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

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

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

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

2018-03-01

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 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!

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
- Disjunctive plans
- SAT

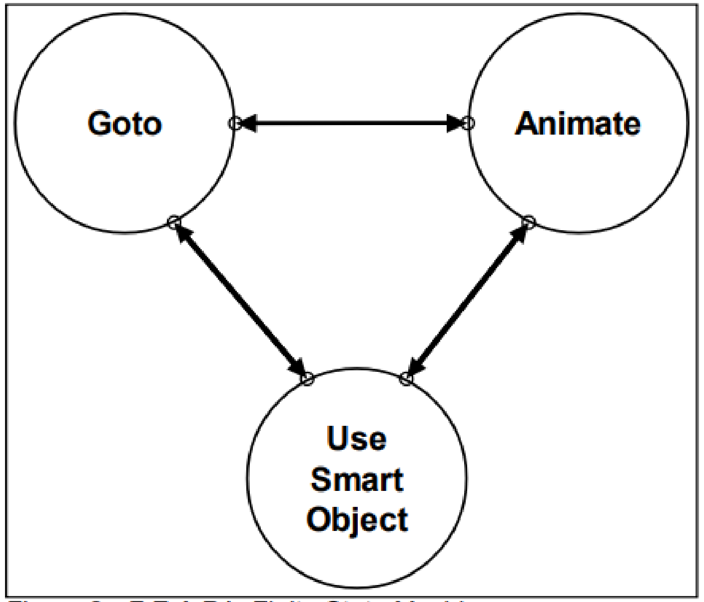
We are going to discuss these forms

ACTION/BEHAVIOR PLANNING WITH A*

Action/Behavior Planning vs. Path planning

- Same algorithm, different action representation
  - Pathplanning:
    - Cells
    - 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 avhieve goals.

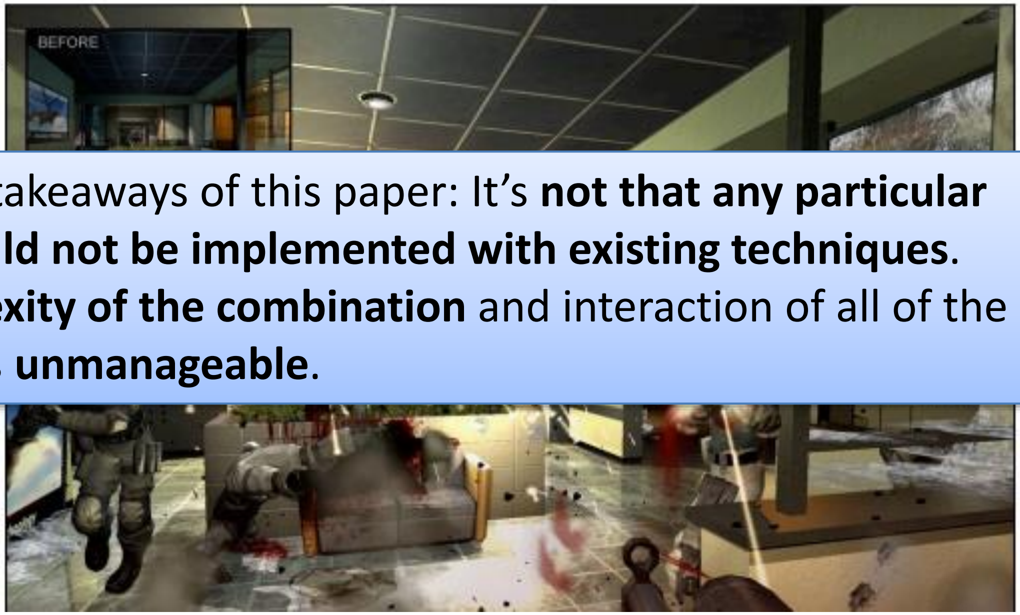


Three States and a Plan: The A.I. of F.E.A.R.

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

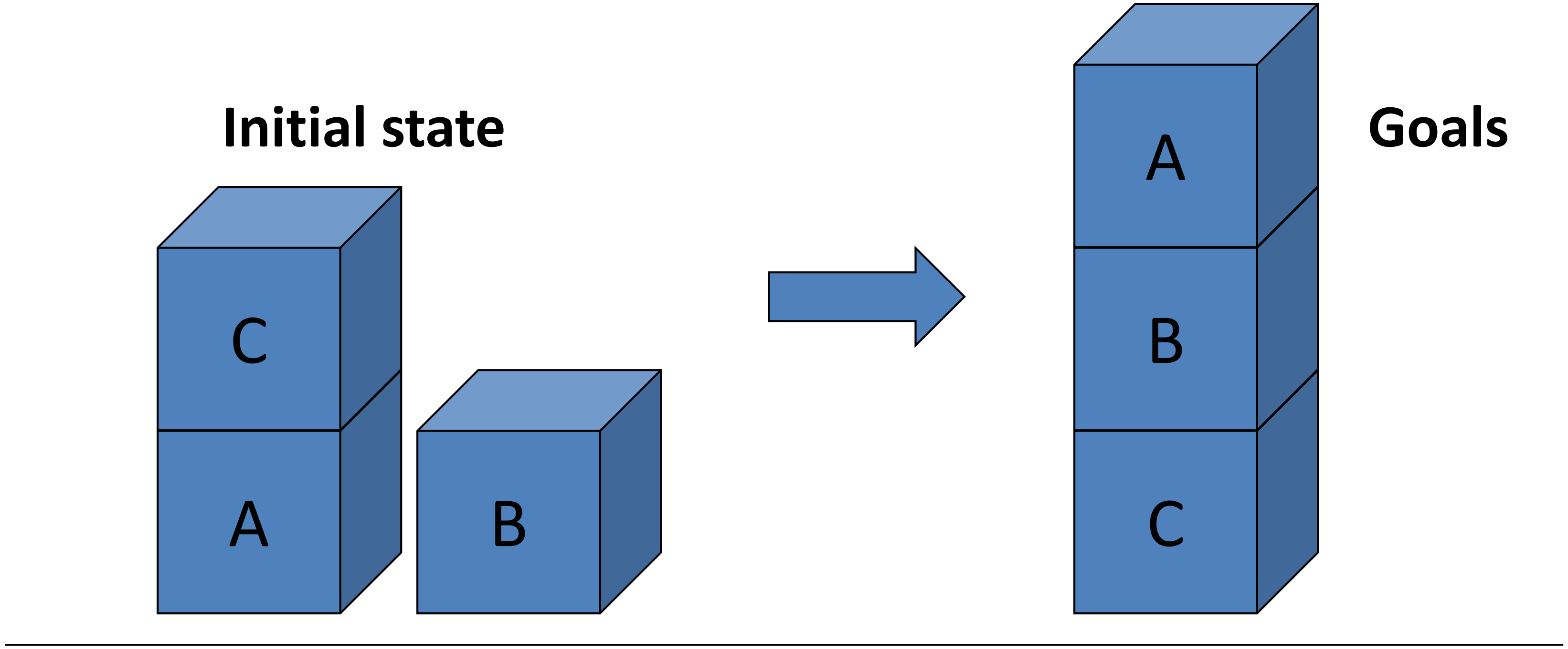
Domain Specific: The plan generation algorithm depends on the particular domain

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

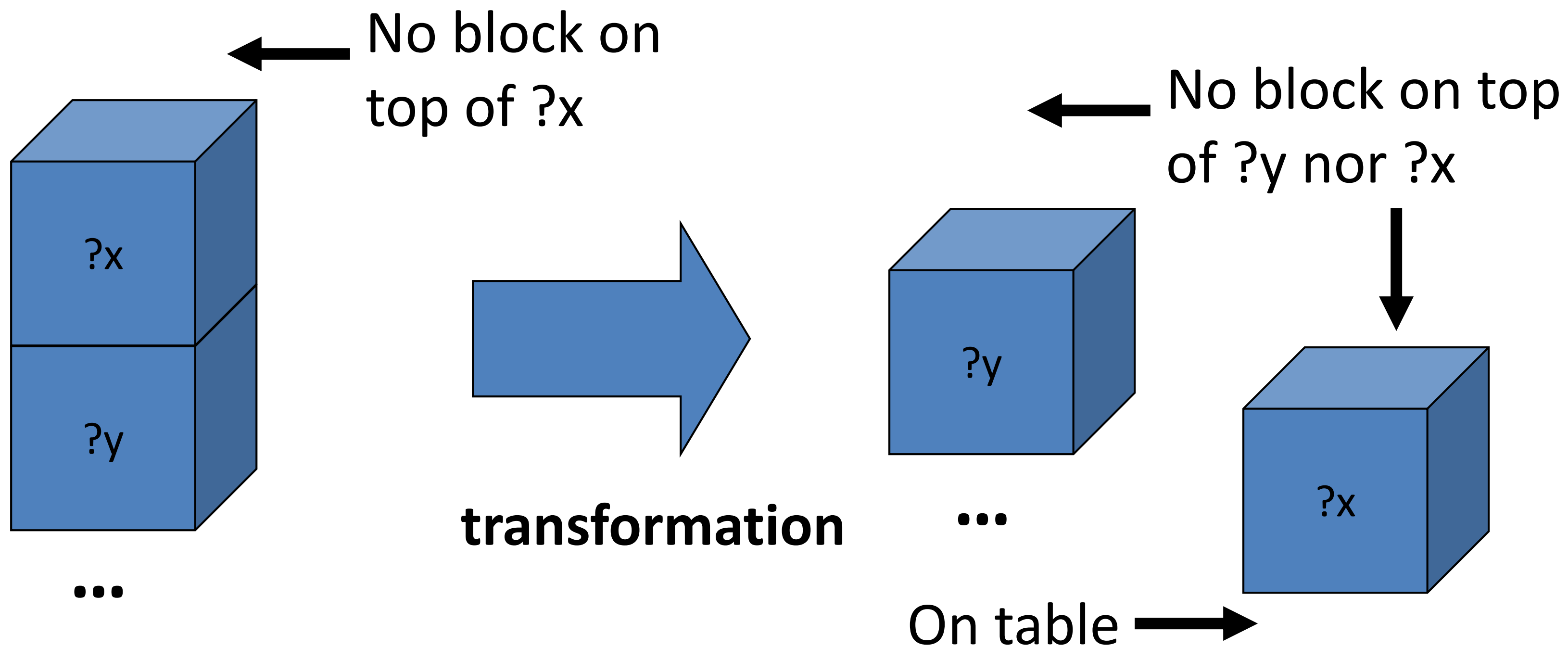
General-Purpose Planning: State & Goals



- **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)

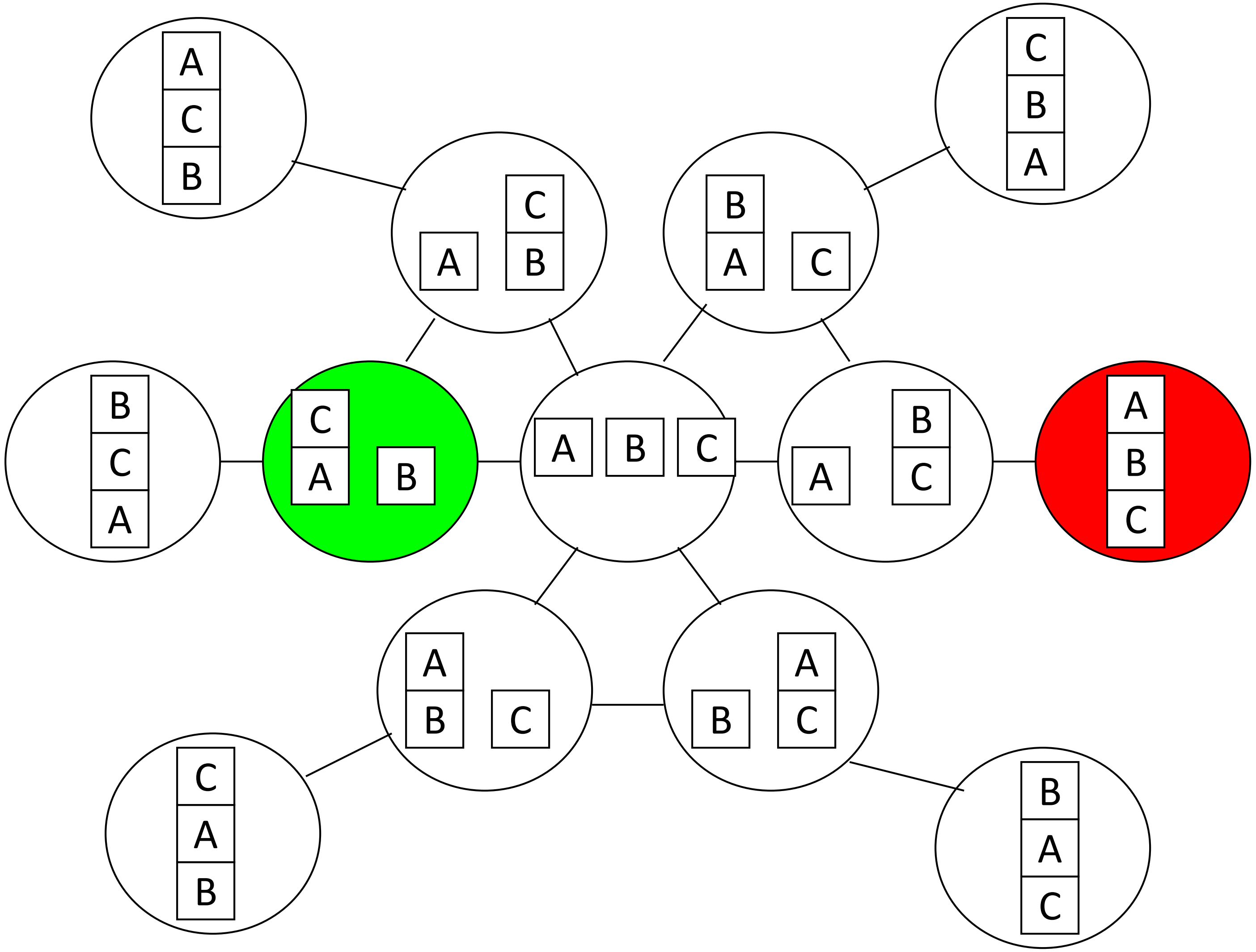
General-Purpose Planning: Operators



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)

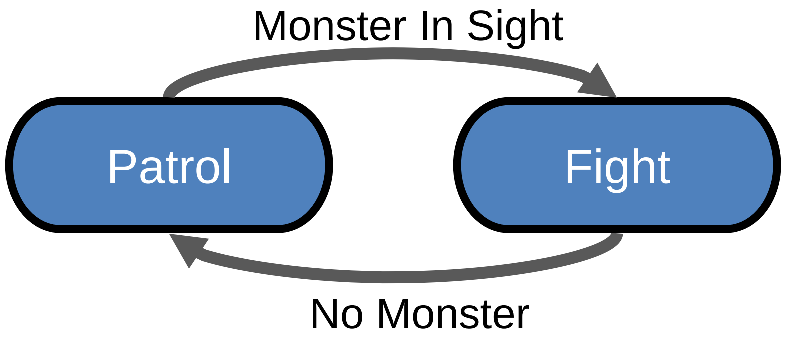
Precondition: at(?p, ?from), plane(?p), airport(?from),
airport(?to), path(?from, ?to), ?from ≠ ?to

Effect: at(?p, ?to), ¬at(?p, ?from)

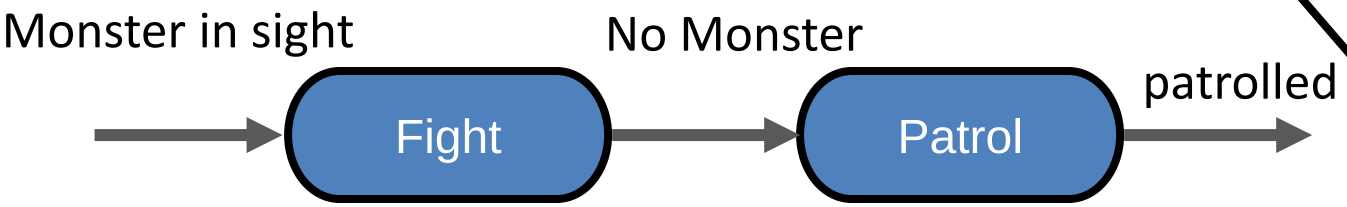
*Also a call to the game engine to play animation or run a function

FSM vs A* 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

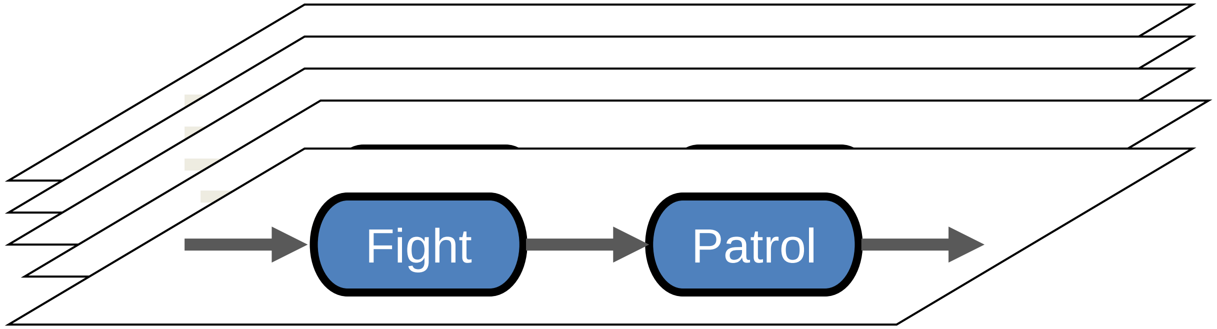
reasoning knowledge

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.
- 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

Other Games with Planning

- Empire: Total War
- Fallout 3
- Killzone

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

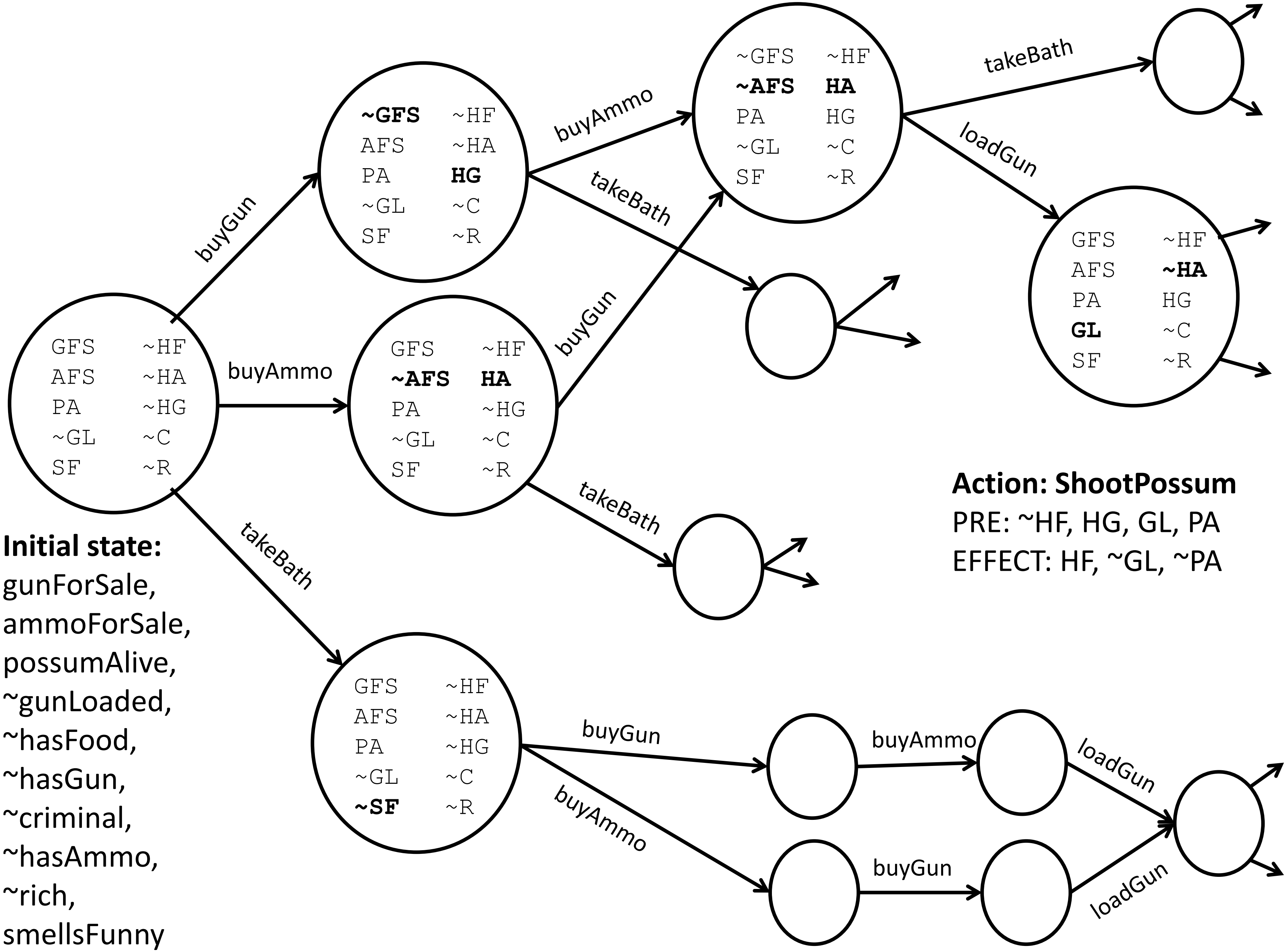
Action: LoadGun
PRE: hasGun, hasAmmo, ~gunLoaded
EFFECT: gunLoaded, ~has Ammo

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

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

Heuristic Search Planning

- 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)

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

PARTIAL ORDER PLANNING

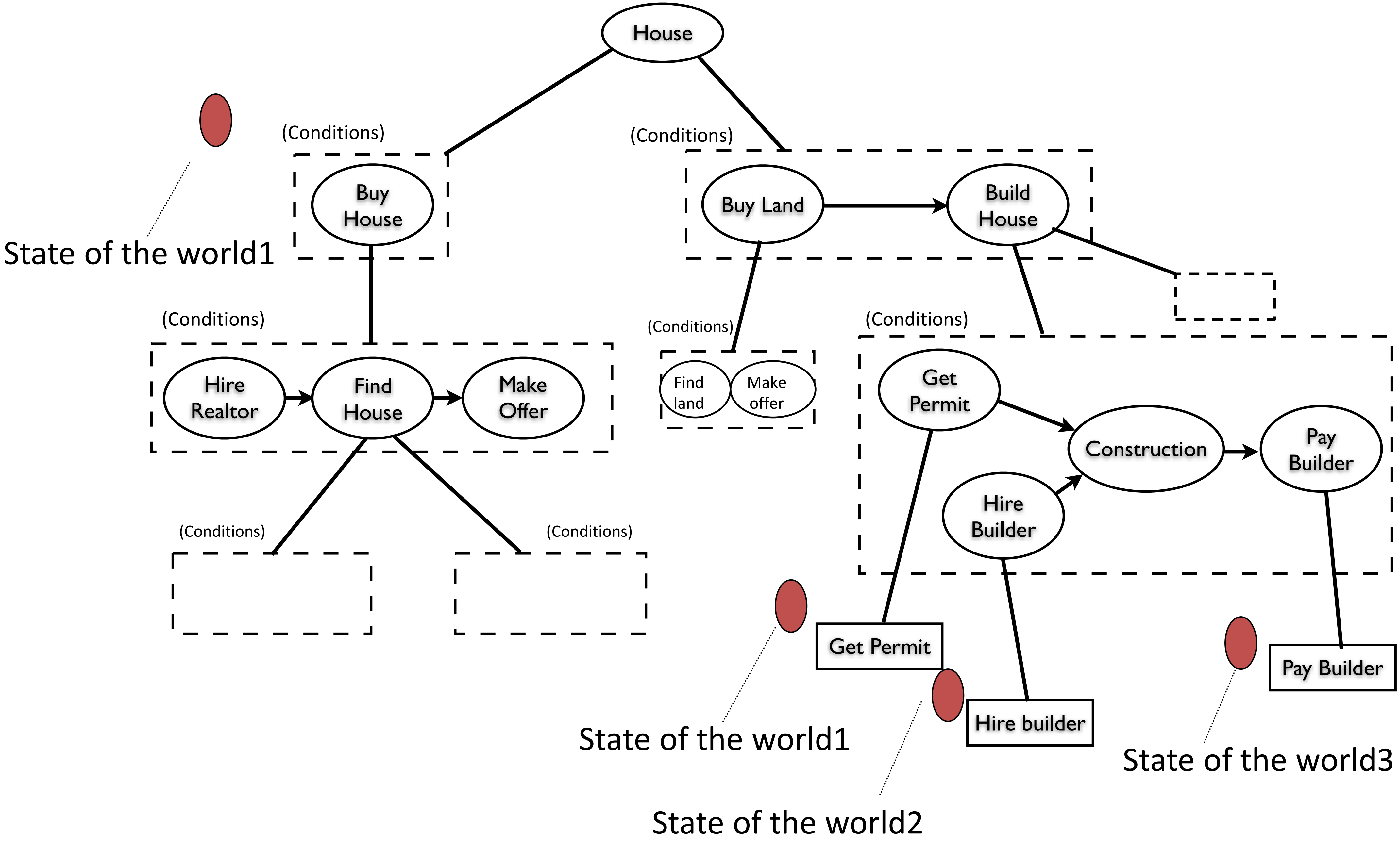
HTN PLANNING

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


House
(Conditions)
Buy House
(Conditions)
Buy Land
Build House
State of the world1
(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
State of the world1
State of the world2
State of the world3

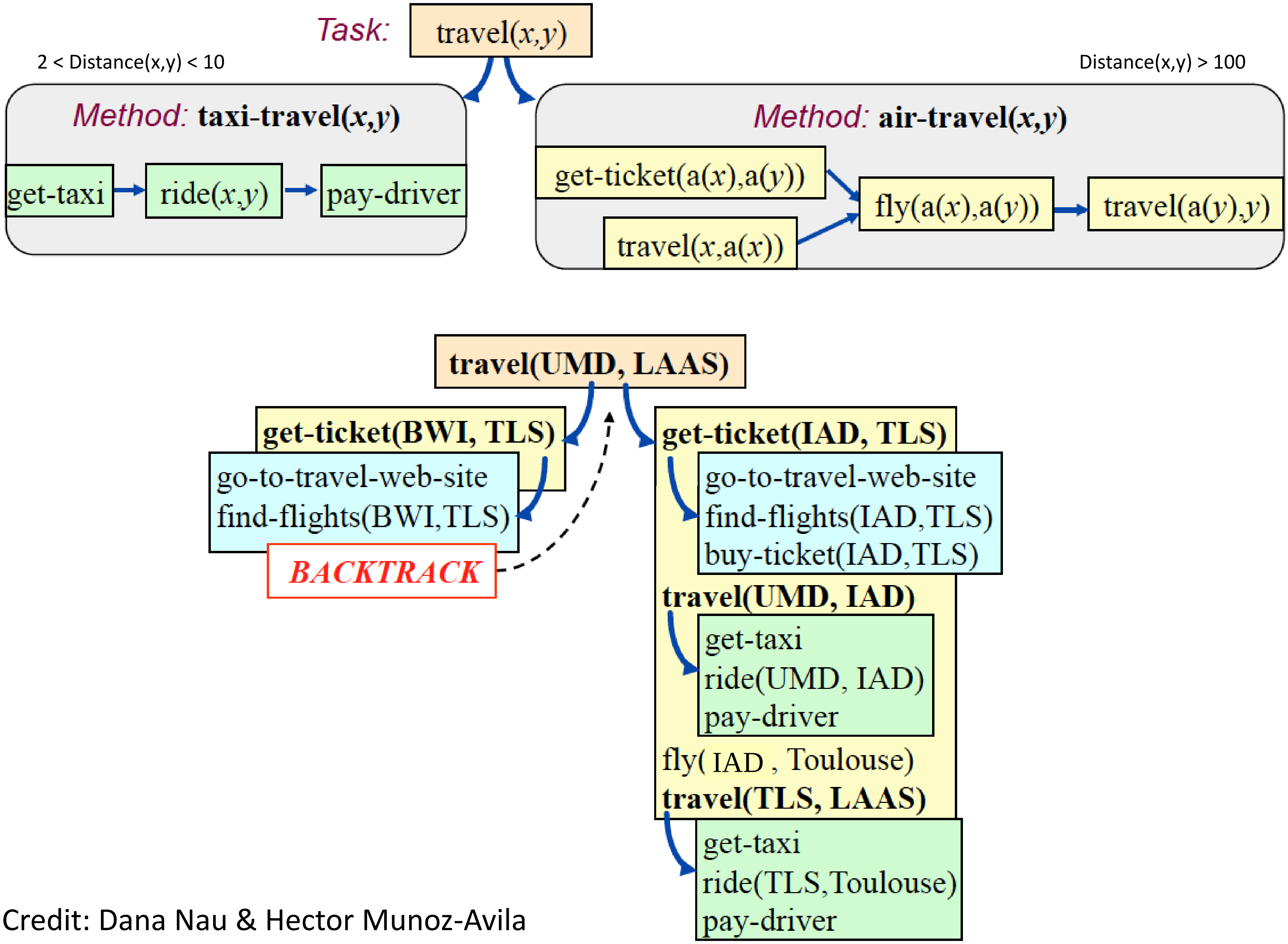
HTN Planning: Idea

- 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

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?

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?

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/

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