Approximation Algorithms for Planning Under Uncertainty

Andrey Kolobov

Computer Science and Engineering

University of Washington, Seattle

Why Approximate?

Difficult example applications:

- **Inventory management**

  "How much X to order from the supplier every day until the end of the season?"

  Lots of uncertainty, large planning horizon

- **Landing damaged aircraft**

  "What's the safest way to land a damaged airplane?"

  Large expected plan length, possibility of a disaster

Formal Models: Finite-Horizon MDPs

Puterman, 1994

FH MDP is a tuple <*S, A, D, T, R, s_0*>, where:

- *S* is a finite state space
- *D* is a sequence of time steps *(1,2,3, …, L)* up to a finite *horizon L*
- *A* is a finite action set
- *T*: *S* x *A* x *S* x *D* → [0, 1] is a transition function
- *R*: *S* x *A* x *S* x *D* → $\mathbb{R}$ is a reward function
- s_0 is the initial state

Solution: policy π*: S x D → A maximizing exp. reward from s_0

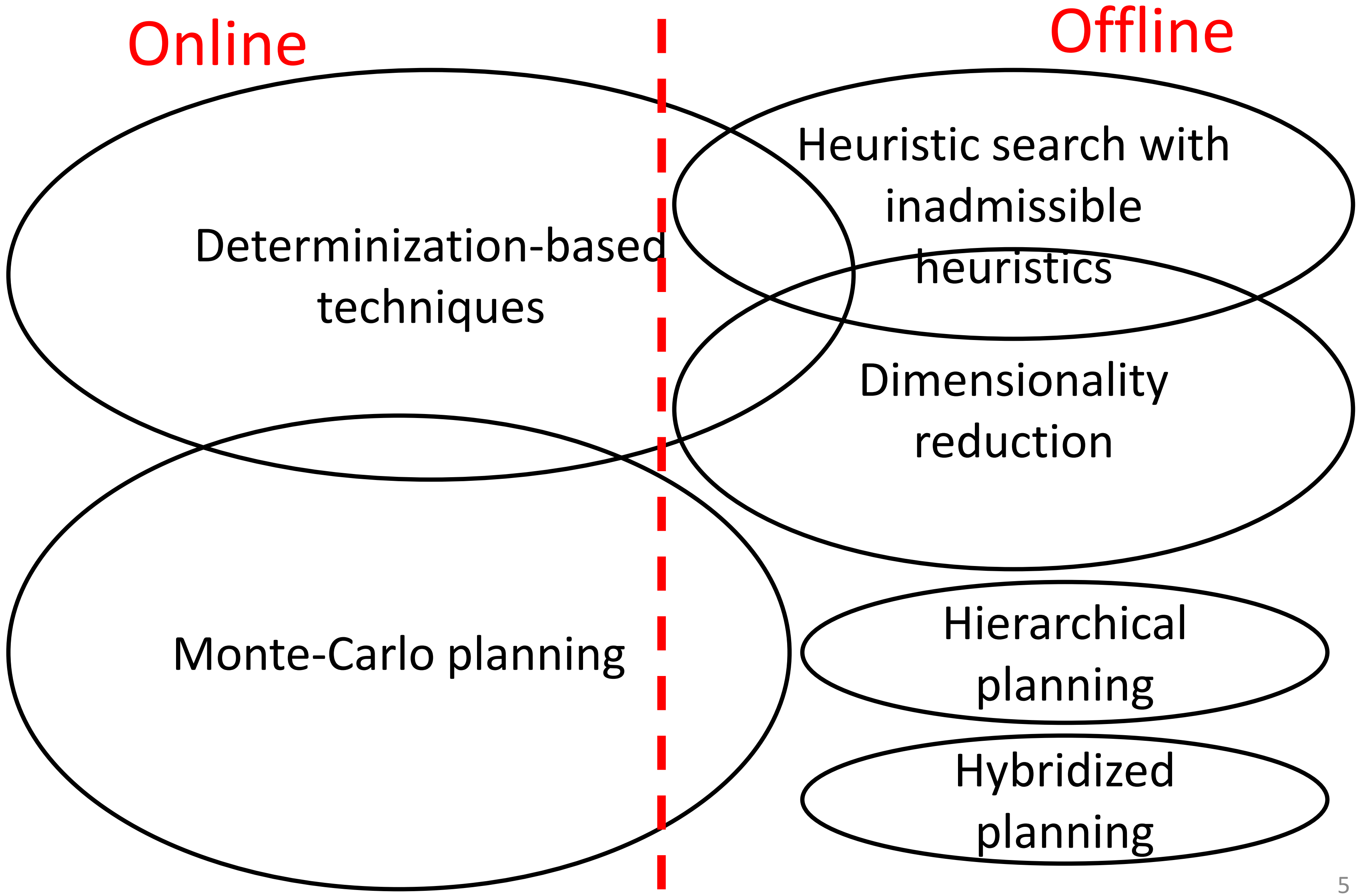
Formal Models: Goal-oriented MDPs

Goal-oriented MDP is a tuple <S, A, T, C, G, s_0, p>, where:

- S is a finite state space
- A is a finite action set
- T: $S \times A \times S \rightarrow [0, 1]$ is a stationary transition function
- C: $S \times A \times S \rightarrow \mathbb{R}^+$ is a stationary *cost function* (= -R: $S \times A \times S \rightarrow \mathbb{R}^-$)
- G is a set of absorbing cost-free goal states
- s_0 is the initial state
- p is a penalty for entering a dead end

Solution: policy $\pi^*: S \rightarrow A$ minimizing expected cost from s_0

Overview of the State of the Art



Outline

✓ Motivation + Overview
- **Determinization-based Algorithms**
- Heuristic Search with Inadmissible Heuristics
- Dimensionality Reduction
- Monte-Carlo Planning
- Hierarchical Planning
- Hybridized Planning

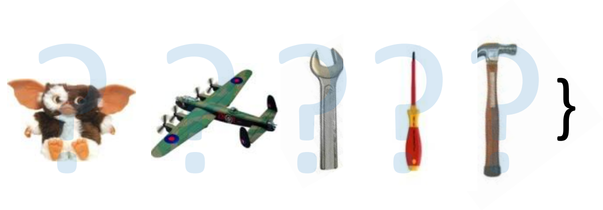
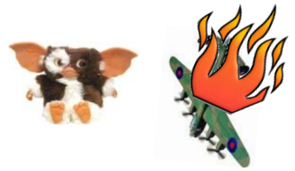
Determinization-based Techniques

- High-level idea:
  1. Compile MDP into its *determinization*
  2. Generate plans in the determinization
  3. Use the plans to choose an action in the curr. state
  4. Execute, repeat

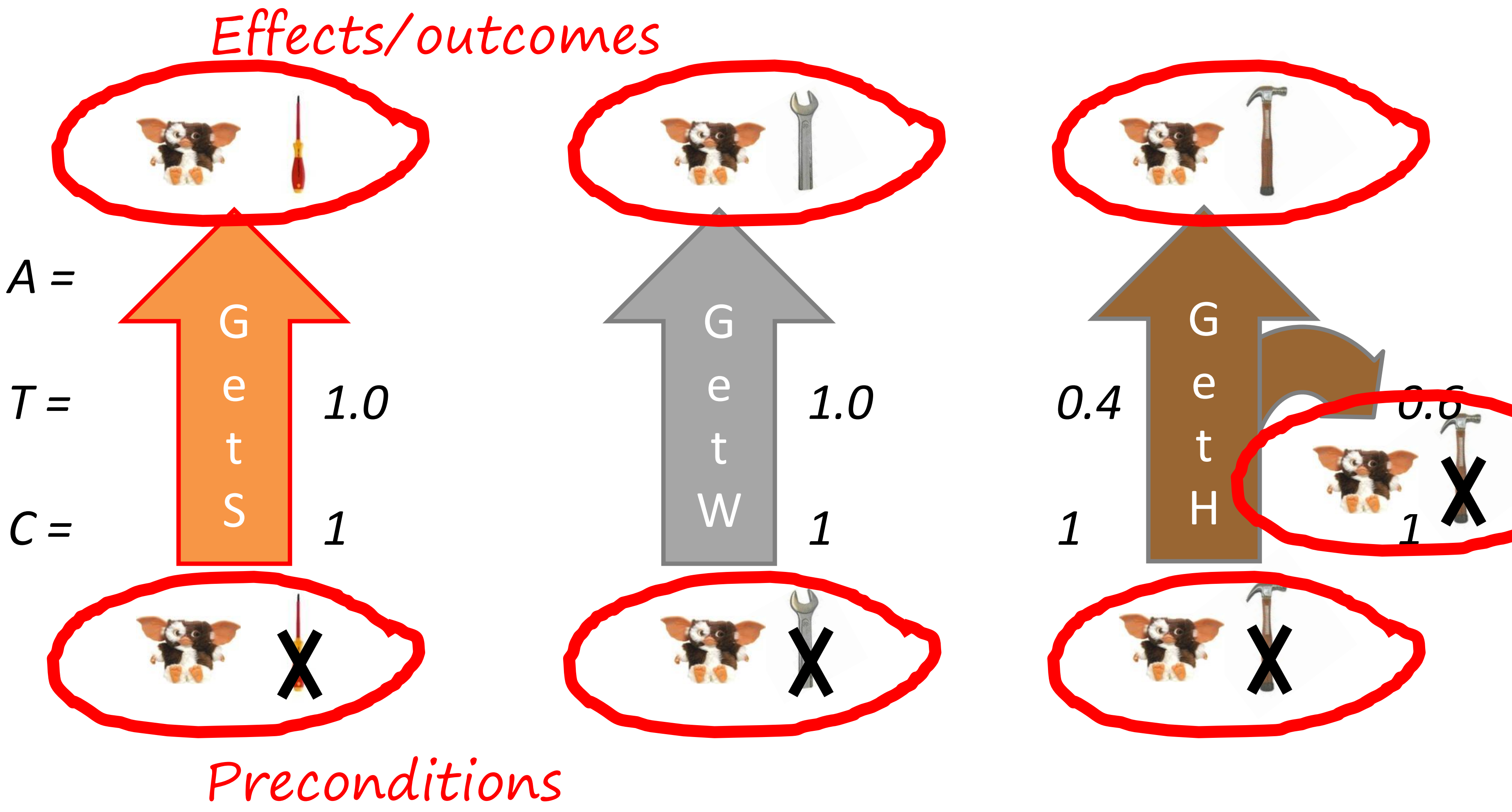
- Key insight:
  – Classical planners are *very* fast

- Key assumption:
  – The MDP is goal-oriented and in *factored representation*
  – I.e., MDP components are expressed in terms of *state variables*

Example Factored Goal-Oriented MDP

- Gremlin wants to sabotage an airplane
- Can use tools to fulfill its objective
- Needs to pick up the tools
- X = {  }
- G = 

Example Factored Goal-Oriented MDP

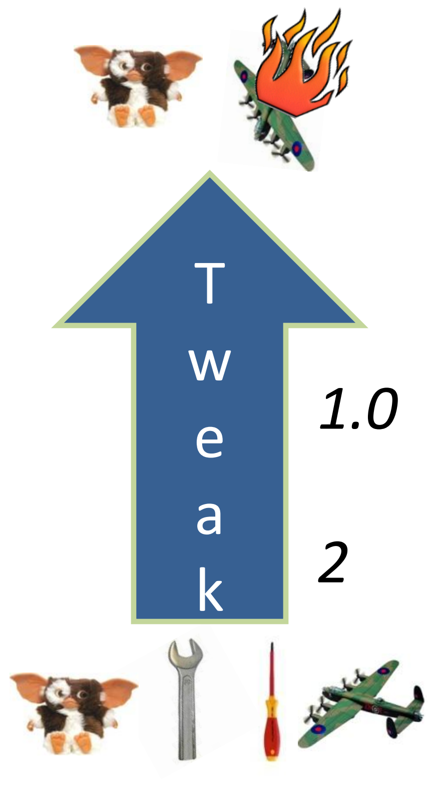


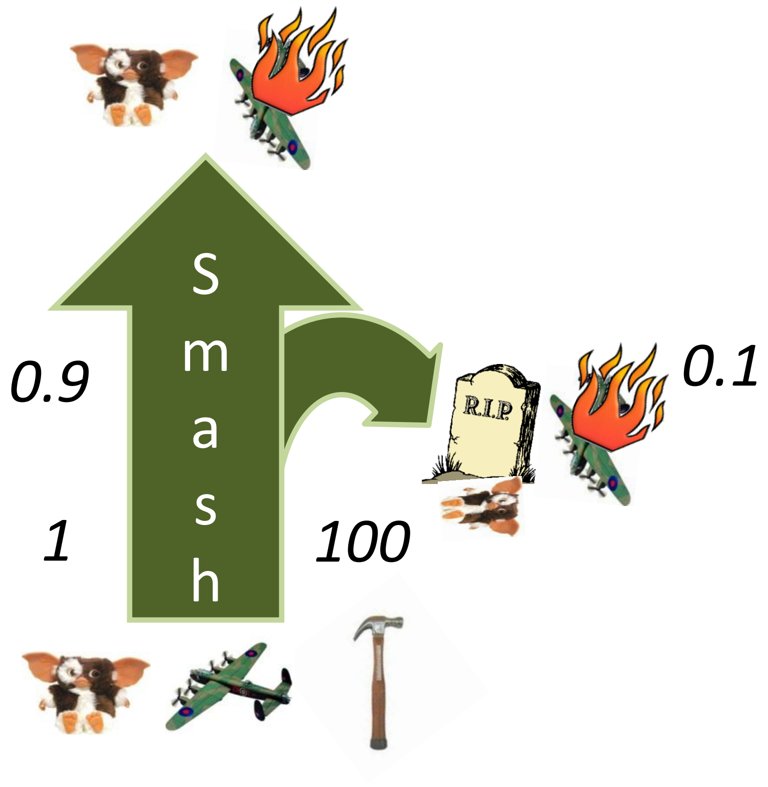
Example Factored Goal-Oriented MDP

A =

T =

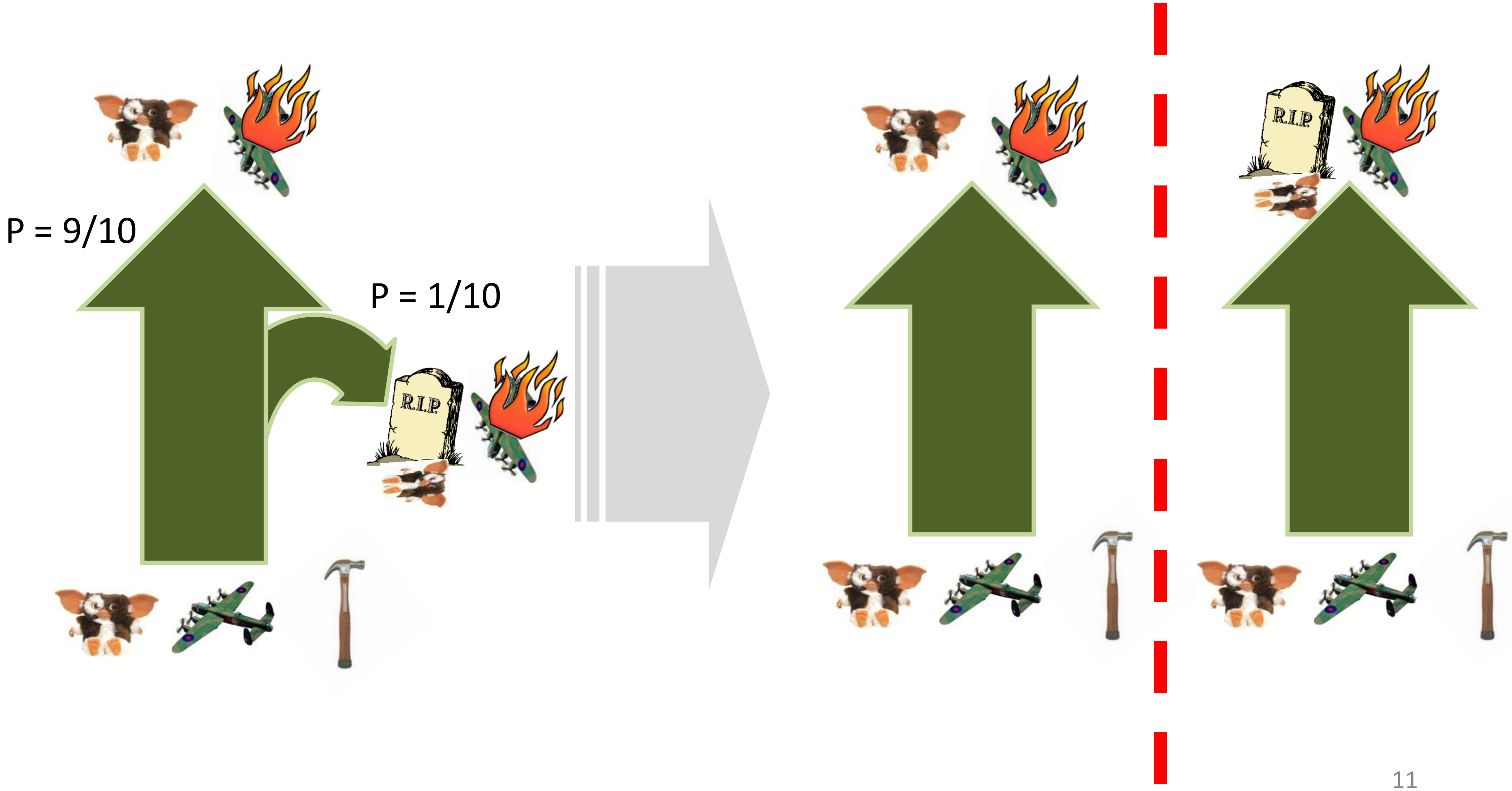
C =



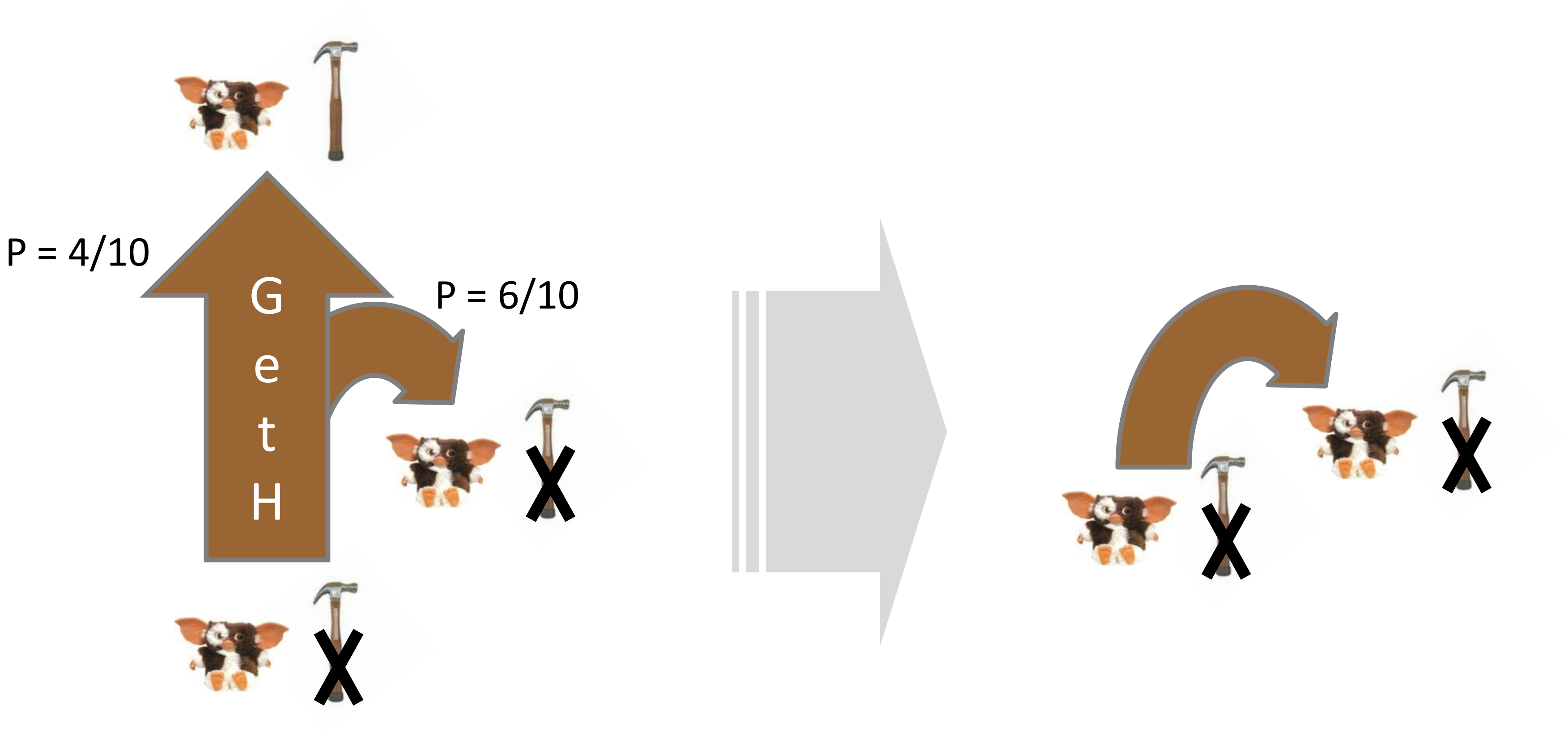


Back to Determinization-based Planning: All-Outcome Determinization

Each outcome of each probabilistic action → separate action



Most-Likely-Outcome Determinization



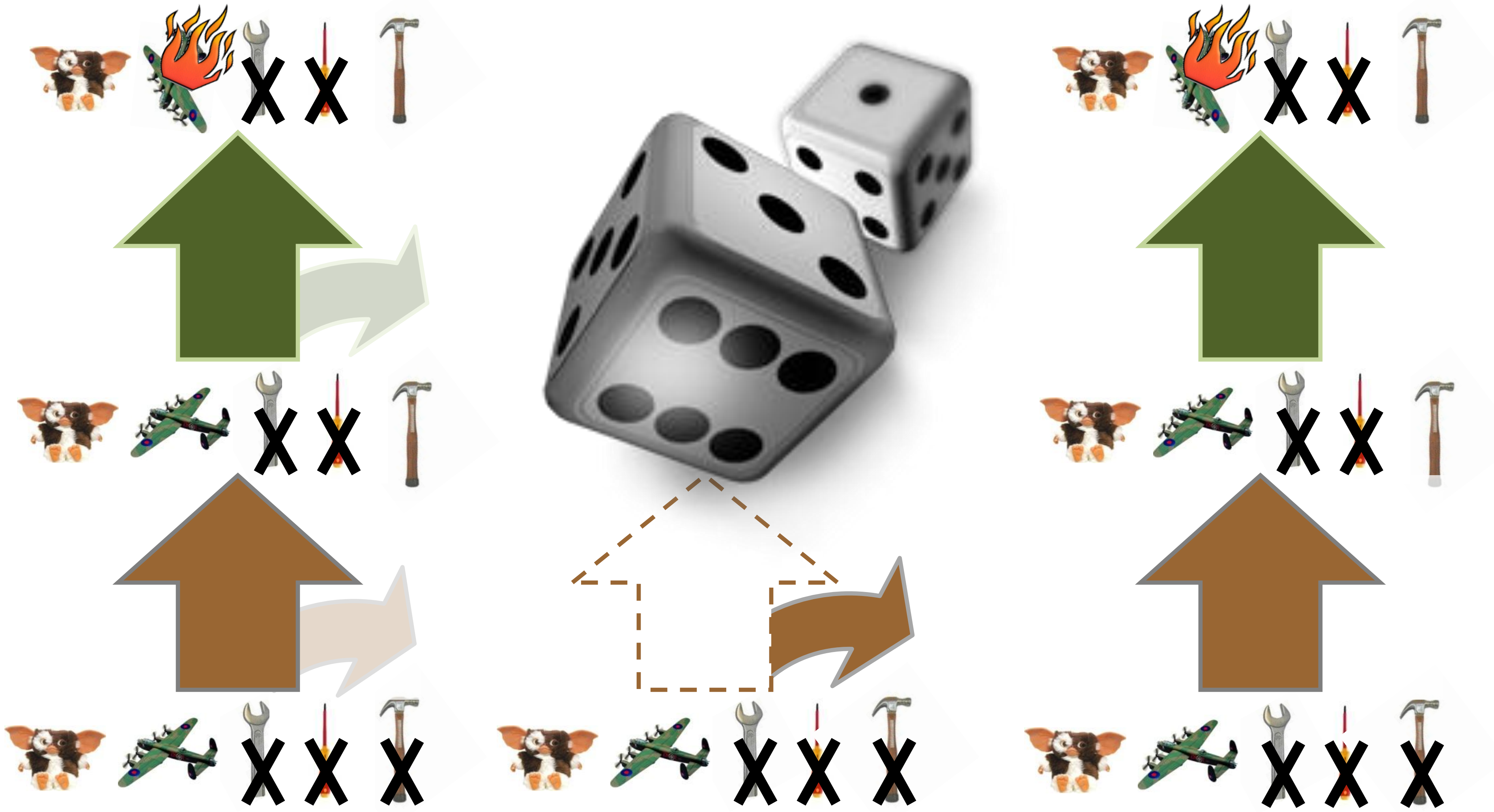
[Yoon, Fern, Givan, 2007]

FF-Replan: Overview & Example

1) Find a goal plan in a determinization

2) Try executing it in the original MDP

3) Replan&repeat if unexpected outcome



FF-Replan: Theoretical Properties

- Super-efficient on goal-oriented MDPs w/o dead ends
  - Won two IPPCs (-2004 and -2006) against much more sophisticated techniques
- Ignores probability of deviation from the found plan
  - Results in long-winded paths to the goal
  - Troubled by *probabilistically interesting MDPs* [Little, Thiebaux, 2007]
    - There, an unexpected outcome may lead to catastrophic consequences
- In particular, breaks down in the presence of dead ends

FF-Hindsight: Putting “Probabilistic” Back Into Planning

- FF-Replan is oblivious to probabilities
  - Its main undoing
  - How do we take them into account?

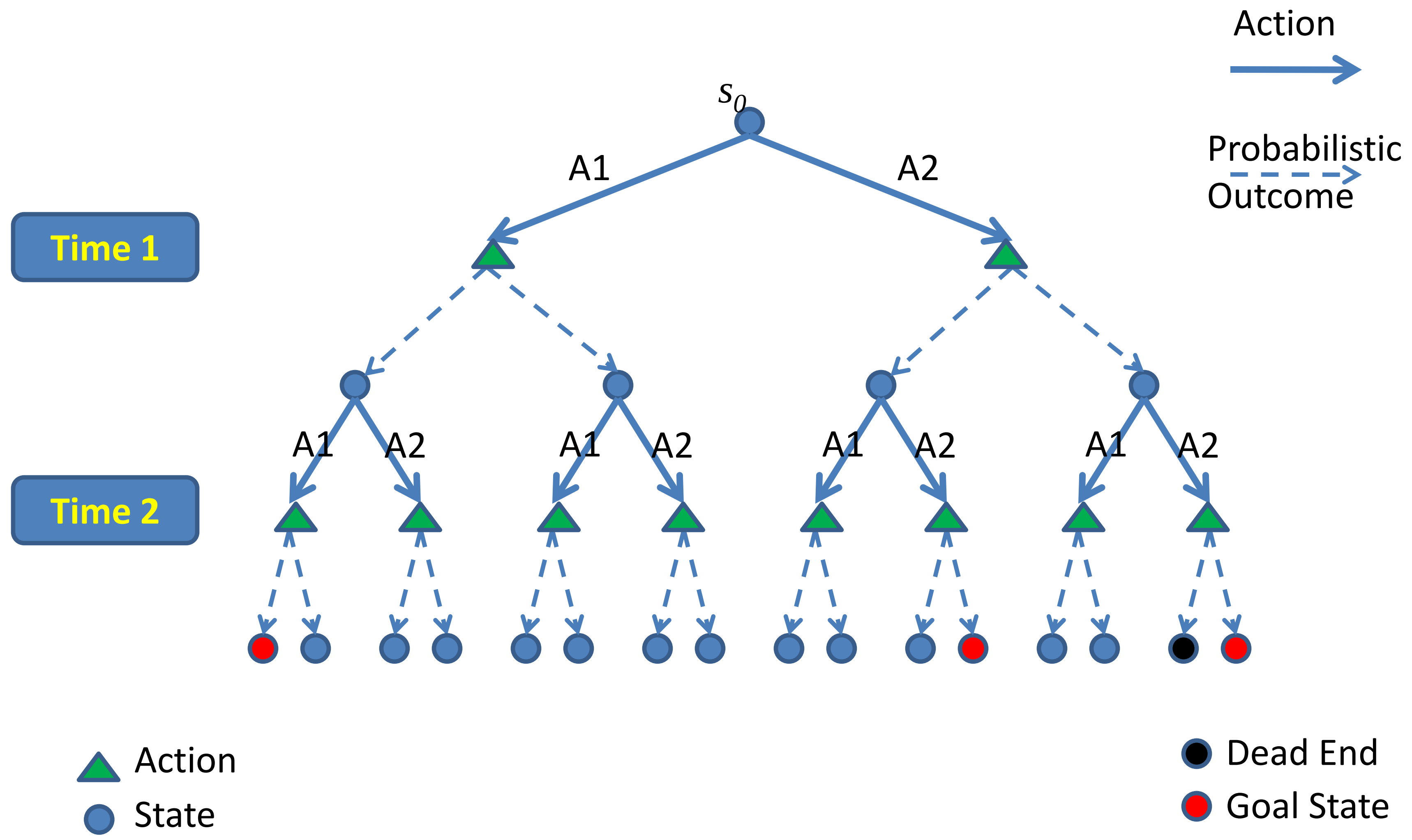
- Suppose you knew the future…
  - Knew would happen if you executed action *a* *t* steps from now
  - The problem would be deterministic, could easily solve it

- Don’t actually know the future…
  - …but can sample several futures probabilistically
  - Solutions tell which actions are likely to start successful policies

- Basic idea behind *FF-Hindsight*

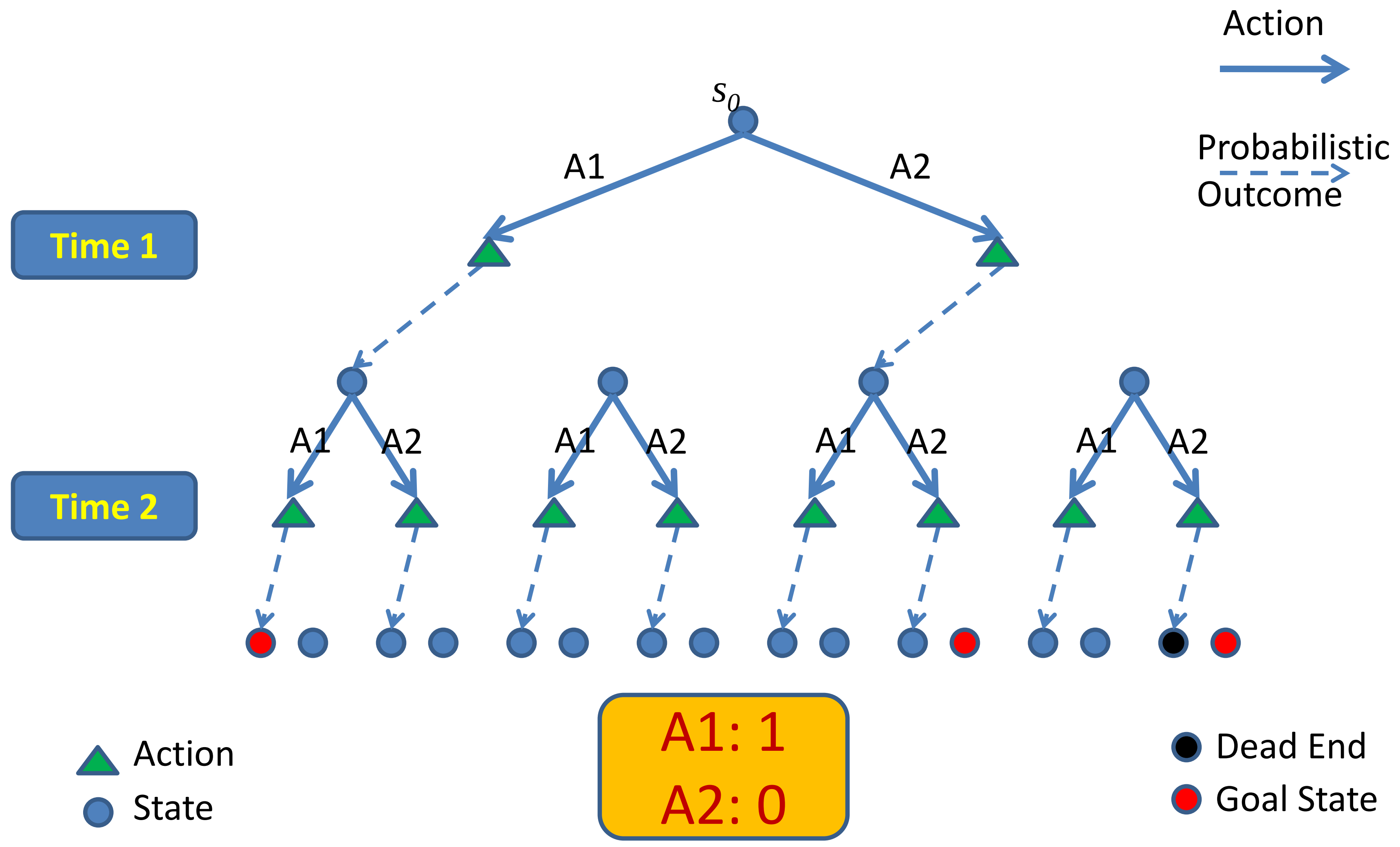
FF-Hindsight: Example

[Yoon, Fern, Givan, Kambhampati, 2010]



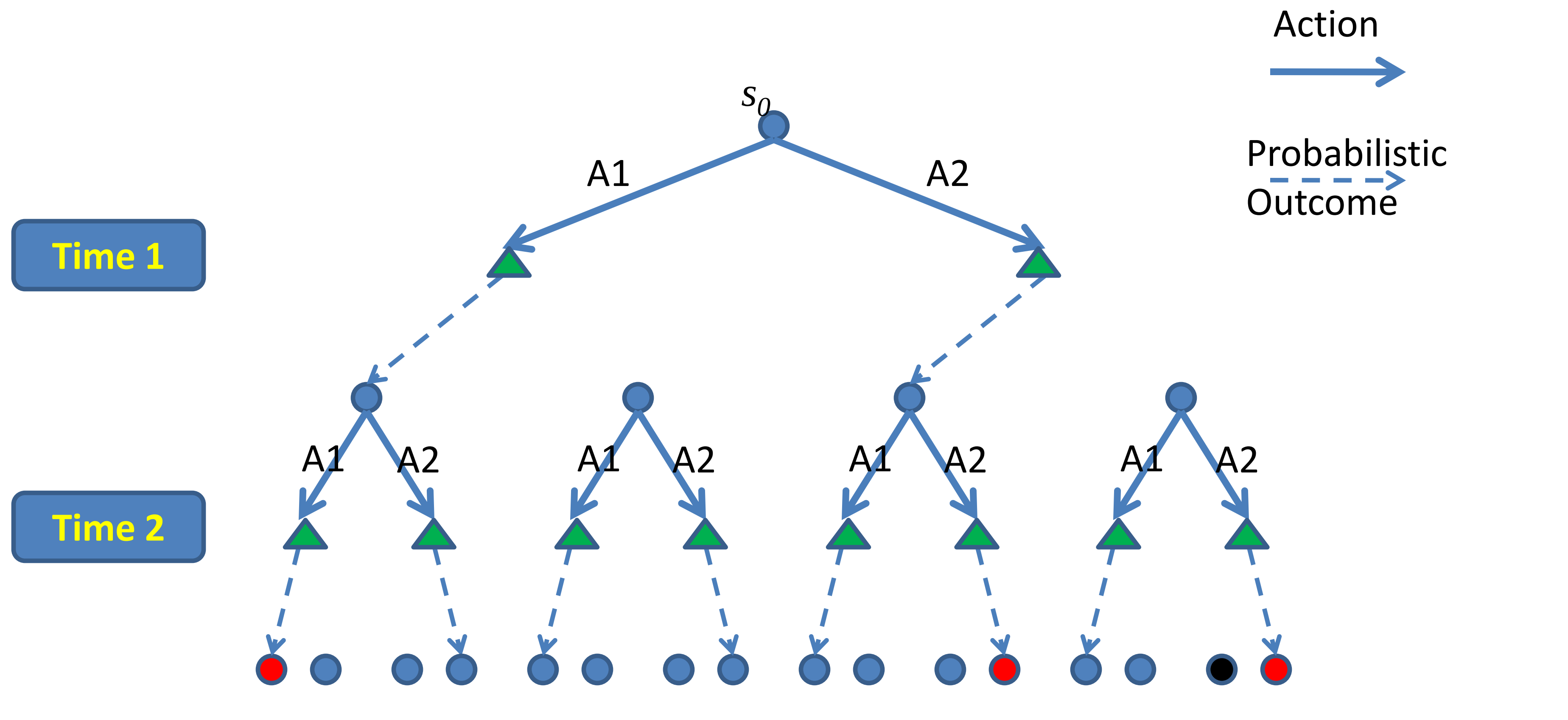
Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati

FF-Hindsight: Sampling a Future-1



Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati

FF-Hindsight: Sampling a Future-2



▲ Action

● State

A1: 2

A2: 1

● Dead End

● Goal State

Slide courtesy of S. Yoon, A. Fern, R. Givan, and R. Kambhampati

Providing Solution Guarantees

- FF-Replan provides no solution guarantees
  - May have probabaility of reaching the goal $P_G = 0$ on MDPs with dead ends, even if $P^*_G > 0$

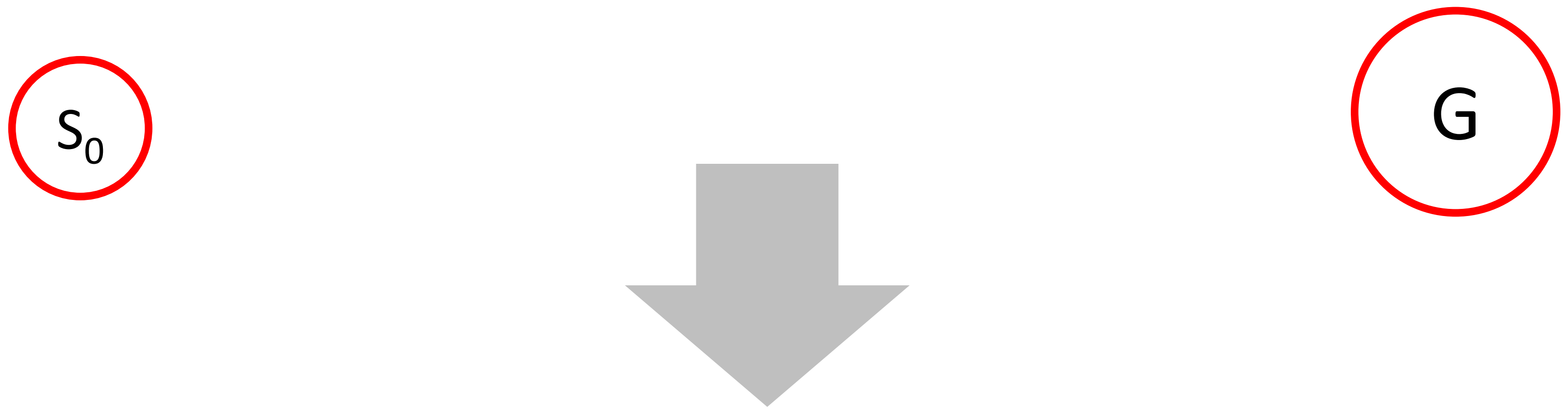
- FF-Hindsight provides only theoretical guarantees
  - Practical implementations too distinct from theory

- **RFF (Robust FF) provides quality guarantees in practice**
  - Constructs a policy tree out of deterministic plans

RFF: Initialization

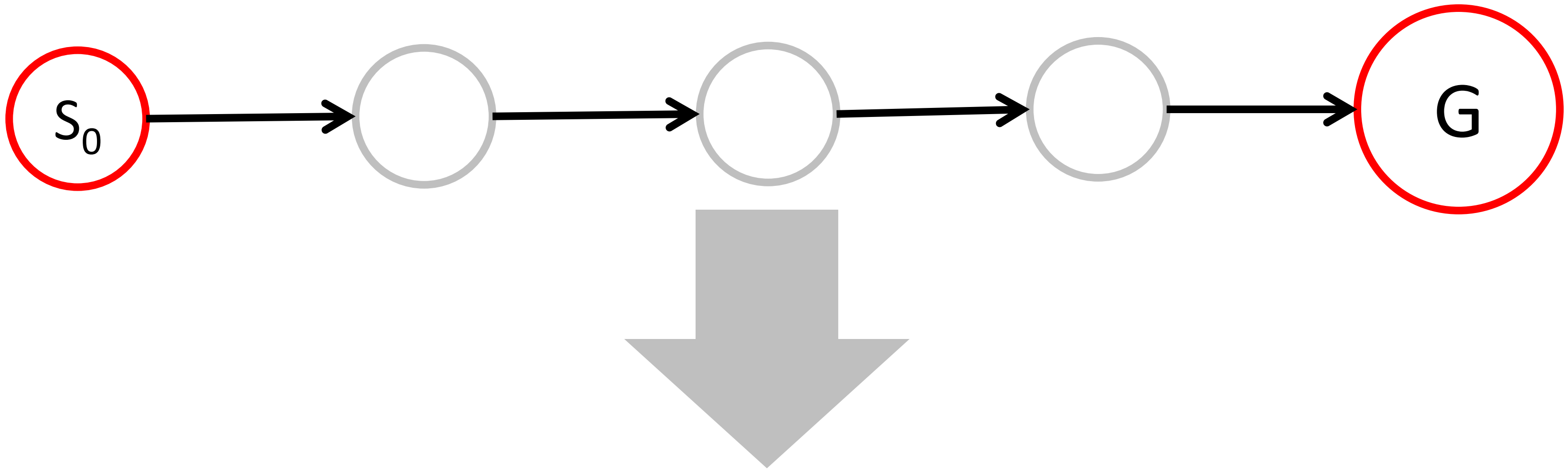
[Teichteil-Königsbuch, Infantes , Kuter, 2010]

1. Generate either the AO or MLO determinization. Start with the policy graph consisting of the initial state s_0 and all goal states G



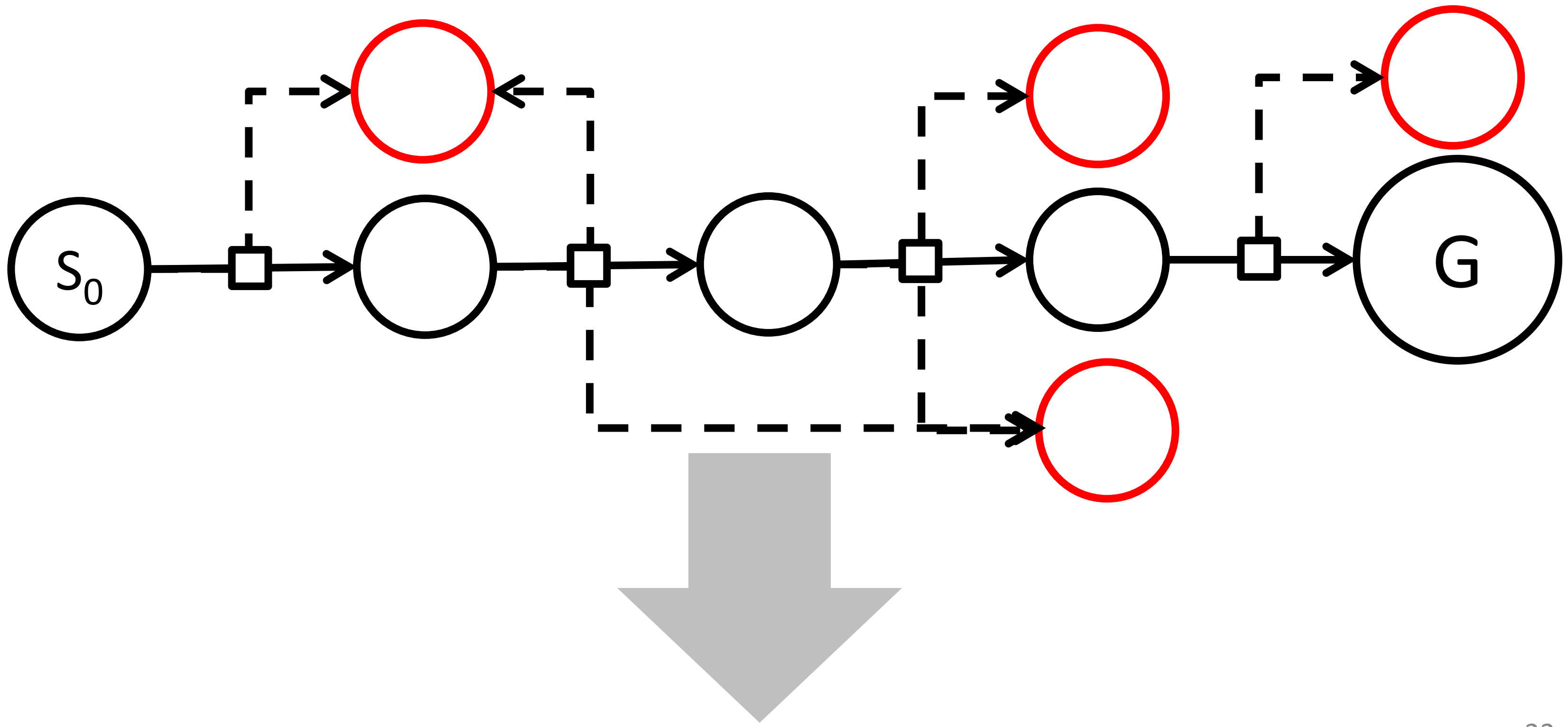
RFF: Finding an Initial Plan

2. Run FF on the chosen determinization and add all the states along the found plan to the policy graph.



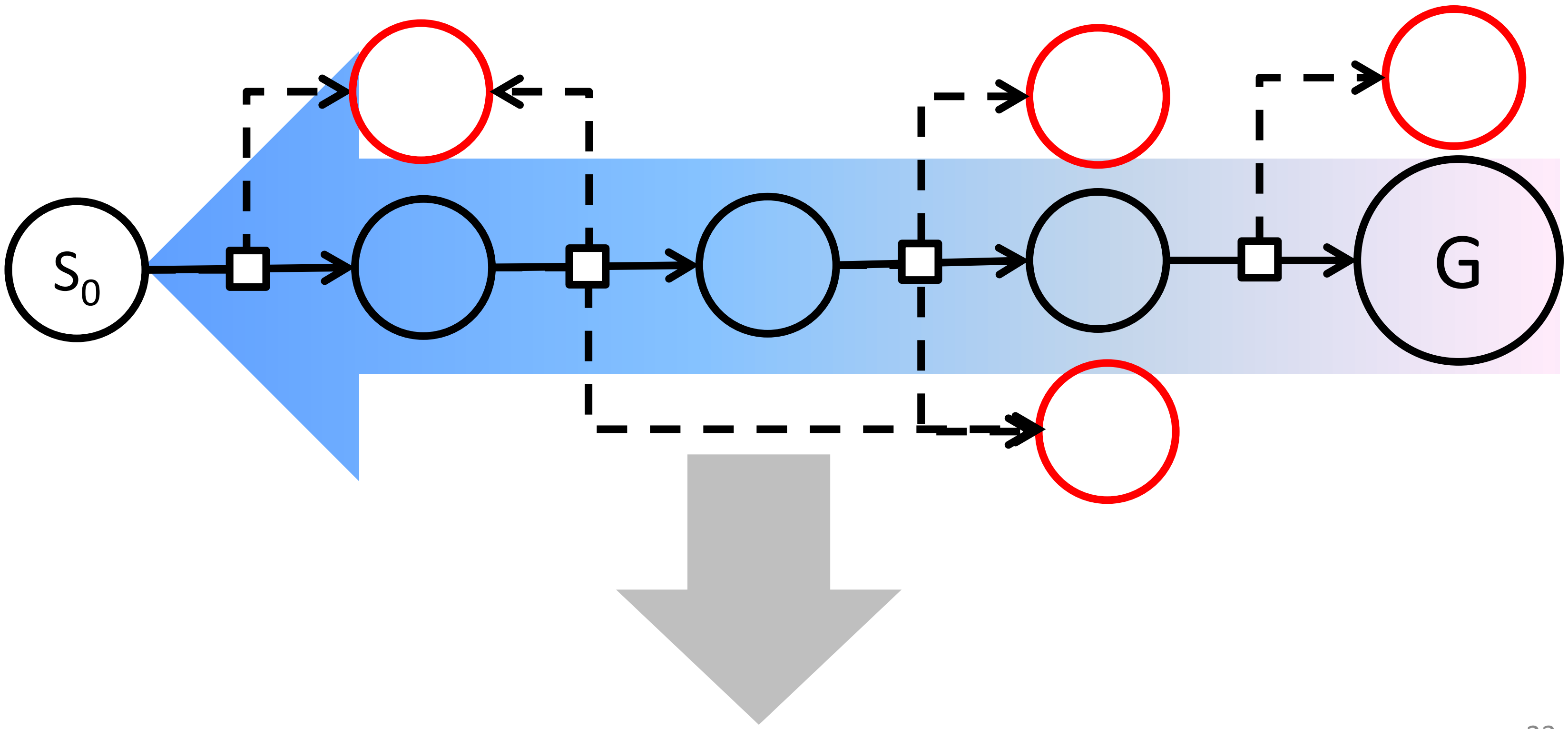
RFF: Adding Alternative Outcomes

3. Augment the graph with states to which other outcomes of the actions in the found plan could lead and that are not in the graph already. They are the policy graph's *fringe states*.



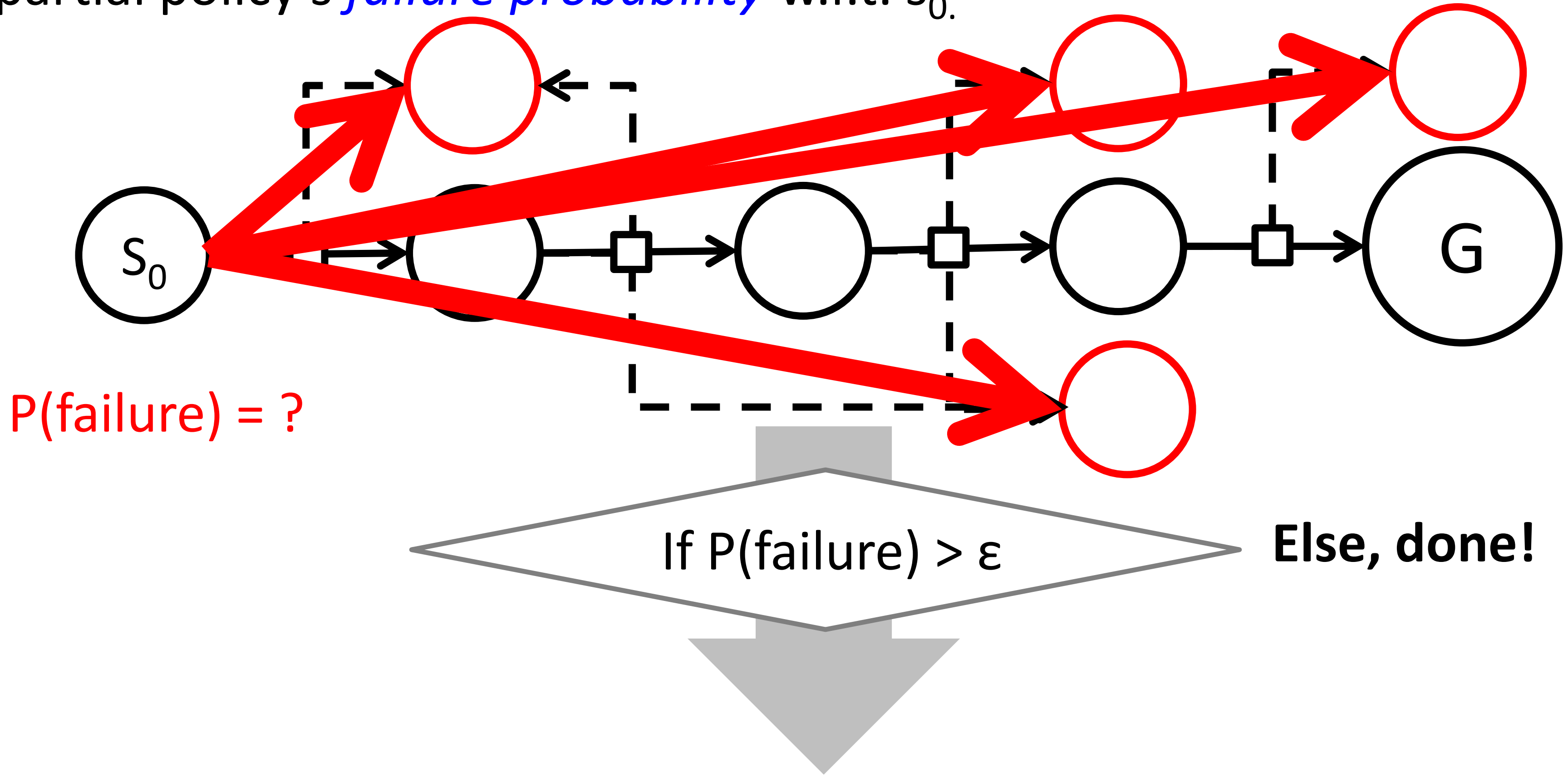
RFF: Run VI

4. Run VI to propagate heuristic values of the newly added states.
This possibly changes the graph's fringe and helps avoid dead ends!



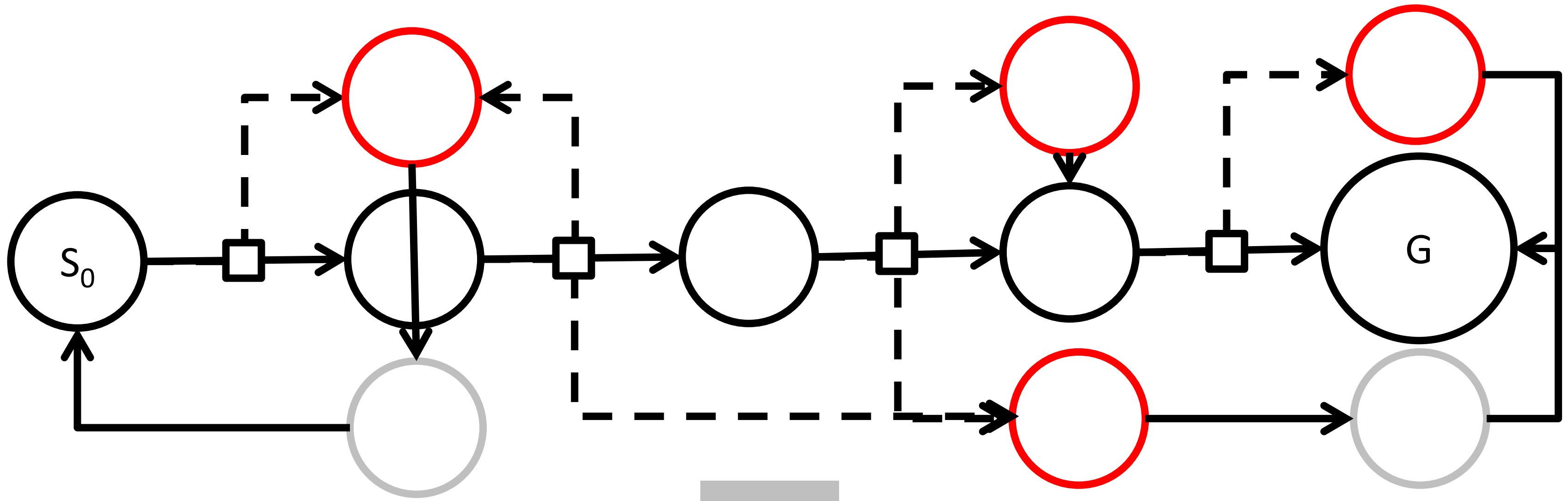
RFF: Computing Replanning Probability

5. Estimate the probability *P(failure)* of reaching the fringe states (e.g., using Monte-Carlo sampling) from s_0. This is the current partial policy's *failure probability* w.r.t. s_0.



RFF: Finding Plans from the Fringe

6. From each of the fringe states, run FF to find a plan to reach the goal or one of the states already in the policy graph.



Go back to step 3: Adding Alternative Outcomes

Summary of Determinization Approaches

- Revolutionized goal-oriented MDPs' approximation techniques
  - Harnessed the speed of classical planners
  - Eventually, started to take into account probabilities

- Classical planners help by quickly finding paths to a goal
  - Takes "probabilistic" MDP solvers a while to find them on their own

- However…
  - Still almost completely disregard expect cost of a solution
  - Often assume uniform action costs (since many classical planners do)
  - So far, not useful on finite-horizon MDPs turned goal-oriented
    - Reaching a goal in them is trivial, need to approximate reward more directly
  - Impractical on problems with large numbers of outcomes

Outline

- ✓ Motivation + Overview
- ✓ Determinization-based Algorithms
- **Heuristic Search with Inadmissible Heuristics**
- Dimensionality Reduction
- Monte-Carlo Planning
- Hierarchical Planning
- Hybridized Planning

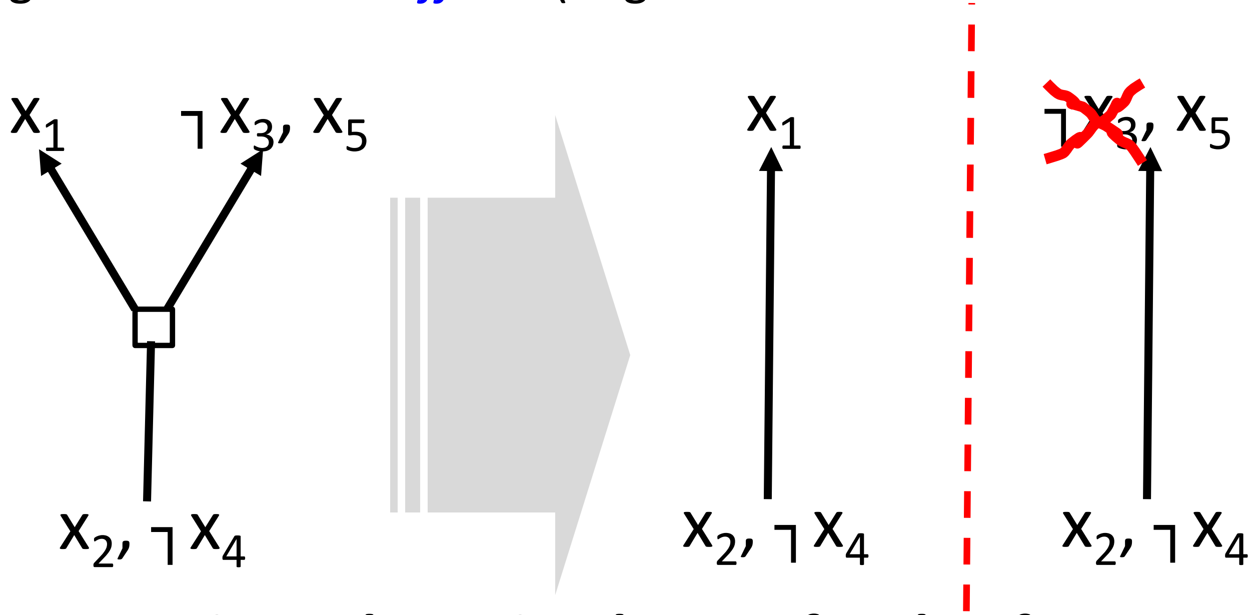
Inadmissible Heuristic Search

- Why?
  - May require less space than admissible heuristic search
- Apriori, no reason to expect an arbitrary inadmissible heuristic to yield a small solution
  - But, empirically, those based on determinization often do
- Same algos as for admissible HS, only heuristics differ

The FF Heuristic

[Hoffmann and Nebel, 2001]

- Taken directly from deterministic planning
- Uses the all-outcome determinization of an MDP
  - But ignores the *delete effects* (negative literals in action outcomes)



- ***$h_{FF}(s)$* = approximately optimal cost of a plan from *s* to a goal in the delete relaxation**
- Very fast due to using the delete relaxation
- Very informative

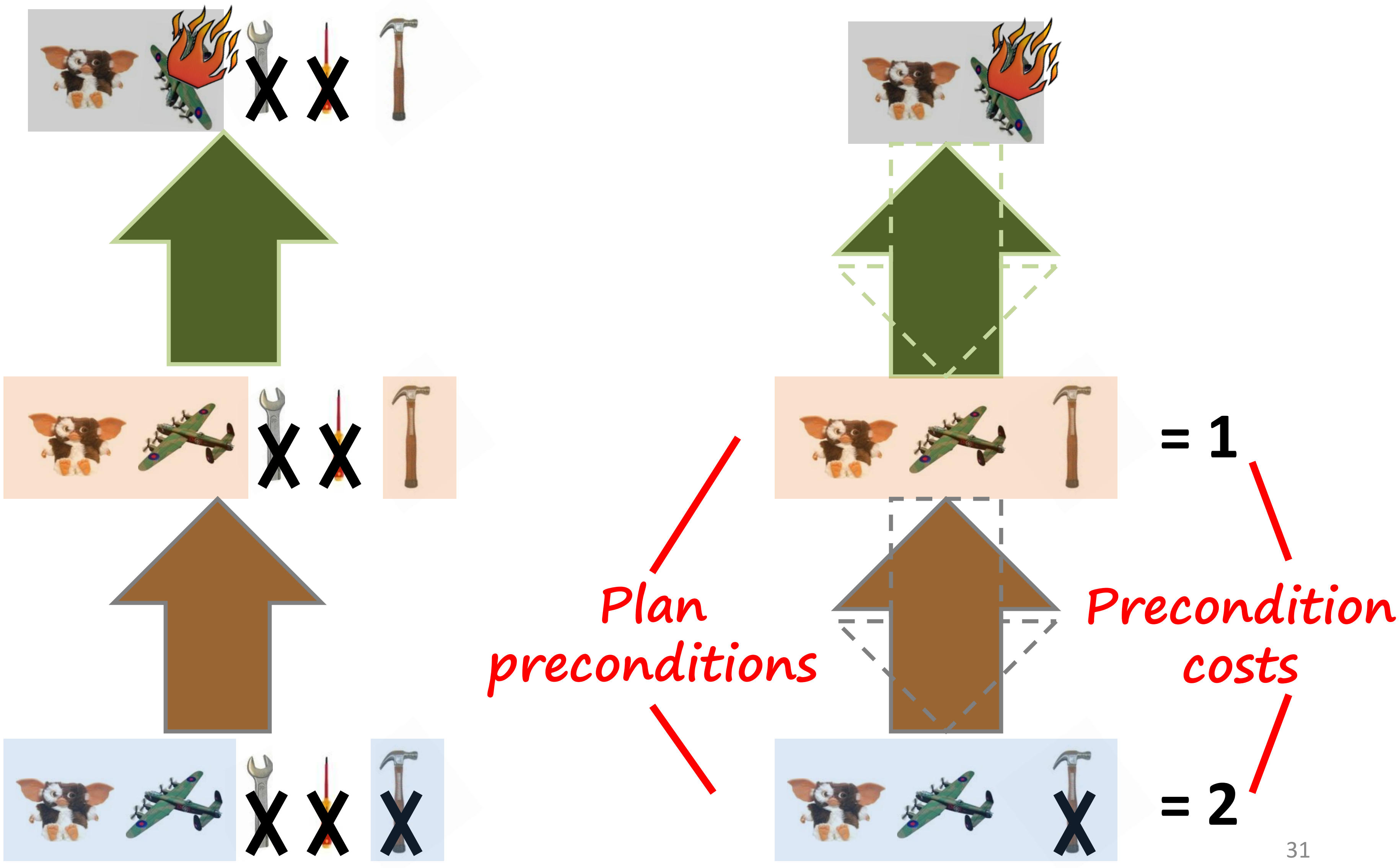
The GOTH Heuristic

[Kolobov, Mausam, Weld, 2010a]

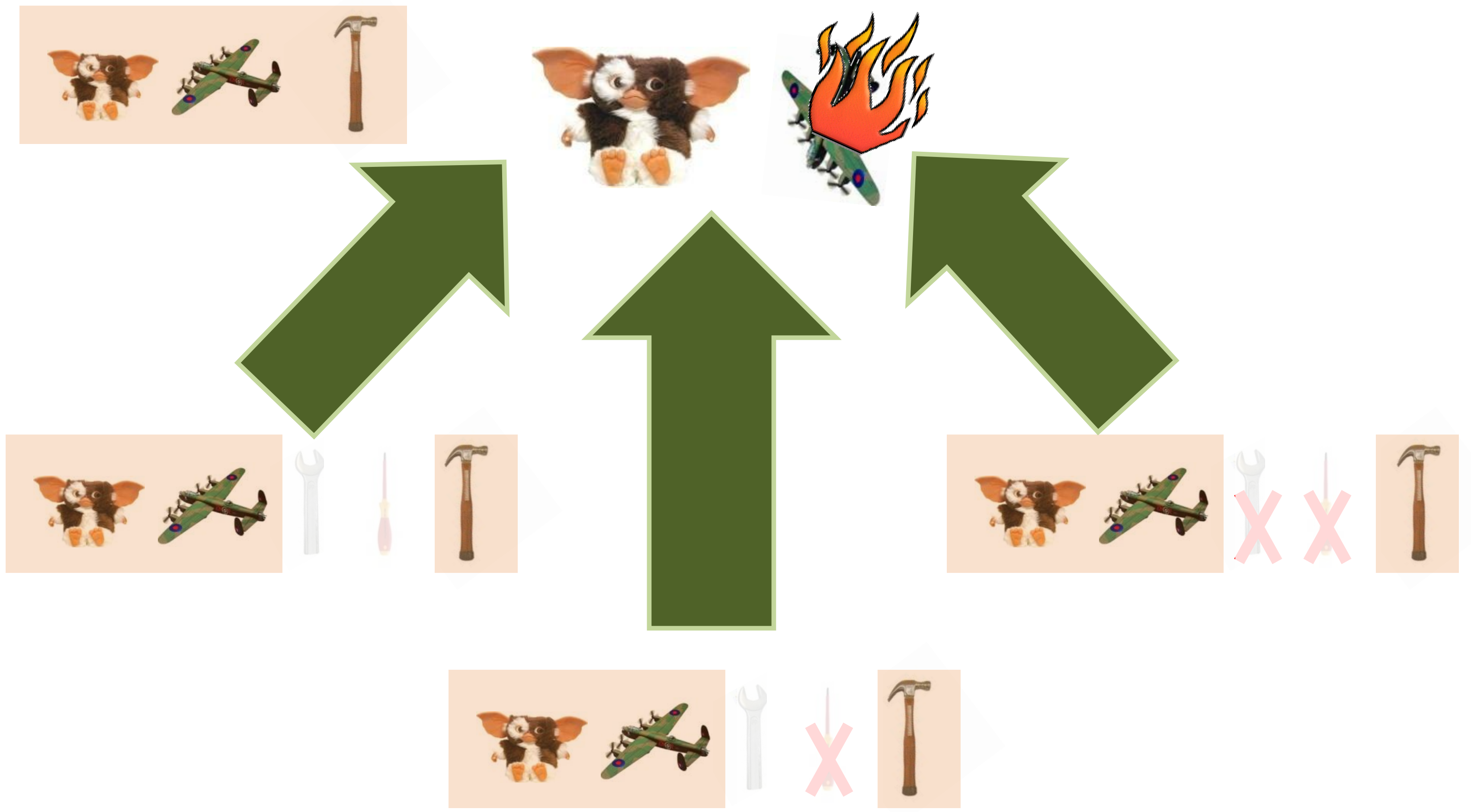
- Designed for MDPs at the start (not adapted classical)

- Motivation: would be good to estimate *h(s)* as cost of a non-relaxed deterministic goal plan from *s*
  - But too expensive to call a classical planner from every *s*
  - Instead, call from only a few *s* and generalize estimates to others

Regressing Trajectories

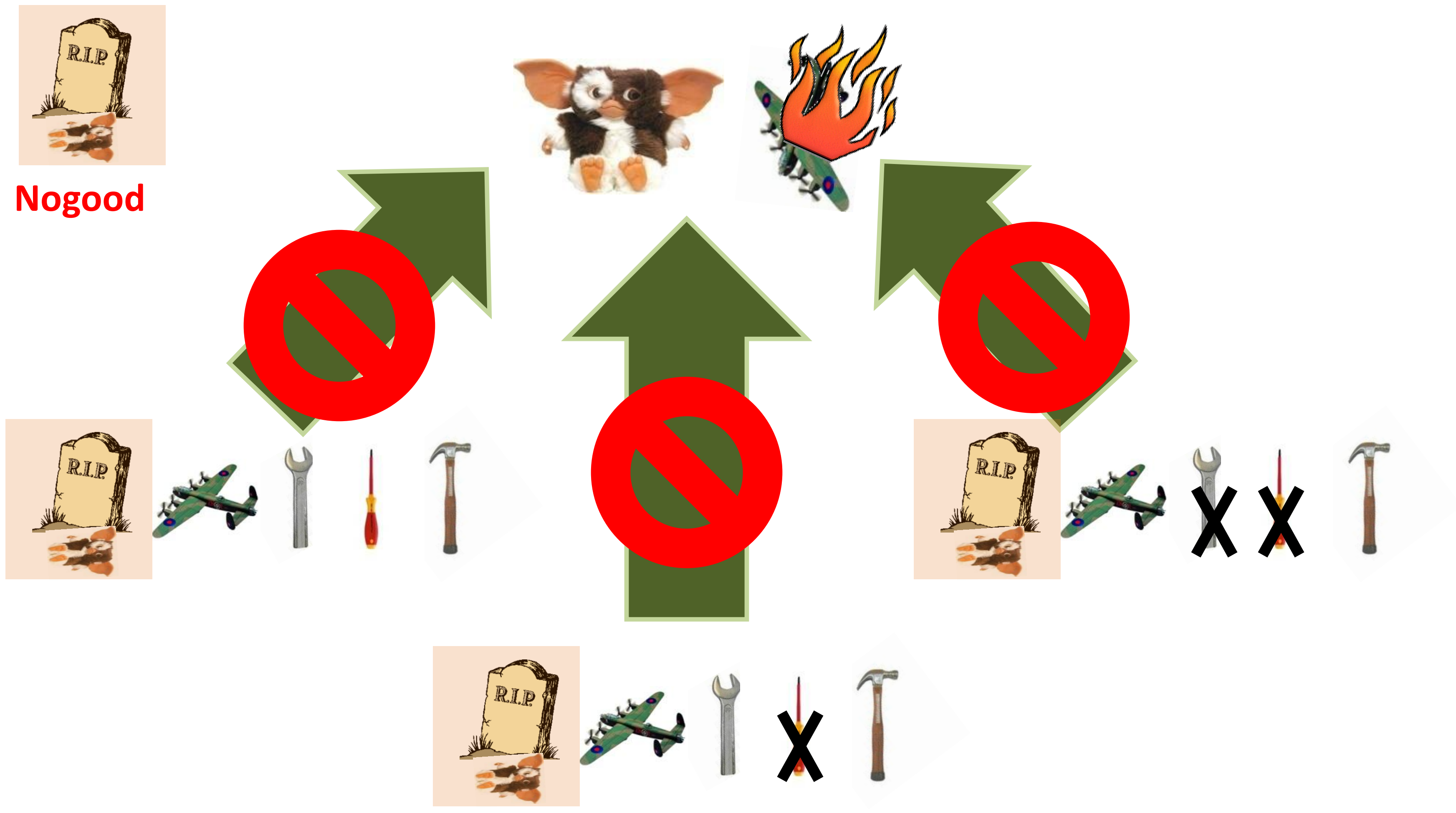


Plan Preconditions



Nogoods

[Kolobov, Mausam, Weld, 2010a]



Computing Nogoods

- Machine learning algorithm
  - Adaptively scheduled generate-and-test procedure
- Fast, sound
- Beyond the scope of this tutorial…

Estimating State Values

- Intuition
  - Each plan precondition cost is a “candidate” heuristic value
- Define $h_{GOTH}(s)$ as MIN of all available plan precondition values applicable in *s*
  - If none applicable in *s*, run a classical planner and find some
  - Amortizes the cost of classical planning across many states
- More informative than h_{FF}

Open Questions in Inadmissible HS

- **Still not clear when and why determinization-based inadmissible heuristics appear to work well**
  - Due to an experimental bias (MDPs with uniform action costs)?

- Need more research to figure it out…

Outline

- ✓ Motivation + Overview
- ✓ Determinization-based Algorithms
- ✓ Heuristic Search with Inadmissible Heuristics
- **Dimensionality Reduction**
- Monte-Carlo Planning
- Hierarchical Planning
- Hybridized Planning

Dimensionality Reduction: Motivation

- No approximate methods so far explicitly try to save space
- Dimensionality reduction attempts to do exactly that
  - Insight: V^* and π^* are functions of ~$|S|$ parameters (states)
  - Replace it with an approximation with r << $|S|$ params …
  - … in order to save space
- **How to do it?**
  - Obtain r basis functions b_i, let $V^*(s) \approx \sum_i w_i\, b_i(s)$, find w_i

ReTrASE

[Kolobov, Mausam, Weld, 2009]

- Derives *basis functions* from the all-outcomes determinization
  - one plan precondition in GOTH => one basis function
  - $b_p(s) = 1$ if plan precodition p holds in state s, ∞ otherwise

- Sets $V(s) = min_i\ w_i b_i(s)$

- Learns weights w_i by running modified Bellman backups

Approximate Policy Iteration

[Guestrin, Koller, Parr, Venkataraman, 2003]

- Reminder: Policy Iteration
  - Policy evaluation
  - Policy improvement

- Approximate Policy Iteration
  - Policy evaluation: compute the best linear approx. of V^{π}
  - Policy improvement: same as for PI

- Does API converge?
  - In theory, no; can oscillate if linear approx. for some policies coincide
  - In practice, usually, yes

Approximate Policy Iteration

- Let $V(s) \approx \sum_i w_i\, b_i(s)$, where b_i's are given by the user

- If a user gives a set B of basis functions, how do we pick $w_1, \ldots, w_{|B|}$ s.t. $|V^\pi - \sum_i w_i\, b_i|$ is the smallest?

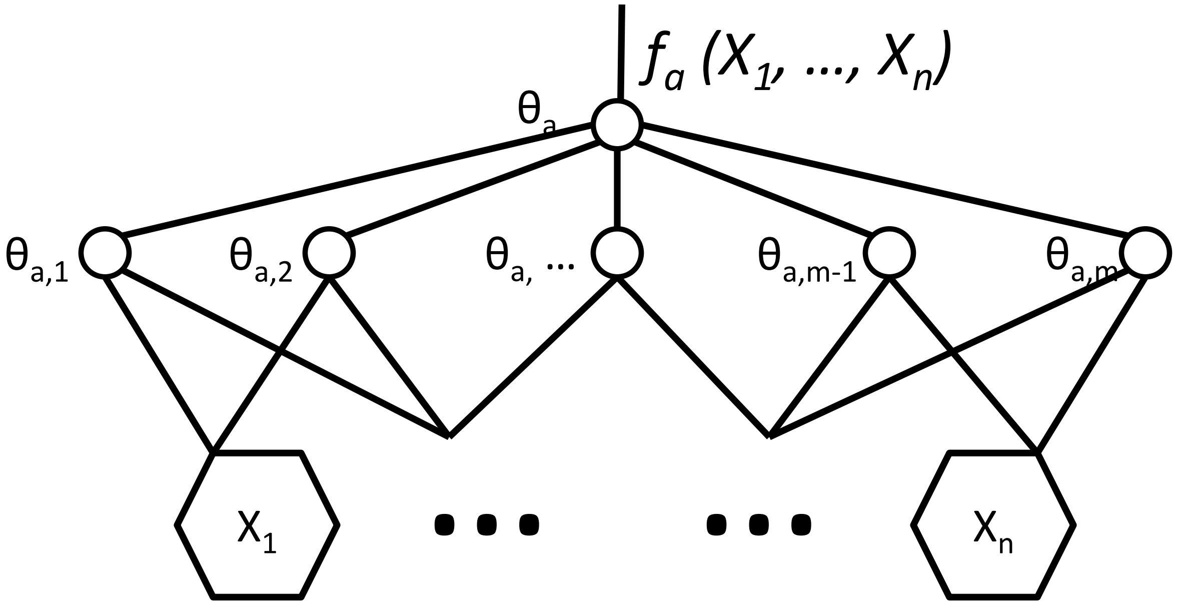
- Insight: assume each b depends on at most $z \ll |X|$ vars

- Then, can formulate LP with only $O(2^z)$ constraints to find w_i's
  - Much smaller than $|S| = O(2^{|X|})$

FPG

[Buffet and Aberdeen, 2006, 2009]

- **Directly learns a policy, not a value function**
- For each action, defines a *desirability function*



- Mapping from state variable values to action "quality"
  - Represented as a neural network
  - Parameters to learn are network weights $\theta_{a,1}, \ldots, \theta_{a,m}$ for each *a*

FPG

- Policy (distribution over actions) is given by a softmax

$$\pi_{\vec{\theta}}(s,a) = \frac{e^{f_{a|\vec{\theta}_a}(s)}}{\sum_{a' \in \mathcal{A}} e^{f_{a'|\vec{\theta}_{a'}}(s)}},$$

- To learn the parameters:
  - Run trials
  - After taking each action, compute the gradient w.r.t. weights
  - Adjust weights in the direction of the gradient
  - Makes actions causing expensive trajectories to be less desirable

Outline

- ✓ Motivation + Overview
- ✓ Determinization-based Algorithms
- ✓ Heuristic Search with Inadmissible Heuristics
- ✓ Dimensionality Reduction
- **Monte-Carlo Planning**
- Hierarchical Planning
- Hybridized Planning

Monte-Carlo Planning: Motivation

- Characteristics of the *Inventory Management* problem:
  - Enormous reachable state space
  - High-entropy *T* ($2^{|X|}$ outcomes per action, many likely ones)
    - Building determinizations can be super-expensive
    - Doing Bellman backups can be super-expensive

- Solution: ***Monte-Carlo Tree Search***
  - Does not manipulate *T* or *R* explicitly – no Bellman backups
  - Relies on a *world simulator* – indep. of MDP description size

UCT: An MCTS Algorithm

- UCT [Kocsis & Szepesvari, 2006]
  - Plans online finds best action for the current state
  - Similar to learning a policy for a multiarmed bandit

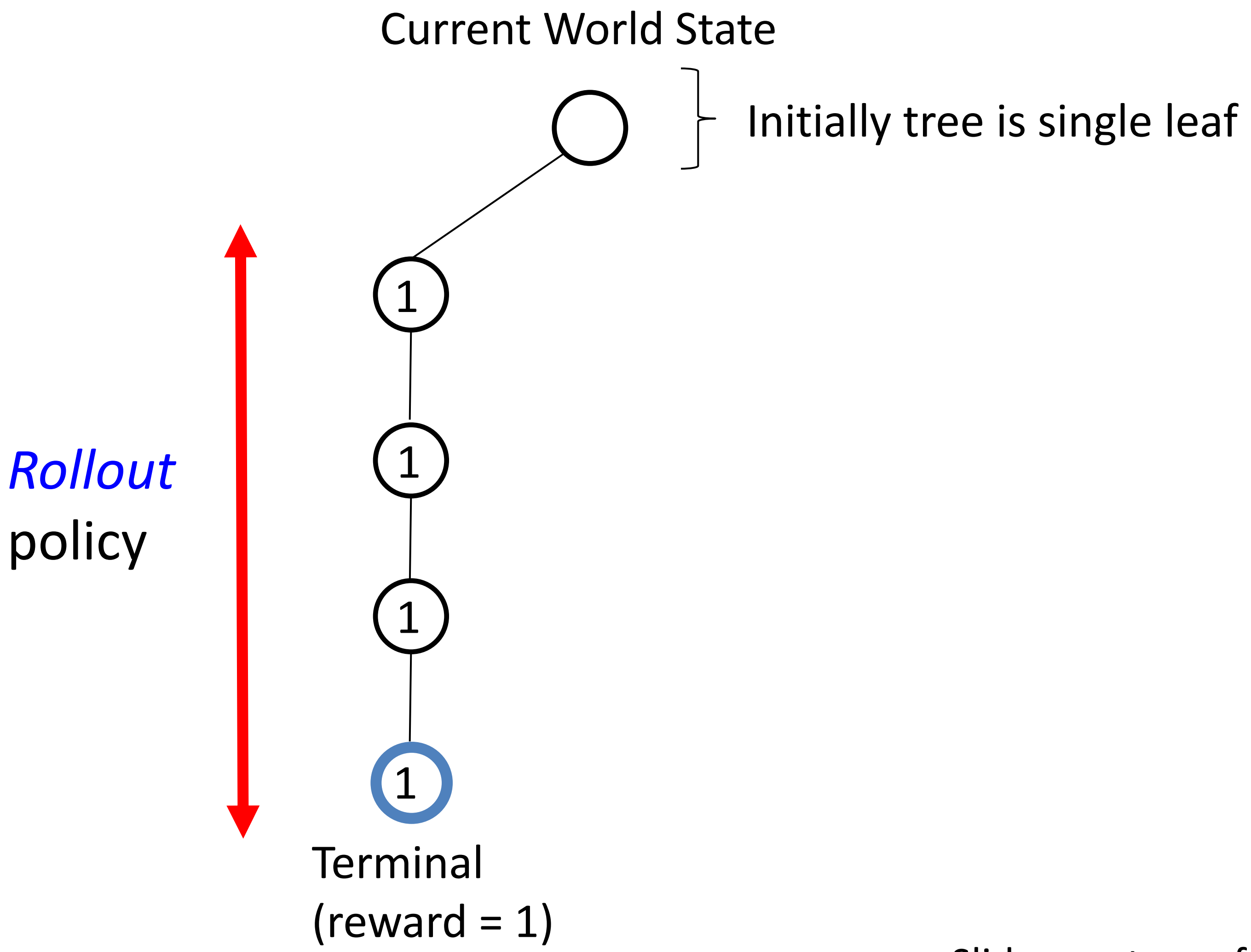
- Success stories:
  - Go (thought impossible in ‘05, human grandmaster level at 9x9 in ‘08)
  - Klondike Solitaire (wins 40% of games)
  - General Game Playing Competition
  - Real-Time Strategy Games
  - Probabilistic Planning Competition (-2011) [Keller & Eyerich, ICAPS’12]
  - The list is growing…

UCT: An MCTS Algorithm

- Rough idea:
  - Planning online, want to compute a policy for the current state
  - Have some “default” starting policy, the *rollout policy*
  - Repeatedly generate length-*L* trajectories starting at curr. state
  - Propagate reward along the trajectories
  - In the process, in states where you haven’t tried all actions, use the rollout policy
  - In states where you have tried all actions, use a *tree policy*
  - What is the rollout and the tree policy?

UCT Example

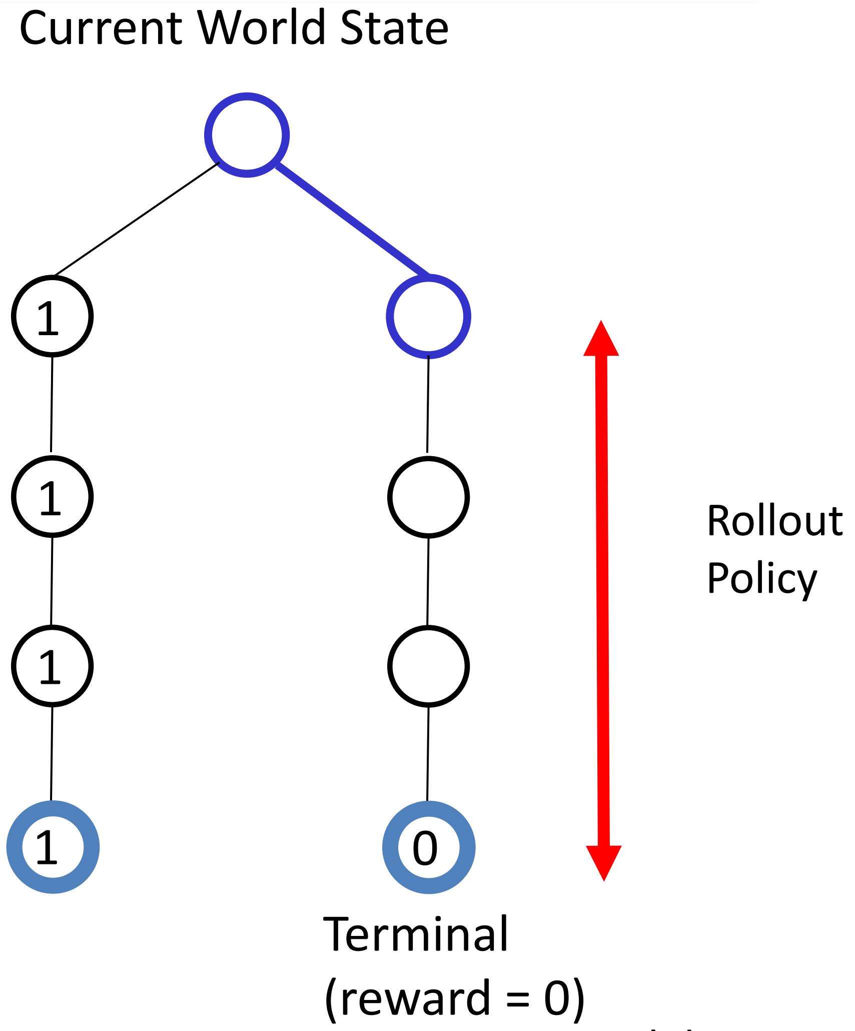
At a leaf node perform a random *rollout*



Slide courtesy of A. Fern

UCT Example

Must select each action at a node at least once

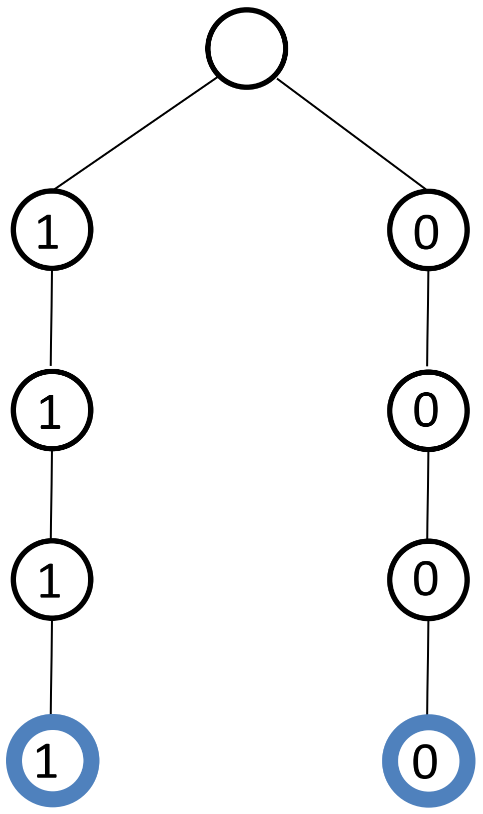


Slide courtesy of A. Fern

UCT Example

Must select each action at a node at least once

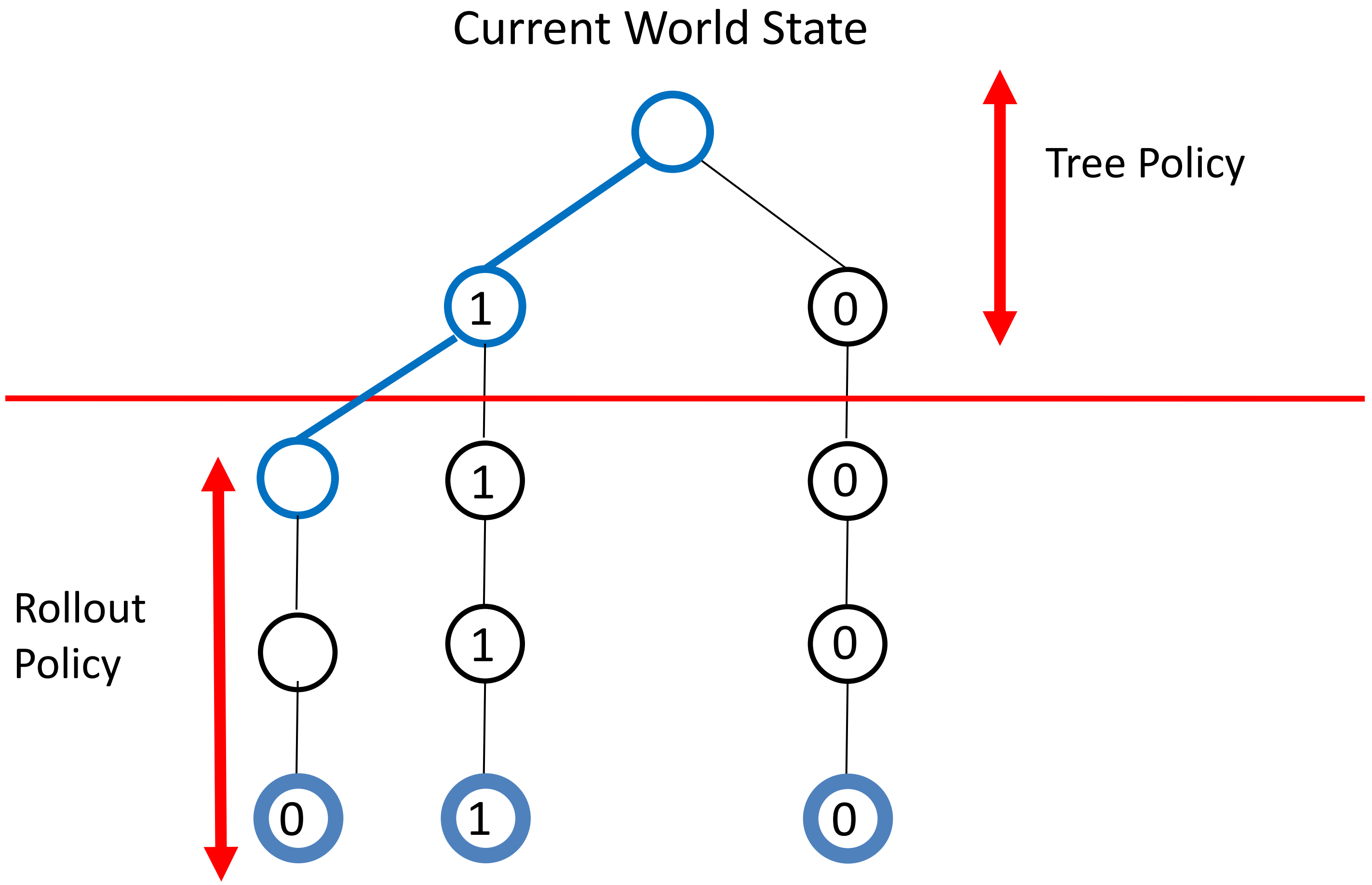
Current World State



Slide courtesy of A. Fern

UCT Example

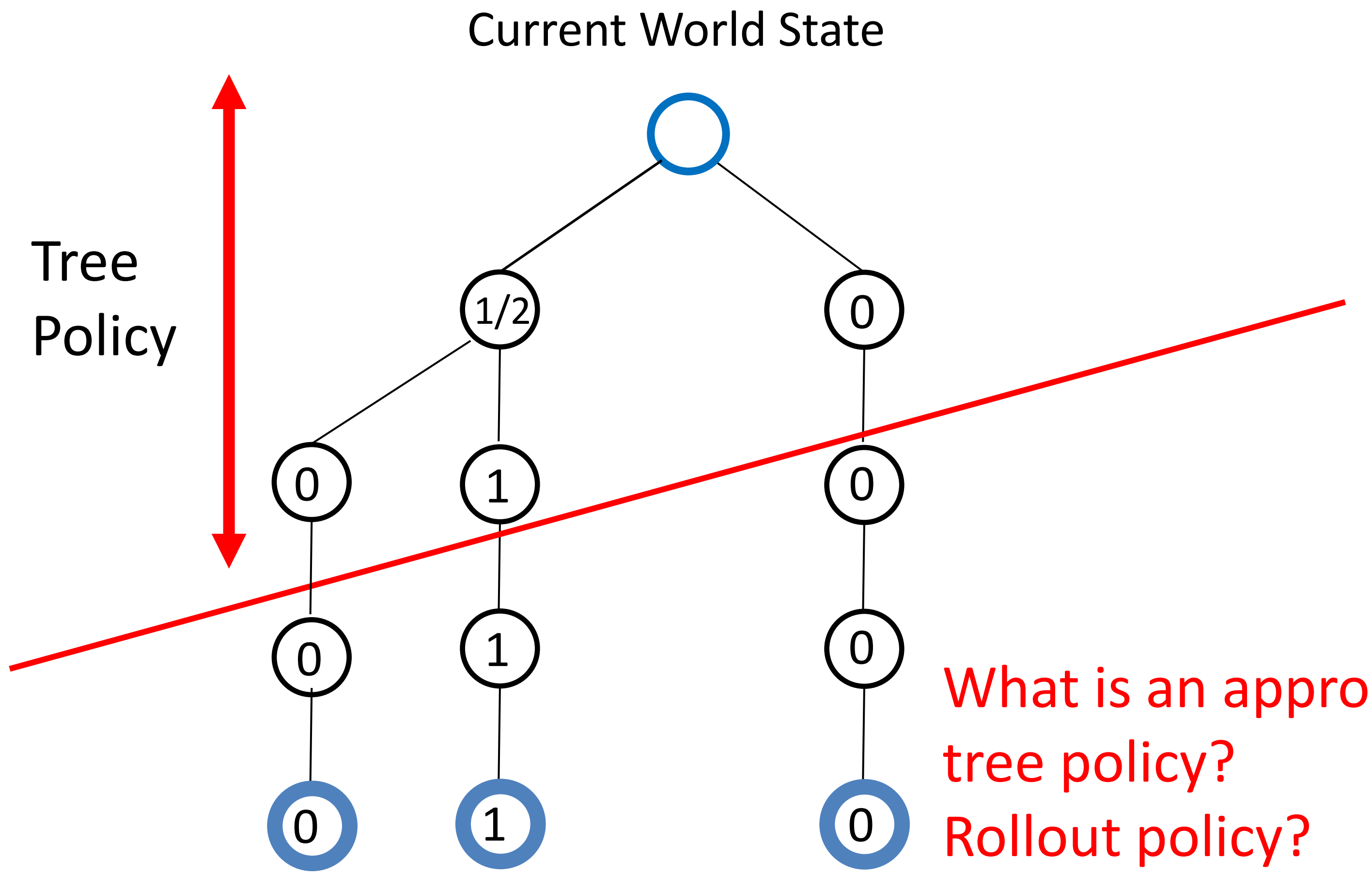
When all node actions tried once, select action according to tree policy



Slide courtesy of A. Fern

UCT Example

When all node actions tried once, select action according to tree policy



What is an appropriate tree policy?
Rollout policy?

Slide courtesy of A. Fern

UCT Details

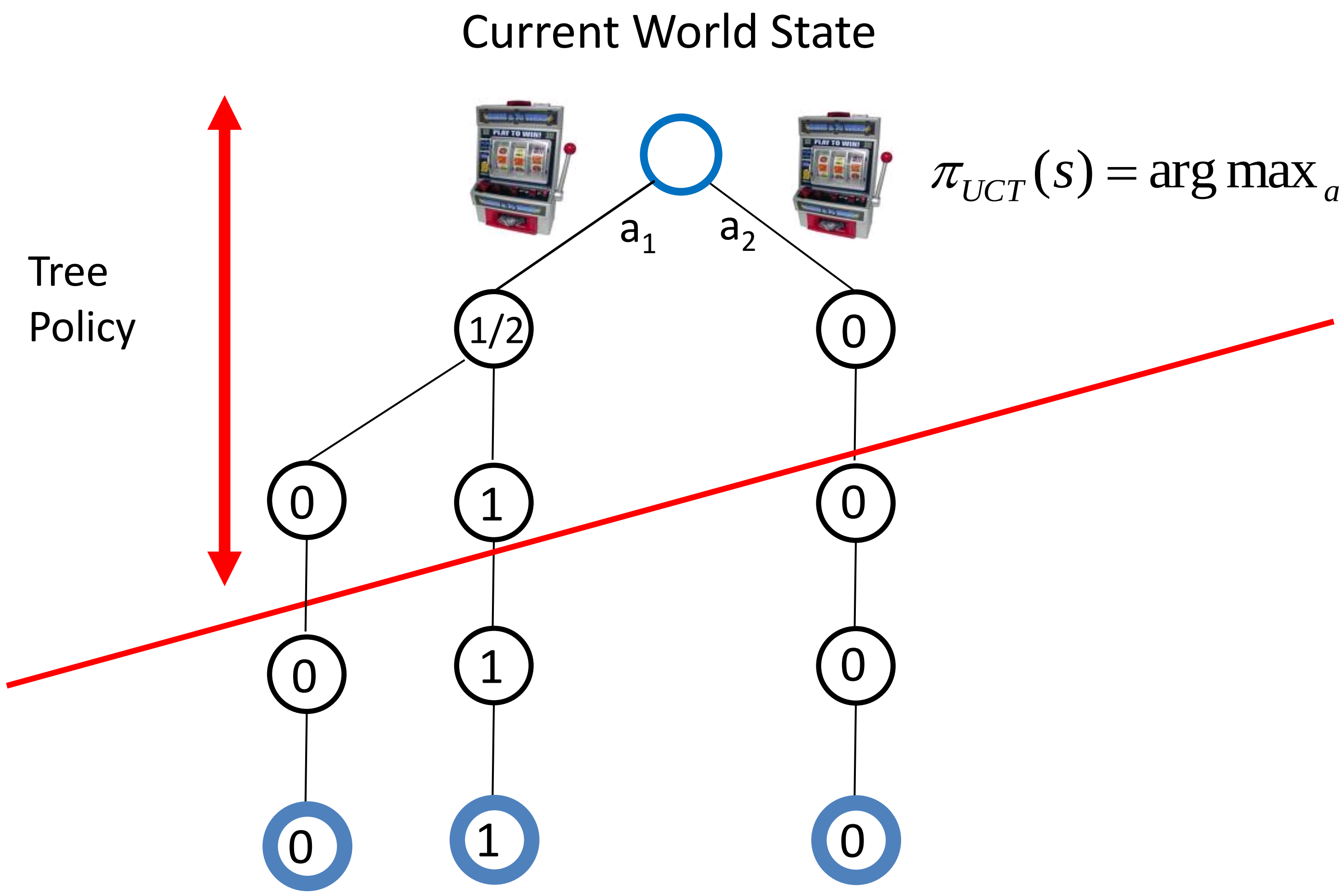
- Rollout policy:
  - Basic UCT uses random
- Tree policy:
  - Want to balance exploration and exploitation
  - Q(s,a) : average reward received in current trajectories after taking action a in state s
  - n(s,a) : number of times action a taken in s
  - n(s) : number of times state s encountered

$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + \underbrace{\boxed{c}\sqrt{\frac{\ln n(s)}{n(s,a)}}}_{\text{Exploration term}}$$

Theoretical constant that must be selected empirically in practice.

UCT Example

When all node actions tried once, select action according to tree policy



$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Slide courtesy of A. Fern

UCT Summary & Theoretical Properties

- The more simulations, the more accurate
  - Guaranteed to pick suboptimal actions exponentially rarely after convergence (under some assumptions)

- Possible improvements
  - Initialize the state-action pairs with a heuristic (need to pick a weight)
  - Think of a better-than-random rollout policy

- Best student paper at ICAPS'13 is on UCT!
  - Keller, Helmert, "Trial-Based Heuristic Tree Search for Finite-Horizon MDPs", presented on June, 12 after lunch

UCT Caveat

- Optimizes *cumulative regret*
  - Total amount of regret during state space exploration
  - Appropriate in RL

- In planning, *simple regret* is more appropriate
  - Regret during state space exploration is fictitious
  - Only regret of the final policy counts!

BRUE: Monte-Carlo *Planning*

[Feldman and Domshlak, 2012]

- Modifies UCT to minimize simple regret directly
- In UCT, simple regret diminishes at a polynomial rate in # samples
- In BRUE, simple regret diminishes at an exponential rate
- Simple modification to UCT
  - For every trajectory, sample a number K = [1,L]
  - Use random policy up to step K, tree or random afterwards

Outline

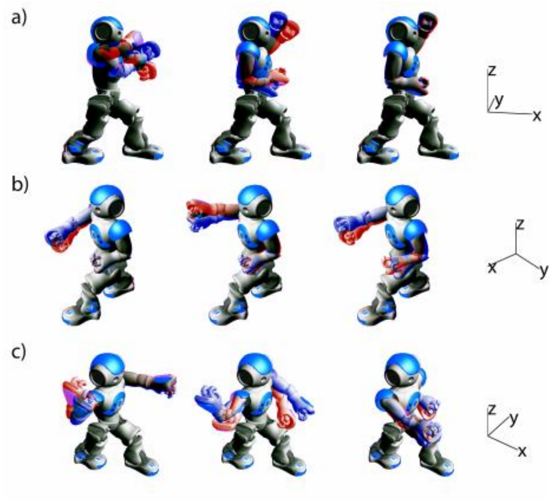
- ✓ Motivation + Overview
- ✓ Determinization-based Algorithms
- ✓ Heuristic Search with Inadmissible Heuristics
- ✓ Dimensionality Reduction
- ✓ Monte-Carlo Planning
- **Hierarchical Planning**
- Hybridized Planning

Hierarchical Planning: Motivation

- Some MDPs are too hard to solve w/o prior knowledge
  - Also, arbitrary policies for such MDPs may be hard to interpret
- Need a way to bias the planner towards “good” policies
  - And to help the planner by providing guidance
- **That’s what hierarchical planning does**
  - Given some prespecified (e.g., by the user) parts of a policy …
  - … planner “fills in the details”
  - Essentially, breaks up a large problem into smaller ones

Hierarchical Planning with Options

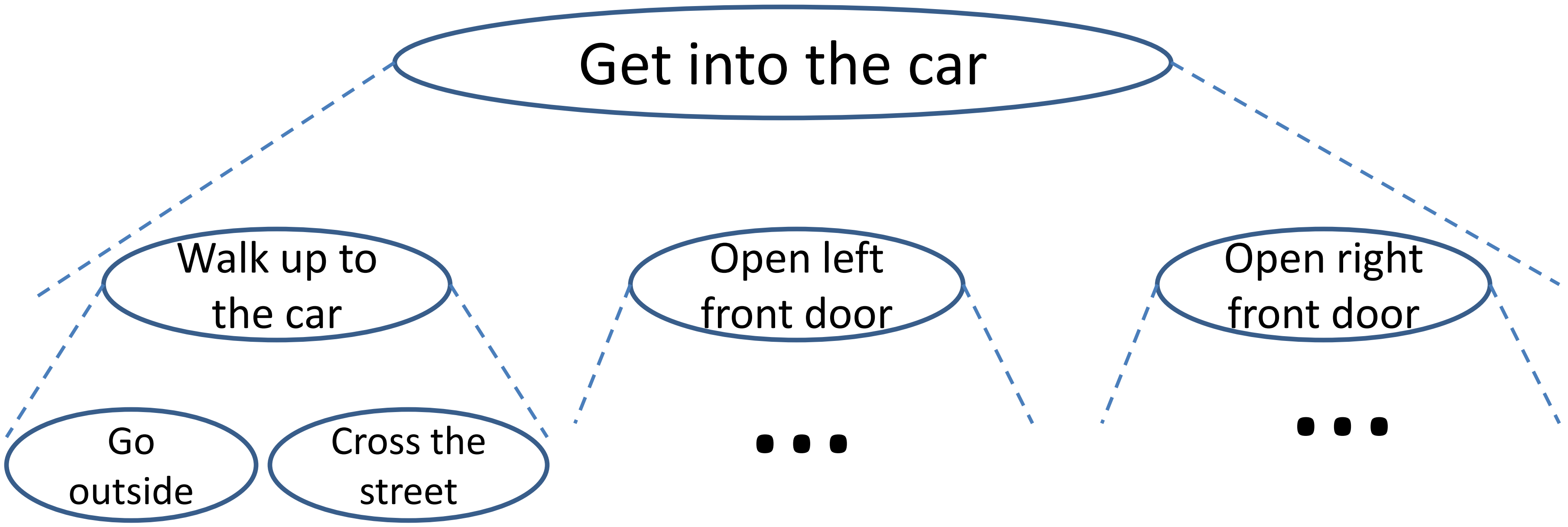
- Suppose you have precomputed policies (*options*) for some primitive behaviors of a robot



- Suppose you want to teach the robot how to dance
  - *You* give a hier. planner options for robot's primitive behaviors
  - *The planner* computes a policy for dancing that uses options as subroutines

Task Hierarchies

- The user breaks down a task into a hierarchy of subgoals



- The planner chooses which subgoals to achieve at each level, and how
  - Subgoals are just hints
  - Not all subgoals may be necessary to achieve the higher-level goal

Hierarchies of Abstract Machines (HAMs)

- More general hierarchical representation
- Each machine is a finite-state automaton w/ 4 node types

Execute action *a*

Call another machine *H*

Choose a machine from $\{H_1, \ldots, H_n\}$ and execute it

Stop execution/return control to a higher-level machine

- The user supplies a HAM
- The planner needs to decide what to do in *choice nodes*

Optimality in Hierarchical Planning

- Hierarchy constraints may disallow globally optimal π^*
- Next-best thing: a *hierarchically optimal policy*
  - The best policy obeying the hierarchy constraints
  - Not clear how to find it efficiently
- A more practical notion: a *recursively optimal policy*
  - A policy optimal at every hierarchy level, assuming that policies at lower hierarchy levels are fixed
  - Optimization = finding optimal policy starting from lowest level
- Hierarchically optimal doesn’t imply recursively optimal, and v. v.
  - But hierarchically optimal is always at least as good as recursively optimal

Learning Hierarchies

- Identifying useful subgoals
  - States in “successful” and not in “unsuccessful” trajectories
  - Such states are similar to *landmarks*

- Breaking up an MDP into smaller ones
  - State abstraction (removing variables irrelevant to the subgoal)

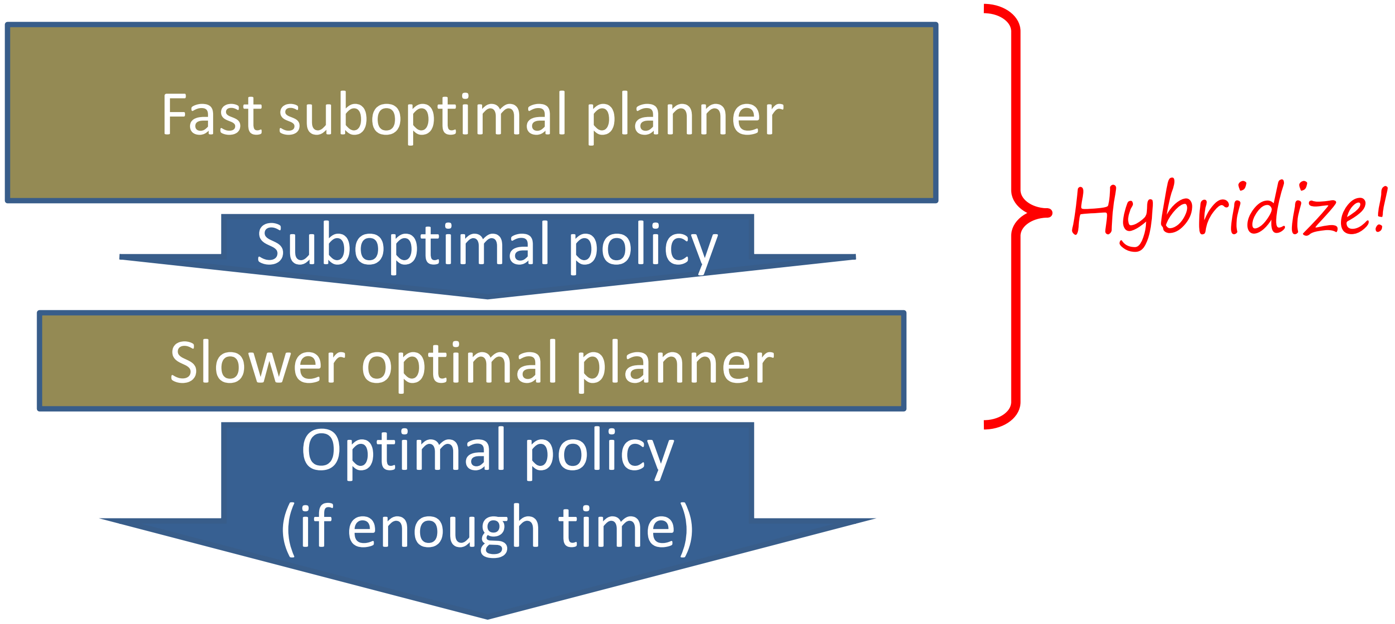
- Still very much an open problem

Outline

- ✓ Motivation + Overview
- ✓ Determinization-based Algorithms
- ✓ Heuristic Search with Inadmissible Heuristics
- ✓ Dimensionality Reduction
- ✓ Monte-Carlo Planning
- ✓ Hierarchical Planning
- **Hybridized Planning**

Hybridized Planning: Motivation

- Sometimes, need to arrive at a provably "reasonable" (but possibly suboptimal) solution ASAP



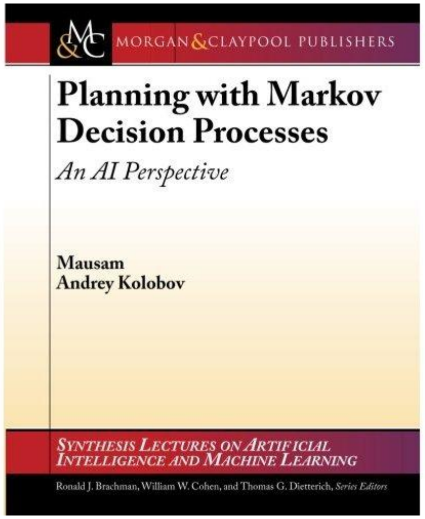
Hybridized Planning

[Mausam, Bertoli, Weld, 2007]

- **Hybridize MBP and LRTDP**
- MBP is a non-deterministic planner
  - Gives a policy guaranteed to reach the goal from everywhere
  - Very fast
- LRTDP is an optimal probabilistic planner
  - Amends MBP's solution to have a good expected cost
- **Optimal in the limit, produces a good policy quickly**

Summary

- Surveyed 6 different approximation families
  - Determinization-based
  - Inadmissible heuristic search
  - Dimensionality reduction
  - Monte-Carlo planning
  - Hierarchical planning
  - Hybridized planning



MORGAN & CLAYPOOL PUBLISHERS

Planning with Markov Decision Processes

An AI Perspective

Mausam
Andrey Kolobov

SYNTHESIS LECTURES ON ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Ronald J. Brachman, William W. Cohen, and Thomas G. Dietterich, *Series Editors*

- Address different "difficult" aspects of MDPs

- Takeaway: no silver bullet