Disclaimer: I use these notes as a guide rather than a comprehensive coverage of the topic. They are neither a substitute for attending the lectures nor for reading the assigned material.

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

- https://www.gdcvault.com/play/1024912/Beyond-Killzone-Creating-New-AI

Announcements

- Exam topics
  - World representation: Grid, Mesh, etc.
  - Navigation: Local (Steering) & Global (Search)
  - Decision making: Dtree, FSM, Btree, Rules, A* & HTN Planning
- Questions:
  - Compare/contrast approaches; pros & cons
  - Understand when and why it is appropriate to use different techniques
- **Question:** Which of the following are true about X and Y? (Circle all that apply)
  - X executes faster than Y
  - Y can do everything X can do
  - X & Y both rely on Z
  - There is nothing X does better than Y
- **Question:** How can the use of X reduce computation time in Y?

Can bring 2 normal sheets (8.5x11) of paper with info on both sides

Sidebar – Architectures

- The "**Game Loop**"

  while game is running

  process inputs

  update game world

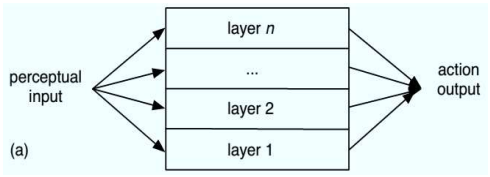
  generate outputs

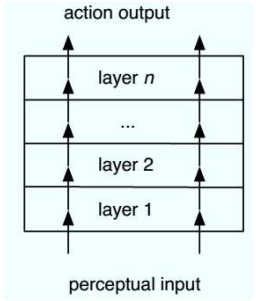
```
while( user doesn't exit )
  check for user input
  run AI
  move enemies
  resolve collisions
  draw graphics
  play sounds
end while
```

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

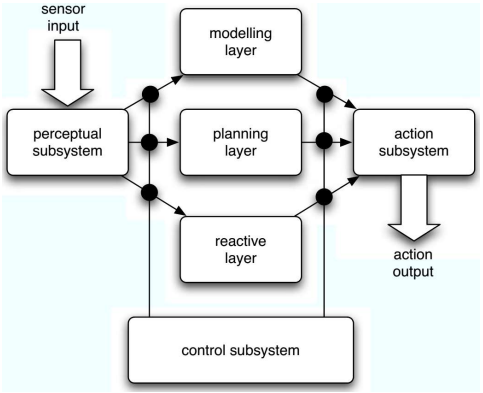
http://www.informit.com/articles/article.aspx?p=2167437&seqNum=3

Decision Making: Architectures…

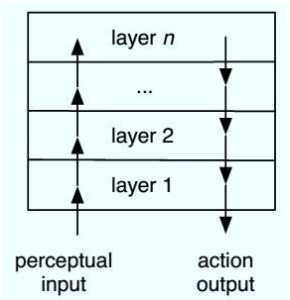




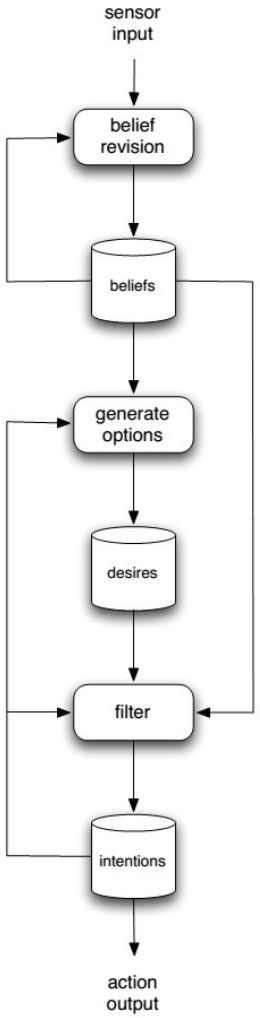
One-pass



Brooks Subsumption Arch.



Two-pass



BDI

See Multiagent Systems 2nd ed, Gerhard Weiss

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?

N-1: Production/rule systems

- Based on the current game state, activate a set of rules, pick from those based on some heuristic (e.g. best matches conditions)
- Pros:
  - Don't need to specify response to every contingency
  - Can respond to novel conditions
- Cons:
  - Hard to author robust rule systems
  - Emergence vs. over-engineering
  - Hard to debug

- Use in Games Industry, e.g. Environment-sensitive dialog generation (dynamic dialog gen)
  - Overwatch: https://www.youtube.com/watch?v=yqIKRL_5f6o&t=13s&spfreload=10
  - 2Bots1Wrench (GDC2012): https://www.youtube.com/watch?v=j4eIu6LxdZg
  - L4D2: https://www.youtube.com/watch?v=T5-2EnX5-K0
- See also:
  - Inform 7 rbs for interactive fiction
    - Eg rules chart: https://emshort.files.wordpress.com/2009/01/rules-chart4.pdf
  - Online Colossal Cave Adventure: https://www.amc.com/shows/halt-and-catch-fire/exclusives/colossal-cave-adventure

Rule Matching

Query

```
{ who: nick, concept: onHit, curMap:circus, health: 0.66, nearAllies: 2, hitBy: zombieclown }
```

PASS Rule 1: `{ who = nick, concept = onHit } → "ouch!"`

FAIL Rule 2: `{ who = nick, concept = onReload } → "changing clips!"`

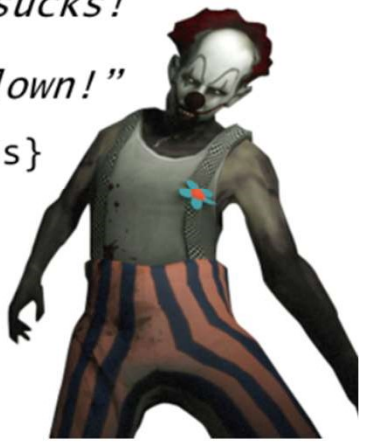
FAIL Rule 3: `{ who = nick, concept = onHit, health < 0.3 } → "aaargh I'm dying!"`

PASS Rule 4: `{ who = nick, concept = onHit, nearAllies > 1 } → "ow help!"`

PASS Rule 5: `{ who = nick, concept = onHit, curMap = circus } → "This circus sucks!"`

PASS Rule 6: `{ who = nick, concept = onHit, hitBy = zombieclown } → "Stupid clown!"`

PASS Rule 7: `{ who = nick, concept = onHit, hitBy = zombieclown, curMap = circus}`
`→ "I hate circus clowns!"`



Turn the Environment into Facts

Tag = “soda_can”

Tag = “barrel”

Tag = “bird”

Turn the Environment into Facts

Tag = “soda_can”

Tag = “barrel”

Tag = “bird”

Agent Knowledge Base

Current Decision Making Problems

- Shallowness & Realism
  - All of these techniques have the agent take **next** "best" move, but **don't look further** into the future than the next state. Agents ought to be **motivated by goals**
- Adaptability
  - Each technique is more general/adaptive than the last (Rule Systems >Behavior Trees > FSMs > Decision Tree). But can still **struggle to perform *well* in unanticipated** situations.
- Heavy Design Burden
  - All three of these techniques require a heavy authoring burden on designers (FSMs: states and transitions, B trees: nodes and structure, Rules: all conditions and each individual rule)

Decision Making: Planning

2019-10-16

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

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

What is Planning?

- Finding a **sequence of actions** to achieve a **goal**
  - Problems requiring **deliberate** thought ahead of time and a sequence of actions
- Basic planning comes down to **search**: "behavior planning using A*"
  - Given: state of the world, a goal, and models of actions
  - Find: sequence of actions (a plan) that achieves the goal
- Need to find appropriate heuristic

Some Examples

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

- **Route search**: Find a route from GT to New Orleans
- **Project management**: Construct a project plan for organizing an event (e.g., Music Midtown)
- **Military operations**: Develop an air campaign
- **Information gathering**: Find and reserve an airline ticket to travel from Hartsfield to Miami
- **Game playing**: plan the behavior of a computer controlled player
- **Resource control**: Plan the stops of several of elevators in a skyscraper building

Answer: ALL!

Classes of General-Purpose Planners

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

Action/Behavior Planning with A*

- state
- plan
- Hierarchical Tasks

We are going to discuss these forms

- Disjunctive plans
- SAT

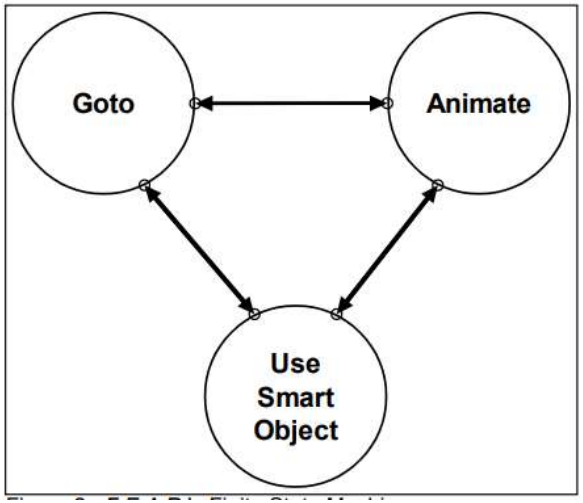
"A planning system tells the A.I. what its goals and actions are, and lets the A.I. decide how to sequence actions to satisfy goals. […] Planning is a formalized process of searching for sequence of actions to satisfy a goal" – Orkin

ACTION/BEHAVIOR PLANNING WITH A*

Action/Behavior Planning vs. Path planning

- Same algorithm, different action representation
  - Pathplanning:
    - Cells / Waypoints / Path nodes
    - Node links
  - Action/Behavior planning with A*:
    - Goals
    - Actions, with
      - Action name
      - Precondition
      - Effect (add/remove)

- As with all things in AI, how we represent the problem and the attributes of the problem is going to be key.
- In planning systems, agents have goals and a set of actions. Agent decides how to apply those actions to achieve goals.

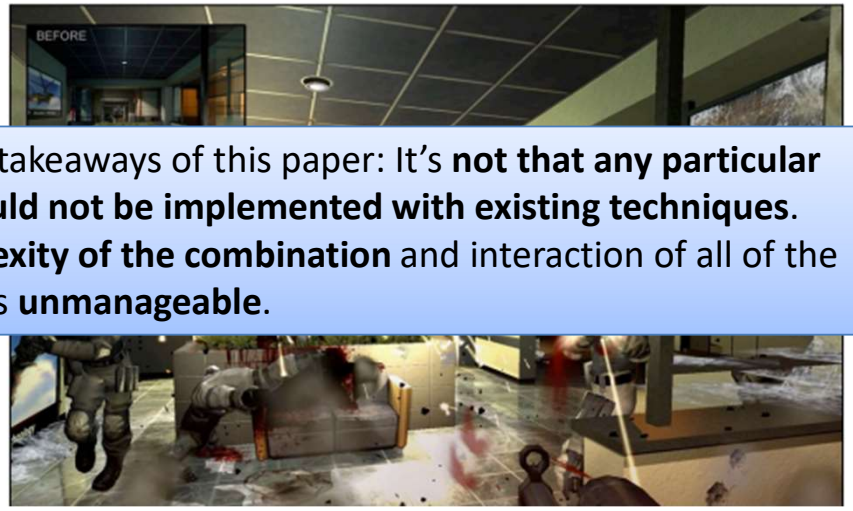


Three States and a Plan: The A.I. of F.E.A.R.

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.



This is one of the main takeaways of this paper: It's **not that any particular behavior in F.E.A.R. could not be implemented with existing techniques**. Instead, it is the **complexity of the combination** and interaction of all of the behaviors that becomes **unmanageable**.

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

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

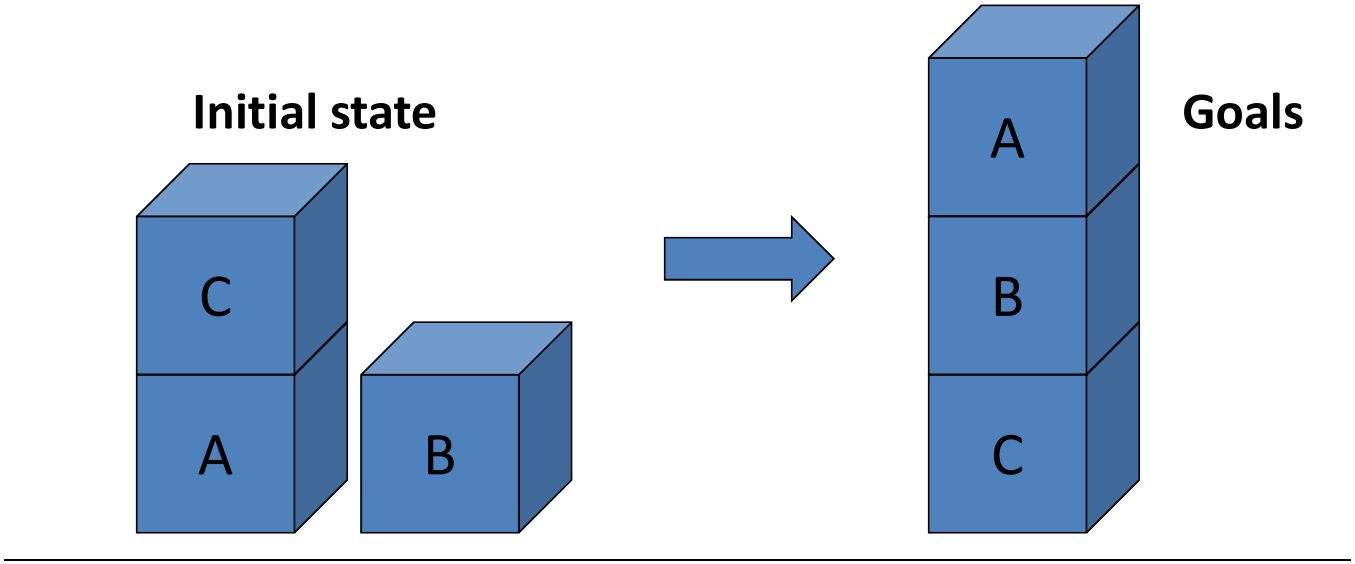
Domain Specific: The plan generation algorithm depends on the particular domain

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

STRIPS (Fikes & Nilsson)

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

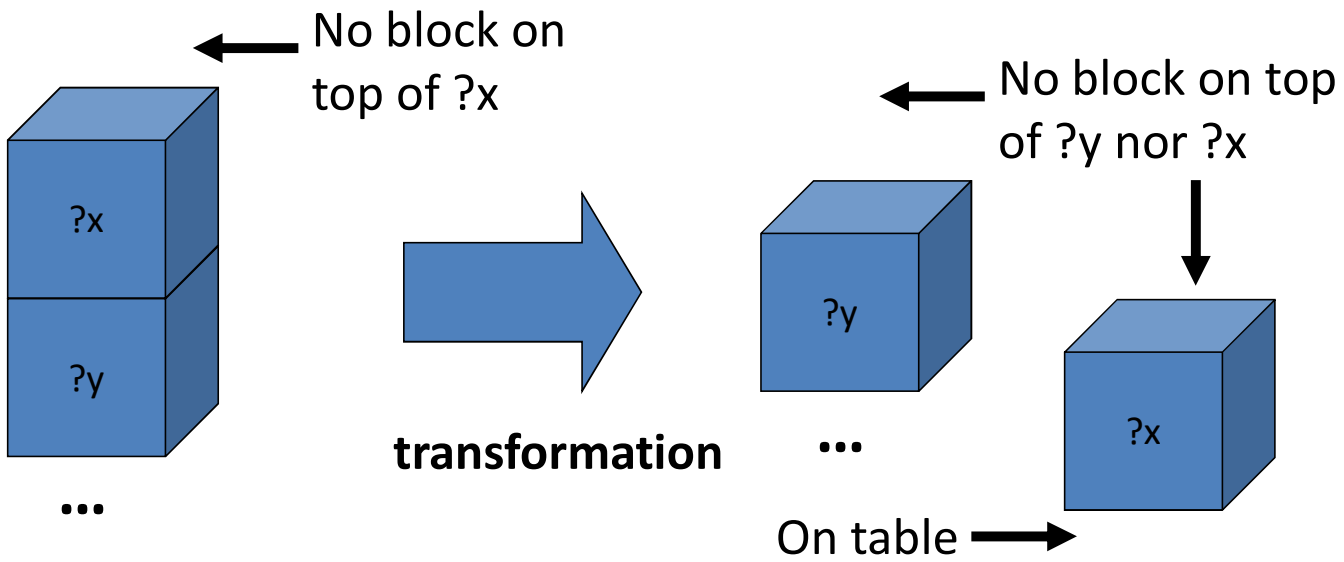
General-Purpose Planning: State & Goals



- **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)

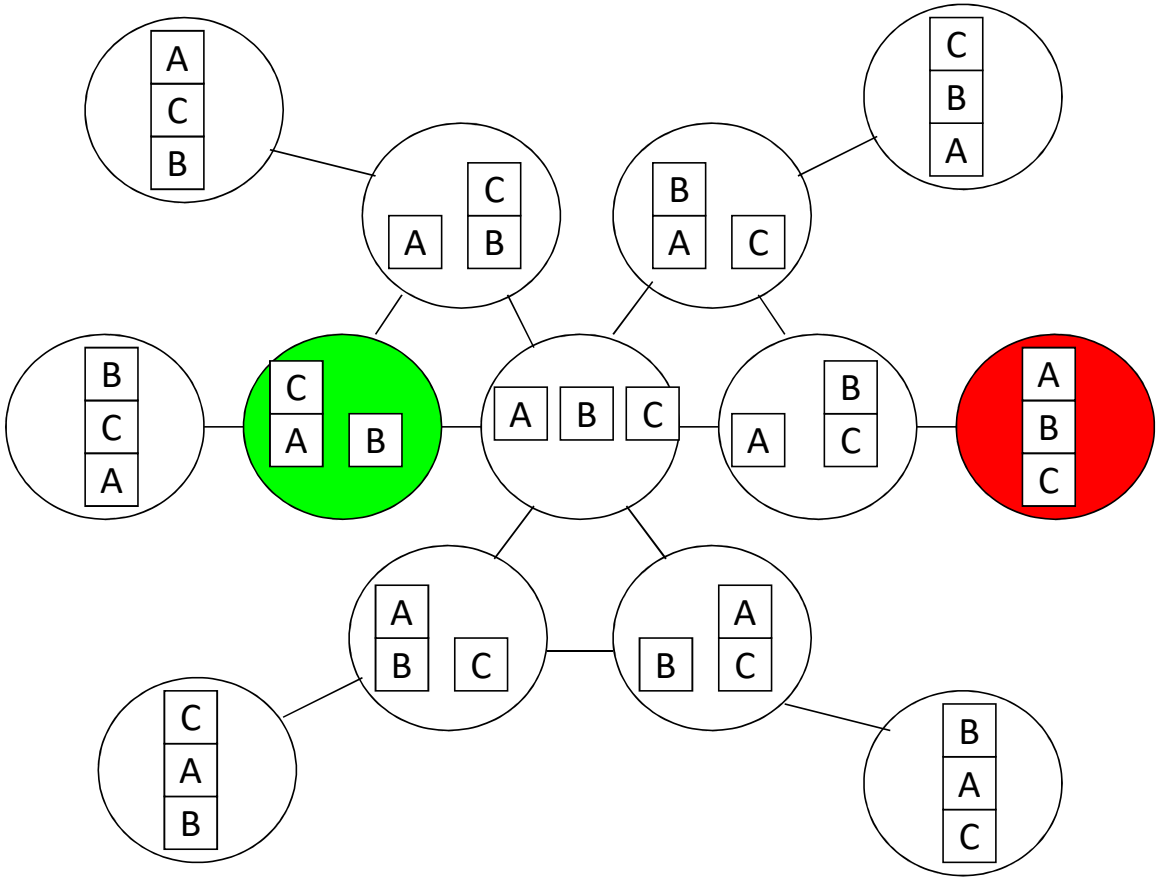
General-Purpose Planning: Operators



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)

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)

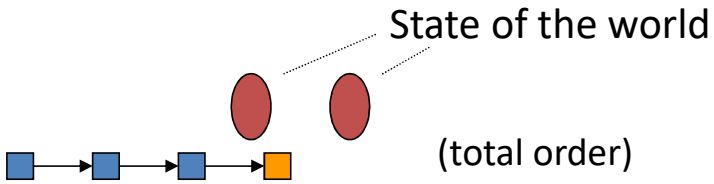
Precondition: at(?p, ?from), plane(?p), airport(?from),
airport(?to), path(?from, ?to), ?from ≠ ?to

Effect: at(?p, ?to), ¬at(?p, ?from)

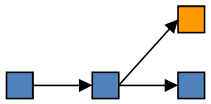
*Also a call to the game engine to play animation or run a function

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



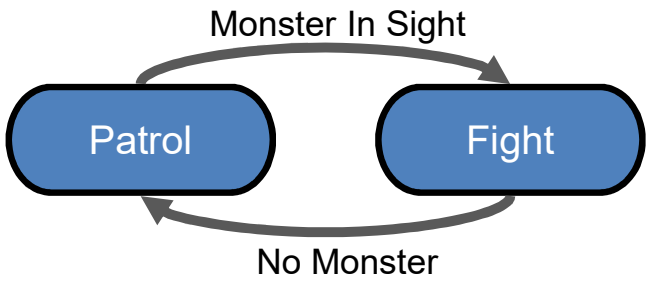
- **Plan-space** planners transform the plans. These planners search for a a plan satisfying certain conditions



(partial-order, least-commitment)

FSM vs A* Planning

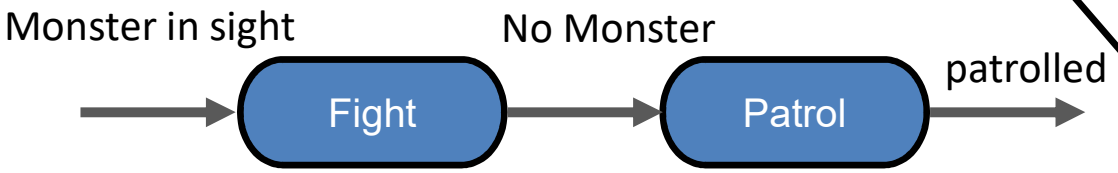
FSM:



Planning Operators

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

A resulting plan:



Neither is more powerful than the other

But Planning Gives More Flexibility

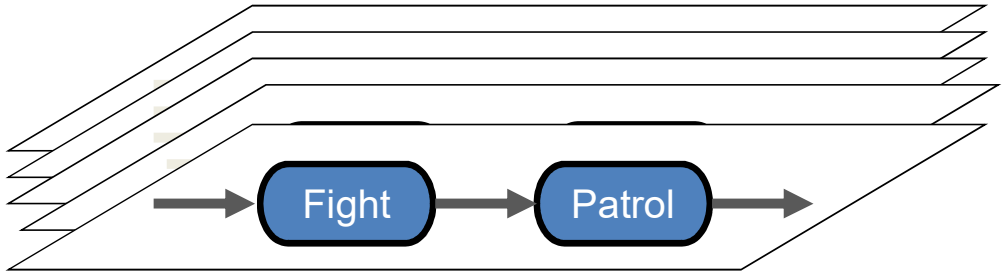
- "Separates implementation from data" --- Orkin

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.
- With planning, “easy” to add goals and actions

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

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

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

F.E.A.R. Differences from STRIPS

- Action costs
  - Actions have non-uniform costs in FEAR; FEAR uses A* to navigate the state-space.
- World state representation
  - The world state representation only allows for the planner to consider one enemy and one weapon at a time.
  - Therefore, systems outside of the planner choose which weapon and which enemy to deal with, and the planner only considers those.

Other Games with Planning

- Empire: Total War
- Fallout 3
- Killzone series

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

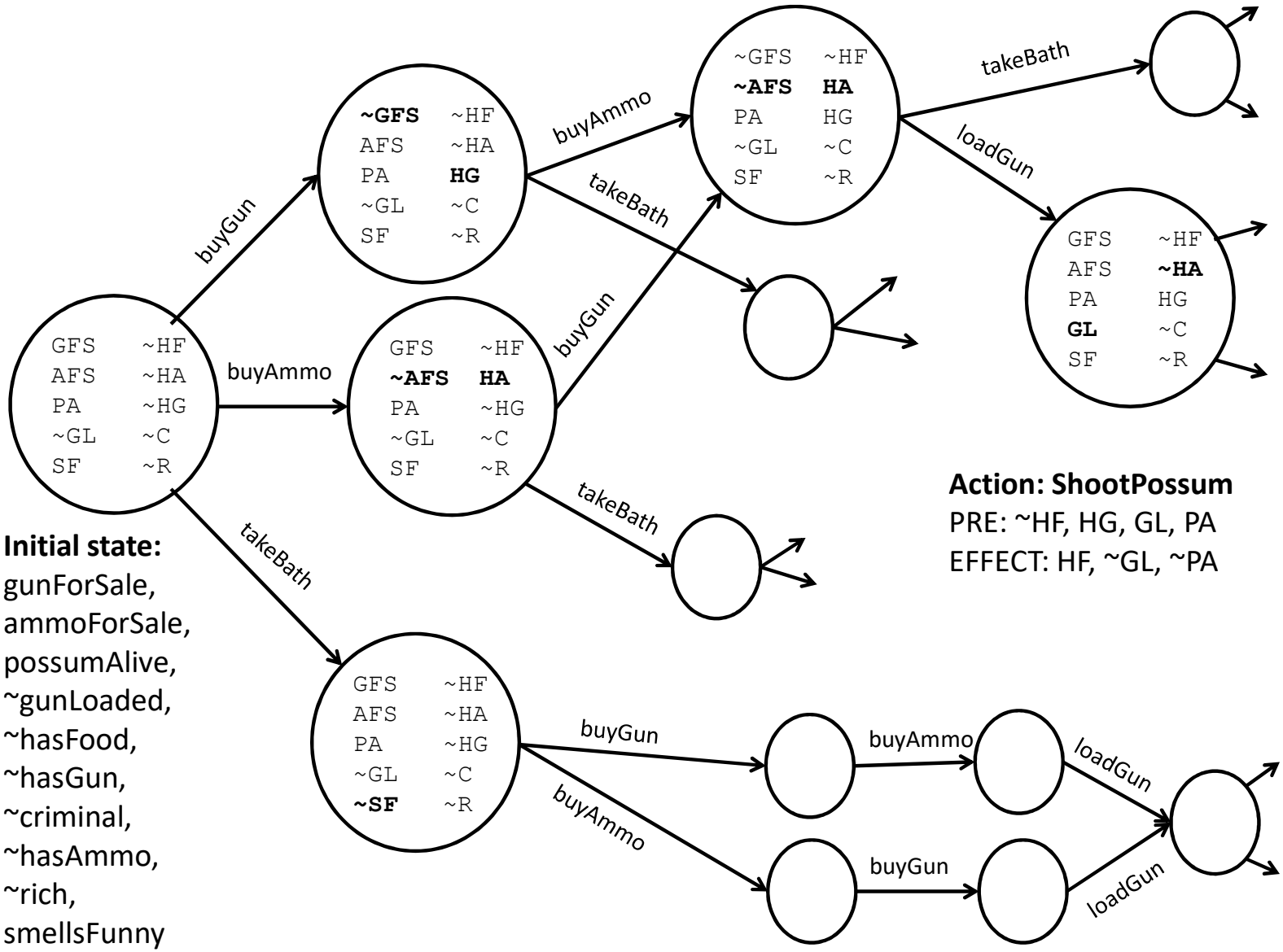
Action: LoadGun
PRE: hasGun, hasAmmo, ~gunLoaded
EFFECT: gunLoaded, ~has Ammo

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



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

Action: ShootPossum
PRE: ~HF, HG, GL, PA
EFFECT: HF, ~GL, ~PA

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 and AI

"Heuristic Search Hypothesis. The solutions to problems are represented as symbol structures. A physical symbol system exercises its intelligence in problem solving by search-that is, by generating and progressively modifying symbol structures until it produces a solution structure." – Alan Newell

"A physical symbol system has the necessary and sufficient means for general intelligent action."

— Allen Newell and Herbert A. Simon

https://amturing.acm.org/award_winners/newell_3167755.cfm

https://www.nobelprize.org/prizes/economic-sciences/1978/simon/biographical/

Heuristics

- Informs decision into which node (state) to expand
  - A function that estimates how close a state is to a goal
  - Encapsulates domain knowledge for a particular search problem
- Admissible heuristics allow for A*
  - Relax the planning problem (no deletes) and use subgoal independence assumption
  - Inadmissible heuristics break optimality because good plans get left on the frontier
  - A* expands mainly toward the goal, but maintains alternatives in frontiers just in case
  - As heuristics get closer to the true cost, fewer nodes are expanded but usually more work is done per node to compute the heuristic itself

- Heuristic cost should never overestimate the actual cost of a node. This is an “admissible” heuristic
  - i.e. it must be “optimistic” so that we never overlook a node that is actually good
  - A* is optimally efficient: given the information in h, no other optimal search method can expand fewer nodes
  - Memory is a problem for the A* algorithms. Consider IDA*, SMA*
- A problem with less restrictions on its operators is called a relaxed problem
  - The optimal solution of the original problem is also a solution to the relaxed problem and must therefore be at least as expensive as the optimal solution to the relaxed problem

On memory

- In spite of its optimality and completeness, A* still has problems:
  - For most problems, the number of nodes on the frontier of the search space is still exponential in the length of the solution.
  - That is, the search tree can still grow to be as "bushy" as in Breadth-first Search. So there can be problems with respect to the amount of memory needed to run A* search.
- General drawbacks of heuristic (informed) search include the following:
  - need to keep everything on the frontier in memory
  - need to be able to do cost comparisons quickly — if the heuristic function is extremely complex, the search will not be fast.
  - need to choose good heuristics — this is not easy for many problems!
- When at a loss for a good heuristic function, consider a relaxed version of the problem.
  - An exact solution to a relaxed problem might be a good heuristic for the real problem.
  - For example, consider a sliding tile puzzle. One relaxed version of the puzzle is one in which the tiles can simply be picked up and put into place. Therefore, one possible heuristic is to count the number of tiles out of position, since simply placing them would solve the relaxed problem.

http://www.cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html

Heuristic Search Planning

- Computes heuristic values for each precondition 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)

Benefits of A* 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

- Potentially more realistic combat behavior (goal oriented actions)
- Make it harder to learn how to defeat enemy tactics
- Emergent solutions
- Unanticipated situations can be realistically handled
- Less authoring

Potential Drawbacks of A* Planning

- Slower responses / longer computation
- Less reliable patterns of behavior
- Removal of authorial intent
- Less understandable/explainable than BTree

PARTIAL ORDER PLANNING

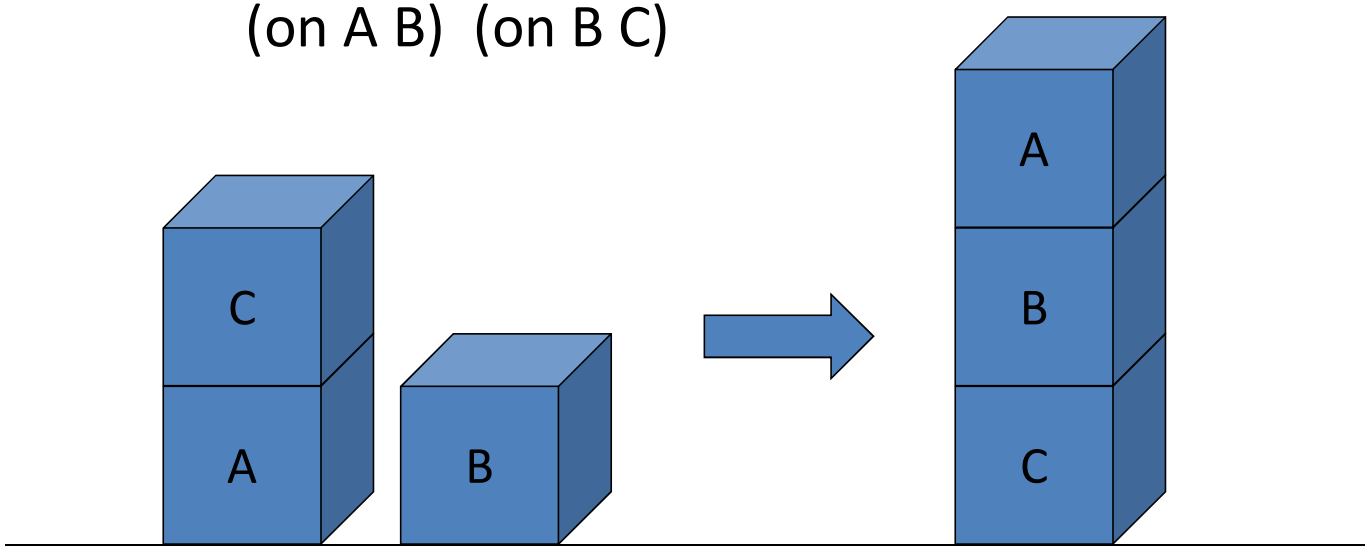
Partial-Order Planning

- 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?

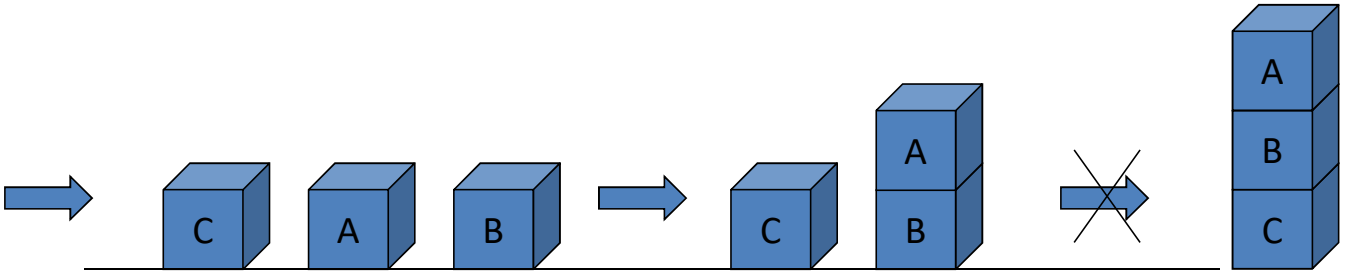
- 1. Motivation: “Sussman Anomaly”
  - Two subgoals to achieve:

    (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

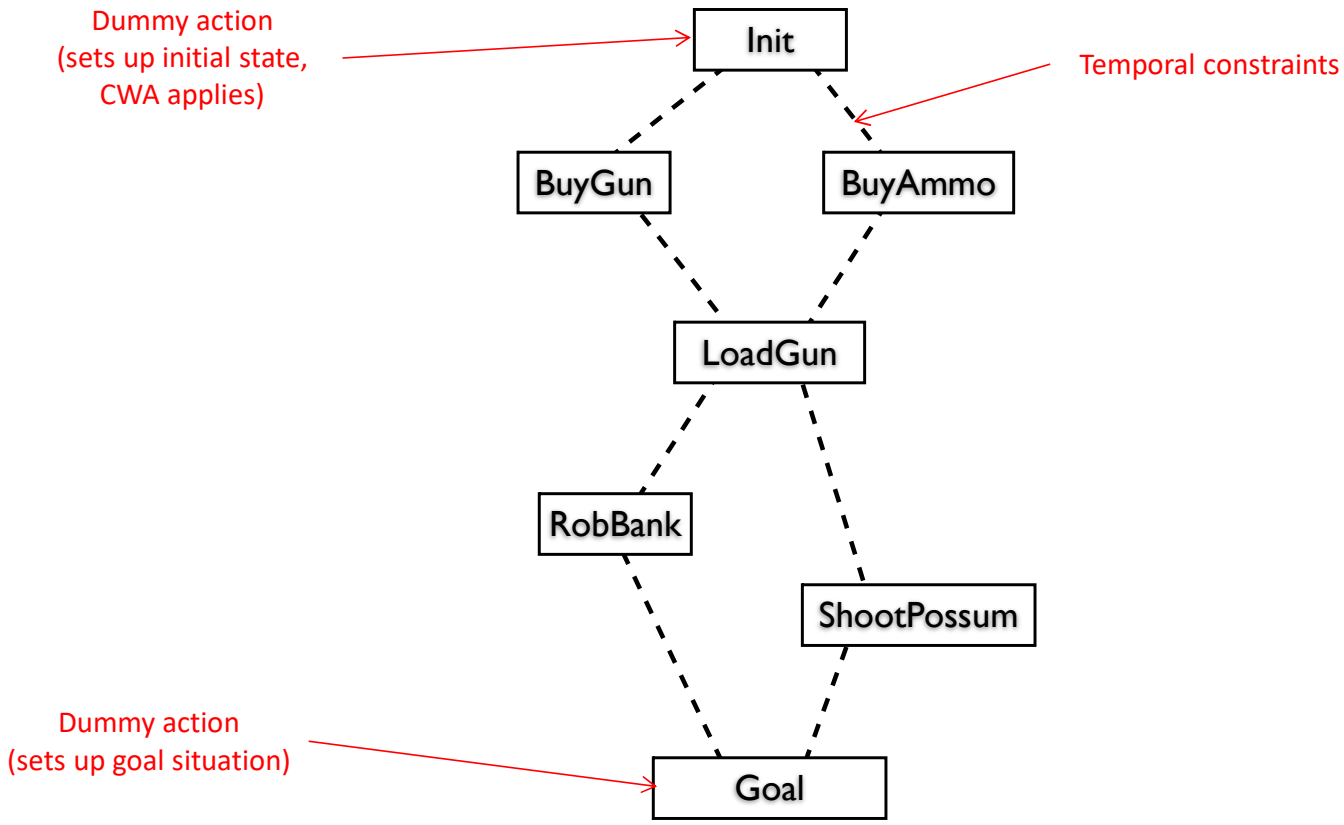
- Conflict
  - An action C conflicts with A -^{p}--> B if C has ¬p as an effect AND C can occur between A and B
- Consistent Plans
  - No cycles in ordering constraints
  - No conflicts with causal links
- Solution
  - Consistent plan with no open preconditions

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

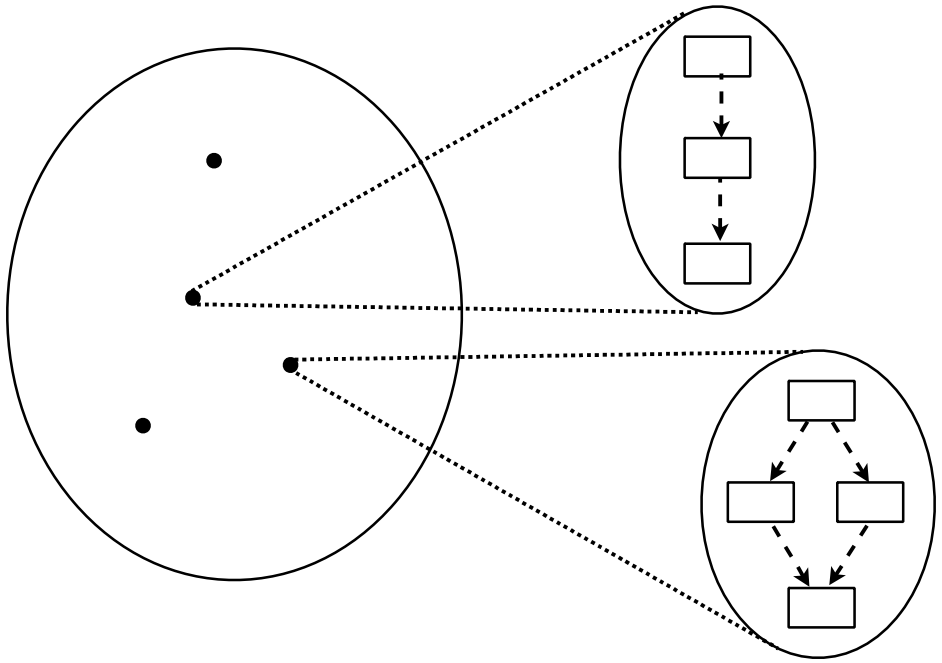
- Less understanding of how to create accurate heuristics for POP than total-order planning.
- Obvious heuristic: number of open preconditions
- Most-constrained-variable



- 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

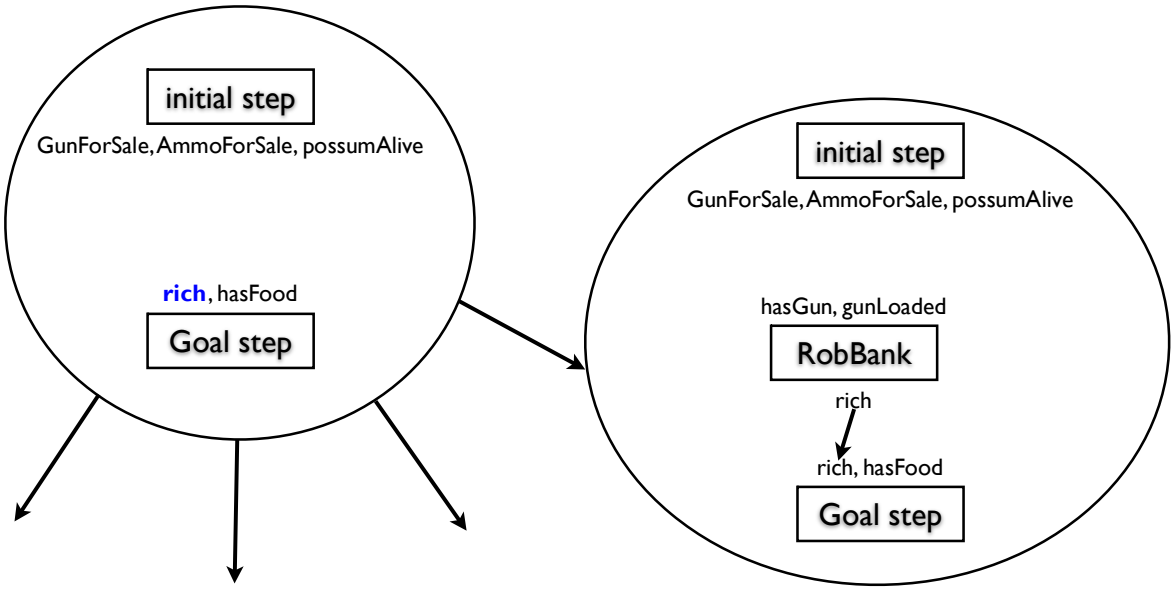
Two flaws: Nothing makes either goal condition true

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

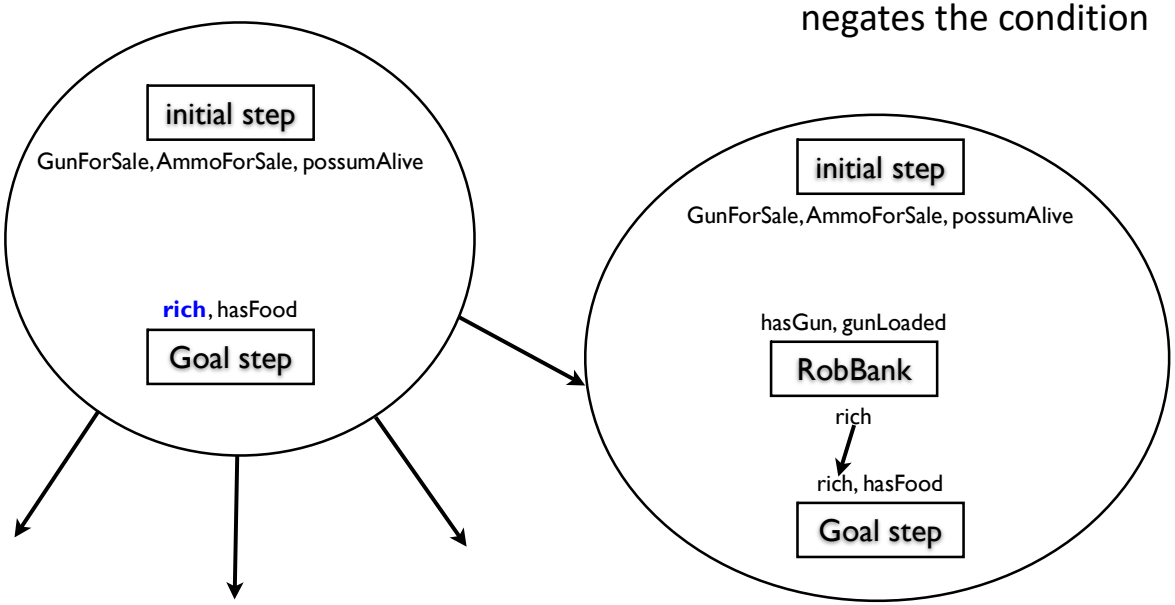
* Does it matter what order I pick flaws?

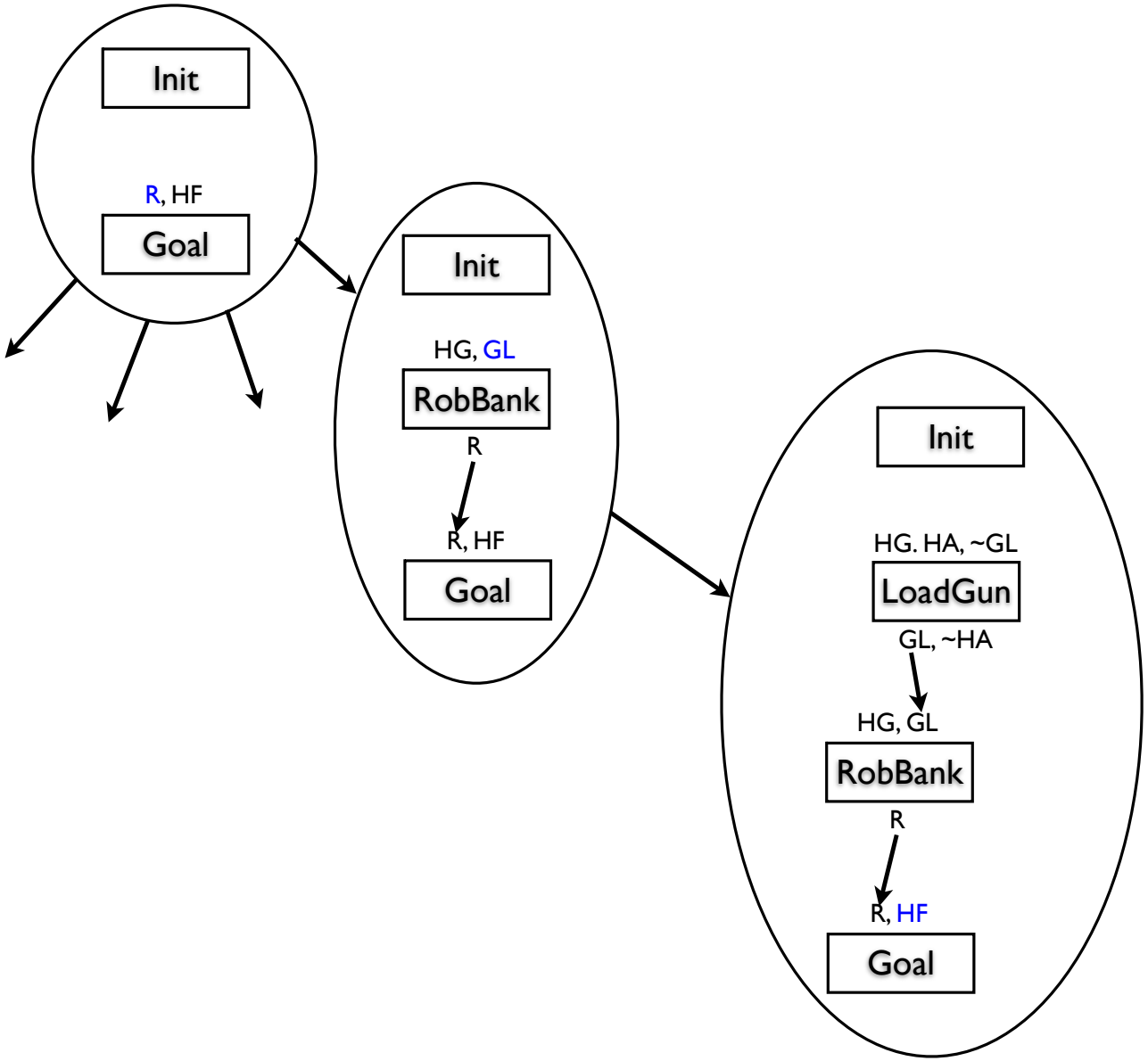
Open condition flaw:

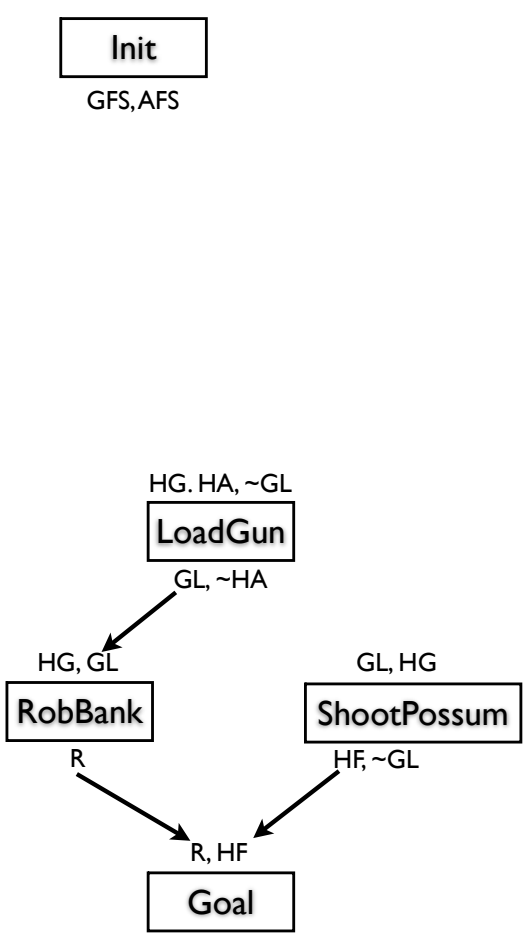
- A precondition that is not satisfied
- Pick an operator that has an effect unifying with the condition
  - Strategy #1: Add a new action
  - Strategy #2: Reuse an action

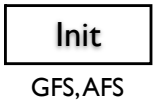
Causal link

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



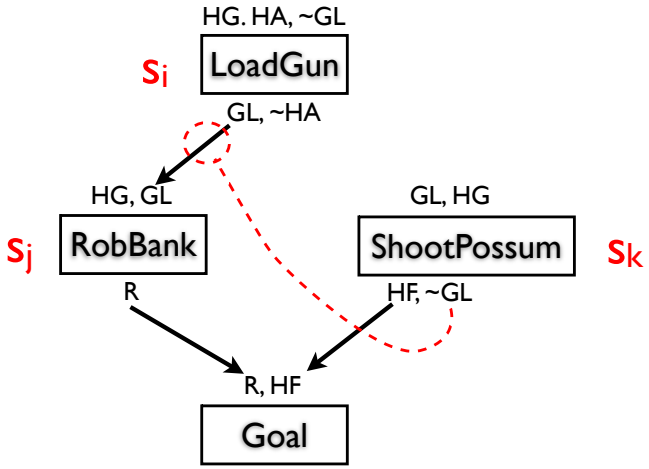

Init
R, HF
Goal
Init
HG, GL
RobBank
R
R, HF
Goal
Init
HG. HA, ~GL
LoadGun
GL, ~HA
HG, GL
RobBank
R
R, HF
Goal


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

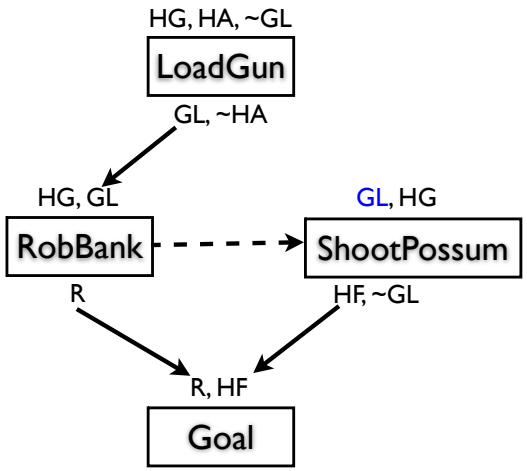


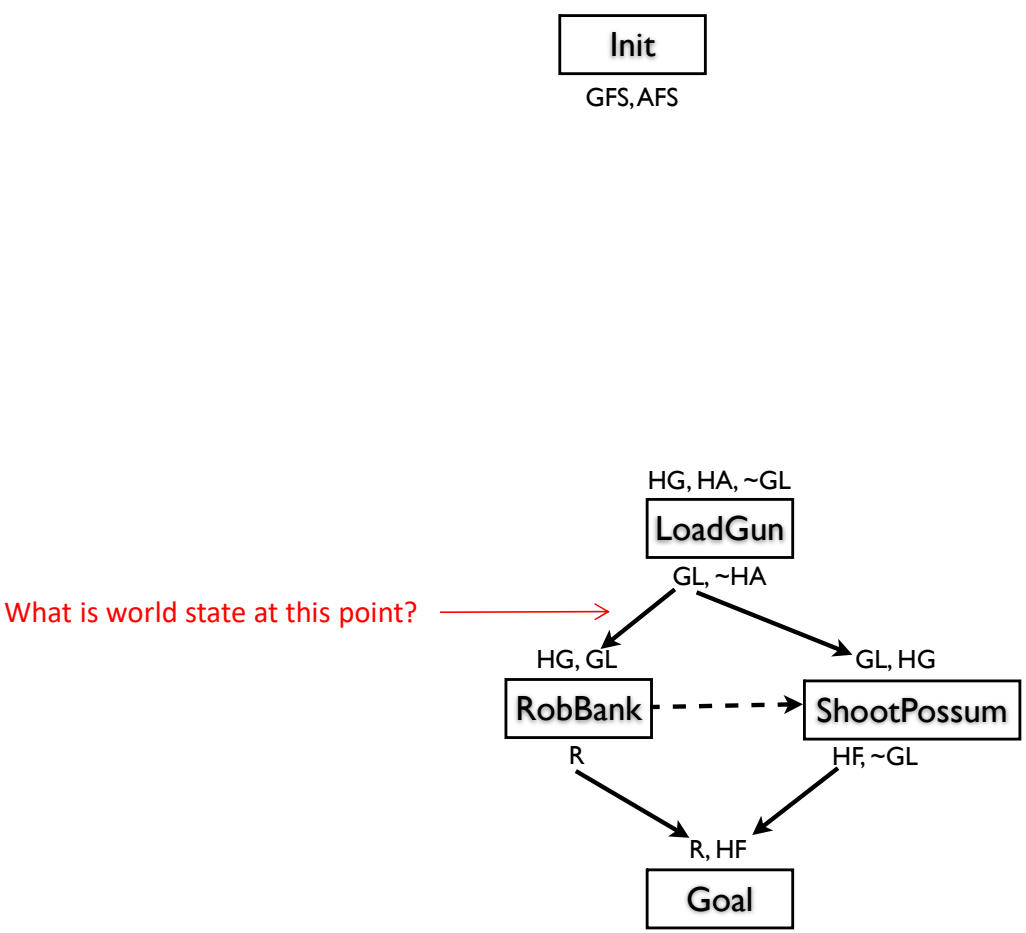
Causal threat

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

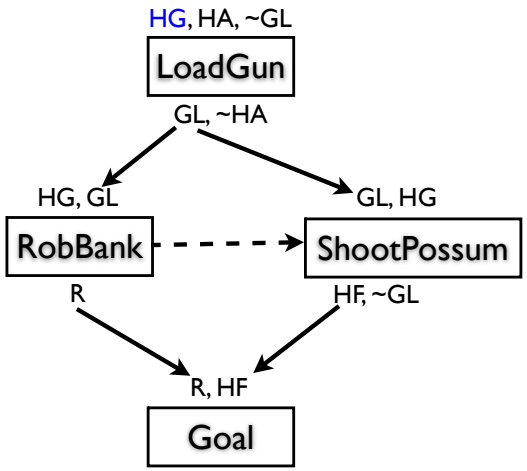


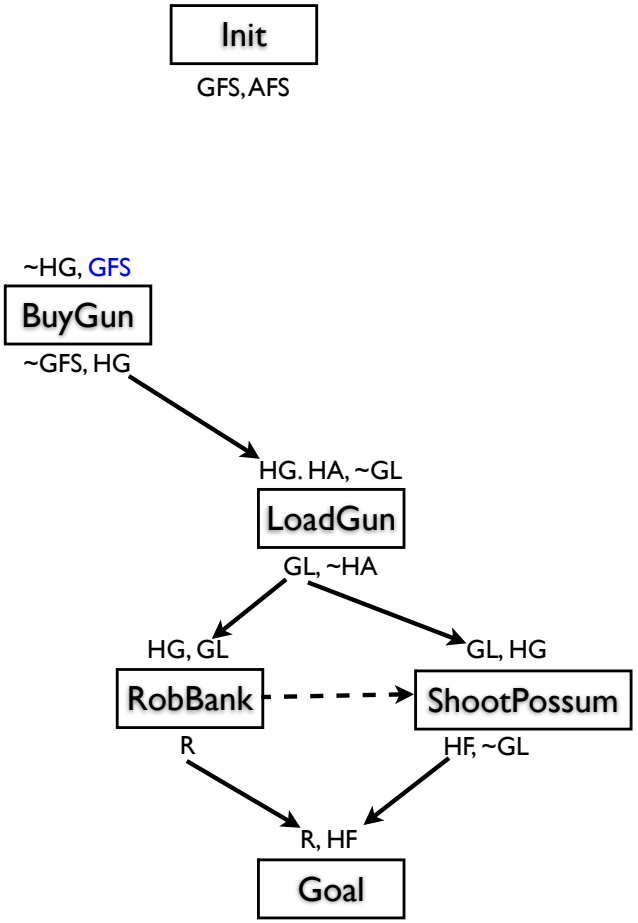

Init
GFS, AFS

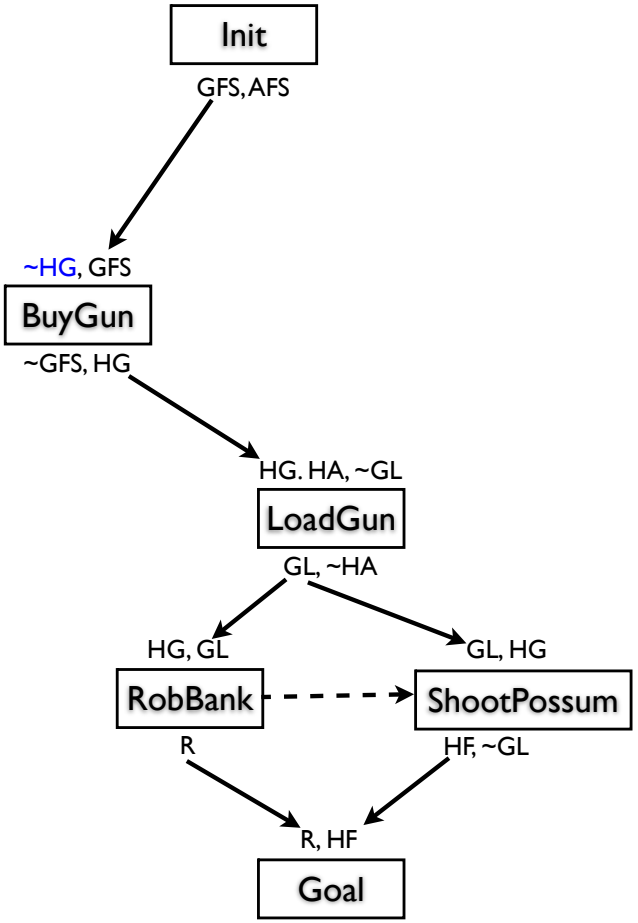

HG, HA, ~GL
LoadGun
GL, ~HA
HG, GL
GL, HG
RobBank
ShootPossum
R
HF, ~GL
R, HF
Goal

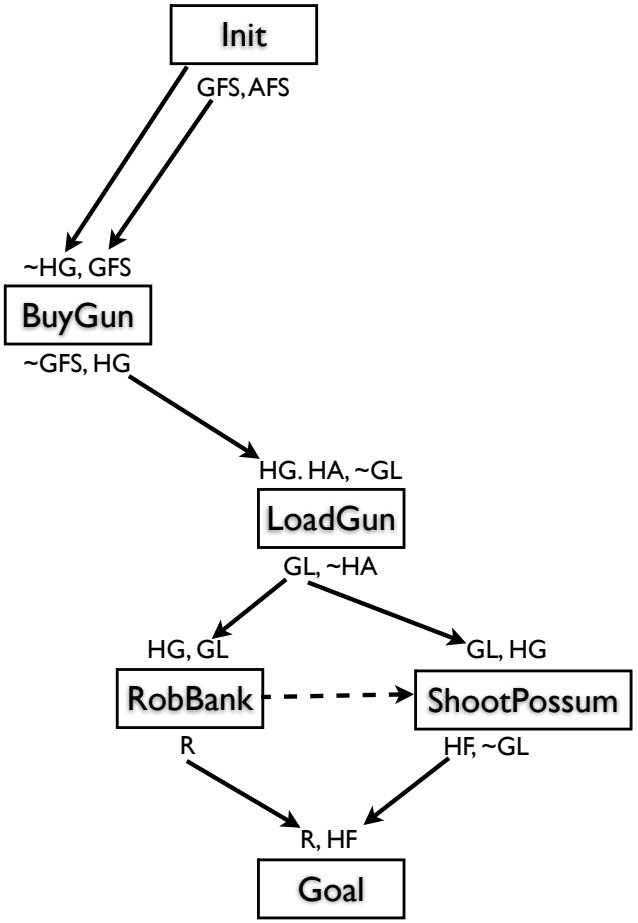

Init
GFS, AFS
HG, HA, ~GL
LoadGun
GL, ~HA
What is world state at this point?
HG, GL
GL, HG
RobBank
ShootPossum
R
HF, ~GL
R, HF
Goal


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
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
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
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)
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)

Must read: http://www.cs.umd.edu/~nau/papers/nau2013game.pdf

http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter29_Hierarchical_AI_for_Multiplayer_Bots_in_Killzone_3.pdf

HTN PLANNING

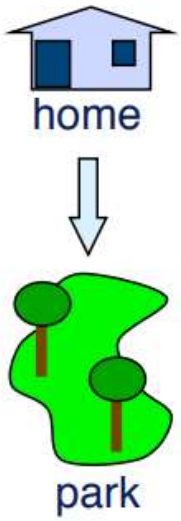
Hierarchical Task Network (HTN) Planning

- 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 Network (HTN) Planning

- Hierarchical decomposition of plans
- Initial plan describes high-level actions
  - E.g. “Build House”, “Find Player”, etc
- Refine plans using action decompositions
- Process continues until the agent reaches primitive actions

States and Tasks



- **State**: description of the current situation
  - » I'm at home, I have €20, there's a park 8 km away
- **Task**: description of an activity to perform
  - » Travel to the park
- Two kinds of tasks
  - » ***Primitive*** task: a task that corresponds to a basic action
  - » ***Compound*** task: a task that is composed of other simpler tasks

http://www.cs.umd.edu/~nau/papers/nau2013game.pdf

Operators

- **Operators**: parameterized descriptions of what the basic actions do
  - » ***walk*** from location x to location y
    - Precond: agent is at x
    - Effects: agent is at y
  - » ***call taxi*** to location x
    - Precond: (none)
    - Effects: taxi is at x
  - » ***ride taxi*** from location x to location y
    - Precond: agent and taxi are at x
    - Effects: agent and taxi at y, agent owes 1.50 + ½ distance(x,y)
  - » ***pay driver***
    - Precond: agent owes amount of money r, agent has money $m \geq r$
    - Effects: agent owes nothing, agent has money $m - r$
- **Actions**: operators with arguments

http://www.cs.umd.edu/~nau/papers/nau2013game.pdf

Methods

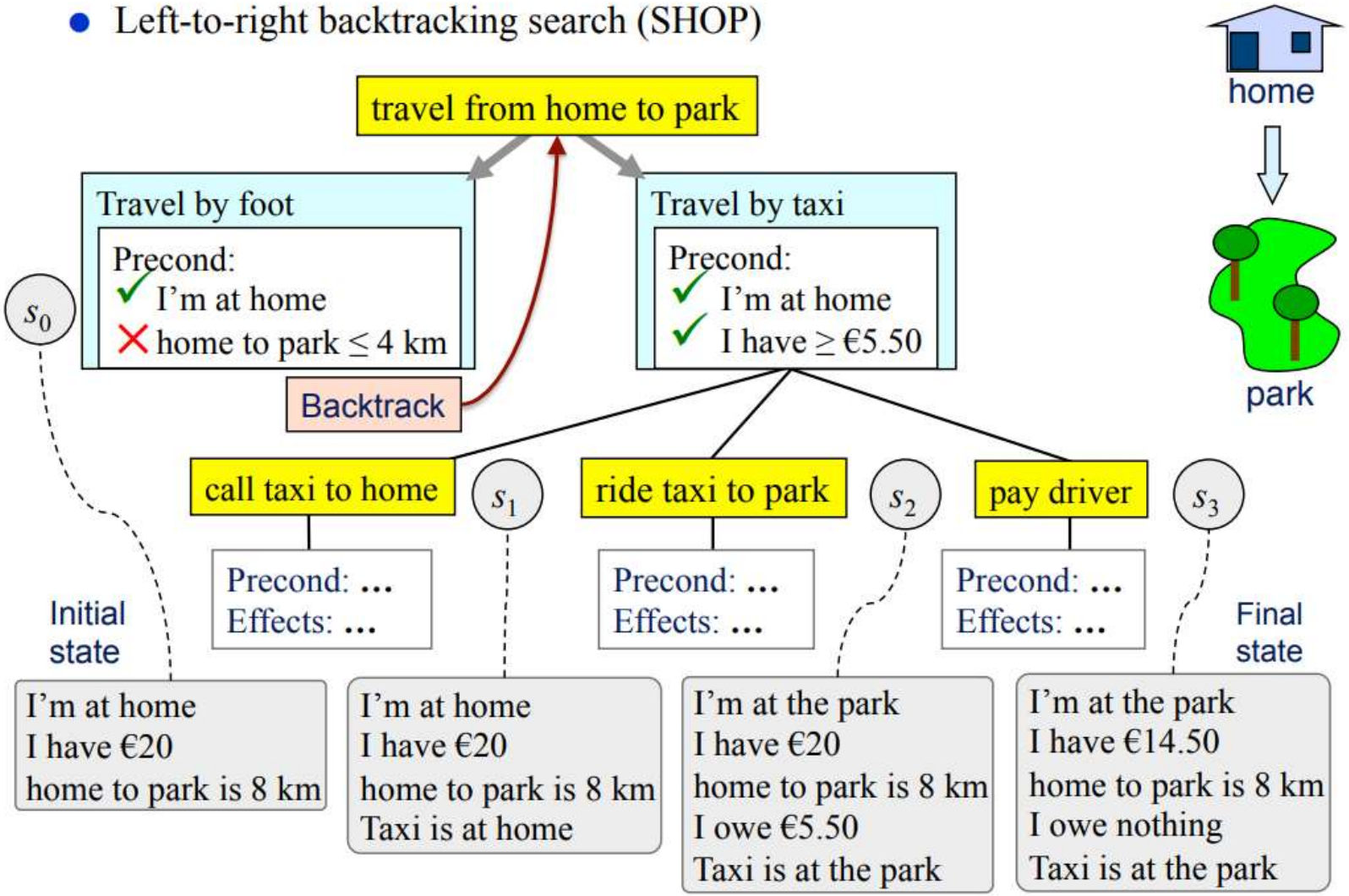
- Method: parameterized description of a possible way to perform a compound task by performing a collection of subtasks
- There may be more than one method for the same task

  - ***travel by foot*** from x to y
    - Task: travel from x to y
    - Precond: agent is at x, distance to y is ≤ 4 km
    - Subtasks: walk from x to y
  - ***travel by taxi*** from x to y
    - Task: travel from x to y
    - Precond: agent is at x, agent has money $\geq 1.5 + \frac{1}{2}$ distance(x,y)
    - Subtasks: call taxi to x,
      ride taxi from x to y,
      pay driver

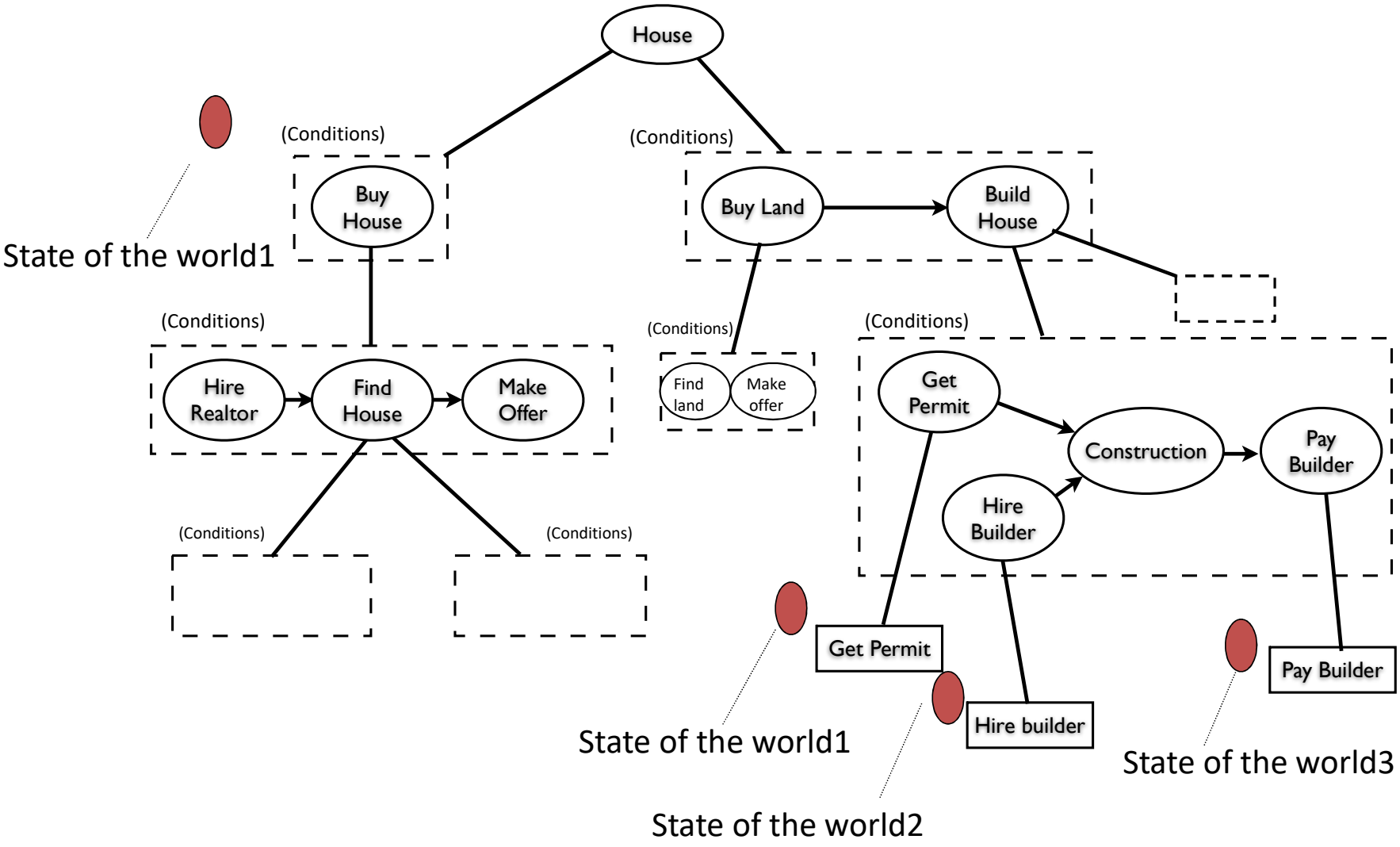
http://www.cs.umd.edu/~nau/papers/nau2013game.pdf

Simple Travel-Planning Problem

- Left-to-right backtracking search (SHOP)



http://www.cs.umd.edu/~nau/papers/nau2013game.pdf


House
(Conditions)
(Conditions)
Buy House
Buy Land
Build House
State of the world1
(Conditions)
(Conditions)
(Conditions)
Hire Realtor
Find House
Make Offer
Find land
Make offer
Get Permit
Construction
Pay Builder
Hire Builder
(Conditions)
(Conditions)
Get Permit
Pay Builder
Hire builder
State of the world1
State of the world3
State of the world2

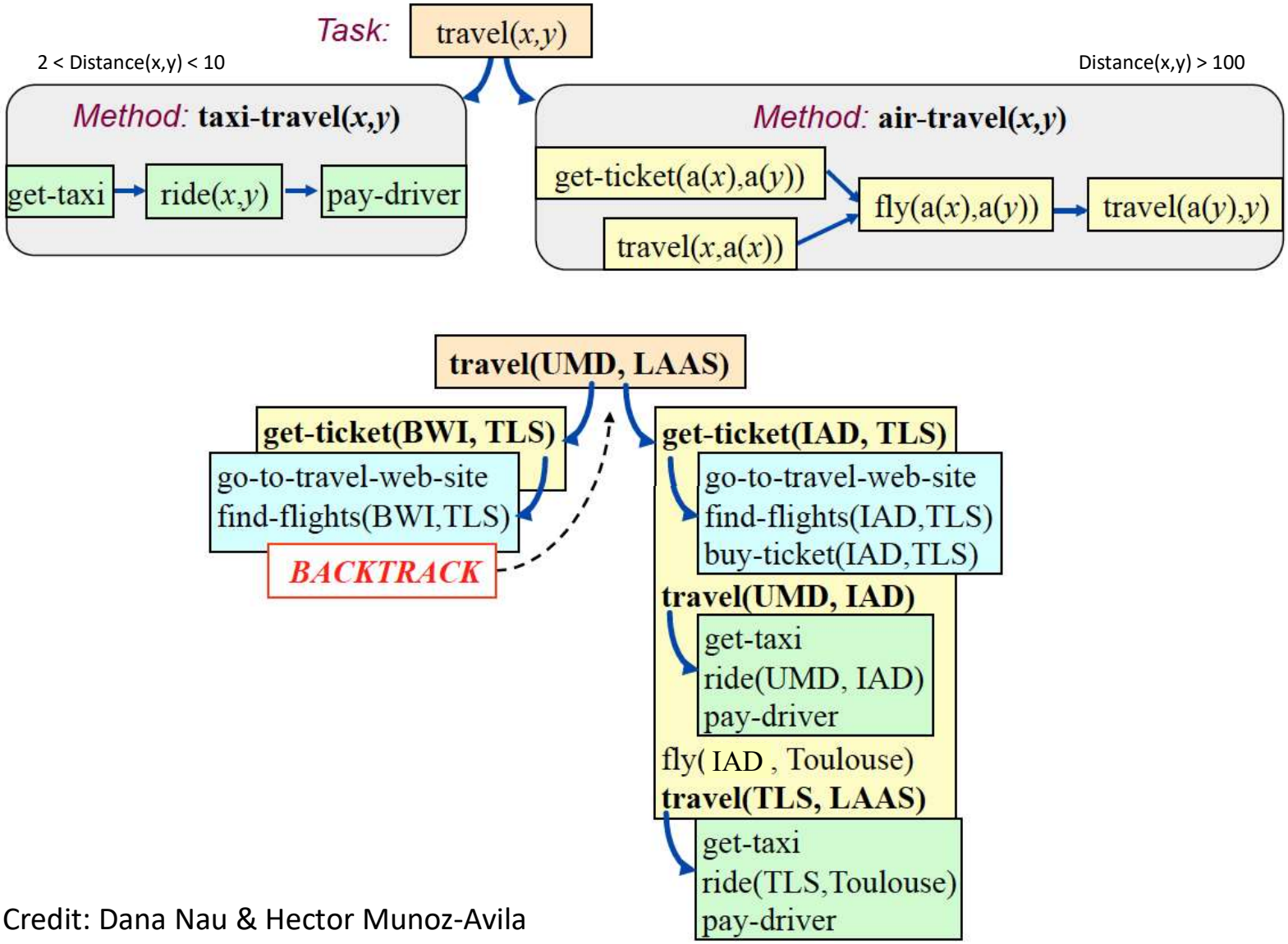
HTN Planning: Idea

- Each “task” segments the planning problem into smaller and smaller problems
- Only plan out the high-level plan at first
  - Replan if conditions break it
- Only plan out with primitive actions the current high-level task

- *Note*: can use heuristic A* planning algorithms discussed for primitive action planning

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



Credit: Dana Nau & Hector Munoz-Avila

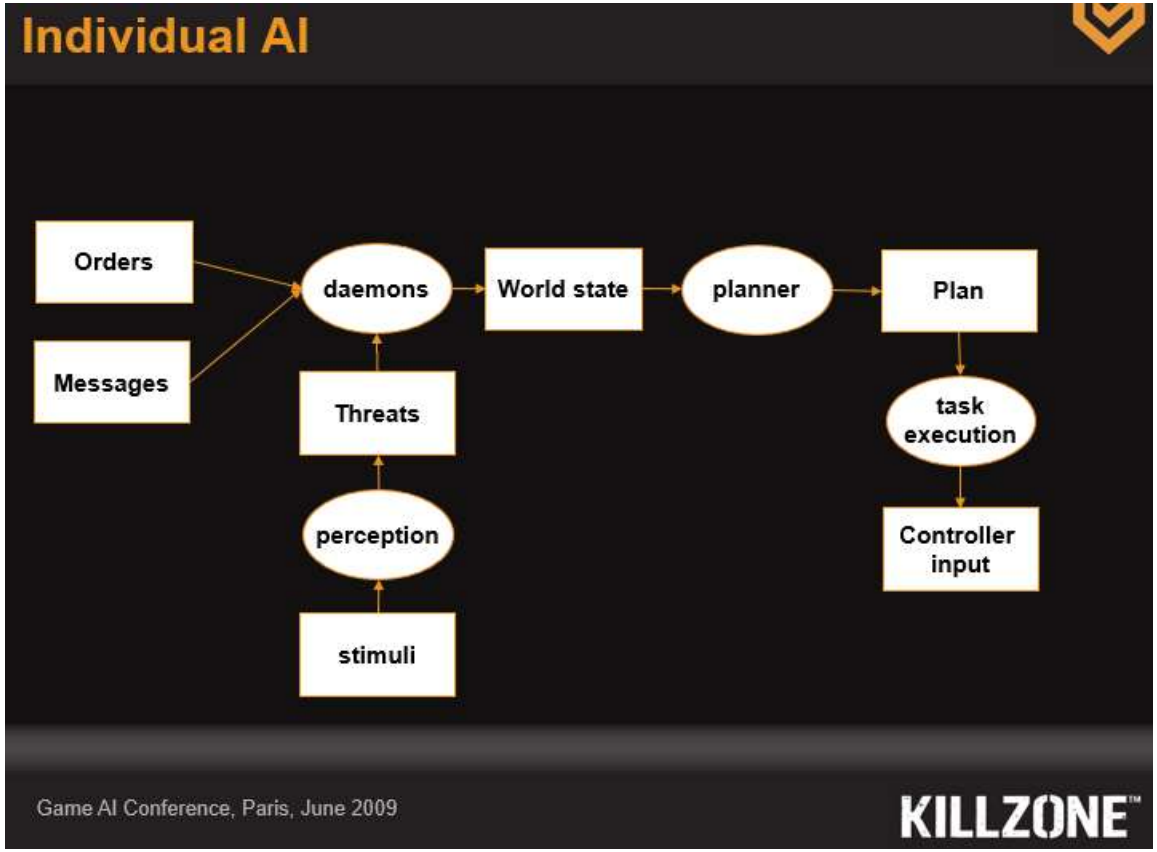
SHOP2

```
(: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



https://www.guerrilla-games.com/read/killzone-2-multiplayer-bots

KILLZONE 2

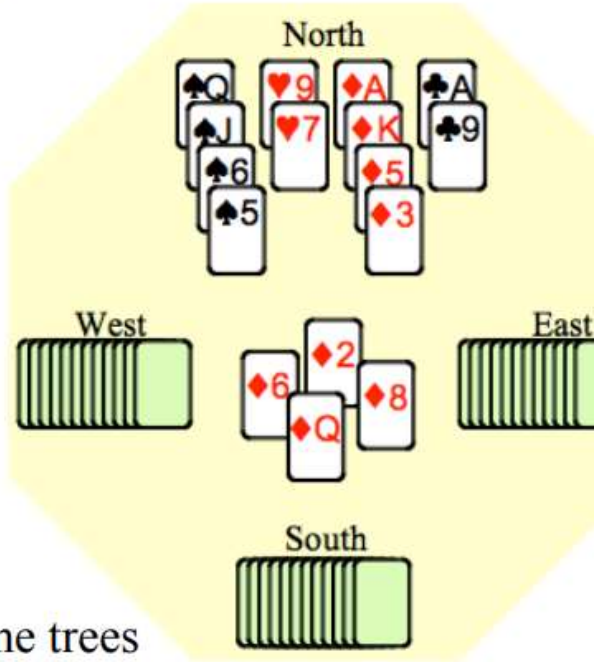
- Special-purpose HTN planner for planning at the squad level
  - » Method and operator syntax similar to SHOP's and SHOP2's
  - » Quickly generates a linear plan that would work if nothing interferes
  - » Replan several times per second as the world changes
- **Why it worked:**
  - » Very different objective from a bridge tournament
  - » Don't ***want*** to look for the best possible play
  - » Need actions that appear believable and consistent to human users
  - » Need them very quickly

http://www.cs.umd.edu/~nau/papers/nau2013game.pdf

Bridge

- Ideal: game-tree search (all lines of play) to compute expected utilities
- Don't know what cards other players have
  - » Many moves they *might* be able to make
    - worst case about 6×10^{44} leaf nodes
    - average case about 10^{24} leaf nodes
- About 1½ minutes available

Not enough time – need smaller tree



- ***Bridge Baron***
  - » 1997 world champion of computer bridge
- Special-purpose HTN planner that generates game trees
  - » Branches ⇔ standard bridge card plays (finesse, ruff, cash out, …)
  - » Much smaller game tree: can search it and compute expected utilities
- **Why it worked**:
  - » Special-purpose planner to generate trees rather than linear plans
  - » Lots of work to make the HTN methods as complete as possible

http://www.cs.umd.edu/~nau/papers/nau2013game.pdf

Planning Versus Games

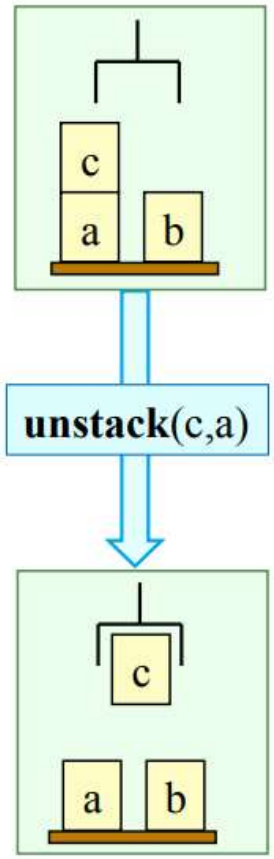
- Pyhop resolves these incompatibilities

	Typical AI Planning	Typical Games
State	Set of propositions	Data structures
Actions	Add/delete propositions	Modify data structures
Agents	One	Many
World	Static	Dynamic
Time available	Whatever the planner needs	Small
Objective	Find complete solution	Find partial solution
Execution	Starts after planning ends	Simultaneous with planning

- Are there general solutions for these?
  - » Or do they need to be game-specific?

http://www.cs.umd.edu/~nau/papers/nau2013game.pdf

Propositions Versus State Variables



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

{loc(a)=table, clear(a)=0, loc(c)=a, clear(c)=1, loc(b)=table, clear(b)=1, holding=nothing}

unstack(x,y)
Precond: on(x,y), clear(x), handempty
Effects: ¬on(x,y), ¬clear(x), clear(y), holding(x), ¬handempty

unstack(x,y)
Precond: loc(x) = y, $y \neq$ table, clear(x) = 1, holding = nothing
Effects: loc(x) = hand, clear(x) = 0, clear(y) = 1, holding = x

- Classical representation:
  - » State: set of propositions
  - » Actions add/delete them
- PDDL is based on this
- Reason is largely historical
  - » AI planning evolved out of AI theorem proving

- State-variable representation:
  - » State: variable bindings
  - » Actions change the values
- Same expressive power
- More compatible with conventional computer programming

http://www.cs.umd.edu/~nau/papers/nau2013game.pdf

Individual AI : HTN Planner

```
(:method (select_weapon_and_attack_as_turret ?inp_threat)
    ( branch_use_bullets   // Only use bullets against humanoids and turrets.
        (and (or (threat ?inp_threat humanoid) (threat ?inp_threat turret) )
            (distance_to_threat ?inp_threat ?threat_distance)
            (call lt ?threat_distance @weapon_bullet_max_range)  )
        ( (attack_as_turret_using_weapon_pref ?inp_threat wp_bullets))
    )
    ( branch_use_rockets // Don't use rockets against humanoids and turrets.
        (and (not (threat ?inp_threat humanoid)) (not (threat ?inp_threat turret))
            (distance_to_threat ?inp_threat ?threat_distance)
            (call lt ?threat_distance @weapon_rocket_max_range) )
        ((attack_as_turret_using_weapon_pref ?inp_threat wp_rockets))
    )
)
```


https://www.guerrilla-games.com/read/killzone-2-multiplayer-bots

Individual AI : Plan monitoring

Plan fails when current task fails

Abort current plan preemptively when
- Better plan available
- Current plan no longer feasible

So, we keep planning @5hz, but:
- Prevent twitchy behavior
- Prevent unnecessary checks (optimizations)

- Combine planning and monitoring using **continue branches**
  - Branch with “continue” as only task in plan
  - When encountered during planning, keep current plan.

https://www.guerrilla-games.com/read/killzone-2-multiplayer-bots

Individual AI : Random Numbers

Individual bot domain

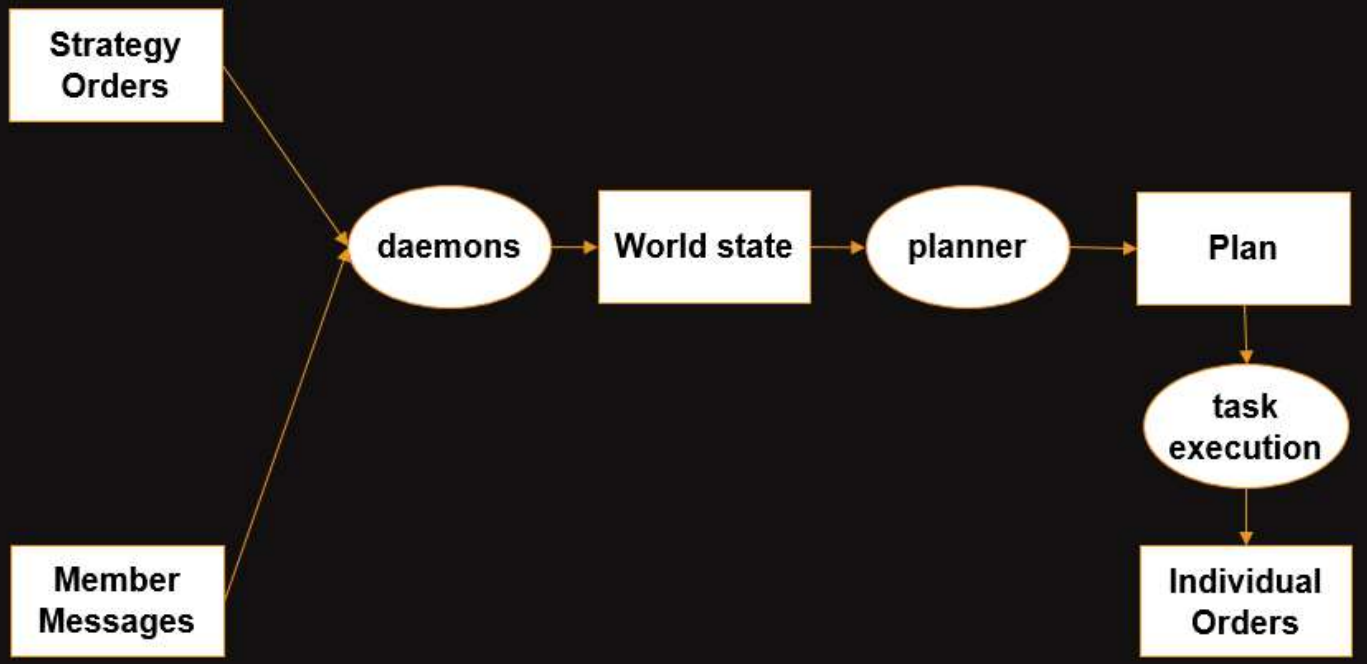
- 360 methods
- 1048 branches
- 138 behaviors
- 147 continue branches

During multiplayer game (14 bots / max. 10 turrets / max. 6 drones / squads)

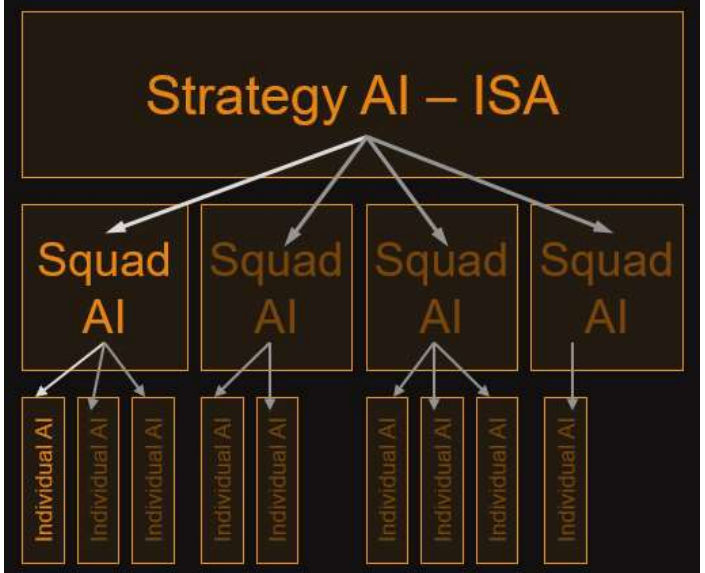
- Approx. 500 plans generated per second
- Approx. 8000 decompositions per second
- Avg. 15 decompositions per planning.
- Approx 24000 branch evaluations per second.

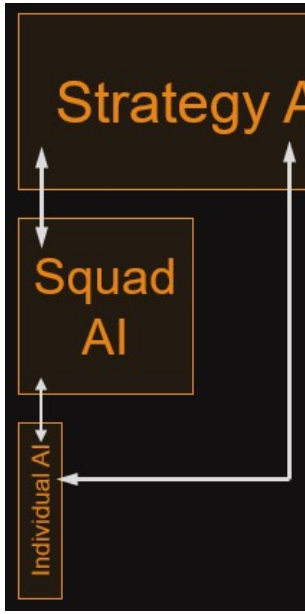
https://www.guerrilla-games.com/read/killzone-2-multiplayer-bots

Squad AI


Strategy Orders
Member Messages
daemons
World state
planner
Plan
task execution
Individual Orders

Architecture





- Strategy to Squad: Orders: Defend, Advance, ...
- Squad to Strategy: Feedback: Order failed

- Squad to Individual: Orders: MoveTo
- Individual to Squad: Combat information

- Strategy to Individual: Orders: Assasination target
- Individual to Strategy: Request reassignment

Given state s, Tasks T, Domain D
Let P = empty plan
Let T_0 = {t∈T | no task comes before t}
Loop
    If T_0 is empty, return P
    Pick any t ∈ T_0
    If t is primitive
        Modify s according to effects
        Add t to P
        Update T by removing t
        T_0 = {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
            T_0 = {t∈subtasks | no task comes before t}
        Else
            T_0 = {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?~~
  - Search time?
  - Authoring time?
  - Optimality?
  - Novelty/Predictability?
  - Partial solutions?

- HTN: Takes advantage of knowledge of how things are done. Often is much faster, but decomposition could slow it down depending on your level of granularity. Less/no improvisation

- A*: Opportunistic discovery and creativity. Harder for player to learn/anticipate.

- POP: Efficient handling of semi-decomposable problems

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

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

- 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/
- Killzone 3:
  - http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter29_Hierarchical_AI_for_Multiplayer_Bots_in_Killzone_3.pdf

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/
  - https://bitbucket.org/dananau/pyhop/src/default/
- 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