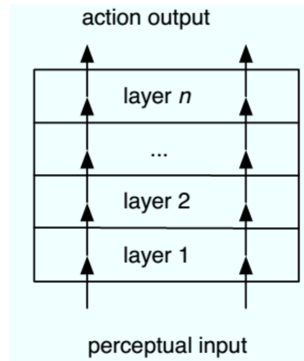
http://www.informit.com/articles/article.aspx?p=2167437&seqNum=3

Decision making

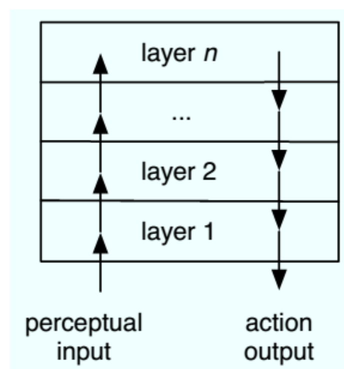




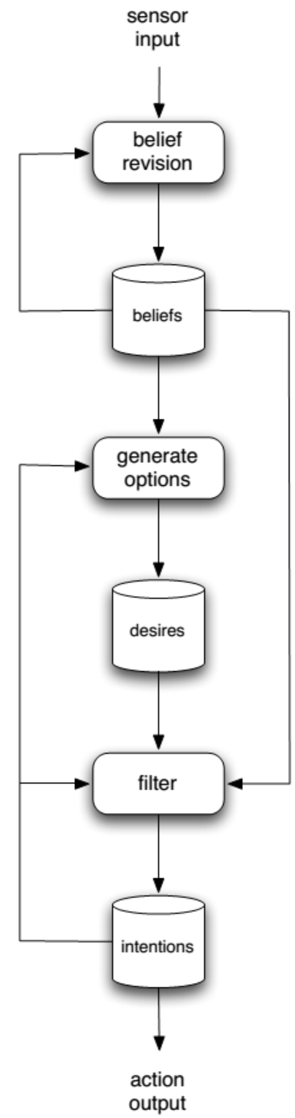
Brooks Subsumption Arch.



One-pass



Two-pass



BDI

See Multiagent Systems 2nd ed, Gerhard Weiss

(mdps)

An **agent** is a **computational entity** such as a software program or a robot that is **situated in some environment** and that to some extent is able to **act autonomously** in order to achieve its **design objectives**. As **interacting entities**, agents do not simply exchange data but are **actively engaged in cooperative and competitive scenarios**. They may **communicate** on the basis of semantically rich languages, and they **achieve agreements** and **make decisions** on the basis of processes such as negotiation, argumentation, voting, auctioning, and coalition formation. As intelligent entities, agents **act flexibly, that is, both reactively and deliberatively**, in a variety of environmental circumstances on the basis of processes such as planning, learning, and constraint satisfaction. As **autonomous entities**, agents have **far-reaching control over their behavior** within the frame of their objectives**, possess decision authority** in a wide variety of circumstances, and are **able to handle complex and unforeseen situations on their own** and without the intervention of humans or other systems. And as entities situated in some environment, **agents perceive their environment** at least partially and **act upon their environment without being in full control of it**.

Class N-2

1. How can we describe decision making?
2. What makes FSMs so attractive?
3. What might make us not choose an FSM?
4. Two drawbacks of FSMs, and how to fix?
5. What are the performance dimensions we tend to assess?
6. What are two methods we discussed to learn about changes in the world state?

Class N-1

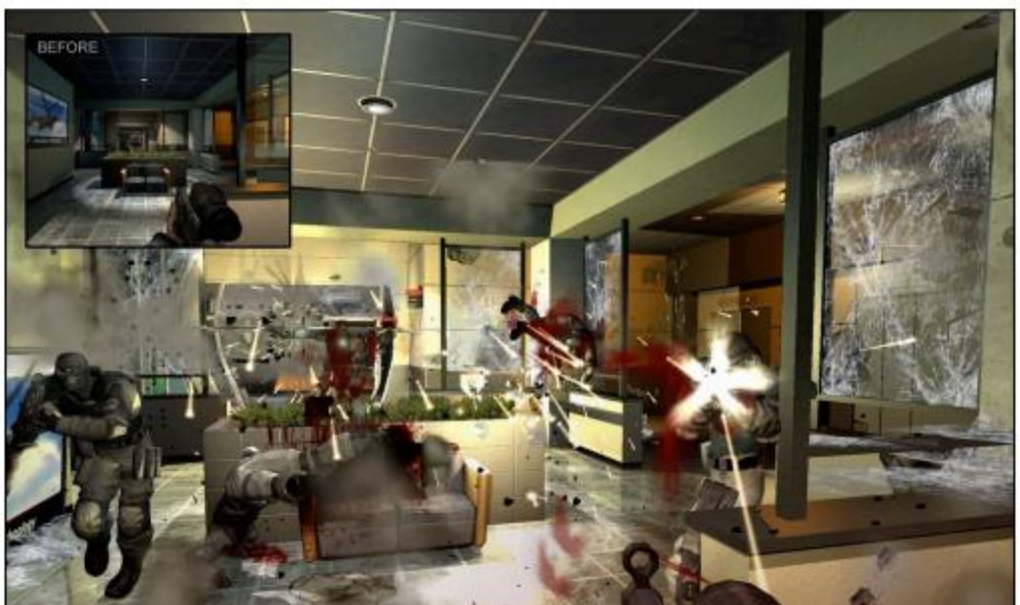
1. How can we describe decision making?
2. What do the algorithms we've seen share?
3. What are the dimensions we tend to assess?
4. FSMs/Btrees: ____ :: Planning : _____
5. For the 2nd blank, we need m_____s.
6. When is reactive appropriate? Deliberative?
7. What is the 'hot-potato' passed around (KE)?
8. H______ have helped in most approaches.
9. Which approach should you use?

Three States and a Plan: The A.I. of F.E.A.R.

Jeff Orkin
Monolith Productions / M.I.T. Media Lab, Cognitive Machines Group
http://www.jorkin.com

If the audience of GDC was polled to list the most common A.I. techniques applied to games, undoubtedly the top two answers would be A* and Finite State Machines (FSMs). Nearly every game that exhibits any A.I. at all uses some form of an FSM to control character behavior, and A* to plan paths. *F.E.A.R.* uses these techniques too, but in unconventional ways. The FSM for characters in *F.E.A.R.* has only three states, and we use A* to plan sequences of actions as well as to plan paths. This paper focuses on applying planning in practice, using *F.E.A.R.* as a case study. The emphasis is demonstrating how the planning system improved the process of developing character behaviors for *F.E.A.R.*

We wanted *F.E.A.R.* to be an over-the-top action movie experience, with combat as intense as multiplayer against a team of experienced humans. A.I. take cover, blind fire, dive through windows, flush out the player with grenades, communicate with teammates, and more. So it seems counter-intuitive that our state machine would have only three states.



http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf

Decision Making: Planning

2016-06-16

With extra thanks to Dana Nau, Hector Munoz-Avila, and Mark Riedl

OOB Slides

- Now Decision Making
  - Getting there: Steering
  - Getting there: graphs, search, movement (path planning)
  - Classic AI vs Game AI

Decision Making

- Classic AI:
  - making the optimal choice of action (given what is known or is knowable at the time) that maximizes the chance of achieving a goal or receiving a reward (or minimizes penalty/cost)
- Game AI:
  - choosing the right goal/behavior/animation to support the experience
- Today: One of closest overlaps

Planning in Games

- Motivation: more realism
  - Agents should be motivated by goals
- FSM vs. planning
  - FSM tells the agent what to do
  - With planning, agent is given a goal and figures out what to do

Planning

- Finding a sequence of actions to achieve a goal
- Basic planning comes down to search
- Need to find appropriate heuristic

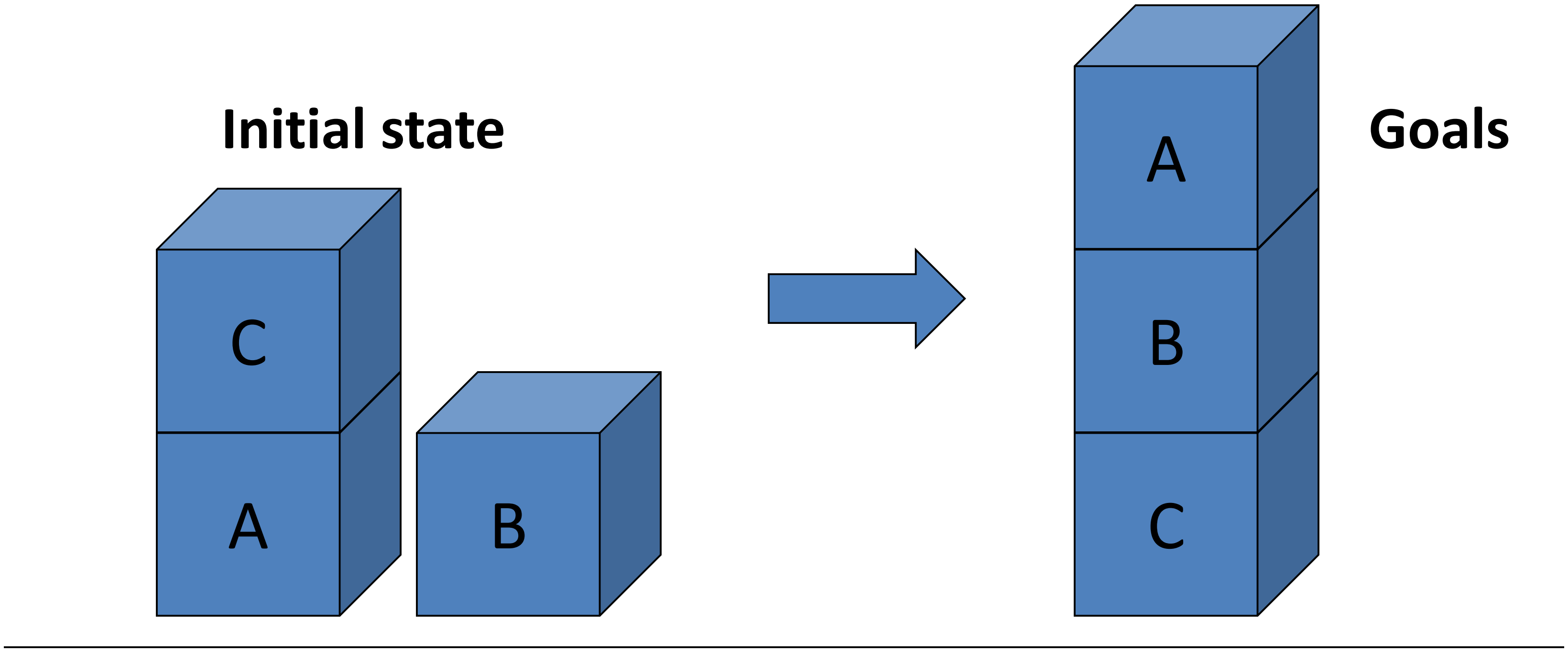
Action Planning vs. Path planning

- Same algorithm
- Different action representation
  - Pathplanning:
    - Cells
    - Node links
  - Action planning:
    - Action data structure
      - Action name
      - Precondition
      - Effect (add/remove)

Representation

- Representation is key
- States
- Goals
- Actions

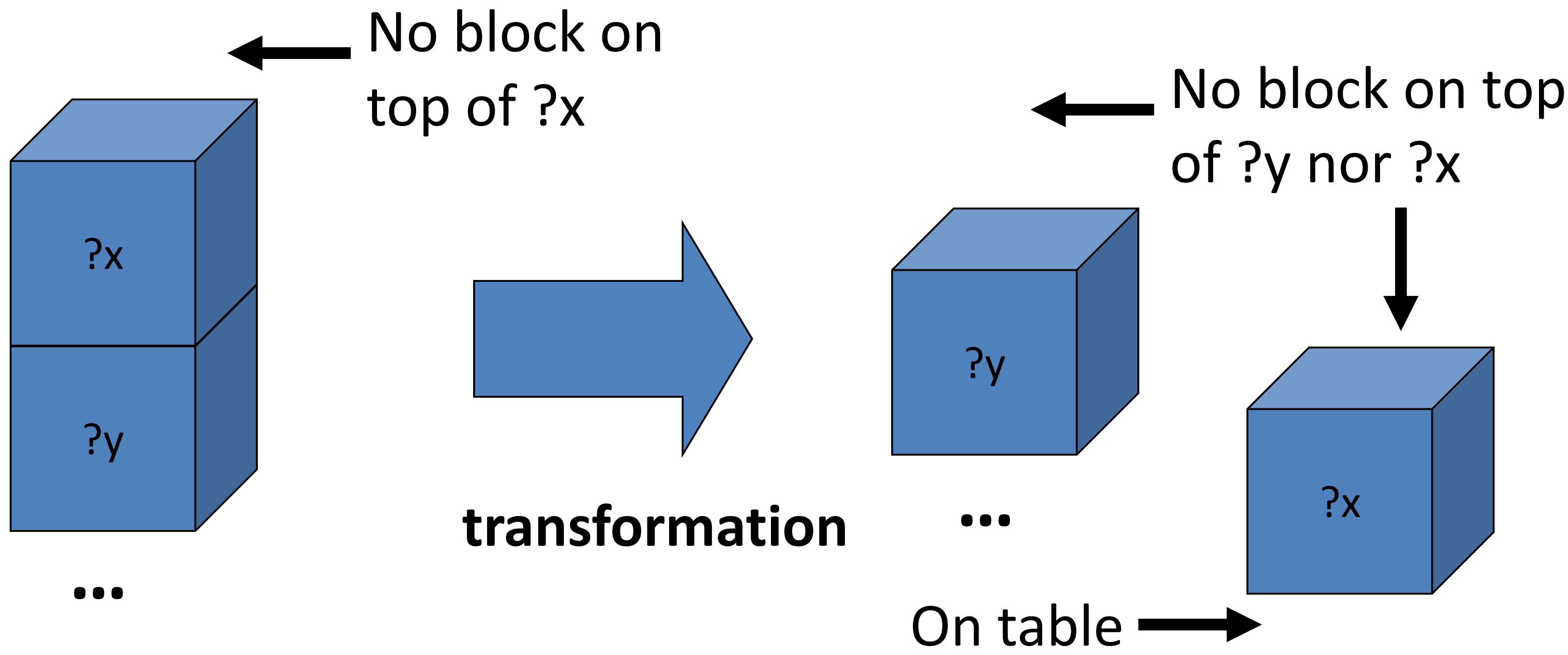
General-Purpose Planning: State & Goals



- **Initial state**: (on A Table) (on C A) (on B Table) (clear B) (clear C)
- **Goals**: (on C Table) (on B C) (on A B) (clear A)

(Ke Xu)

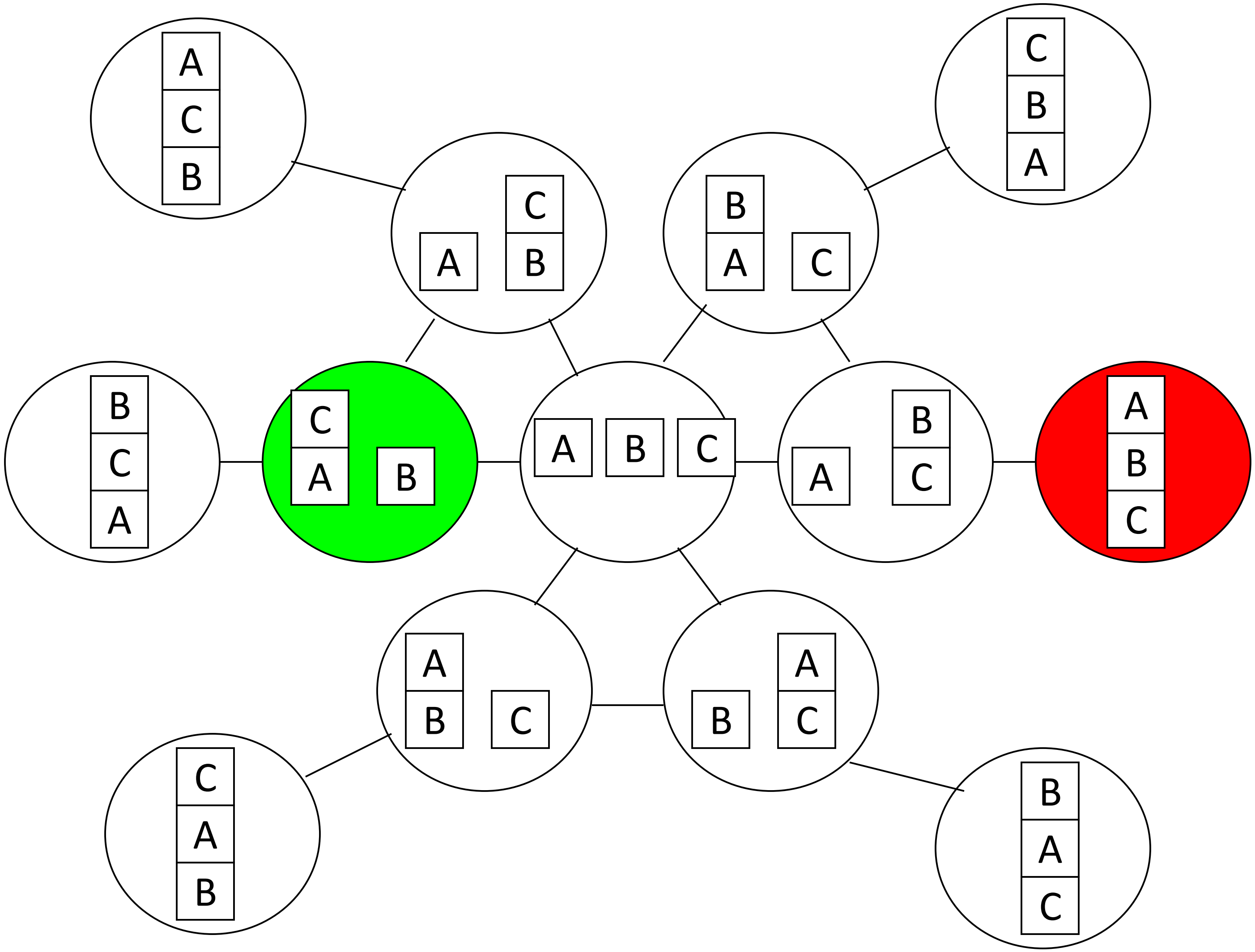
General-Purpose Planning: Operators



Operator: (Unstack ?x)

- **Preconditions**: (on ?x ?y) (clear ?x)
- **Effects**:
  - **Add**: (on ?x table) (clear ?y)
  - **Delete**: (on ?x ?y)

Search Space (World States)



(Michael Moll)

Some Examples

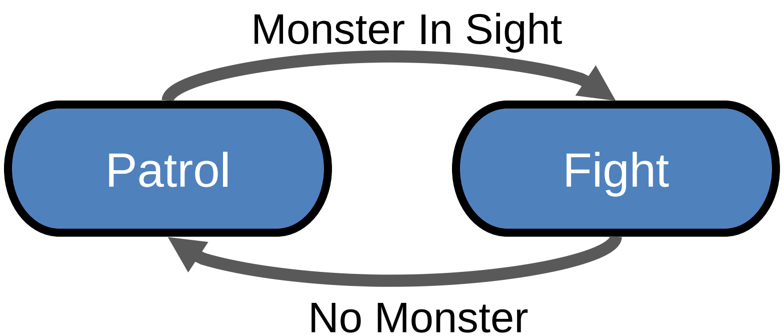
Which of the following problems can be modeled as AI planning problems?

- **Route search**: Find a route between Lehigh University and the Naval Research Laboratory
- **Project management**: Construct a project plan for organizing an event (e.g., the Musikfest)
- **Military operations**: Develop an air campaign
- **Information gathering**: Find and reserve an airline ticket to travel from Newark to Miami
- **Game playing**: plan the behavior of a computer controlled player
- **Resources control**: Plan the stops of several of elevators in a skyscraper building.

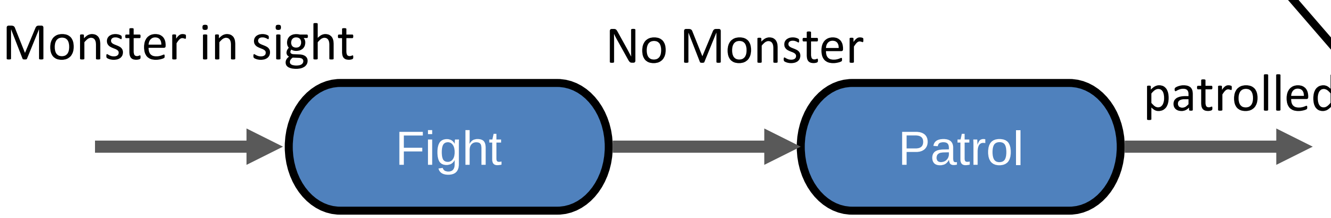
Answer: ALL!

FSM vs AI Planning

FSM:



A resulting plan:



Planning Operators

- **Patrol**
  - Preconditions:
    No Monster
  - Effects:
    patrolled
- **Fight**
  - Preconditions:
    Monster in sight
  - Effects:
    No Monster

Neither is more powerful than the other

But Planning Gives More Flexibility

- "Separates implementation from data" --- Orkin

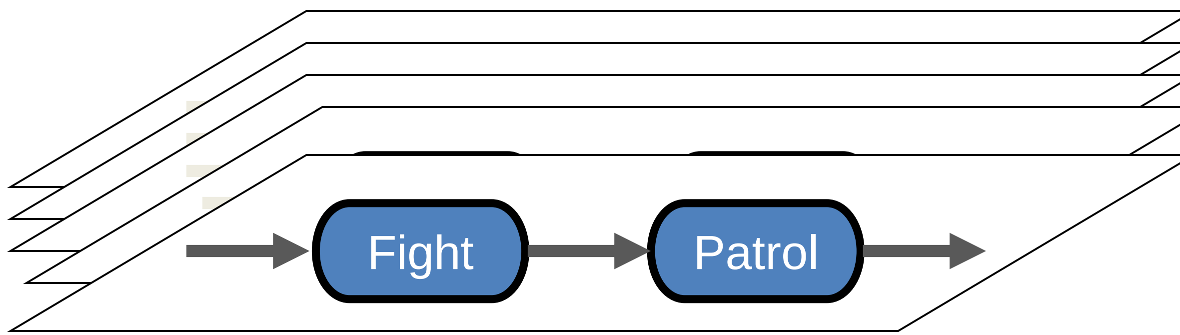
reasoning knowledge

Planning Operators

- **Patrol**
  - ➢Preconditions: No Monster
  - ➢Effects: patrolled
- **Fight**
  - ➢Preconditions: Monster in sight
  - ➢Effects: No Monster

…

Many potential plans:



…

If conditions in the state change making the current plan unfeasible: replan!

FSMs vs. Planning

- FSMs tell agents how to behave in every situation
- In planning systems, agents have goals and a set of actions. Agent decides how to apply those actions to goals.

But… Does Classical Planning Work for Games?

F.E.A.R. not!

Planning in F.E.A.R.

- Agents need to autonomously use environment to satisfy their goals
- Agent will not do anything without a goal
- Agent types defined by what actions are available to them

Benefits of Planning in F.E.A.R.

- Decoupled goals and actions
- Layered behaviors
- Dynamic problem solving

Decoupled Goals and Actions

- Each character has own Action Set
- Allows for late additions in character types
- Allows for shared information between goals

Layered Behaviors

- Developer Goals:
  - Agents should always try to stay covered.
  - Agents should never leave cover unless threatened and other cover is available
  - Agents should fire from cover as best they can

Layered Behaviors

- Basic Goal: KillEnemy
  - Satisfied by *Attack* action
- Additional Goal: Dodge
  - Satisfied by *DodgeShuffle* or *DodgeRoll*
- Goals and actions for melee attacks, taking cover, etc.
- With planning, easy to add goals and actions

Dynamic Problem Solving

- Replanning gives the AI the power to adjust to new scenarios
- AI records obstacles in memory and uses that knowledge later during replanning

Differences from STRIPS

- Action costs
- World state representation

Other Games with Planning

- Empire: Total War
- Fallout 3
- Killzone

General Purpose vs. Domain-Specific

Planning: find a sequence of actions to achieve a goal

General purpose: symbolic descriptions of the problems and the domain. The plan generation algorithm the same

Advantage: - opportunity to have clear semantics
Disadvantage: - symbolic description requirement

Domain Specific: The plan generation algorithm depends on the particular domain

Advantage: - can be very efficient
Disadvantage: - lack of clear semantics
- knowledge-engineering for plan generation

Classes of General-Purpose Planners

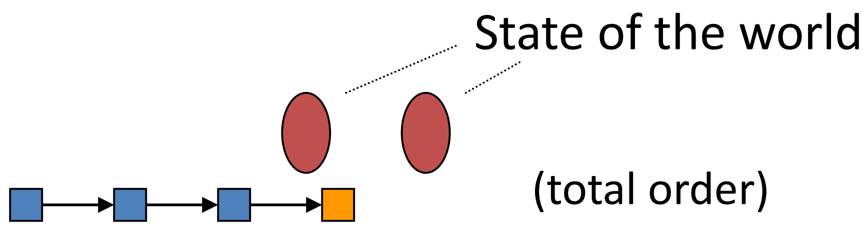
General purpose planners can be classified according to the space where the search is performed:

- state
- plan
- Hierarchical
- Disjunctive plans
- SAT

We are going to discuss these forms

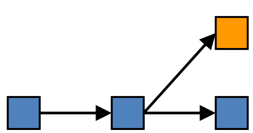
State- and Plan-Space Planning

- **State-space** planners transform the state of the world. These planners search for a sequence of transformations linking the starting state and a final state



(total order)

- **Plan-space** planners transform the plans. These planners search for a a plan satisfying certain conditions



(partial-order, least-commitment)

STRIPS (Fikes & Nilsson)

- States
  - at(plane1, Atlanta)
- Goals
  - A particular state, or part of a particular state
- Actions (“operators”)
  - Action Schema

STRIPS actions

State: airport(LAX), airport(ATL), at(plane1, ATL),
at(plane2, LAX), path(ATL, LAX), ~at(plan1, LAX), …

↑
(All other things that are true or non-true)

Fly (?p, ?from, ?to)

Precondition: at(?p, ?from), plane(?p), airport(?from),
airport(?to), path(?from, ?to), ?from ≠ ?to

Effect: at(?p, ?to), ¬at(?p, ?from)

*Also a call to the game engine to play animation or run a function

Initial state: gunForSale, ammoForSale, possumAlive, ~gunLoaded, ~hasFood, ~hasGun, ~criminal, ~hasAmmo, ~rich, smellsFunny

Goal state: rich, hasFood

Action: RobBank
PRE: ~rich, hasGun, gunLoaded
EFFECT: rich, criminal

Action: ShootPossum
PRE: ~hasFood, hasGun, gunLoaded, possumAlive
EFFECT: hasFood, ~gunLoaded, ~possumAlive

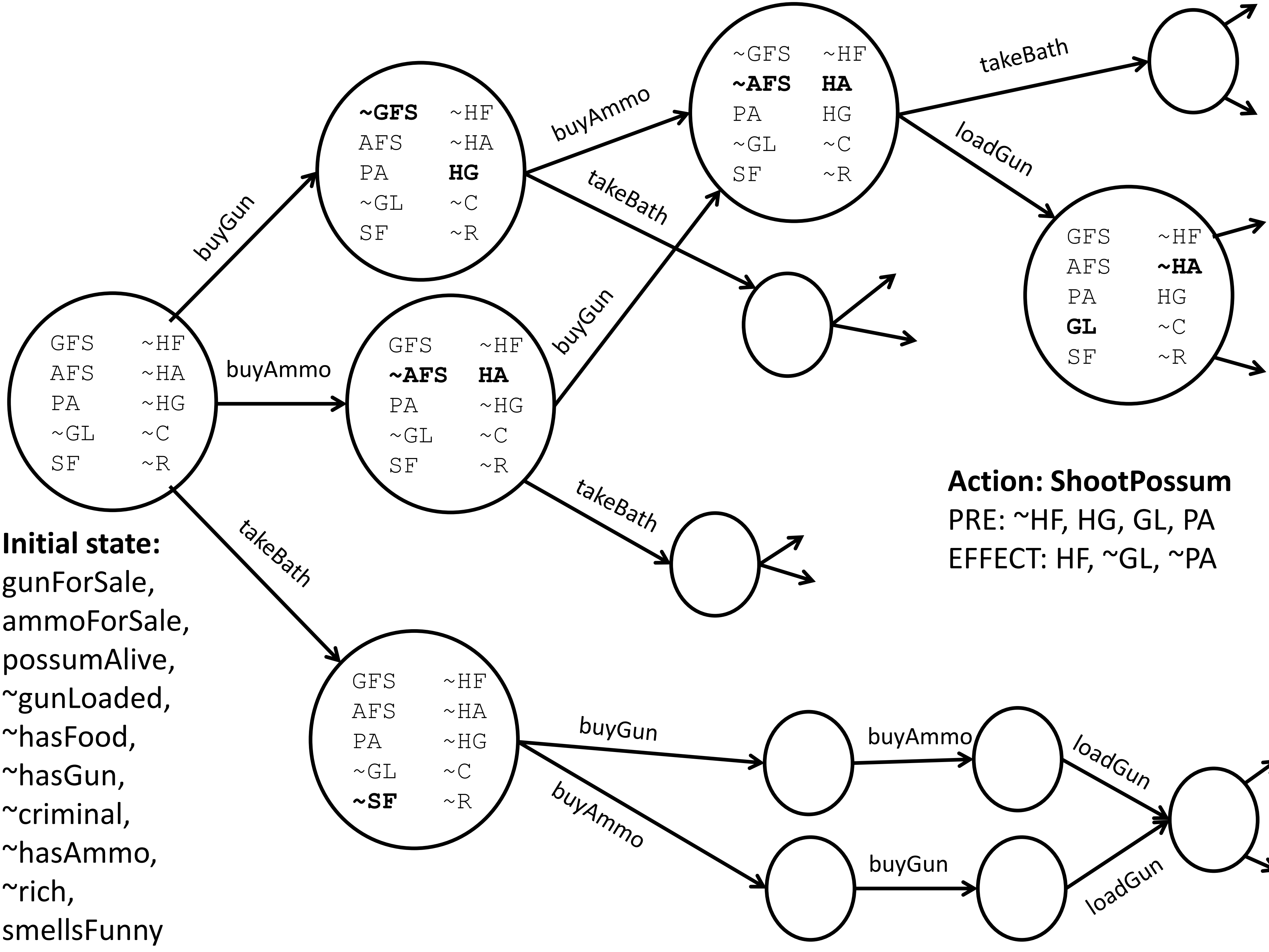
Action: LoadGun
PRE: hasGun, hasAmmo, ~gunLoaded
EFFECT: gunLoaded, ~has Ammo

Action: BuyGun
PRE: gunForSale, ~hasGun, ~criminal
EFFECT: ~gunForSale, hasGun

Action: BuyAmmo
PRE: ammoForSale, ~hasAmmo
EFFECT: ~ammoForSale, hasAmmo

Action: TakeBath
PRE: smellsFunny
EFFECT: ~smellsFunny

Action: PlayInMud
PRE: ~smellsFunny
EFFECT: smellsFunny



Action: ShootPossum

PRE: ~HF, HG, GL, PA

EFFECT: HF, ~GL, ~PA

Initial state:

gunForSale,
ammoForSale,
possumAlive,
~gunLoaded,
~hasFood,
~hasGun,
~criminal,
~hasAmmo,
~rich,
smellsFunny

Forward Planning

- State-space search
- Start with initial state
- Applicable actions are those whose preconditions are satisfied by current state.
- Goal test
- Optional action cost
- Any complete graph search algorithm (e.g. A*)

Backward Planning

- State-space search
- Benefit: only consider relevant actions
- Actions must be consistent
- Graph search algorithm

Good, Bad, and Ugly

- Forward chaining

Good, Bad, and Ugly

- Forward chaining
  - Irrelevant actions cause high branching factor

Good, Bad, and Ugly

- Forward chaining
  - Irrelevant actions cause high branching factor
- Backward chaining

Good, Bad, and Ugly

- Forward chaining
  - Irrelevant actions cause high branching factor
- Backward chaining
  - Practical branching factor can be much lower because it only considers necessary actions
  - Total ordering susceptible to long backtracks when effects negate earlier decisions
- Start thinking: More informed? Total order?

Heuristics

- f() = g() + h()
- g() is sum of action costs, which can be arbitrary
- How do you estimate the distance to the goal?

Heuristics

- Informs decision into which node (state) to expand
- Admissible heuristics allow for A*
  - Relax the planning problem
  - Subgoal independence assumption

Heuristic Search Planning

- Works using backward chaining
- Computes heuristic values for each preconditions based on graph analysis
  - Benefit: Only do it once as pre-computation step
- Heuristic
  1. Cost of action is *maximum* over costs of preconditions (admissible, but not informed)
  2. Cost of action is *sum* over costs of preconditions (informed, but not admissible)

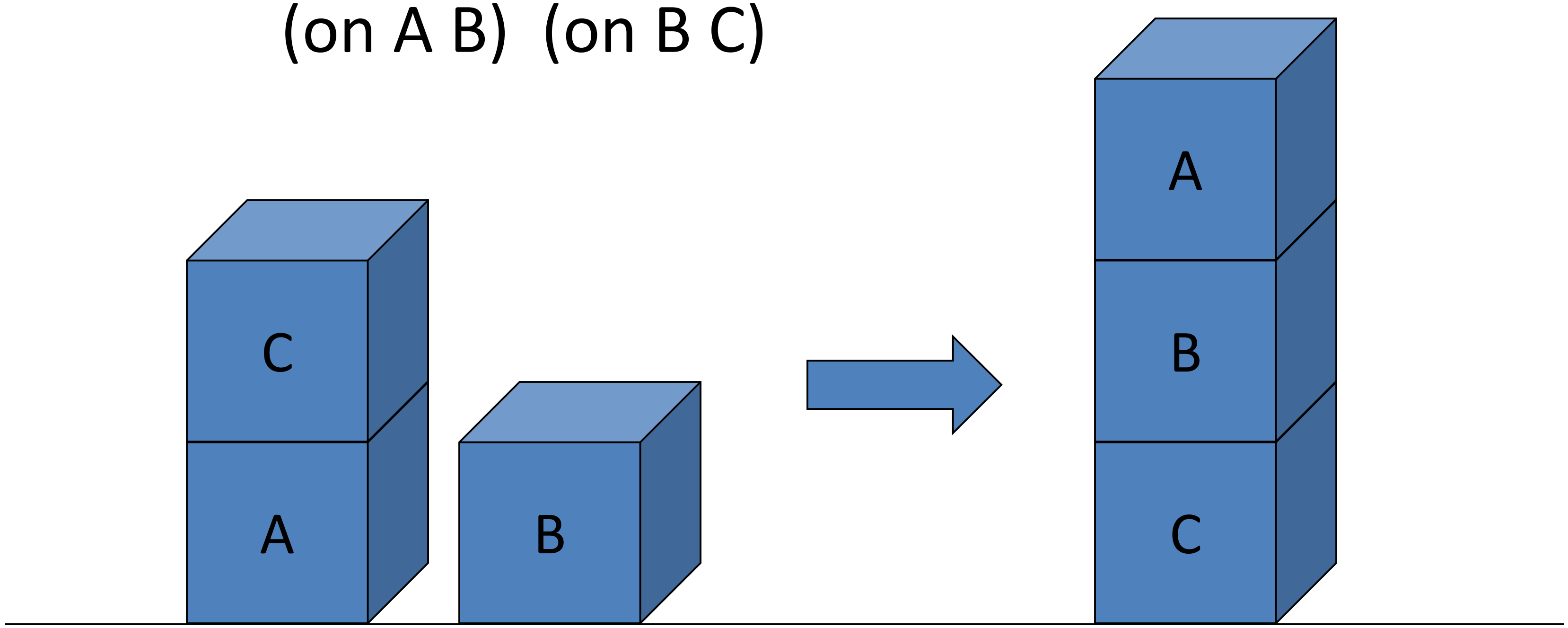
Partial-Order Planning

- Avoid total ordering (previous examples)
- Partial ordering treats every precondition as a sub-problem to be solved independently
- Reconcile solutions to sub-problems when they interact with each other
- Don’t commit to any ordering before strictly necessary
  - Least-commitment planning

Why Plan-Space Planning?

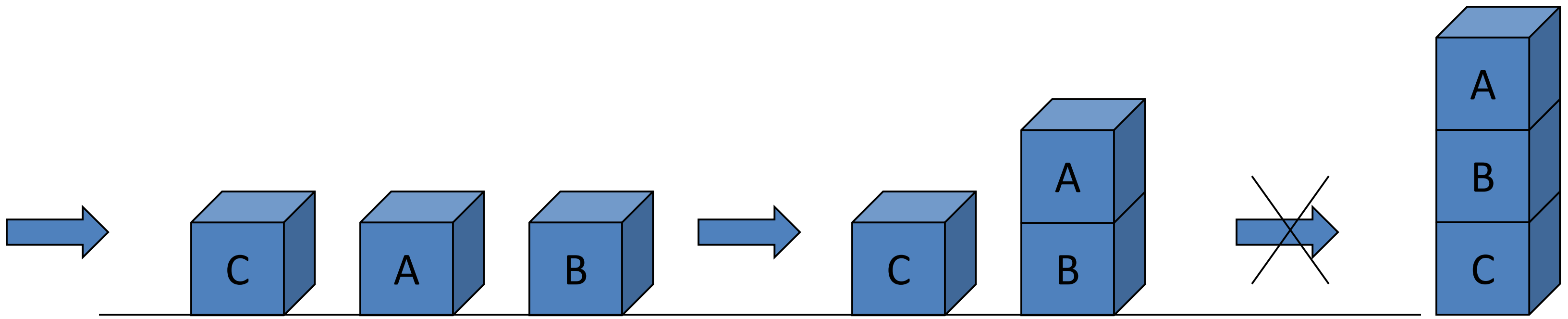
- 1. Motivation: “Sussman Anomaly”
  - Two subgoals to achieve:

    (on A B) (on B C)



Why Plan-Space Planning?

- Problem of state-space search:
  - Try (on A B) first:
    - put C on the Table, then put A on B



    - Accidentally wind up with A on B when B is still on the Table
    - We can not get B on C without taking A off B
    - Try to solve the first subgoal first appears to be mistaken

Partial-Order Planning

- Plan-search rather than state-search
- Plans are made up of:
  - Actions used
  - Ordering constraints
  - Causal links
  - Open preconditions

Partial-Order Planning

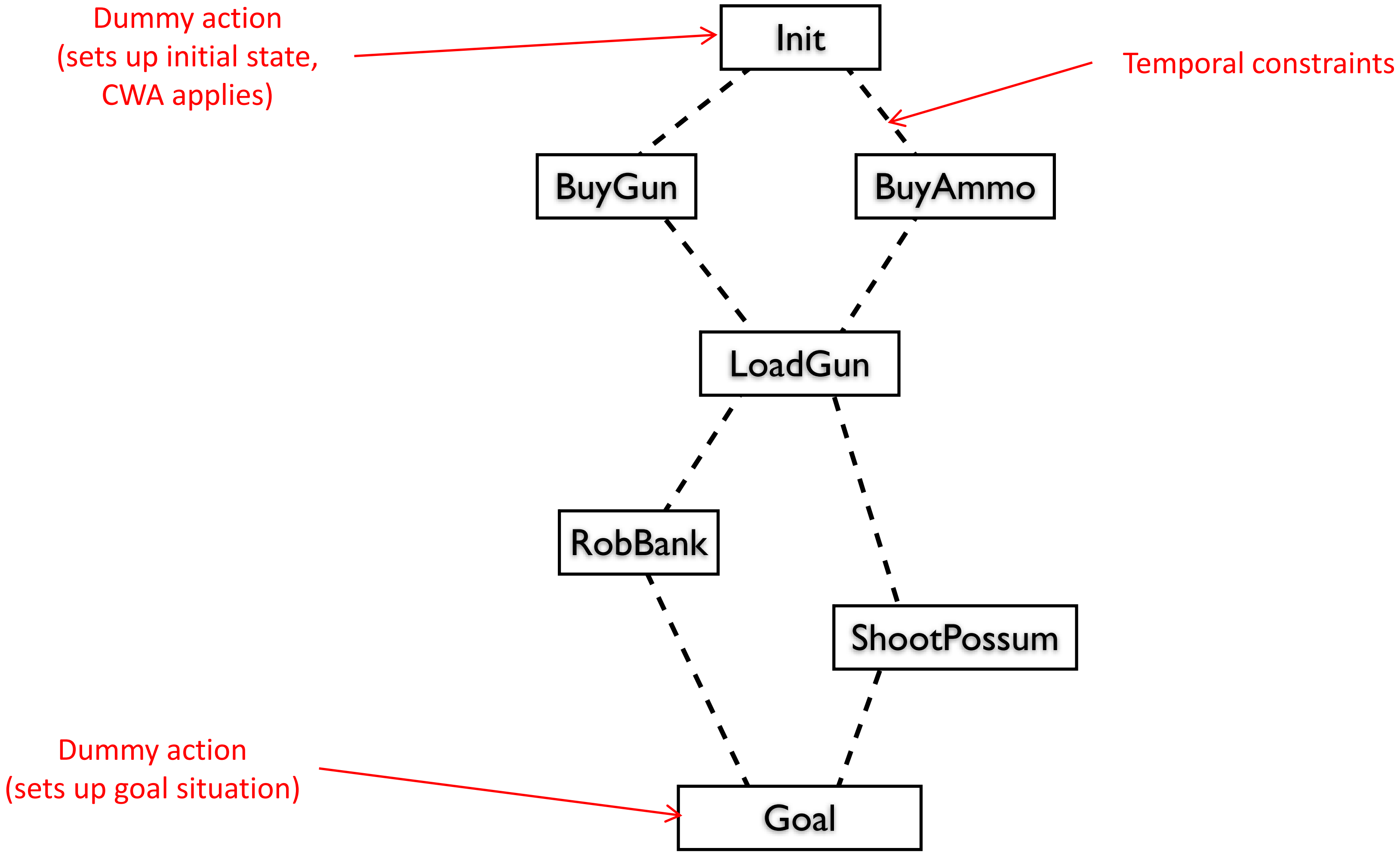
- Conflict
  - An action C conflicts with A -^{p}--> B if C has ¬p as an effect AND C can occur between A and B
- Consistent Plans
  - No cycles in ordering constraints
  - No conflicts with causal links
- Solution
  - Consistent plan with no open preconditions

POP Algorithm

- Start with initial plan [Start, Finish] where Start<Finish.
- Arbitrarily pick one open precond *p*
- Generate successor plans for every possible consistent way of selecting an action A that achieves *p*
- Add new causal link to plan, and resolve conflicts (if necessary)

POP Heuristics

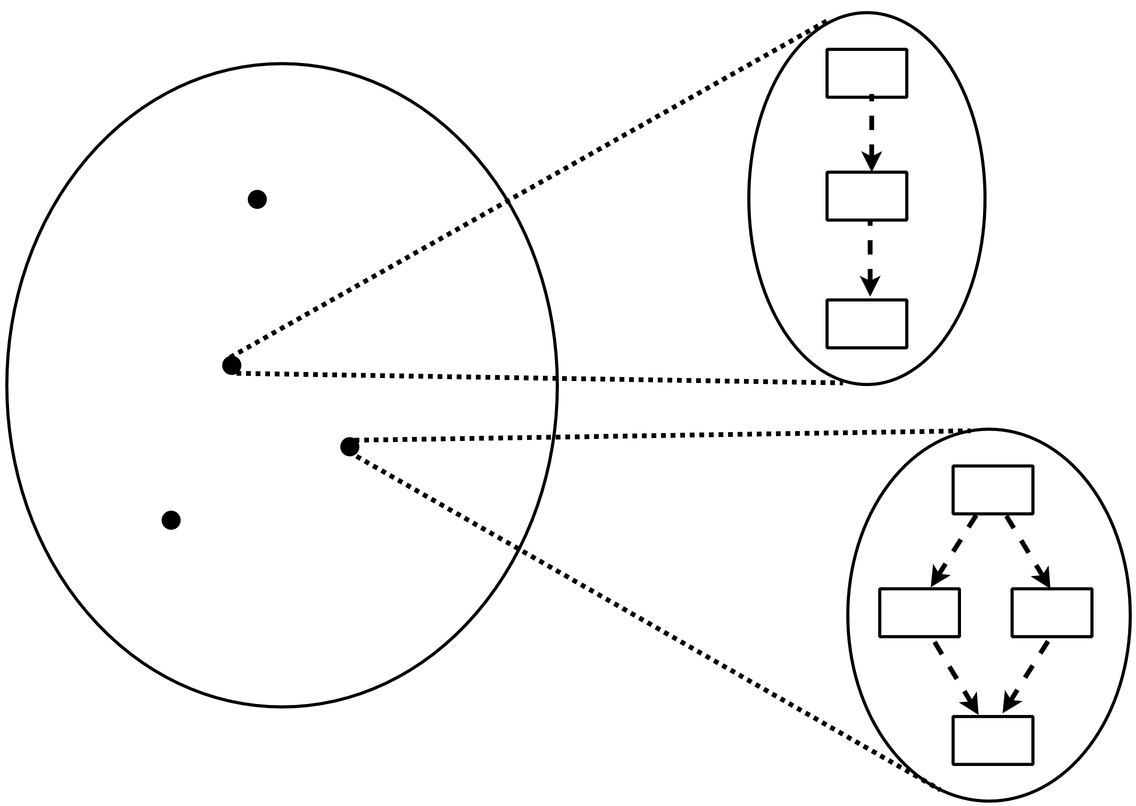
- Less understanding of how to create accurate heuristics for POP than total-order planning.
- Obvious heuristic: number of open preconditions
- Most-constrained-variable



- When it comes time to execute the plan, create a total ordering
- Partial-order plan is a set of total-order plans
- How many total-order plans?

POP Algorithm

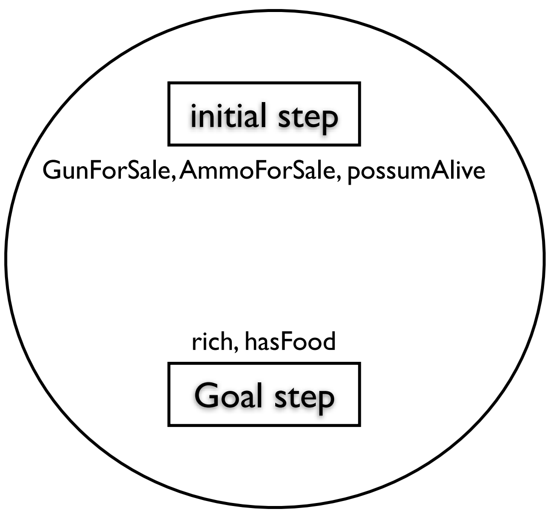
- Plan space search (AKA refinement search)



- **Where do you start?**
- **Where do you end?** (how do you know when you are looking at a valid solution?)
- **How do you move through space?** (how do you generate successors?)

Where do you start?

- The empty plan



Plan has a **flaw**: a reason why it cannot be a solution

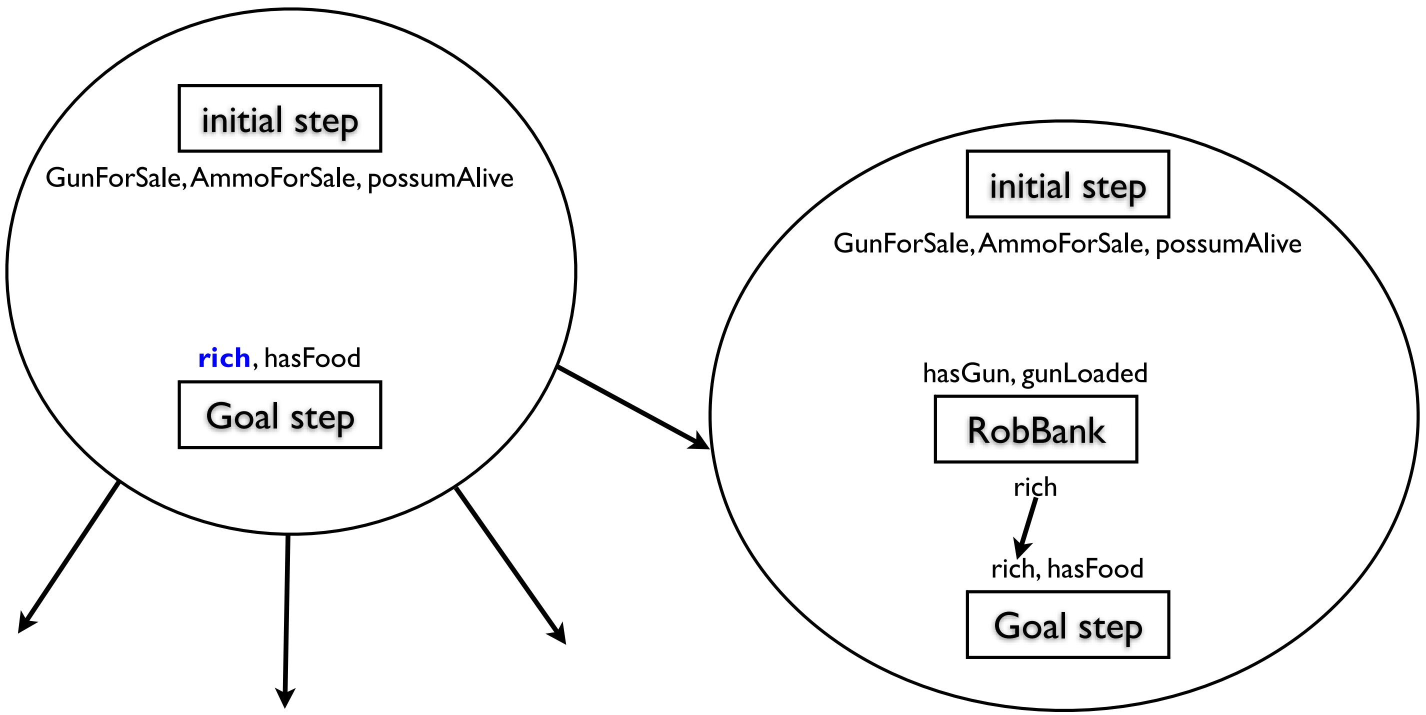
Two flaws: Nothing makes either goal condition true

Where do you stop?

- When you see a plan with no flaws

How do you move?

- Pick a flaw (any flaw)
- Successors are all ways of fixing the flaw
- May introduce new flaws



* All the ways of making rich true

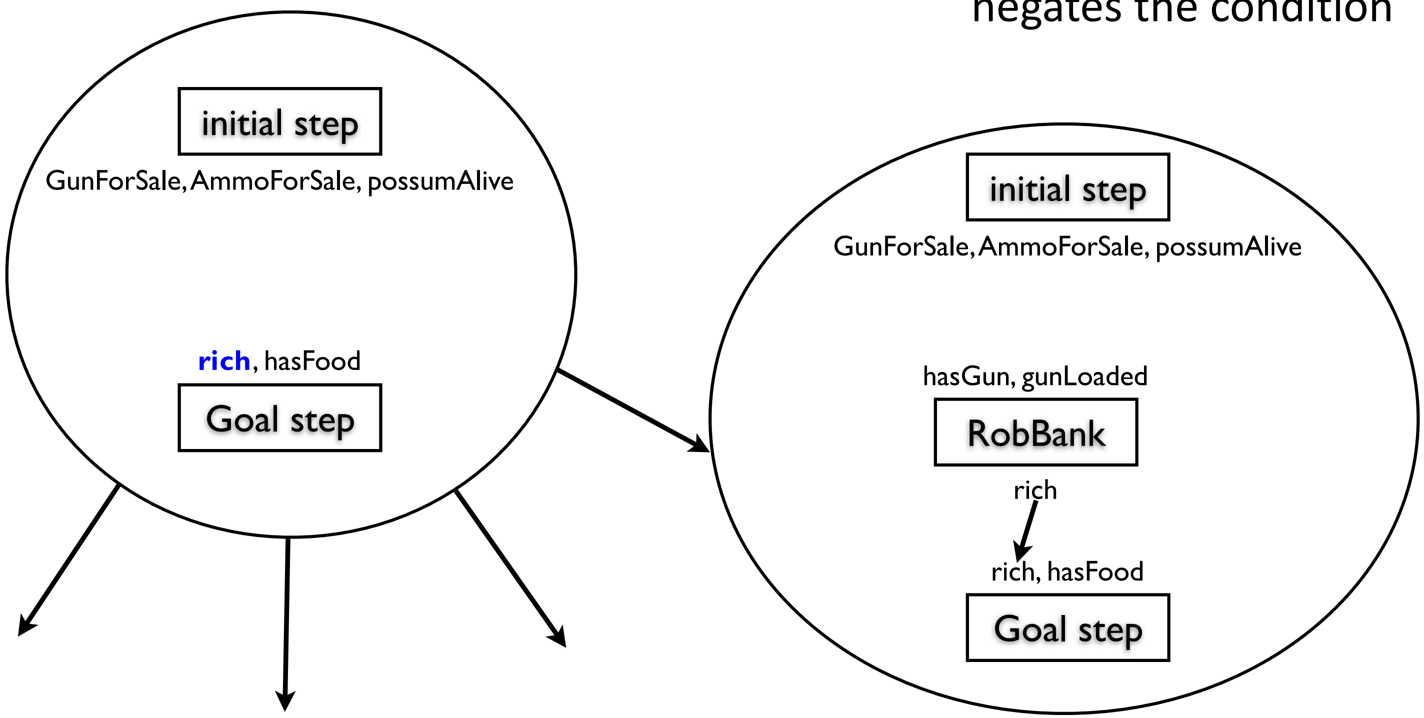
* Does it matter what order I pick flaws?

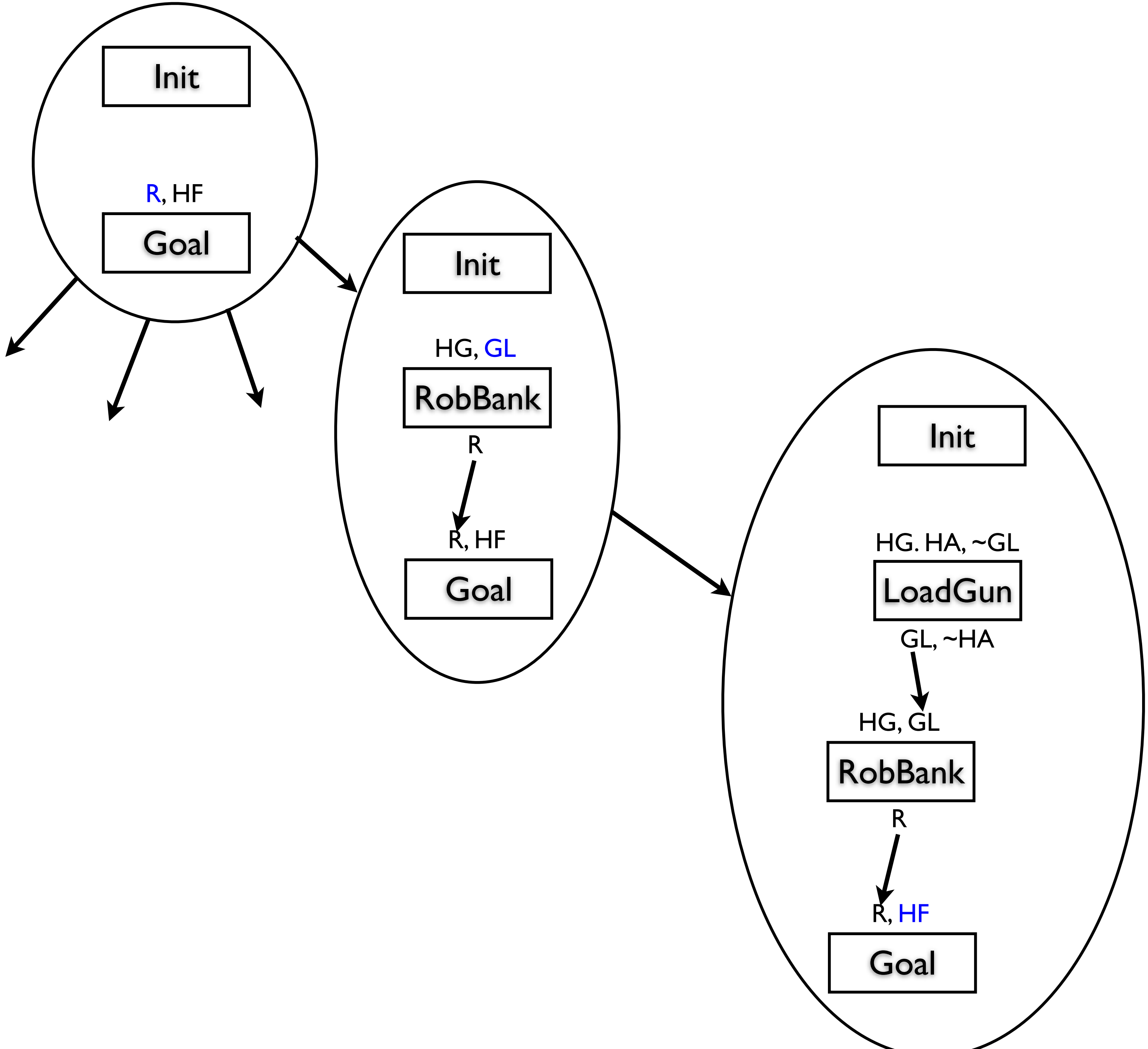
Open condition flaw:

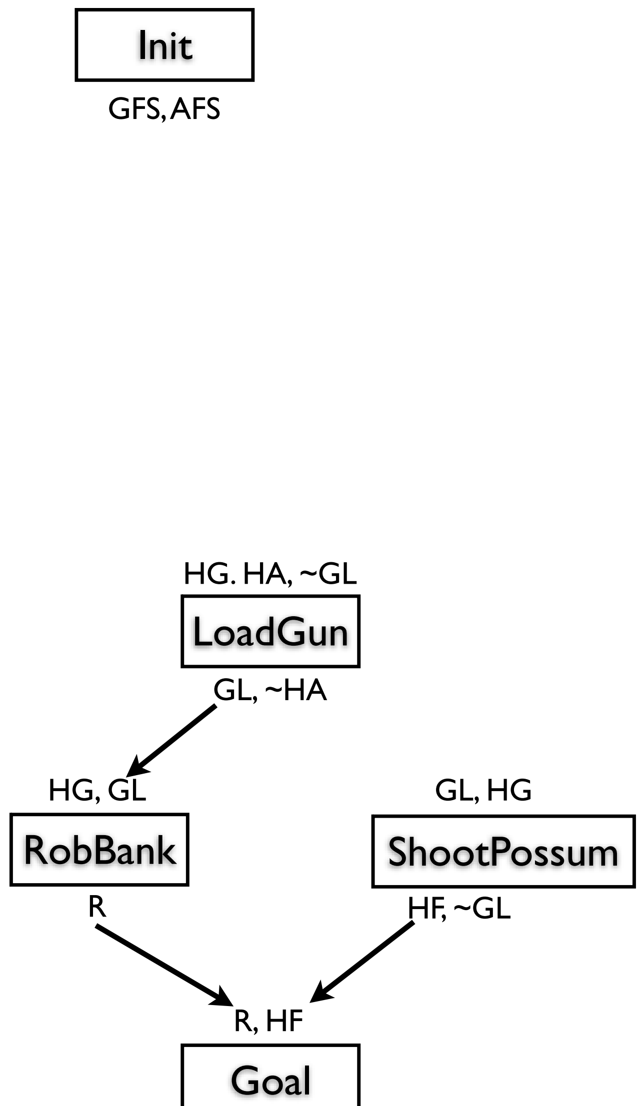
- A precondition that is not satisfied
- Pick an operator that has an effect unifying with the condition
  - Strategy #1: Add a new action
  - Strategy #2: Reuse an action

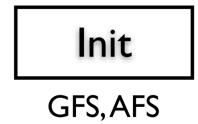
Causal link

- Tells us that a precondition is satisfied
- A protected interval
- Nothing can be put in this interval that negates the condition



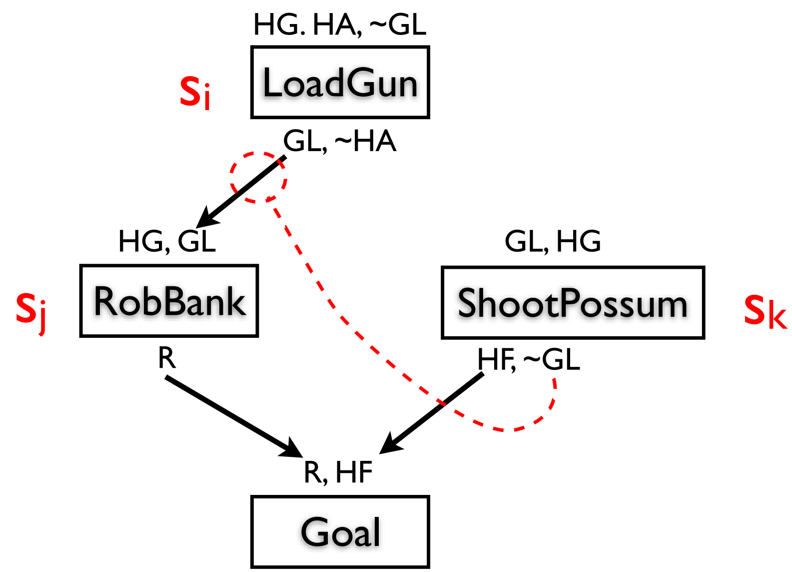

Init
R, HF
Goal
Init
HG, GL
RobBank
R
R, HF
Goal
Init
HG. HA, ~GL
LoadGun
GL, ~HA
HG, GL
RobBank
R
R, HF
Goal

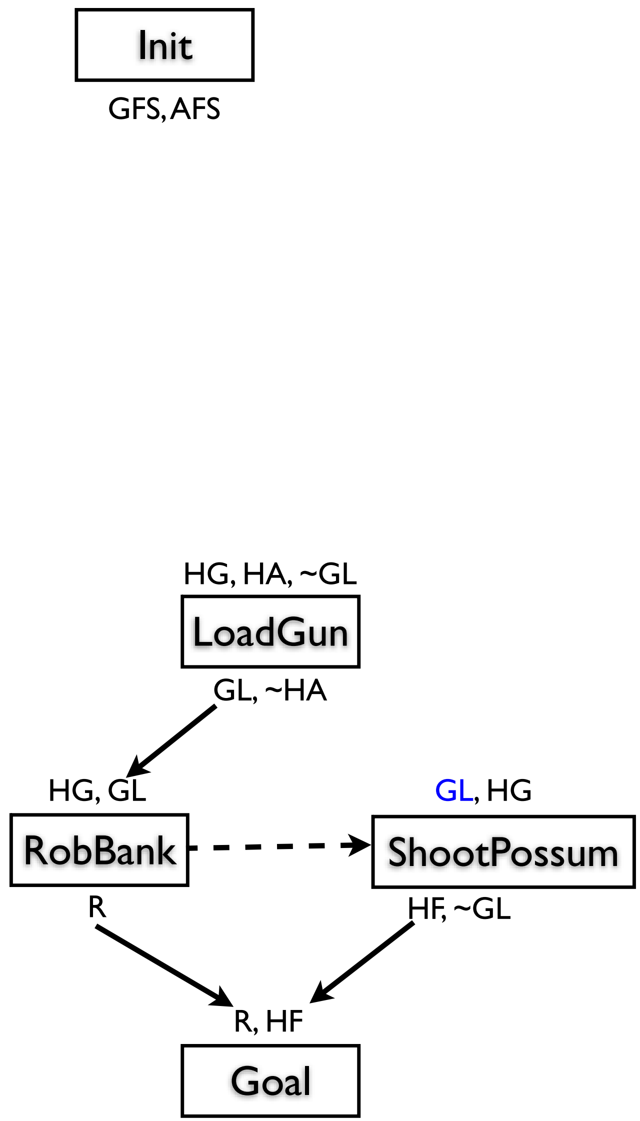

Init
GFS, AFS
HG. HA, ~GL
LoadGun
GL, ~HA
HG, GL
RobBank
R
GL, HG
ShootPossum
HF, ~GL
R, HF
Goal

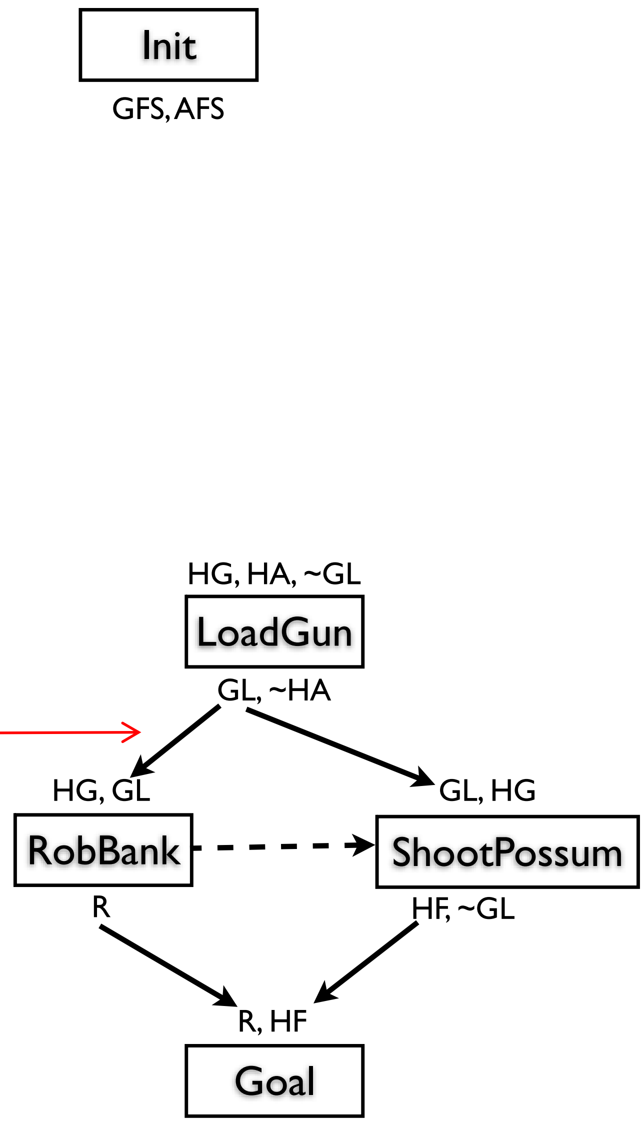


Causal threat

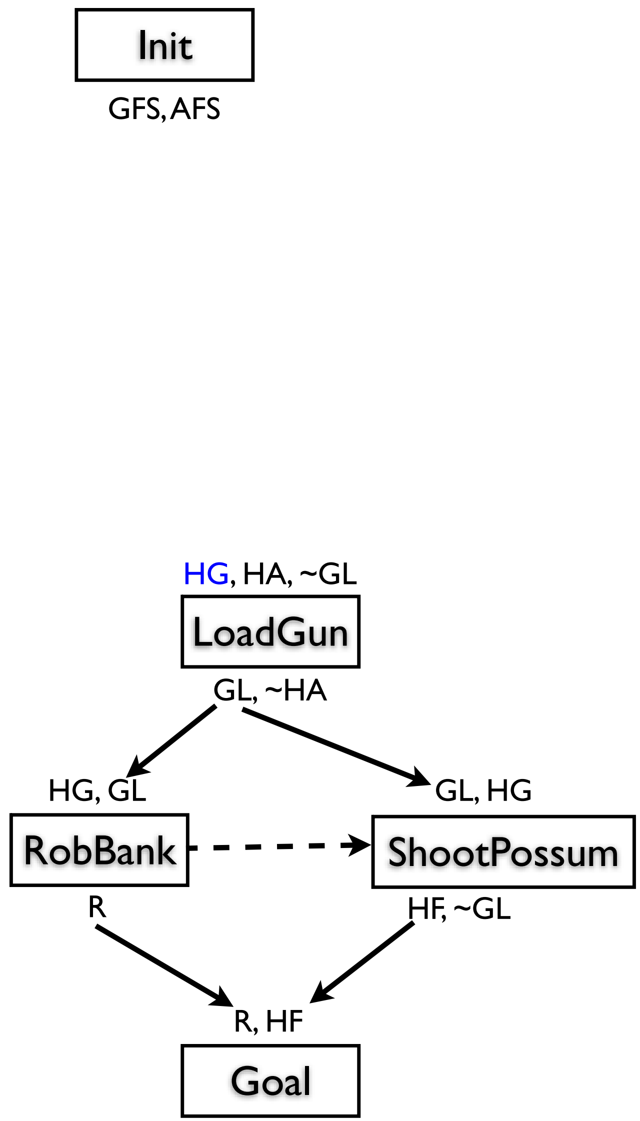
- Effect of an action <u>could</u> negate causal link
- Promote: s_k ordered before s_i
- Demote: s_k ordered before s_j

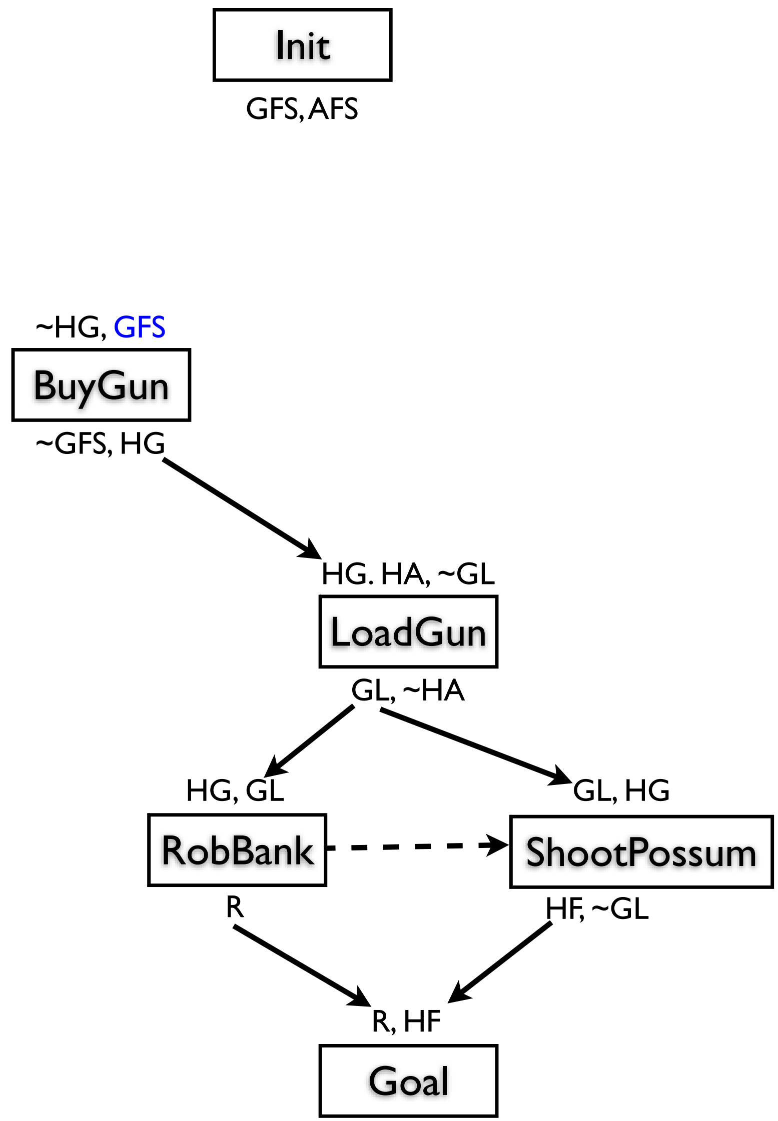


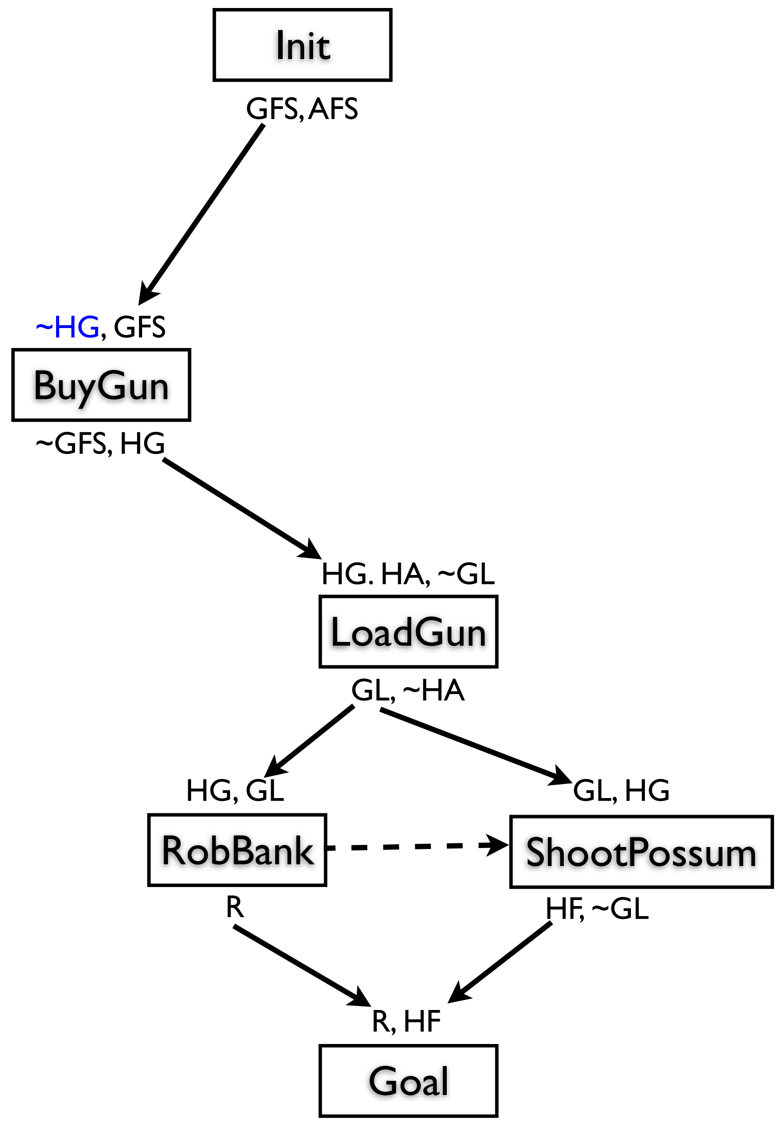

Init
GFS, AFS
HG, HA, ~GL
LoadGun
GL, ~HA
HG, GL
RobBank
GL, HG
ShootPossum
R
HF, ~GL
R, HF
Goal



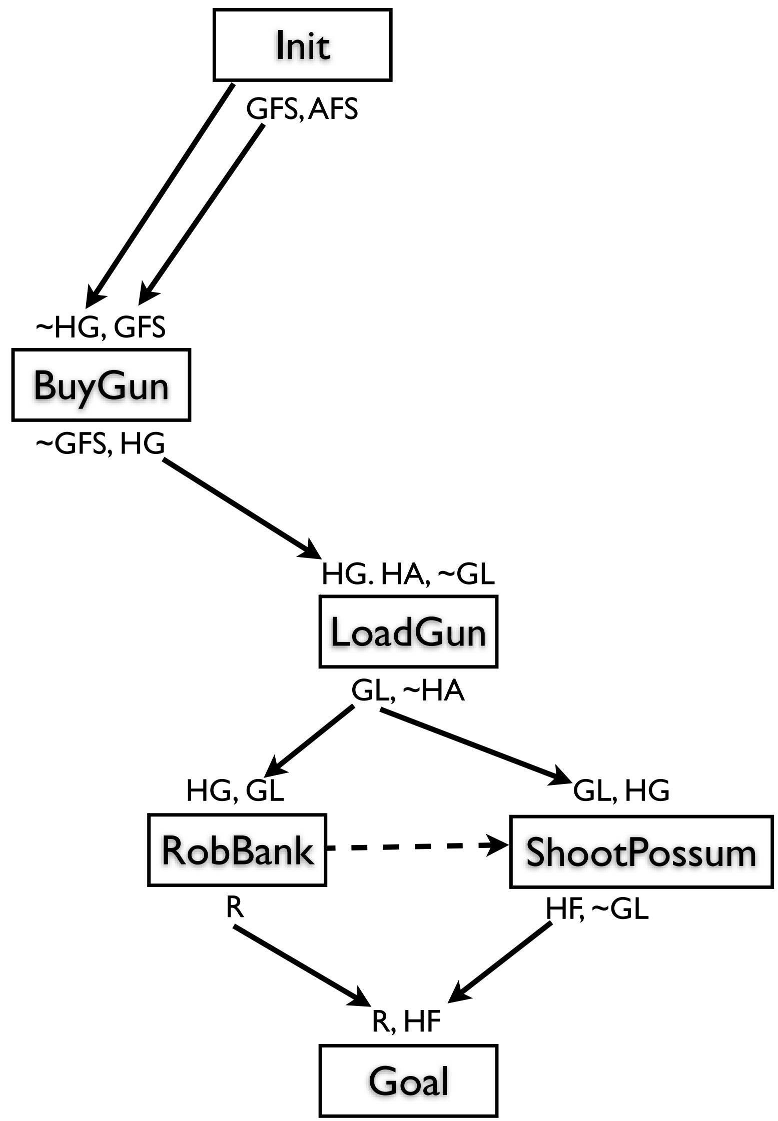
What is world state at this point?


Init
GFS, AFS
HG, HA, ~GL
LoadGun
GL, ~HA
HG, GL
GL, HG
RobBank
ShootPossum
R
HF, ~GL
R, HF
Goal


Init
GFS, AFS
~HG, GFS
BuyGun
~GFS, HG
HG. HA, ~GL
LoadGun
GL, ~HA
HG, GL
RobBank
R
GL, HG
ShootPossum
HF, ~GL
R, HF
Goal



Init
GFS, AFS
~HG, GFS
BuyGun
~GFS, HG
HG. HA, ~GL
LoadGun
GL, ~HA
HG, GL
RobBank
GL, HG
ShootPossum
R
HF, ~GL
R, HF
Goal



Init
GFS, AFS
~HG, GFS
BuyGun
~GFS, HG
HG. HA, ~GL
LoadGun
GL, ~HA
HG, GL
GL, HG
RobBank
ShootPossum
R
HF, ~GL
R, HF
Goal

```
agenda = { make_empty_plan(init, goal) }
current = pop(agenda)

WHILE agenda not empty and current has flaws DO:
        flaw = pick_flaw(current)
        IF flaw isa open condition flaw DO:
                FOREACH op in library that has an effect that unifies with o.c. DO:
                        successors += make_new_plan_from_new(…)
                FOREACH op in current that is before and has an effect that uifies with o.c.
DO:
                        successors += make_new_plan_reuse(…)
                IF a condition in init unifies with o.c. DO:
                        successors += make_new_plan_from_init(…)
                IF a condition is negative and CWA applies DO:
                        successors += make_new_plan_from_cwa(…)
        ELSE IF flaw isa causal threat flaw DO:
                successors += make_new_plan_promote(…)
                successors += make_new_plan_demote(…)
        agenda = agenda + successors
        current = pop(agenda)          <------------------ Insert sort
END WHILE

RETURN current or nil
```

POP Heuristic

- Domain independent heuristic
  - # flaws
  - Length of plan (# of actions)
- Domain dependent heuristic
  - Preference for certain properties of the solution (don’t rob banks)

HTN PLANNING

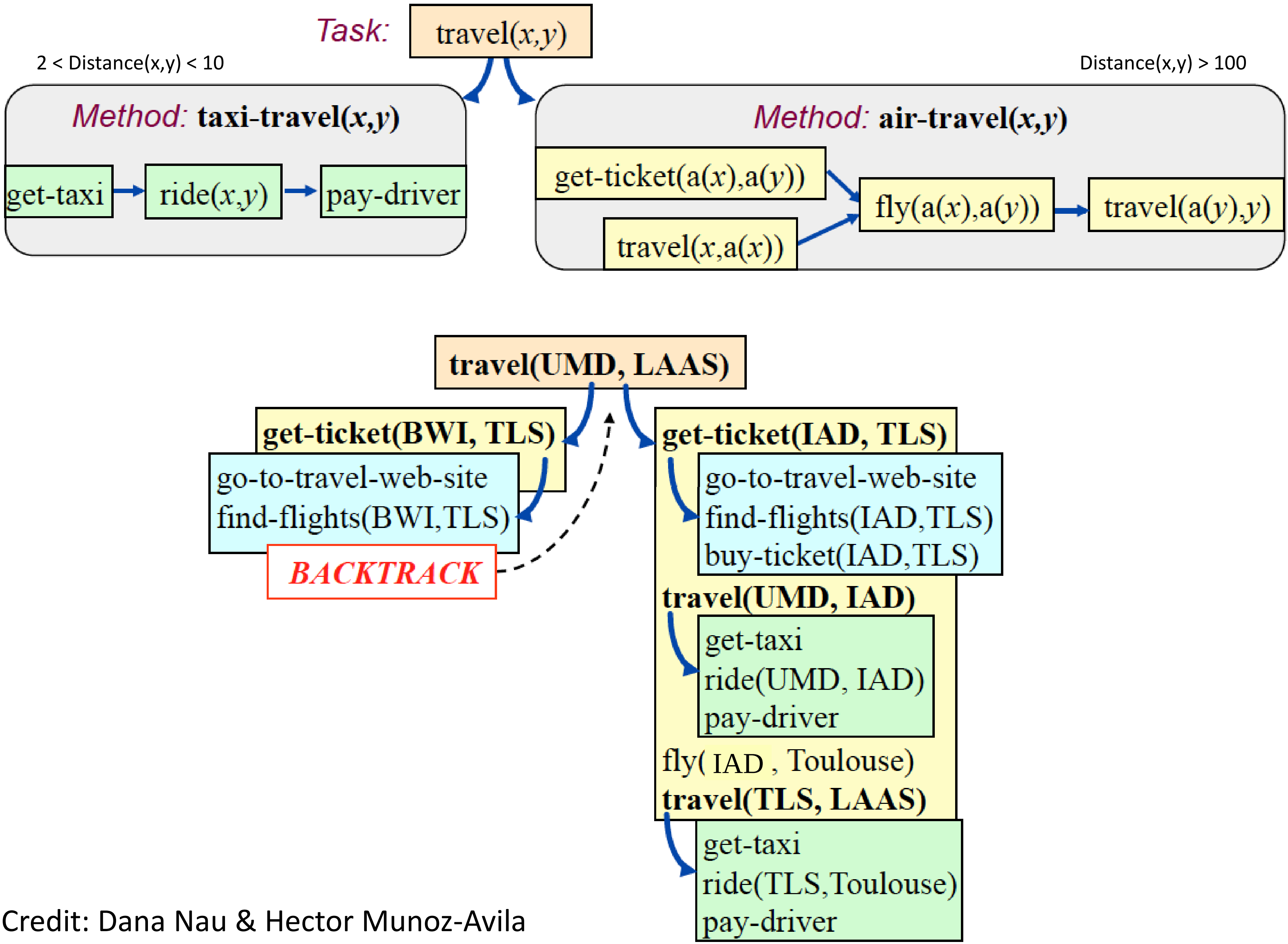
- F.E.A.R AI: https://www.youtube.com/watch?v=rf2T_j-FlDE
- Dana Nau HTN and games presentation http://www.cs.umd.edu/~nau/papers/nau2013game.pdf
- Killzone 2 AI: https://www.youtube.com/watch?v=7oWKCLdsGTE
- http://www.ign.com/boards/threads/killzone-2-enemy-a-i-is-it-up-there-with-fear-as-1.177634641/

Hierarchical Task Network Planning

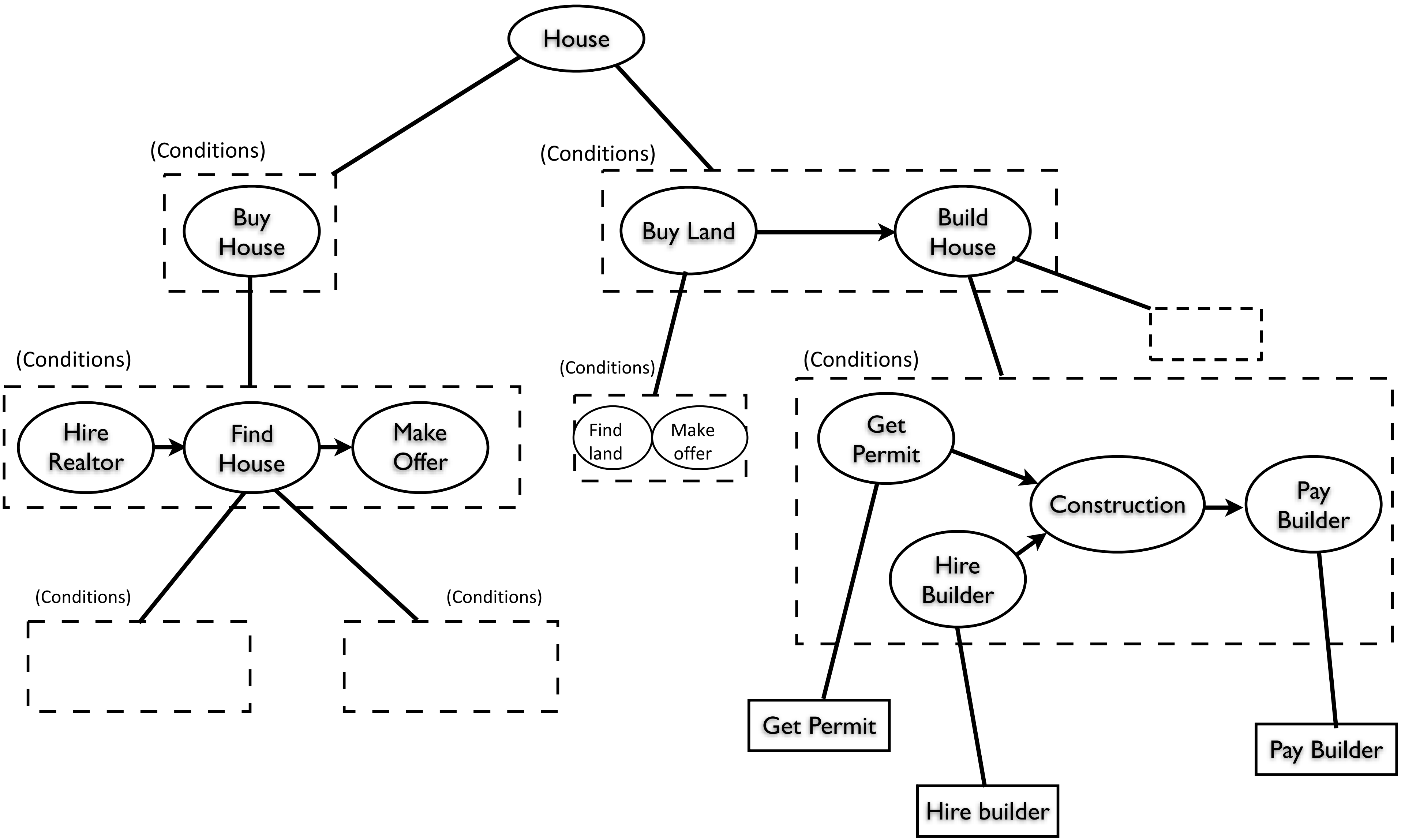
- Sometimes you know how to do things
- Example: going on a trip
  - Domain-independent planner: lots of combinations of vehicles and routes
  - Experienced human: a few recipes
    - Buy air plane ticket
    - Go from home to airport
    - Fly to other airport
    - Go from airport to destination
- Describe recipes as tasks that can be decomposed to sub-tasks (tasks == goals)

Hierarchical Task Networks

- Hierarchical decomposition of plans
- Initial plan describes high-level actions [e.g. BuildHouse]
- Refine plans using action decompositions
- Process continues until you reach primitive actions



Credit: Dana Nau & Hector Munoz-Avila



House
(Conditions)
Buy House
(Conditions)
Buy Land
Build House
(Conditions)
Hire Realtor
Find House
Make Offer
(Conditions)
Find land
Make offer
(Conditions)
Get Permit
Construction
Pay Builder
Hire Builder
(Conditions)
(Conditions)
Get Permit
Hire builder
Pay Builder

HTN Planner

- Given a task…
- Pick method with conditions that match the current world state (or pick randomly)
- Planning process
  - When you get to primitive, update state, repeat
  - Execute full plan (monitor world state)
- Can also create a partial plan
  - But early decisions can affect later conditions
- Replanning
  - If plan breaks, just pop up a level and re-decompose
  - Keep popping up decomposition fails
- SHOP2

SHOP2

```
(:method
        ; head
        (transport-person ?p ?c2)

        ; precondition
        (and
                (at ?p ?c1)
                (aircraft ?a)
                (at ?a ?c3)
                (different ?c1 ?c3))

        ; subtasks
        (:ordered
                (move-aircraft ?a ?c1)
                (board ?p ?a ?c1)
                (move-aircraft ?a ?c2)
                (debark ?p ?a ?c2)))
```

*primitive actions have preconditions and effects

```
Given state s, Tasks T, Domain D
Let P = empty plan
Let T0 = {t∈T | no task comes before t}
Loop
        If T0 is empty, return P
        Pick any t ∈ T0
        If t is primitive
                Modify s according to effects
                Add t to P
                Update T by removing t
                T0 = {t∈T | no task comes before t}
        Else
                Let M = a method for t with true preconditions in state s
                If M is empty return FAIL
                Modify T: remove t, add subtasks of M (note order constraints)
                If M has subtasks
                        T0 = {t∈subtasks | no task comes before t}
                Else
                        T0 = {t∈T | no task comes before t}
Repeat
```

HTN vs. A* Planning

- What are the advantages or disadvantages of HTN planning? A* planning? Partial-order planning?

Planning Under Uncertainty

- What if actions can fail?

Planning Under Uncertainty

- What do you do if you end up in a state you do not desire?

Planning Under Uncertainty

- What do you do if you end up in a state you do not desire?
  - Replan
  - Create a policy

Benefits of Planning

- Decouple goals and actions
  - Can create new character types (mimes vs. mutants)
  - State machines become unmanageable by design team
- Dynamic problem solving
  - Ability to re-plan when failure occurs

Planning and Games – Future

- Plan recognition
- Story generation
- Where else?

Reactive Planning

- Real-time decision making by performing one action every instant
- Instead of focusing on state, focus on action
- Examples
  - State-action table
  - Universal plan
  - Behavior trees
  - Rule systems

Resources

- Planning in modern games:
  - http://aigamedev.com/open/review/planning-in-games/
  - Nau HTN planning in Killzone: http://www.cs.umd.edu/~nau/papers/nau2013game.pdf
  - G.O.A.P: http://web.media.mit.edu/~jorkin/goap.html
  - Workshop at ICAPS 2013: http://icaps13.icaps-conference.org/technical-program/workshop-program/planning-in-games/
  - The AI of F.E.A.R.: http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf
- SHOP, JSHOP, SHOP2, JSHOP2, Pyhop (HTN planners)
  - http://www.cs.umd.edu/projects/shop/
- Scala impl. of partial-order planning
  - https://github.com/boyangli/Scalpo
- Other planners:
  - http://www.cs.cmu.edu/~jcl/compileplan/compiling_planner.html
- Facing your F.E.A.R. lecture: https://www.youtube.com/watch?v=rf2T_j-FlDE