Exploiting Expert Knowledge in Factored POMDPs

Felix Müller, Christian Späth,
Thomas Geier, and Susanne Biundo

Institute of Artificial Intelligence

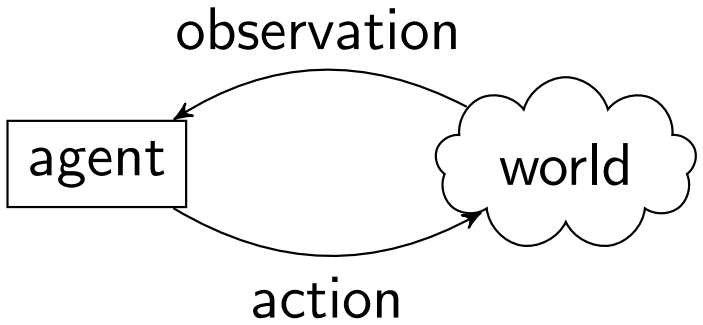
August 29th, 2012

ulm university universität ulm

Introduction

What is a POMDP? Partially Observable Markov Decision Process:

- formalism for probabilistic planning under partial observability
- acting in POMDP domains: execute action, receive observation, update state estimation, execute action, ...



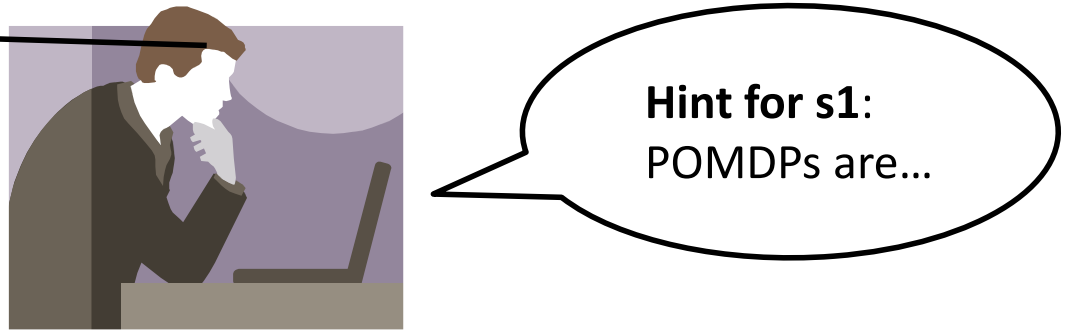
- $T(s, a, s') = Pr(s'|a, s)$: probability for ending up in s' after executing a in s
- $Z(s', a, o) = Pr(o|a, s')$: probability for observing o after executing a and ending up in s'
- execution terminates after fixed number of executed actions

Example

Skill Teaching domain from IPPC 2011:

- teach series of interdependent skills to a student
- skill proficiency levels low, medium, high (not observed)
- available actions increase skill proficiency:
  - give hint (only at low proficiency): giveHint(?s)
  - ask multiple choice question: askProb(?s)
  - only work if prerequisite skills are at high proficiency
- observations: question answered correctly? answeredRight(?s)

Skill	**Proficiency**
s1	Low
s2	High
s3	Medium



Rewards and Policies

Determining appropriate actions:

- $R(s, a)$: immediate cost of executing a in s
  $\rightarrow$ example: penalty not yet learned skills
- goal: minimize expected accumulated cost
  $\rightarrow$ example: raise all skill levels to high proficiency a.s.a.p.
- $b_0(s)$: initial belief (probability distribution over states)
- policy $\pi : B \rightarrow A$ maps belief states to actions

Motivation

Motivation for our approach:

- POMDPs model many interesting problems (elevator control, traffic control, user-centered planning, . . .)
- in practice currently: hand-crafted solutions from domain experts
- idea: exploit expert knowledge in planning!
- how? adapt HTN (hierarchical task network) planning:
  - exploits expert knowledge through action hierarchies
  - successfully applied in practice (evacuation planning, material selection for manufacturing, . . .)[1]
  - for deterministic domains

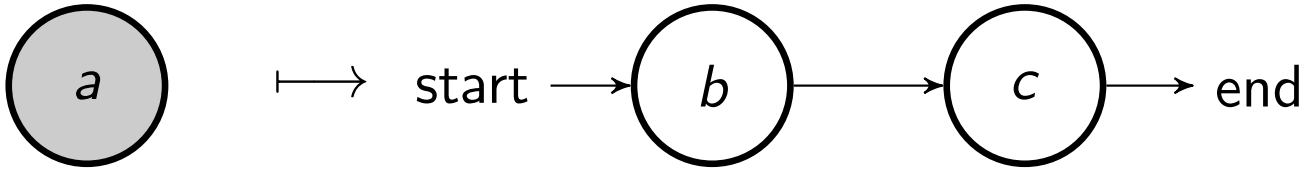
→ generalize HTN planning to POMDPs

[1] Nau et al., Applications of SHOP and SHOP2. IEEE Int. Systems, 2004.

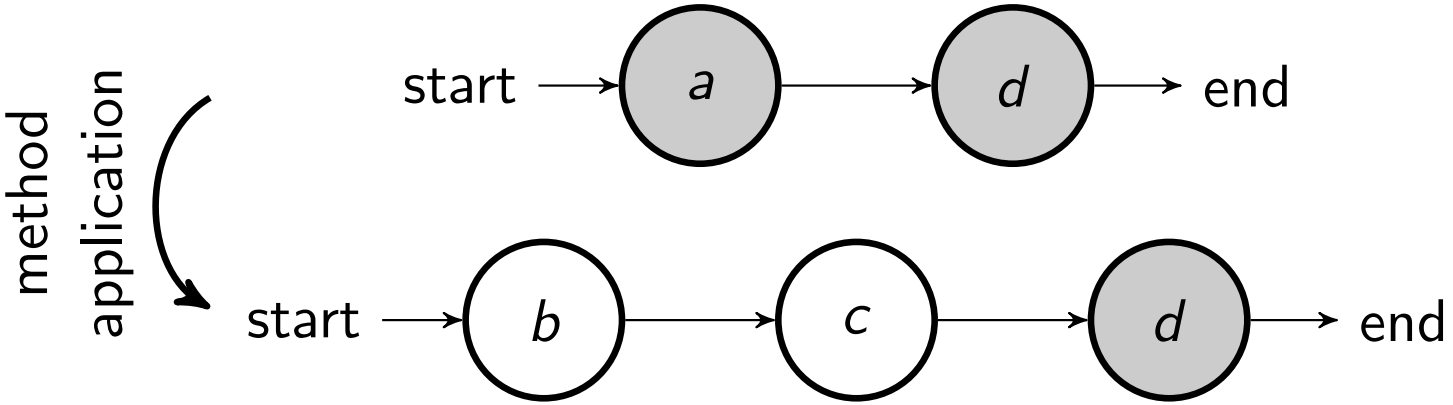
HTN Planning at a Glance

Hierarchical Task Network planning (simplified):

- plans in deterministic domain: sequences of actions
- action hierarchies
  - abstract actions: higher-level abstraction of tasks
  - methods: abstract action $\mapsto$ sequence of actions



- plan construction: refine initial abstract plan into fully primitive plan



Suitable Policy Representation

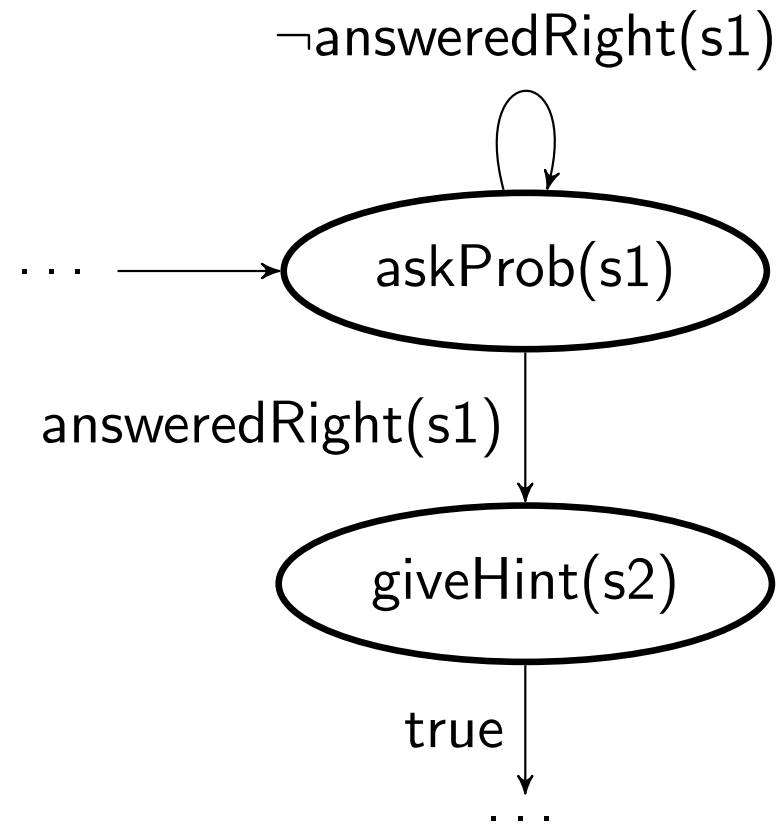
Logical Finite State Controller (LFSC) policy representation

Structure:

- directed graph
- node labels: actions
- edge labels: observation formulas

Execution:

- execute action of current node
- receive observation (= truth assignment to observation variables)
- follow edge where received observation fulfills formula
- repeat



HTN + POMDPs

Combining HTN and POMDPs:

HTN planning:

- plans: sequences of actions
- methods: abstract action $\mapsto$ sequence of actions

Intuition for HTN in POMDPs:

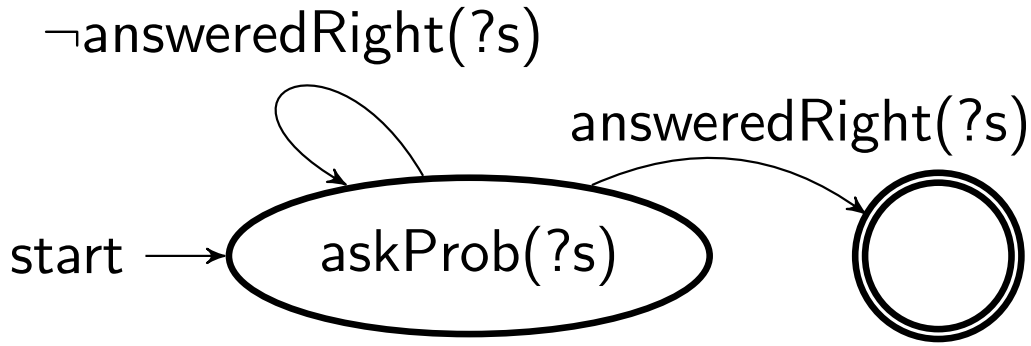
- plans (resp. policies): LFSCs
- methods: abstract action $\mapsto$ LFSC

Question: how does method application work?

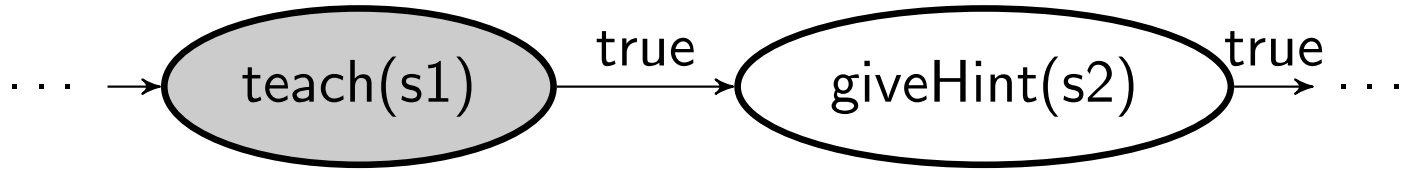
- applying methods requires implementing controllers to have start and end points
- *method controllers*

Method Application Example

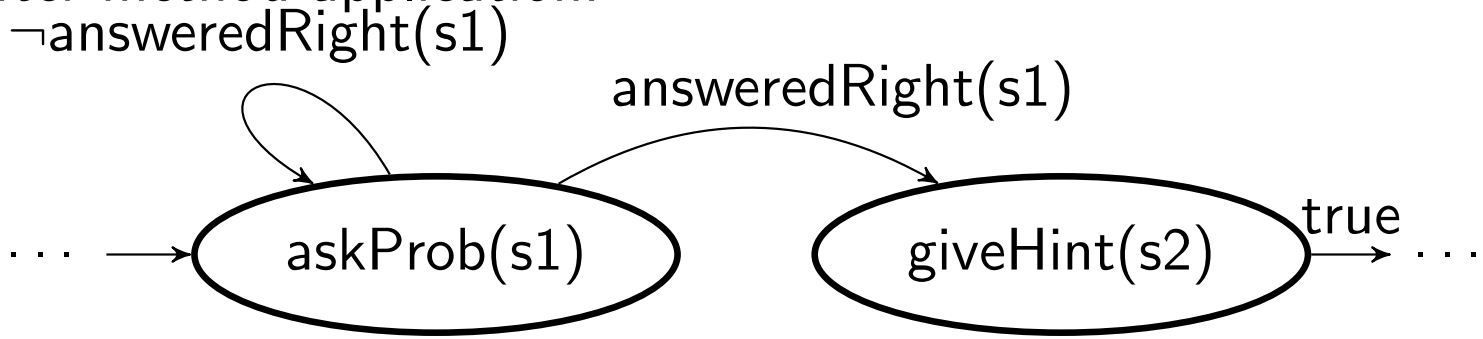
- method for abstract action teach(?s):



- controller containing abstract action teach(s1):



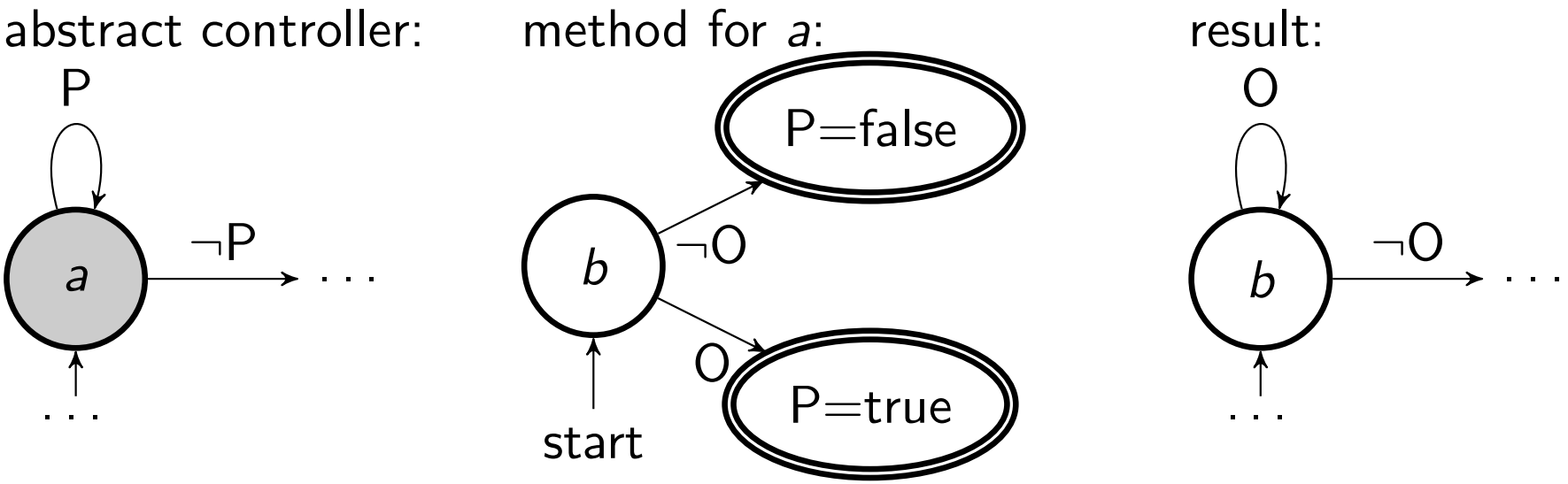
- after method application:



Abstract Observations and Method Application

Passing on information gleaned during method execution:

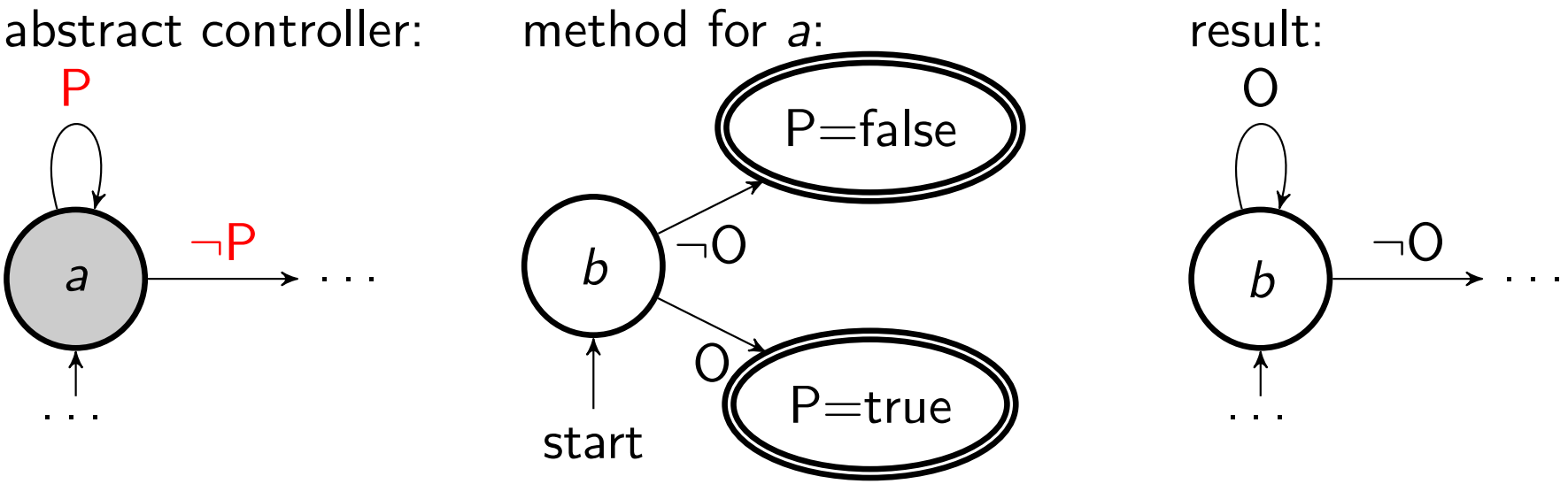
- abstract observation variables associated with abstract actions
- method controllers have multiple terminal nodes labeled with abstract observations
- method application: match terminal node labels and abstract observation formulas



Abstract Observations and Method Application

Passing on information gleaned during method execution:

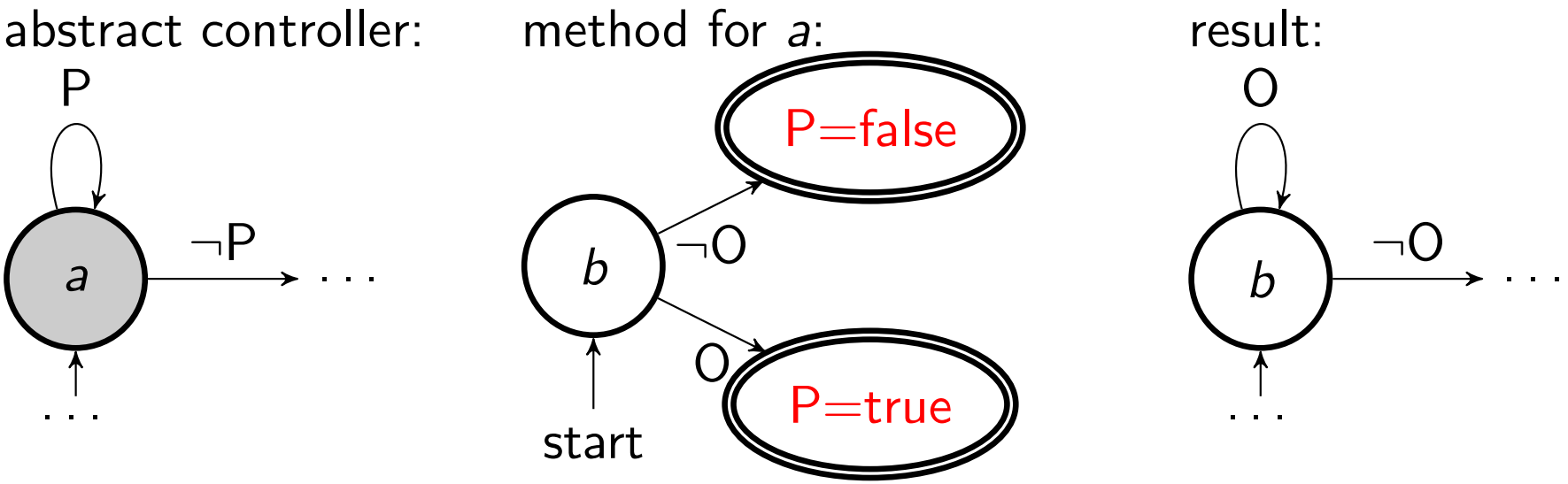
- abstract observation variables associated with abstract actions
- method controllers have multiple terminal nodes labeled with abstract observations
- method application: match terminal node labels and abstract observation formulas



Abstract Observations and Method Application

Passing on information gleaned during method execution:

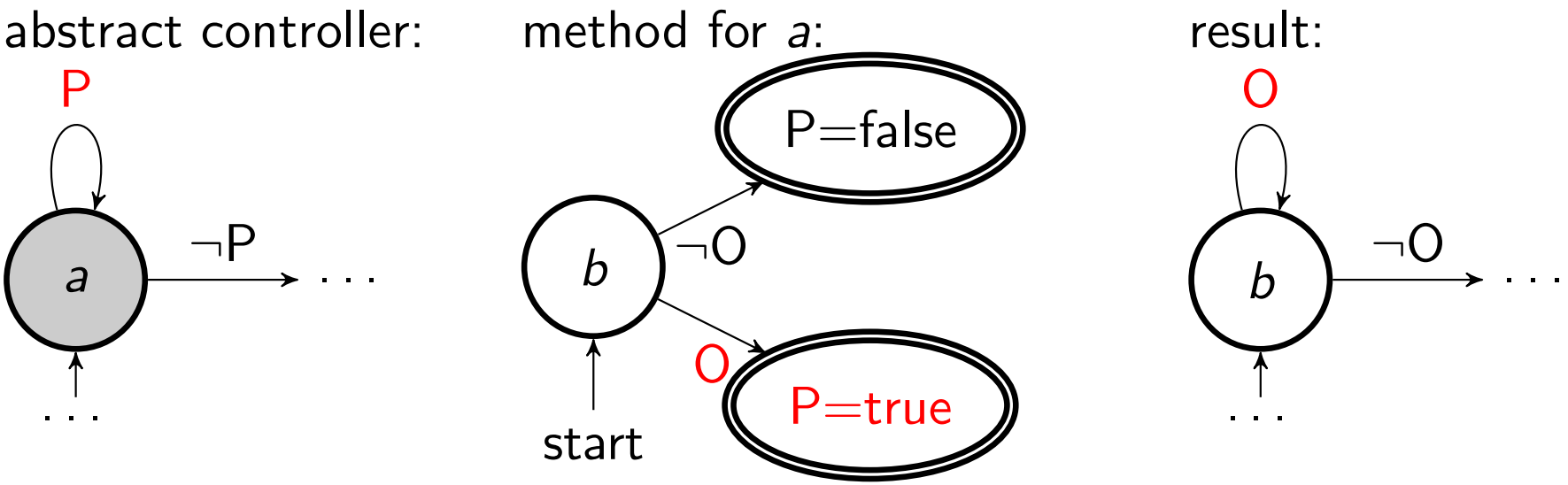
- abstract observation variables associated with abstract actions
- method controllers have multiple terminal nodes labeled with abstract observations
- method application: match terminal node labels and abstract observation formulas



Abstract Observations and Method Application

Passing on information gleaned during method execution:

- abstract observation variables associated with abstract actions
- method controllers have multiple terminal nodes labeled with abstract observations
- method application: match terminal node labels and abstract observation formulas



Planning

Defining hierarchies:

- language: extension of RDDL (Relational Dynamic Influence Diagram Language)
- model hierarchy for domain
- model initial controller for problem instance

Planning in hierarchical POMDPs:

- planning problem: RDDL domain + action hierarchy + RDDL problem instance + initial controller
- solution: least-cost primitive controller that can be generated

Algorithms I

First applied algorithm: A*

- search in policy space
  - search nodes represent controllers
  - start at initial controller, apply methods to find good primitive controller
- prerequisite: algorithm to compute controller value
  - factored POMDP: enumerating states and observations infeasible
  - calculation directly on factored representation using Algebraic Decision Diagrams (ADDs)
- path costs $g(C)$ = "guaranteed" costs of C
- heuristic $h(C)$ = estimates for abstract actions in C

Algorithms II

Second applied algorithm: UCT (Upper Confidence Bound applied to Trees)

- instance of Monte Carlo Tree Search (anytime algorithm)
- incrementally expand search tree in memory by sampling search space
- also search in policy space
- drawing a sample: apply methods until primitive, simulate execution once
  - remember number of times node was visited
  - remember average value of node
  - use node information for generating next sample

Experiments

Setting:

- hierarchies for Elevator, Skill Teaching, and Navigation domains from 2011 IPPC (24 instances in total)
- A*, UCT with 10s runtime, Symbolic Perseus
- 2 hours timeout

Results:

- Symbolic Perseus: returned policies for 6 of 24 instances
- A*: returned policies for only 3 of 24 instances ($\rightarrow$ time-consuming policy evaluation)
- UCT: returned policies for all instances
  - better policies than Symbolic Perseus
  - 8.4% of the runtime of Symbolic Perseus on average
- caveat: inherently limited comparability

Conclusion and Future Work

Presented approach:

- generalization of HTN planning to partial observability
- RDDL extension for modeling hierarchies
- applied A* and UCT
- performance compares favorably to Symbolic Perseus

Future work:

- factored controller state: higher flexibility

Experiments: Skill Teaching

Cells:

- Upper value: runtime in milliseconds
- lower value: plan quality (accumulated reward)

Instance	A*	UCT 10s	Sym. Perseus	Blind
instance 1	timeout n/a	12,559 48.699	26,943 -16.039	n/a 25.937
instance 2	timeout n/a	10,616 51.774	25,906 20.193	n/a 26.551
instance 3	timeout n/a	11,181 28.263	timeout n/a	n/a -86.6
instance 4	timeout n/a	12,855 -3.646	timeout n/a	n/a -111.645
instance 5	timeout n/a	14,019 -46.2	memout n/a	n/a -247.745
instance 6	timeout n/a	14,812 -106.799	memout n/a	n/a -292.613
instance 7	timeout n/a	18,693 -181.782	memout n/a	n/a -589.307
instance 8	timeout n/a	16,516 -202.218	memout n/a	n/a -537.33
instance 9	timeout n/a	19,070 -270.603	memout n/a	n/a -602.724
instance 10	timeout n/a	25,135 -307.424	memout n/a	n/a -542.868

Experiments: Elevator

Cells:

- Upper value: runtime in milliseconds
- lower value: plan quality (accumulated reward)
- other 6 instances had parallel actions

Instance	A*	UCT 10s	Sym. Perseus	Blind
instance 1	timeout n/a	10,871 -32.580	651,885 -35.418	n/a -44.363
instance 4	timeout n/a	12,184 -74.256	timeout n/a	n/a -88.982
instance 7	timeout n/a	14,486 -113.247	timeout n/a	n/a -133.809
instance 10	timeout n/a	23,926 -148,802	timeout n/a	n/a -177.855

Experiments: Navigation

Cells:

- Upper value: runtime in milliseconds
- lower value: plan quality (accumulated reward)

Instance	A*	UCT 10s	Sym. Perseus	Blind
instance 1	5,597 -10.165	11,669 -10.165	104,042 -40 (-12,355)	n/a -38.605
instance 2	15,936 -10.781	12,269 -10.781	302,217 -40 (-11,696)	n/a -39.178
instance 3	258,135 -12.457	11,799 -12.457	5,824,802 -40 (-14,314)	n/a -39.833
instance 4	memout n/a	11,361 -15.745	memout n/a	n/a -39.991
instance 5	timeout n/a	12,752 -17.924	memout n/a	n/a -39.839
instance 6	memout n/a	13,746 -19.978	memout n/a	n/a -39.995
instance 7	memout n/a	12,057 -22.385	memout n/a	n/a -39.996
instance 8	memout n/a	14,789 -32.042	memout n/a	n/a -39.87
instance 9	memout n/a	13,709 -33.228	memout n/a	n/a -39.997
instance 10	memout n/a	14,968 -33.511	memout n/a	n/a -40

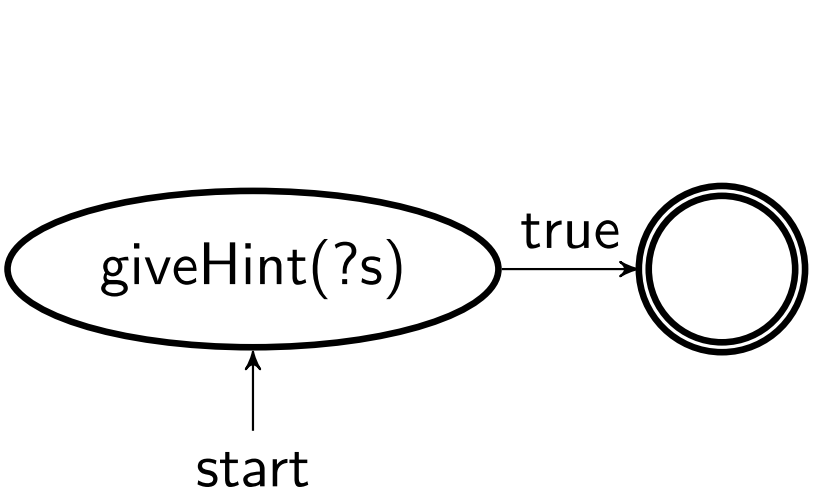
Why Symbolic Perseus?

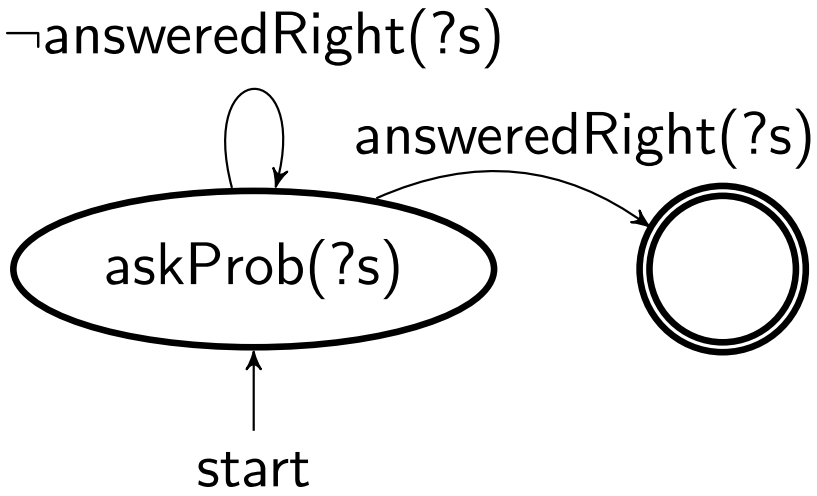
Why no comparison to IPPC 2011 winners?

- two planners scored better than Symbolic Perseus
- competition mode allowed authors interaction with planners
  - parameter tuning
  - manual stopping
  - arbitrary time allocation
- $\Rightarrow$ Problem: difficult to reproduce results
  - should we also tune parameters of other planners?
  - should we also manually stop planners?
  - time allocation?
- Symbolic Perseus usable out of the box

Complete Hierarchy for Skill Teaching

- methods for teach(?s):





- methods for teachAll: