First-order MDPs

Motivation

Scott Sanner
NICTA / ANU

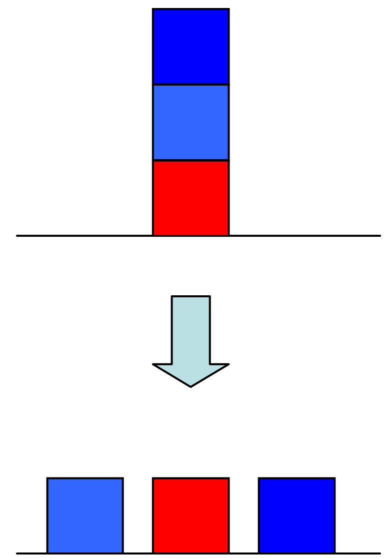
FOMDP Tutorial Outline

- **Motivation**
- Deductive First-order Planning in the Situation Calculus
- FOMDPs and Symbolic Dynamic Programming
- Potential Caveats

Planning Languages

- Common languages:
  - STRIPS
  - PDDL
    - more expressive than STRIPS
    - for example, *universal* and *conditional* effects:

(:action put-all-blue-blocks-on-table
:parameters ( )
:precondition ( )
:effect (forall (?b)
(when (Blue ?b)
(not (OnTable ?b)))))



  - General Game Playing (GGP)
    - one or more agents

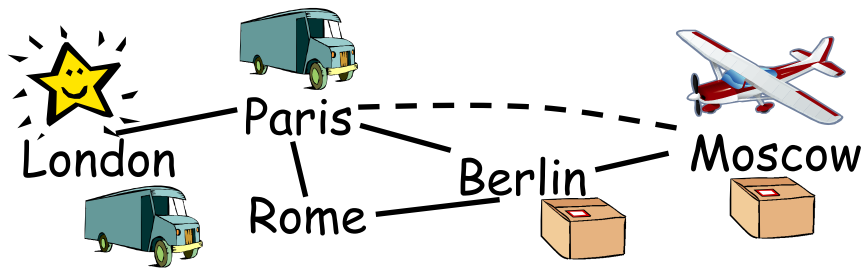
Benefits of Relational Languages

- STRIPS, PDDL, GGP are relational languages…
  - Refer to relational *fluents:*
    - e.g., *BIn(?b,?c), OnTable(?b)*
    - specify relations between objects
    - change over time
  - Use first-order logic to specify…
    - action preconditions
    - action effects
    - goals / rewards
      - e.g., *(forall (?b ?c) ((Destination ?b ?c)* $\Rightarrow$ *(Bin ?b ?c)))*
  - Are *domain-independent* and often compact!

How to Solve?

- Relational planning *problem*:

BoxWorld:




```
(:action  load-box-on-truck-in-city
          :parameters  (?b - box ?t - truck ?c – city)
          :precondition  (and (BIn ?b ?c) (TIn ?t ?c))
          :effect  (and (On ?b ?t) (not (BIn ?b ?c))))
```

- Solve *ground problem* for *each domain instance*?
  - 3 trucks:  2 planes:  3 boxes: 

- Or solve lifted specification for *all* domains at once?

Full Specification: BoxWorld

- **Relational Fluents:** $BoxIn(Box, City)$, $TruckIn(Truck, City)$, $BoxOn(Box, Truck)$
- **Goal:** $[\exists Box : b.BoxIn(b, paris)]$
- **Actions:**
  - $load(Box : b, Truck : t)$:
    - Effects:
      - $when\ [\exists City : c.\ BoxIn(b, c) \wedge TruckIn(t, c)]\ then\ [BoxOn(b, t)]$
      - $\forall City : c.\ when\ [BoxIn(b, c) \wedge TruckIn(t, c)]\ then\ [\neg BoxIn(b, c)]\}$
  - $unload(Box : b, Truck : t)$:
    - Effects:
      - $\forall City : c.\ when\ [BoxOn(b, t) \wedge TruckIn(t, c)]\ then\ [BoxIn(b, c)]$
      - $when\ [\exists City : c.\ BoxOn(b, t) \wedge TruckIn(t, c)]\ then\ [\neg BoxOn(b, t)]$
  - $drive(Truck : t, City : c)$:
    - Effects:
      - $when\ [\exists City : c_1.\ TruckIn(t, c_1)]\ then\ [TruckIn(t, c)]$
      - $\forall City : c_1.\ when\ [TruckIn(t, c_1)]\ then\ [\neg TruckIn(t, c_1)]$

Solving Ground BoxWorld

- Apply planner to BoxWorld grounded w.r.t. domain, e.g.,

#states exponential in #state-vars!

- **Domain Object Instantiation**:
  - $Box = \{box_1, box_2, box_3\}$, $Truck = \{truck_1, truck_2\}$, $City = \{paris, berlin, rome\}$
- **Ground Fluents (i.e., binary state variables)**:

  Exponential #state-vars in arity

  - $BoxIn$: $\{BoxIn(box_1, paris), BoxIn(box_2, paris), BoxIn(box_3, paris), BoxIn(box_1, berlin), BoxIn(box_2, berlin), BoxIn(box_3, berlin), BoxIn(box_1, rome), BoxIn(box_2, rome), BoxIn(box_3, rome)\}$
  - $TruckIn$: $\{TruckIn(truck_1, paris), TruckIn(truck_1, berlin), TruckIn(truck_1, rome), TruckIn(truck_2, paris), TruckIn(truck_2, berlin), TruckIn(truck_2, rome)\}$
  - $BoxOn$: $\{BoxOn(box_1, truck_1), BoxOn(box_2, truck_1), BoxOn(box_3, truck_1), BoxOn(box_1, truck_2), BoxOn(box_2, truck_2), BoxOn(box_3, truck_2)\}$
- **Ground Actions**:

  Exponential #actions in arity

  - $load$: $\{load(box_1, truck_1), load(box_2, truck_1), load(box_3, truck_1)\ load(box_1, truck_2), load(box_2, truck_2)\}, load(box_3, truck_2)\}$
  - $unload$: $\{unload(box_1, truck_1), unload(box_2, truck_1), unload(box_3, truck_1), unload(box_1, truck_2), unload(box_2, truck_2)\}, unload(box_3, truck_2)\}$
  - $drive$: $\{drive(truck_1, paris), drive(truck_1, berlin), drive(truck_1, rome)\ drive(truck_2, paris), drive(truck_2, berlin), drive(truck_2, rome)$
- **Goal**: $[BoxIn(box_1, paris) \vee BoxIn(box_2, paris) \vee BoxIn(box_3, paris)]$

Exponential in #nested quantifiers

A First-order Solution to BoxWorld

- Derive solution deductively at lifted PDDL level:
  - if $(\exists b. BoxIn(b, paris))$
    then do $noop$
  - else if $(\exists b^*, t^*. TruckIn(t^*, paris) \wedge BoxOn(b^*, t^*))$
    then do $unload(b^*, t^*)$
  - else if $(\exists b, c, t^*. BoxOn(b, t^*) \wedge TruckIn(t, c))$
    then do $drive(t^*, paris)$
  - else if $(\exists b^*, c, t^*. BoxIn(b^*, c) \wedge TruckIn(t^*, c))$
    then do $load(b^*, t^*)$
  - else if $(\exists b, c_1^*, t^*, c_2. BoxIn(b, c_1^*) \wedge TruckIn(t^*, c_2))$
    then do $drive(t^*, c_1^*)$
  - else do $noop$

Optimal for *any* domain instantiation!

- Great, but how do I obtain this solution?

Tutorial Overview

- Foundational theory for exploiting first-order structure in planning
  - deterministic and probabilistic
  - representations and implementation

- We cover a *deductive* approach
  - plan solely based on model
  - no simulations or sampled data
    - this would require grounding

- See *Sanner & Boutilier, AI Journal 2008* for discussion / comparison to *inductive* approaches

First-order MDPs

Deterministic Planning in the Situation Calculus

Scott Sanner
NICTA

FOMDP Tutorial Outline

- Motivation
- **Deductive First-order Planning in the Situation Calculus**
- FOMDPs and Symbolic Dynamic Programming
- Potential Caveats

Situation Calculus: Ingredients

- Actions
  - first-order terms with action parameters
  - e.g., *load(b,t), unload(b,t), drive(t,c)*
- Situations
  - term that encodes action history
  - e.g., *s*, s_0, *do(load(b,t),s), do(load(b,t),drive(t,c),s)*
- Fluents
  - relation whose truth value varies b/w situations
  - e.g., *BoxOn(b,t,s), TruckIn(t,c,s), BoxIn(t,c,s)*

Situation Calculus: PDDL to Effects

- Recall BoxWorld PDDL specification…
  - $load(Box : b, Truck : t)$:
    - Effects:
      - $when\ [\exists City : c.\ BoxIn(b, c) \wedge TruckIn(t, c)]\ then\ [BoxOn(b, t)]$
      - $\forall City : c.\ when\ [BoxIn(b, c) \wedge TruckIn(t, c)]\ then\ [\neg BoxIn(b, c)]\}$
  - $unload(Box : b, Truck : t)$:
    - Effects:
      - $\forall City : c.\ when\ [BoxOn(b, t) \wedge TruckIn(t, c)]\ then\ [BoxIn(b, c)]$
      - $when\ [\exists City : c.\ BoxOn(b, t) \wedge TruckIn(t, c)]\ then\ [\neg BoxOn(b, t)]$
  - $drive(Truck : t, City : c)$:
    - Effects:
      - $when\ [\exists City : c_1.\ TruckIn(t, c_1)]\ then\ [TruckIn(t, c)]$
      - $\forall City : c_1.\ when\ [TruckIn(t, c_1)]\ then\ [\neg TruckIn(t, c_1)]$

Situation Calculus: PDDL to Effects

- Translate to positive and negative effect axioms

- $load(Box : b, Truck : t)$:
  - Effects:
    - $[\exists c.\, a = load(b,t) \wedge BoxIn(b,c,s) \wedge TruckIn(t,c,s)] \supset BoxOn(b,t,do(a,s))$
    - $[\exists t.\, a = load(b,t) \wedge BoxIn(b,c,s) \wedge TruckIn(t,c,s)] \supset \neg BoxIn(b,c,do(a,s))$

- $unload(Box : b, Truck : t)$:
  - Effects:
    - $[\exists t.\, a = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t,c,s)] \supset BoxIn(b,c,do(a,s))$
    - $[\exists c.\, a = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t,c,s)] \supset \neg BoxOn(b,t,do(a,s))$

- $drive(Truck : t, City : c)$:
  - Effects:
    - $[\exists c_1.\, a = drive(t,c) \wedge TruckIn(t,c_1,s)] \supset TruckIn(t,c,do(a,s))$
    - $[\exists c.\, a = drive(t,c) \wedge TruckIn(t,c_1,s)] \supset \neg TruckIn(t,c_1,do(a,s))$

Situation Calculus: PDDL to Effects

- Now, merge into positive effect axioms

$$\gamma_F^+(\vec{x}, a, s) \supset F(\vec{x}, do(a, s))$$

  and negative effect axioms

$$\gamma_F^-(\vec{x}, a, s) \supset \neg F(\vec{x}, do(a, s))$$

- Use **rule** to combine multiple effects

$$[(C_1 \supset F) \wedge (C_2 \supset F)] \equiv [(C_1 \vee C_2) \supset F]$$

Frame Problem

- Now we have positive and negative effects

$$\gamma^+_{BoxIn}(\vec{x},a,s) \supset BoxIn(\vec{x},do(a,s)) \qquad \gamma^-_{BoxIn}(\vec{x},a,s) \supset \neg BoxIn(\vec{x},do(a,s))$$
$$\gamma^+_{TruckIn}(\vec{x},a,s) \supset TIn(\vec{x},do(a,s)) \qquad \gamma^-_{TruckIn}(\vec{x},a,s) \supset \neg TruckIn(\vec{x},do(a,s))$$
$$\gamma^+_{BoxOn}(\vec{x},a,s) \supset BoxOn(\vec{x},do(a,s)) \qquad \gamma^-_{BoxOn}(\vec{x},a,s) \supset \neg BoxOn(\vec{x},do(a,s))$$

so we have compactly specified what changes.

- How to compactly specify what does not change?
  - Infamous **Frame Problem**
  - Intuition:
    - "what does not change, remains same"
    - this is Reiter's **Default Solution**
    - but we have to logically formalize it…

Successor State Axioms (SSAs)

- Default solution to frame problem given as SSAs:

Unique names axioms for terms

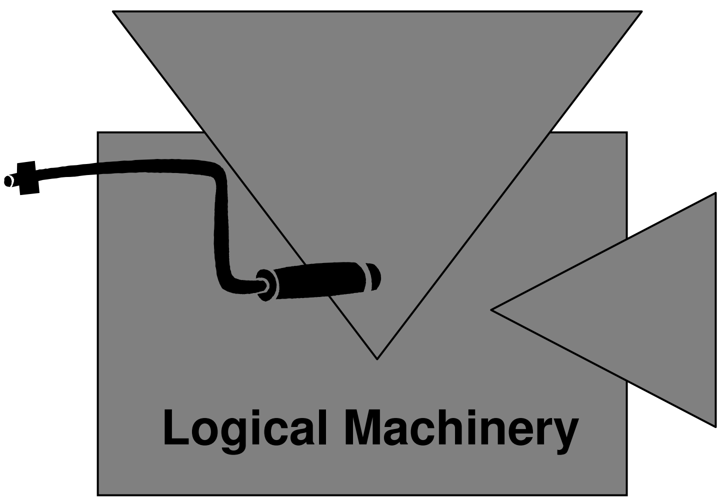
Effect axioms

Unique names for actions / arguments.

Explanation closure axioms

$$\gamma_F^+(\vec{x}, a, s) \supset F(\vec{x}, do(a, s))$$

$$\gamma_F^-(\vec{x}, a, s) \supset \neg F(\vec{x}, do(a, s))$$



SSAs

$$F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s)$$

SSAs

- Shorthand:

$$F(\vec{x}, do(a,s)) \equiv \Phi_F(\vec{x}, a, s)$$
$$\equiv \gamma_F^+(\vec{x}, a, s) \vee F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s)$$

What changes and does not change!

- Reality check:

$$BoxOn(b,t,do(a,s)) \equiv \Phi_{BoxOn}(b,t,a,s)$$
$$\equiv [\exists c.\, a = load(b,t) \wedge BoxIn(b,c,s) \wedge TruckIn(t,c,s)]$$
$$\vee\ BoxOn(b,t,s) \wedge \neg\,[\exists c.\, a = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t,c,s)]$$

$$BoxIn(b,c,do(a,s)) \equiv \Phi_{BoxIn}(b,c,a,s)$$
$$\equiv [\exists t.\, a = unload(b,t) \wedge BoxOn(b,t,s) \wedge TruckIn(t,c,s)]$$
$$\vee\ BoxIn(b,c,s) \wedge \neg\,[\exists t.\, a = load(b,t) \wedge BoxIn(b,c,s) \wedge TruckIn(t,c,s)]$$

$$TruckIn(t,c,do(a,s)) \equiv \Phi_{TruckIn}(t,c,a,s)$$
$$\equiv [\exists c_1.\, a = drive(t,c) \wedge TruckIn(t,c_1,s)]$$
$$\vee\ TruckIn(t,c,s) \wedge \neg\,[\exists c_1.\, a = drive(t,c) \wedge TruckIn(t,c_1,s)]$$

Regression

- Why have we defined SSAs?

- Regression:
  - If φ held after action *a*
    then *regression* is the φ' that held before action *a*

- Exploit following properties:
  - $Regr(\neg\psi) = \neg Regr(\psi)$
  - $Regr(\psi_1 \wedge \psi_2) = Regr(\psi_1) \wedge Regr(\psi_2)$
  - $Regr((\exists x)\psi) = (\exists x) Regr(\psi)$
  - $Regr(F(\vec{x}, do(a, s))) = \Phi_F(\vec{x}, a, s)$

Regression Example

- Given $\exists b.\ BoxIn(b, paris, do(unload(b^*, t^*), s))$
- Regress through *unload(b*,t*)*

$$Regr(\exists b.\ BoxIn(b, paris, do(unload(b^*, t^*), s)))$$

$$= \exists b.\ \Phi_{BoxIn}(b, paris, unload(b^*, t^*), s)$$

$$= \exists b.\ [\exists t.\ unload(b^*, t^*) = unload(b, t) \wedge BoxOn(b, t, s) \wedge TruckIn(t, paris, s)]$$
$$\vee\ BoxIn(b, paris, s)$$
$$\wedge \neg [\exists t.\ unload(b^*, t^*) = load(b, t) \wedge BoxIn(b, paris, s) \wedge TruckIn(t, paris, s)]$$

$$= [\exists b, t.\ b = b^* \wedge t = t^* \wedge BoxOn(b, t, s) \wedge TruckIn(t, paris, s)]$$
$$\vee\ \exists b.\ BoxIn(b, paris, s)$$

make non-empty domain assumption

$$= [(\exists b. b = b^*) \wedge (\exists t. t = t^*) \wedge BoxOn(b^*, t^*, s) \wedge TruckIn(t^*, paris, s)]$$
$$\vee\ \exists b.\ BoxIn(b, paris, s)$$

note free vars b^*, t^*; why?

$$= [BoxOn(b^*, t^*, s) \wedge TruckIn(t^*, paris, s)] \vee \exists b.\ BoxIn(b, paris, s)$$

Regression Example

- But what action instantiation of *unload(b*,t*)* leads to:

$$\exists b.\ BoxIn(b, paris, do(unload(b^*, t^*), s))$$

- Just have to existentially quantify b^*, t^*
  - Can obtain instances via query extraction w.r.t. state KB

$$\begin{aligned} \exists b^*, t^*.\ & Regr(\exists b.\ BoxIn(b, paris, do(unload(b^*, t^*), s))) \\ & = \exists b^*, t^*.\ [BoxOn(b^*, t^*, s) \wedge TruckIn(t^*, paris, s)] \\ & \quad \vee \exists b.\ BoxIn(b, paris, s) \\ & = [\exists b^*, t^*.\ BoxOn(b^*, t^*, s) \wedge TruckIn(t^*, paris, s)] \\ & \quad \vee \exists b.\ BoxIn(b, paris, s) \end{aligned}$$

First-order state & action abstraction!
Don't have to enumerate all states, *b*, t**!

Recap

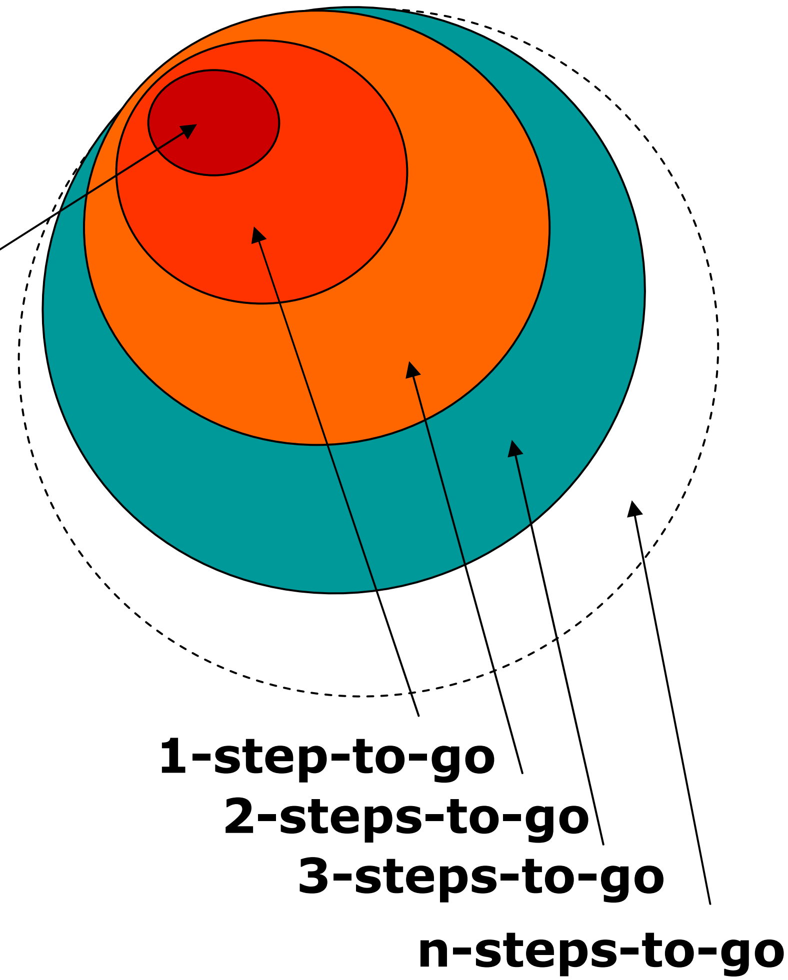
- We translated PDDL to SitCalc theory
  - converted PDDL effects to SitCalc effect axioms
  - derived SSAs from effect axioms
    - using default solution to Frame Problem
- Introduced regression operator
  - extracted action instantiation to achieve goal
- Let the planning begin…

Regression Planning

- Define abstract goal state, e.g.,

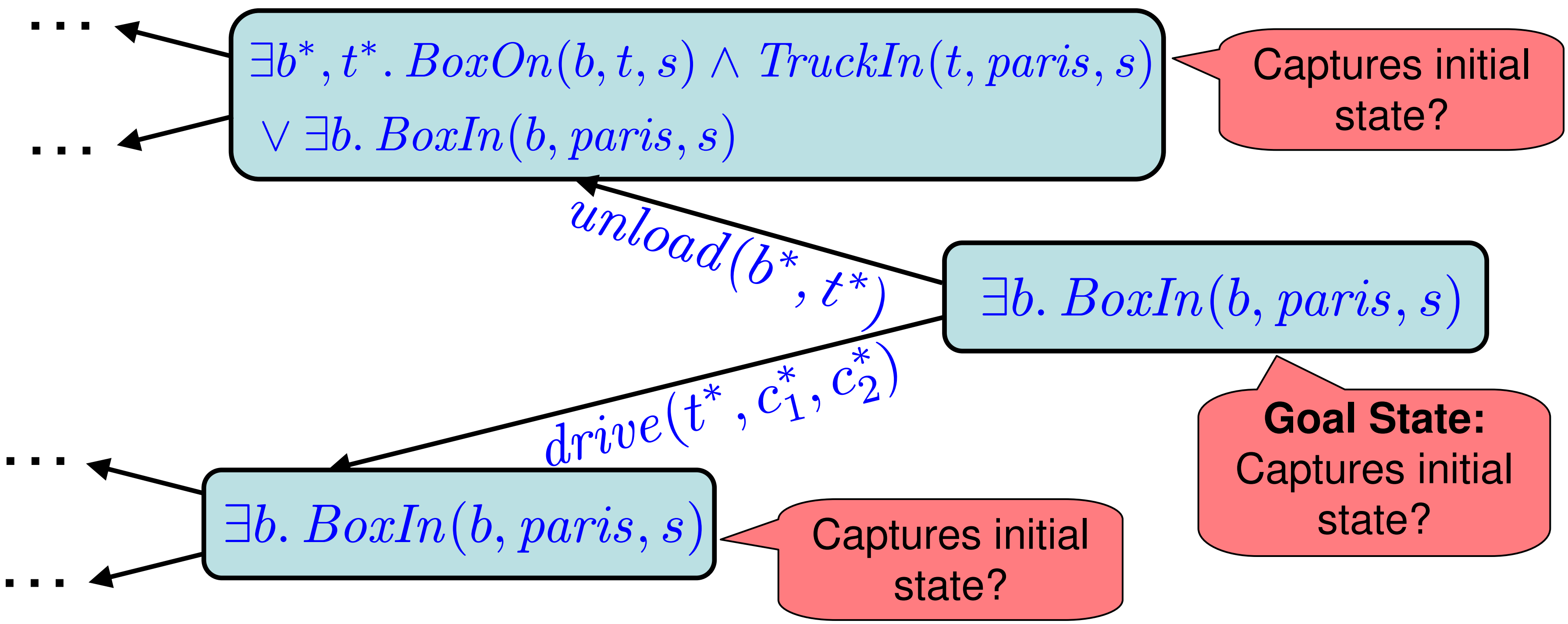
  $\exists b.\, BoxIn(b, paris, s)$

- Check if regression through action sequence holds in initial state



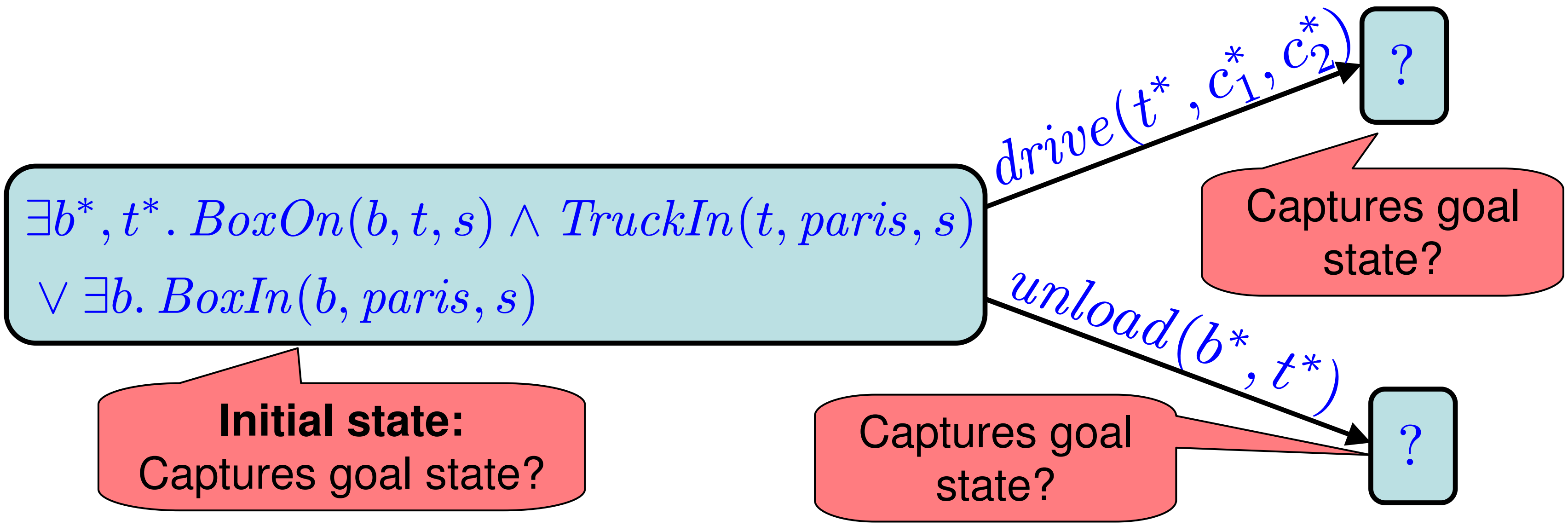
First-order Goal-regression

- We can now do goal regression planning!
  - Provide initial state and sequence of actions
  - Use regression, ∃ to tell whether goal will hold



Progression and Forward-search?

- Can we do lifted *forward-search planning*?



  - *Progression* not first-order definable! (Reiter, 01)
  - Could progress *ground* state
    - But this does not exploit first-order structure

Golog: Restricted Plan Search

- Al**GO**l in **LOG**ic
  - Search the space of sequential action plans
  - Regress actions to initial state to test reachability
  - *Restrict* action space with program:

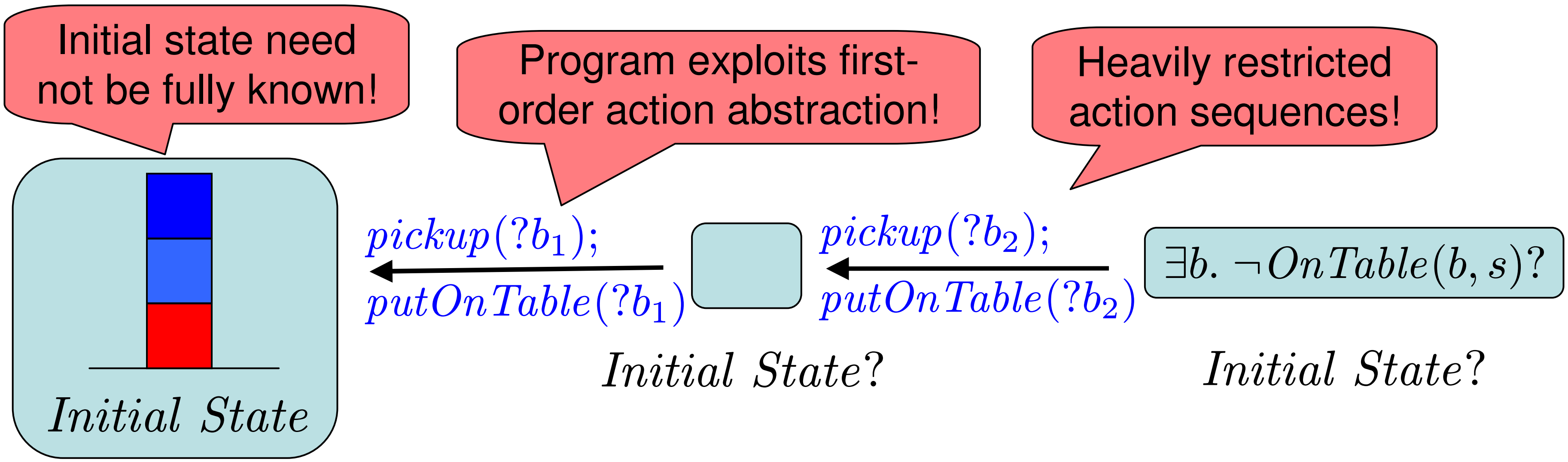
α	primitive action
ϕ?	condition test
(δ_1, δ_2)	sequence
$\textbf{if } \phi \textbf{ then } \delta_2 \textbf{ endIf}$	conditional
$\textbf{while } \phi \textbf{ then } \delta \textbf{ endWhile}$	loop
$(\delta_1 \vert \delta_2)$	nondeterministic choice of action
$\pi\ \vec{x}\ [\delta]$	nondeterministic choice of arguments
δ^*	nondeterministic iteration
$\textbf{proc } \beta(\vec{x})\ \delta \textbf{ endProc}$	procedure call definition
$\beta(\vec{t})$	procedure call

Golog Example

- Golog Program:

$$(\pi\, b\, [\neg OnTable(b, s)?, pickup(b), putOnTable(b)])^*,$$
$$\forall b.\ OnTable(b, s)?$$

- Diagram of Golog Planning:



For Further Reading

- Knowledge in Action:
  In-depth coverage of SitCalc default solution, applications
  (Reiter, 2001)

- Golog
  (Levesque, Reiter, Lesperance, Lin,
  Journal Logic Programming, 1997)

- Extensions
  - ConGolog: concurrent Golog
    (de Giacomo, Lesperance, Levesque, AIJ-00)
  - DT-Golog: decision-theoretic, covered next
    (Soutchanski, Boutilier, Reiter, Thrun, AAAI-20)

For MDPs,
covered next.

Conclusion

- Situation Calculus
  - First-order specification of action theory
  - Default solution addresses Frame Problem
    - Effective approach to PDDL-expressive planning
- Supports Regression Planning
  - Initial state need not be fully specified
  - Can restrict action space with Golog program
  - Exploits state & action abstraction
    - Avoids enumerating all state & action instances!

First-order MDPs

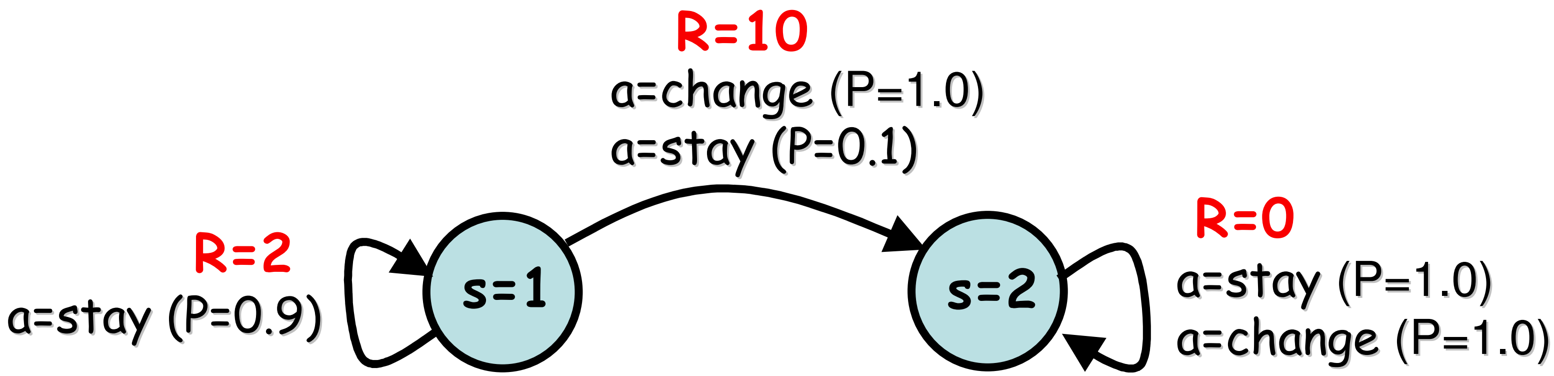
FOMDPs and Symbolic Dynamic Programmming

Scott Sanner
NICTA

FOMDP Tutorial Outline

- Motivation
- Deductive First-order Planning
  in the Situation Calculus
- **FOMDPs and
  Symbolic Dynamic Programming**
- Potential Caveats

MDPs $<S,A,T,R,\gamma>$

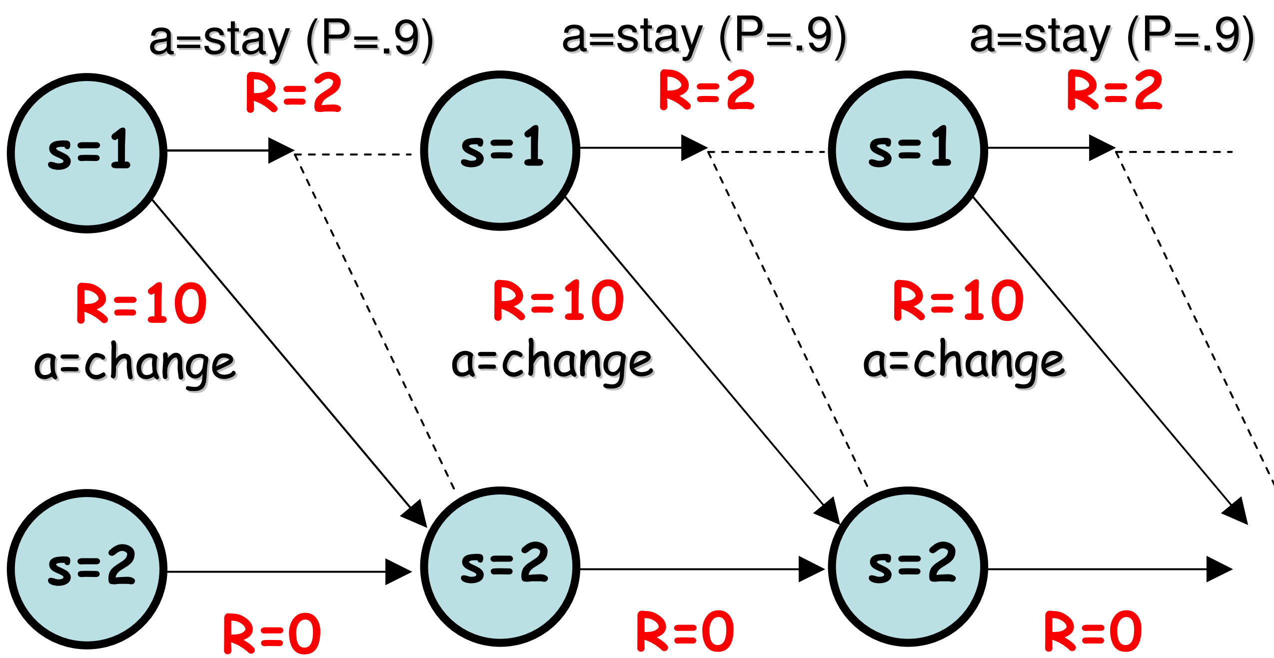


- S = {1,2}; A = {stay, change}
- Reward
  - R(s=1,a=stay) = 2
  - …
- Transitions
  - T(s=1,a=stay,s'=1) = P(s'=1 | s=1, a=stay) = .9
  - …
- Discount γ

How to act in an MDP?

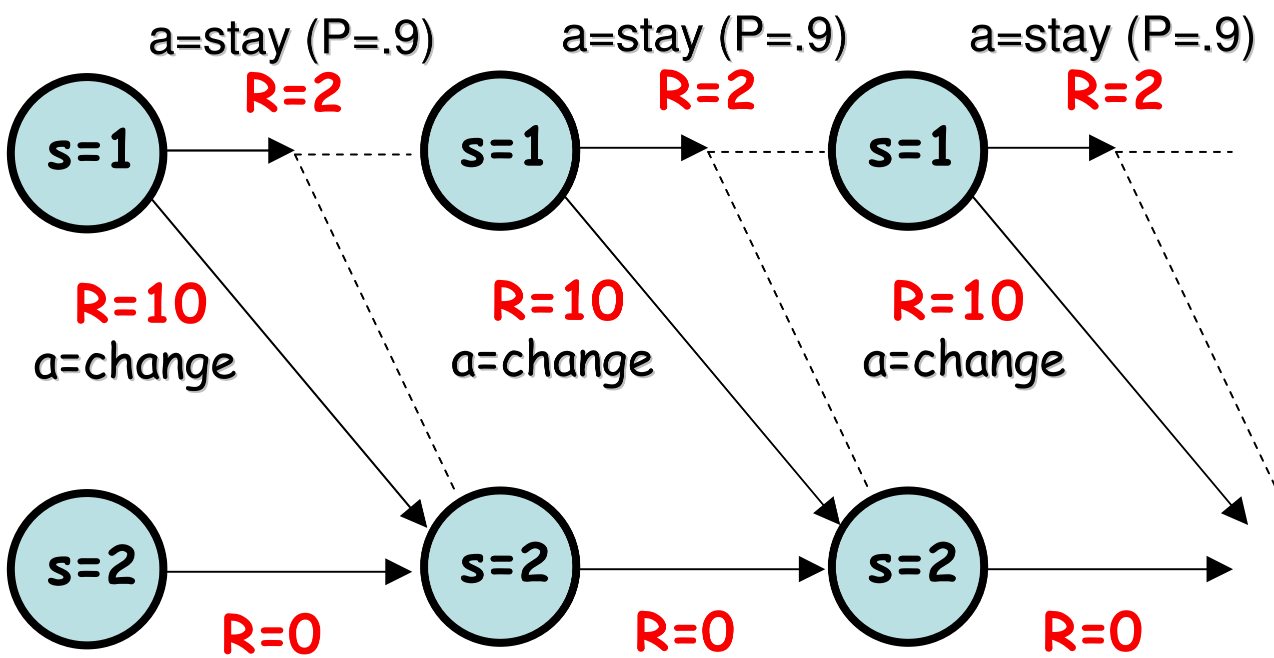
Define policy
$\pi: S \rightarrow A$

What’s the best Policy?



- Immediate vs. long-term gain?

What's the best Policy?



- Must define reward criterion to optimize!
  - Discount factor γ important (γ=.9 vs. γ=.1)

MDP Policy, Value, & Solution

- Define value of a policy π:

$$V_\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t \cdot r_t \middle| s = s_0\right]$$

- Tells how much value you expect to get by following π starting from state *s*

- MDP Optimal Solution:
  - Find optimal policy π* that maximizes value
  - Fortunately: $\exists\pi^*.\,\forall s,\pi.\; V_{\pi^*}(s) \geq V_\pi(s)$

Value Iteration: from finite to ∞ decisions

- Given optimal *t-1-stage-to-go* value function
- How to act optimally with *t* decisions?
  - Take action *a* then act so as to achieve V^{t-1} thereafter

$$Q^t(s,a) := R(s,a) + \gamma \cdot \sum_{s' \in S} T(s,a,s') \cdot V^{t-1}(s')$$

  - What is expected value of best action *a* at decision stage *t*?

$$V^t(s) := \max_{a \in A} \left\{ Q^t(s,a) \right\}$$

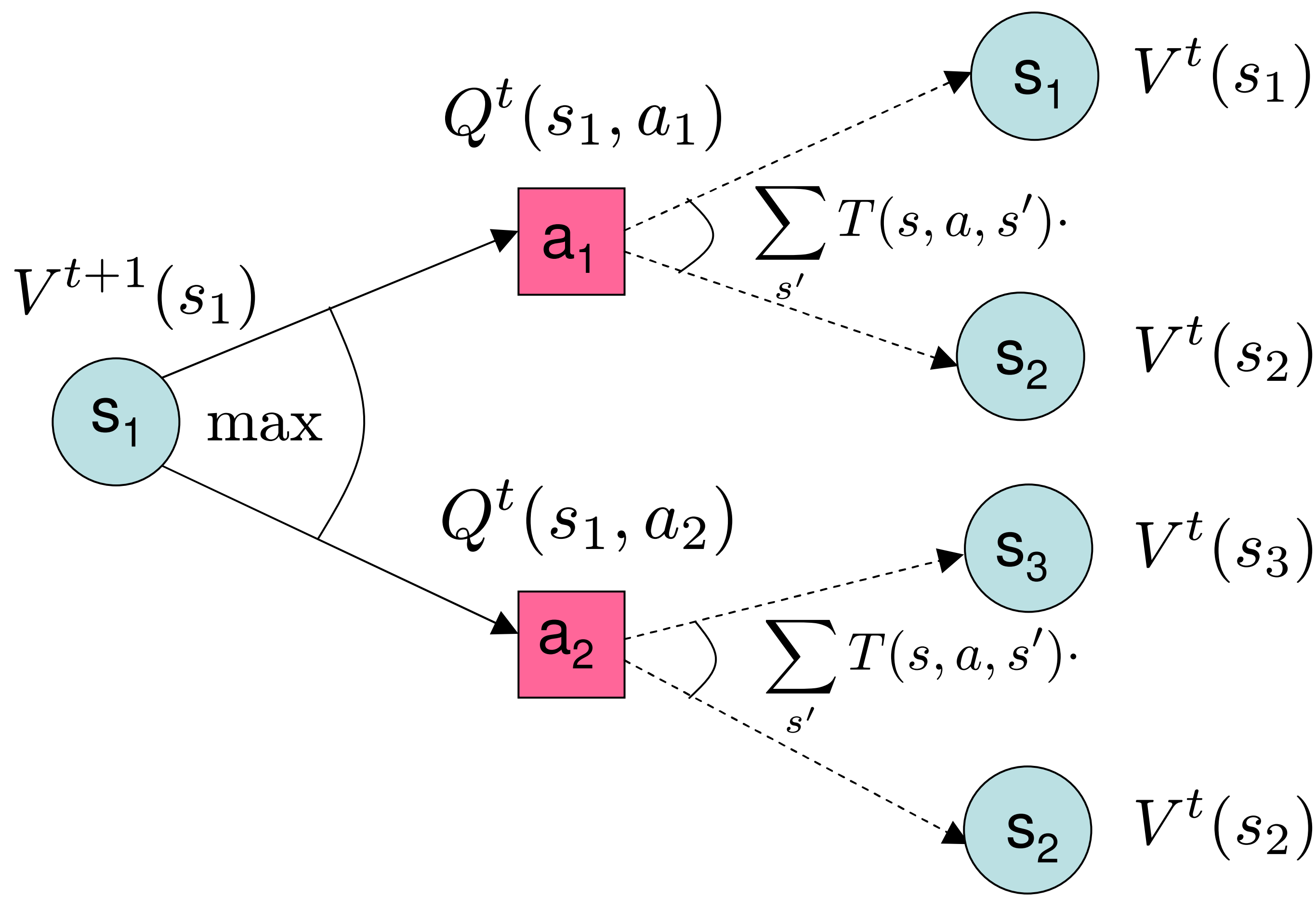
  - At ∞ horizon, get same value (=V*)

$$\lim_{t \to \infty} \max_s |V^t(s) - V^{t-1}(s)| = 0$$

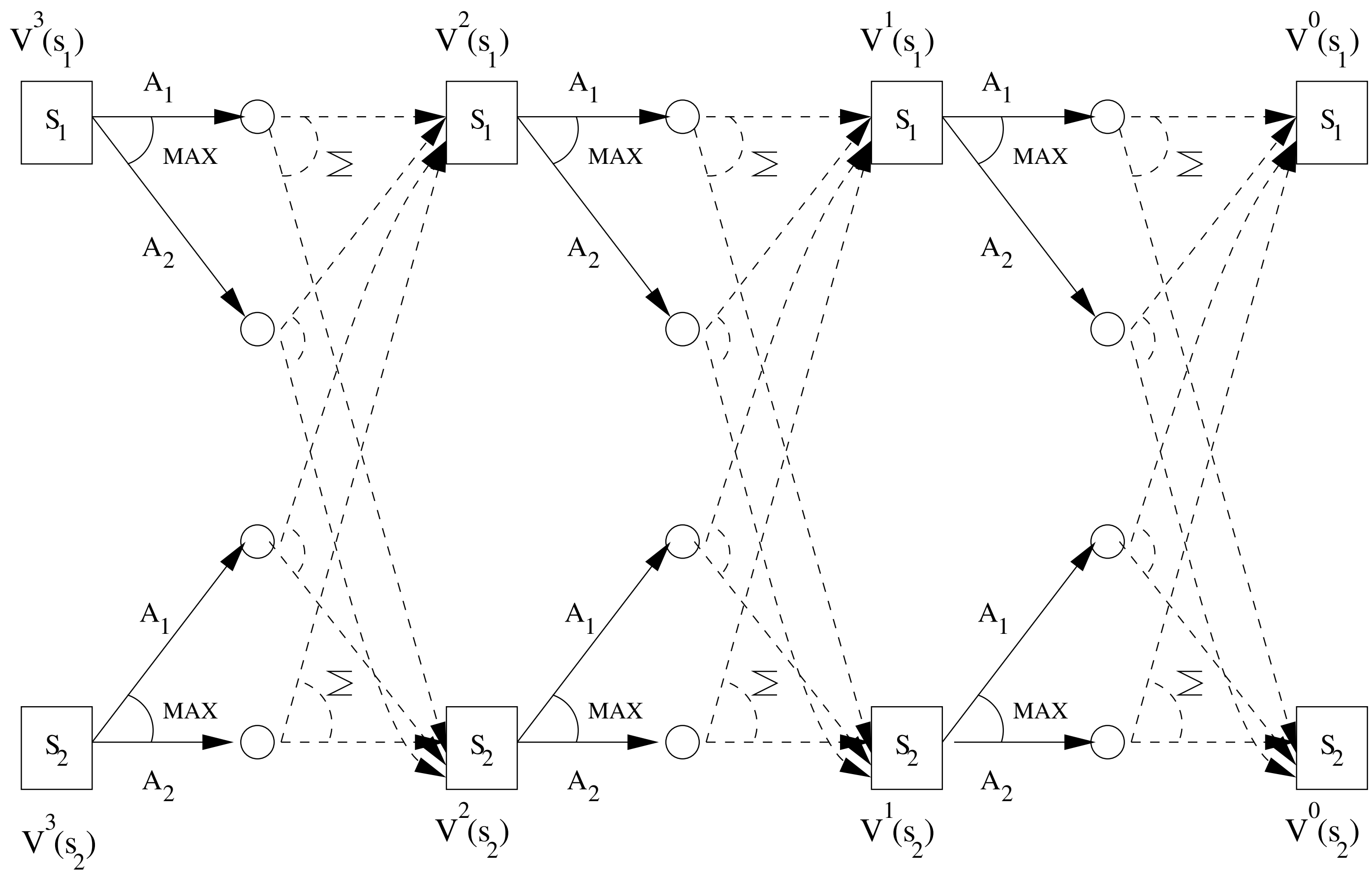
    - π* says act same at each decision stage for ∞ horizon!

Single Dynamic Programming Step

- Graphical view:



Synchronous DP Updates (Value Iteration)



Value Function $\rightarrow$ Policy

- Can derive policy from value function V

- Given arbitrary value V (optimal or not)…
  - A *greedy policy* π_V takes action in each state that maximizes expected value w.r.t. *V*:

$$\pi_V(s) = \arg\max_a \left\{ R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s') \right\}$$

  - If can act so as to obtain *V* after doing action *a* in state *s*, π_V guarantees *V(s)* in expectation

How to Specify & Solve First-order MDPs?

Following:
[Boutilier, Reiter, Price, IJCAI-01]

First-order (FO)MDPs: Case Statement

- $<S,A,T,R>$ for FOMDPs defined in terms of *cases*
  - E.g., express *reward* in *BoxWorld* FOMDP as…

$rCase(s) =$

$\forall b,c.\ Dest(b,c) \Rightarrow BIn(b,c,s)$	1
$\neg$ "	0

- **Operators:** Define unary, binary case operations
  - E.g., can take "cross-sum" $\oplus$ (or $\otimes$, $\ominus$) of cases…

φ	10
$\neg\varphi$	20

$\oplus$

ϕ	3
$\neg\phi$	4

=

$\varphi \wedge \phi$	13
$\varphi \wedge \neg\phi$	14
$\neg\varphi \wedge \phi$	23
$\neg\varphi \wedge \neg\phi$	24

Stochastic Actions & FODTR

- Stochastic actions using deterministic SitCalc:
  - User's stochastic action: $A(x) = load(b,t)$
  - Nature's choice: $n(x) \in \{loadS(b,t), loadF(b,t)\}$
  - Probability distribution over Nature's choice:

$P(loadS(b,t) \mid load(b,t)) =$

$snow(s)$	.1
$\neg snow(s)$	.6

$P(loadF(b,t) \mid load(b,t)) =$

$snow(s)$	.9
$\neg snow(s)$	.4

- First-order decision-theoretic regression
  - FODTR = *expectation* of regression:

$$FODTR[vCase(s),A(x)] = \mathbf{E}_{P(n(x)\mid A(x))}[Regr[vCase(s),n(x)]]$$

Q-functions and Backups

- **FODTR almost gives us a Q-function**

$$FODTR[vCase(unload(b,t))] = \begin{array}{|l|c|}\hline On(b,t,s) & 5 \\ \hline \neg On(b,t,s) & 0 \\ \hline\end{array}$$

  - FODTR specific to action variables
  - Also need to add reward, discount

- **Specify a backup operator for this**

$$B^{unload}[vCase(s)] = rCase(s) \oplus \gamma \begin{array}{|l|c|}\hline \exists b,t.\ On(b,t,s) & 5 \\ \hline \exists b,t.\ \neg On(b,t,s) & 0 \\ \hline\end{array}$$

  - Idea: if *exists* action instance that achieves value
  - Yields a first-order Q-function

Symbolic Dynamic Programming

- What value if 0-stages-to-go?
  - Obviously $V^0(s) = rCase(s)$
- What value if 1-stage-to-go?
  - We know value for each action

$$V^1(s) = \max_s \left\{ \begin{array}{l} \begin{array}{|l|l|} \hline \varphi_1 & 9 \\ \hline \varphi_2 & 0 \\ \hline \end{array} = B^{A_1}[rCase(s)] \\ \\ \begin{array}{|l|l|} \hline \varphi_3 & 3 \\ \hline \varphi_4 & 1 \\ \hline \end{array} = B^{A_2}[rCase(s)] \end{array} \right.$$

  - Now just need max for every state
- Value iteration: (BoutReiPr, IJCAI-01)
  - Obtain V^{n+1} from V^n until $(V^{n+1} \ominus V^n) < \varepsilon$

Symbolic Dynamic Programming

- What value if 0-stages-to-go?
  - Obviously $V^0(s) = rCase(s)$

- What value if 1-stage-to-go?
  - We know value for each action

$$\begin{array}{|c|c|}\hline \varphi_1 & 9 \\ \hline \end{array}$$

$$\begin{array}{|c|c|}\hline \varphi_1 & 9 \\ \hline \varphi_2 & 0 \\ \hline \end{array} = B^{A_1}[rCase(s)]$$

$$V^1(s) =$$

$$\begin{array}{|c|c|}\hline \varphi_3 & 3 \\ \hline \varphi_4 & 1 \\ \hline \end{array} = B^{A_2}[rCase(s)]$$

  - Now just need max for every state

- Value iteration: (BoutReiPr, IJCAI-01)
  - Obtain V^{n+1} from V^n until $(V^{n+1} \ominus V^n) < \varepsilon$

Symbolic Dynamic Programming

- What value if 0-stages-to-go?
  - Obviously $V^0(s) = rCase(s)$
- What value if 1-stage-to-go?
  - We know value for each action

$$\begin{array}{|c|c|}\hline \varphi_1 & 9 \\ \hline \varphi_2 & 0 \\ \hline \end{array} = B^{A_1}[rCase(s)]$$

$$V^1(s) = \begin{array}{|c|c|}\hline \varphi_1 & 9 \\ \hline \textit{else } \varphi_3 & 3 \\ \hline \end{array}$$

$$\begin{array}{|c|c|}\hline \varphi_3 & 3 \\ \hline \varphi_4 & 1 \\ \hline \end{array} = B^{A_2}[rCase(s)]$$

  - Now just need max for every state
- Value iteration: (BoutReiPr, IJCAI-01)
  - Obtain V^{n+1} from V^n until $(V^{n+1} \ominus V^n) < \varepsilon$

Symbolic Dynamic Programming

- What value if 0-stages-to-go?
  - Obviously $V^0(s) = rCase(s)$
- What value if 1-stage-to-go?
  - We know value for each action

$$V^1(s) = \begin{array}{|l|c|}\hline \varphi_1 & 9 \\ \hline \textbf{else } \varphi_3 & 3 \\ \hline \textbf{else } \varphi_4 & 1 \\ \hline \end{array}$$

$$\begin{array}{|l|c|}\hline \varphi_1 & 9 \\ \hline \varphi_2 & 0 \\ \hline \end{array} = B^{A_1}[rCase(s)]$$

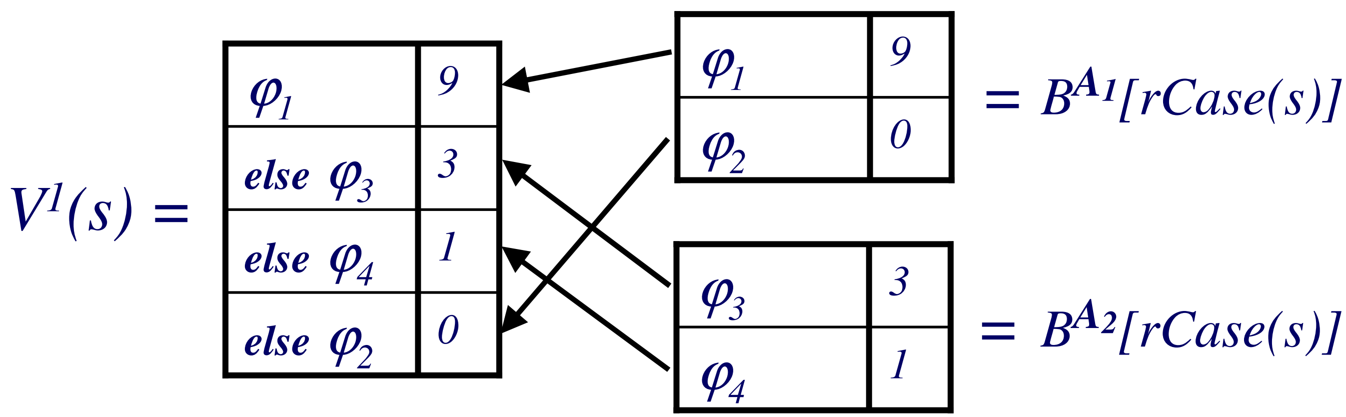
$$\begin{array}{|l|c|}\hline \varphi_3 & 3 \\ \hline \varphi_4 & 1 \\ \hline \end{array} = B^{A_2}[rCase(s)]$$

  - Now just need max for every state

- Value iteration: (BoutReiPr, IJCAI-01)
  - Obtain V^{n+1} from V^n until $(V^{n+1} \ominus V^n) < \varepsilon$

Symbolic Dynamic Programming

- What value if 0-stages-to-go?
  - Obviously $V^0(s) = rCase(s)$
- What value if 1-stage-to-go?
  - We know value for each action



  - Now just need max for every state
- Value iteration: (BoutReiPr, IJCAI-01)
  - Obtain V^{n+1} from V^n until $(V^{n+1} \ominus V^n) < \varepsilon$

[Sanner, Thesis]

First-order ADDs

- **Want to compactly represent:**

$$\text{case} = \begin{array}{|c|c|}\hline \exists x.[A(x) \lor \forall y.A(x) \land B(x) \land \neg A(y)] & 1 \\ \hline \neg \text{ ''} & 0 \\ \hline \end{array}$$

- **Push down quantifiers, expose prop. structure:**

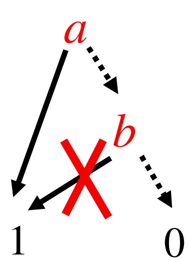
$$[\exists x.A(x)] \lor ([\exists x.A(x) \land B(x)] \land [\forall y.\neg A(y)])$$

Var	*Var* $\Leftrightarrow$ *FOL KB*
a	$\equiv [\exists x.A(x)]$
b	$\equiv [\exists x.A(x) \land B(x)]$

$$\text{case} = \begin{array}{|c|c|}\hline a \lor (b \land \neg a) & 1 \\ \hline \neg \text{ ''} & 0 \\ \hline \end{array}$$

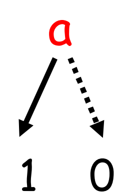
- **Convert to first-order ADD**

case =



First-order CSI!

=



Results for SDP with FOADDs

- Replace case with FO(A)ADDs, e.g. *BoxWorld*

rCase(s) = $\boxed{\exists b.\ BIn(b, Paris, s)}$

10 0

- Use FO(A)ADD ops for structured SDP (using γ=.9)…

vCase(s) = $\boxed{\exists b.\ BIn(b, Paris, s)}$

$100 : \ noop$ $\boxed{\exists b, t. TIn(t, Paris, s) \wedge On(b, t, s)}$

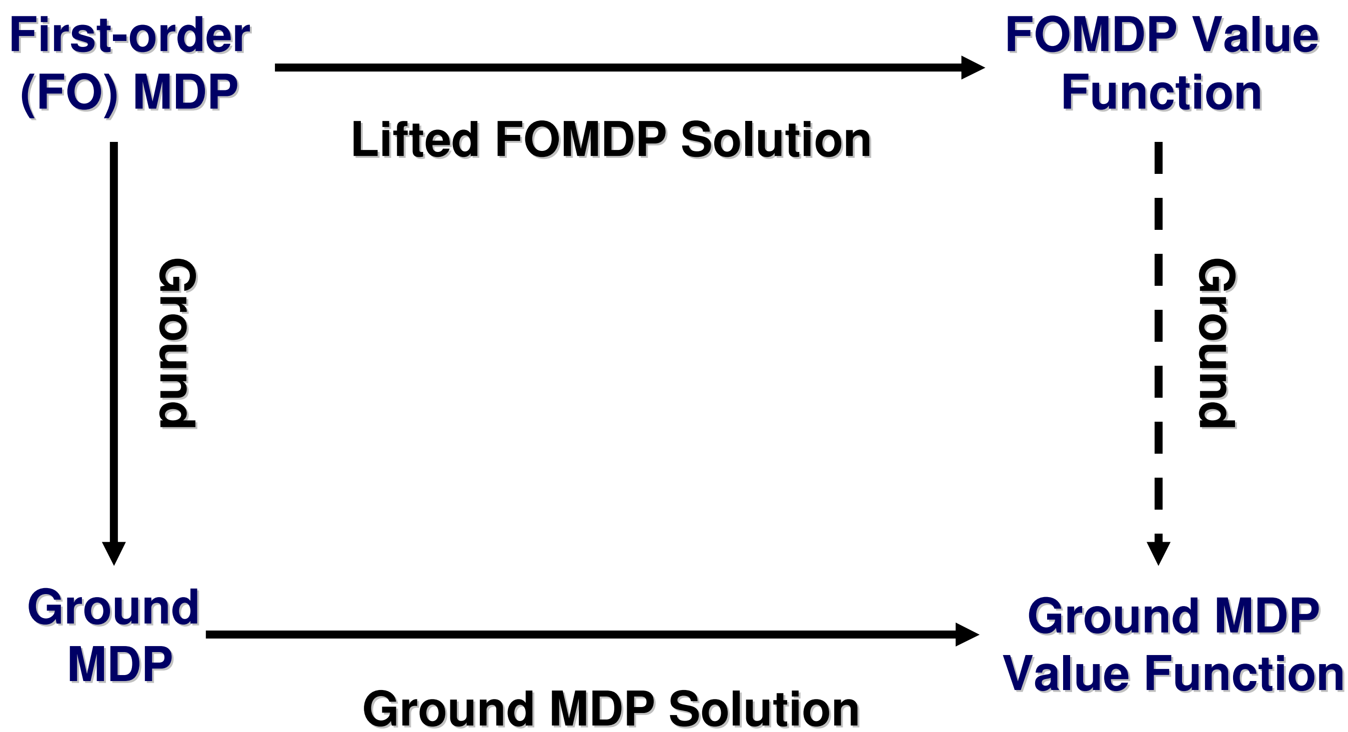
$89 : \ unload(b, t)$ $\boxed{\exists b, t.\ On(b, t, s)}$

$80 : \ drive(t, Paris)$ $\boxed{\exists b, c.\ BoxIn(b, c, s) \wedge \exists t. TIn(t, c, s)}$

$72 : \ load(b, t)$ …

Correctness of SDP

- **Show SDP for FOMDPs is correct w.r.t. ground MDP:**



Related Purely Deductive Approaches

- Value Iteration:
  - **ReBel algorithm**
    (Kersting, van Otterlo, de Raedt, ICML-04)
  - **FOVIA algorithm for fluent calculus**
    (Karabaev & Skvortsova, UAI-05)
  - **First-order decision diagrams (FODDs)**
    (Wang, Joshi, Khardon, IJCAI-07; JK, ICAPS-08; WJK, JAIR-08)

Kristian covers this.

Saket covers this.

- Approximate Linear Programming (ALP)
  - **First-order ALP (FOALP)**
    (Sanner & Boutilier, UAI-05)

3rd place in ICAPS IPPC5 (after FPG, FF-Replan)

- Policy Iteration
  - **Approximate policy iteration (FOAPI)**
    (Sanner & Boutilier, UAI-06)
  - **Modified policy iteration with FODDs**
    (Wang & Joshi, UAI-07)

- Factored FOMDPs – FOMDP extension
  - **Factored SDP and Factored FOALP**
    (Sanner & Boutilier, ICAPS-07)

First-order MDPs

Caveats

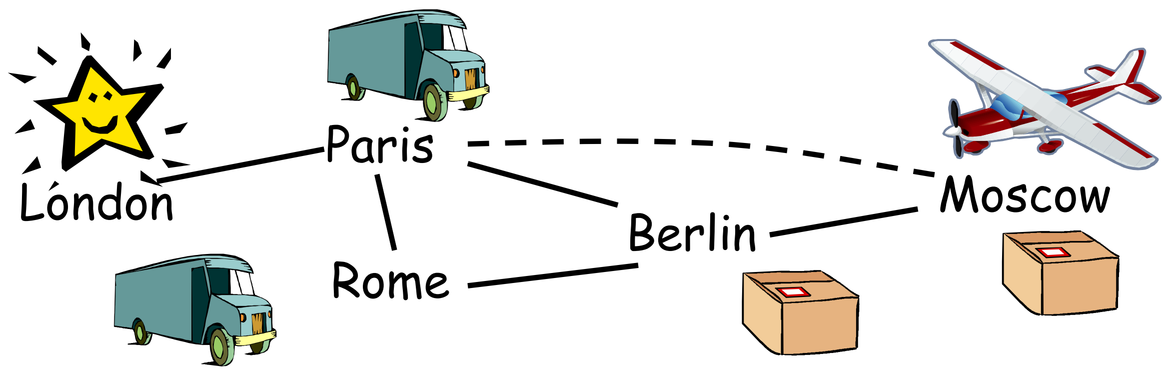
Scott Sanner
NICTA

FOMDP Tutorial Outline

- Motivation
- Deductive First-order Planning
  in the Situation Calculus
- FOMDPs and
  Symbolic Dynamic Programming
- **Potential Caveats**

Caveats of First-order Planning

- Many problems have topologies
  - e.g, reachability constraints in logistics



- If topology not fixed *a priori*…
  - first-order solution must consider ∞ topologies
    - e.g., if *Moscow* reachable from *Rome* in five steps…
  - in general case, leads to ∞ values / policy

Caveats of First-order Planning

- Universal Rewards

$$R(s) = \begin{array}{|l|}\hline \forall b, c.\ Dst(b,c) \rightarrow BoxIn(b,c,s) : 1 \\ \hline \neg \text{``} \qquad\qquad\qquad\qquad\qquad\qquad : 0 \\ \hline \end{array}$$

- Value function must distinguish ∞ cases

$$V^t(s) = \begin{array}{|lr|}\hline \forall b, c.\ Dst(b,c) \rightarrow BoxIn(b,c,s) : & 1 \\ \hline \text{One box not at destination} \quad : & \gamma \\ \hline \text{Two boxes not at destination} \quad : & \gamma^2 \\ \hline \vdots \qquad\qquad\qquad\qquad : & \vdots \\ \hline t-1 \text{ boxes not at destination} : & \gamma^{t-1} \\ \hline \end{array}$$

- Policy will also likely be ∞
  - But some notable exceptions (*put all blocks on table*)

Caveats of First-order Planning

- Unreachable States
  - (P)PDDL domains often under-constrained
  - e.g., domain designer intends
    - *BlocksWorld:* 2 blocks cannot be on a 3rd block
    - *Logistics:* 1 box cannot be in 2 cities at once
- Constraints implicitly obeyed in initial state
  - Then action effects cannot violate constraints
    - Reachable legal states are small subset of all states
  - But (P)PDDL does not constrain legal states!!!

Caveats of First-order Planning

- Unreachable states (continued)
  - If no background theory to restrict legal states
    - First-order planning must solve for *all* states
      - when initial state unknown
    - Where the majority of states are actually illegal!
  - *First-order* planning w/o initial state solves more difficult problem than *search-based* solutions
    - Initial state contains implicit constraint information
    - Reachable state space is small subset of *all* states

Suggests need for hybrid first-order / search-based approaches

Conclusions

- MDP: model of decision-theoretic planning
  - Common solution is dynamic programming
- FOMDPs are one model for lifted decision-theoretic planning
  - Use SitCalc specified action theory
  - Use case to represent reward, probabilities
  - Symbolic dynamic programming = lifted DP
  - **Exploit state & action abstraction for MDPs**