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., $(\forall (?b ?c) ((\text{Destination } ?b ?c) \Rightarrow (\text{Bin } ?b ?c)))$

  – Are *domain-independent* and often compact!
How to Solve?

- Relational planning *problem*:

  ```
  (: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: ![3 trucks]
  - 2 planes: ![2 planes]
  - 3 boxes: ![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 $[\exists City : c. BoxIn(b, c) \land TruckIn(t, c)]$ then $[BoxOn(b, t)]$
      - $\forall City : c. when [BoxIn(b, c) \land TruckIn(t, c)] then [\neg BoxIn(b, c)]$
  - $unload(Box : b, Truck : t)$:
    * Effects:
      - $\forall City : c. when [BoxOn(b, t) \land TruckIn(t, c)] then [BoxIn(b, c)]$
      - when $[\exists City : c. BoxOn(b, t) \land TruckIn(t, c)]$ 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.,

  - **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):**

    - $BoxIn: \{BoxIn(box_1, paris), BoxIn(box_2, paris), BoxIn(box_3, paris), BoxIn(box_1, berlin), BoxIn(box_2, berlin), BoxIn(box_3, berlin), BoxIn(box_1, rome), BoxIn(box_2, rome), BoxIn(box_3, rome)\}$

    - $TruckIn: \{TruckIn(truck_1, paris), TruckIn(truck_1, berlin), TruckIn(truck_1, rome), TruckIn(truck_2, paris), TruckIn(truck_2, berlin), TruckIn(truck_2, rome)\}$

    - $BoxOn: \{BoxOn(box_1, truck_1), BoxOn(box_2, truck_1), BoxOn(box_3, truck_1), BoxOn(box_1, truck_2), BoxOn(box_2, truck_2), BoxOn(box_3, truck_2)\}$

  - **Ground Actions:**

    - $load: \{load(box_1, truck_1), load(box_2, truck_1), load(box_3, truck_1), load(box_1, truck_2), load(box_2, truck_2), load(box_3, truck_2)\}$

    - $unload: \{unload(box_1, truck_1), unload(box_2, truck_1), unload(box_3, truck_1), unload(box_1, truck_2), unload(box_2, truck_2), unload(box_3, truck_2)\}$

    - $drive: \{drive(truck_1, paris), drive(truck_1, berlin), drive(truck_1, rome), drive(truck_2, paris), drive(truck_2, berlin), drive(truck_2, rome)\}$

  - **Goal:** $[BoxIn(box_1, paris) \lor BoxIn(box_2, paris) \lor BoxIn(box_3, paris)]$

- **Exponential #state-vars in arity**

- **Exponential #actions in arity**

- **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 \textit{noop}

  • else if \( \exists b^*, t*. TruckIn(t^*, paris) \wedge BoxOn(b^*, t^*) \)
    then do \textit{unload}(b^*, t^*)

  • else if \( \exists b, c, t*. BoxOn(b, t^*) \wedge TruckIn(t, c) \)
    then do \textit{drive}(t^*, paris)

  • else if \( \exists b^*, c, t^*. BoxIn(b^*, c) \wedge TruckIn(t^*, c) \)
    then do \textit{load}(b^*, t^*)

  • else if \( \exists b, c^*_1, t^*, c_2. BoxIn(b, c^*_1) \wedge TruckIn(t^*, c_2) \)
    then do \textit{drive}(t^*, c^*_1)

  • else do \textit{noop}

• Great, but 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…
  
  - \textit{load}(\text{Box} : b, \text{Truck} : t):
    
    - Effects:
      
      * when $[\exists \text{City} : c. \BoxIn(b, c) \land \TruckIn(t, c)]$ then $[\BoxOn(b, t)]$
      * $\forall \text{City} : c. \text{when } [\BoxIn(b, c) \land \TruckIn(t, c)] \text{ then } [\neg \BoxIn(b, c)]$
  
  - \textit{unload}(\text{Box} : b, \text{Truck} : t):
    
    - Effects:
      
      * $\forall \text{City} : c. \text{when } [\BoxOn(b, t) \land \TruckIn(t, c)] \text{ then } [\BoxIn(b, c)]$
      * when $[\exists \text{City} : c. \BoxOn(b, t) \land \TruckIn(t, c)]$ then $[\neg \BoxOn(b, t)]$
  
  - \textit{drive}(\text{Truck} : t, \text{City} : c):
    
    - Effects:
      
      * when $[\exists \text{City} : c_1. \TruckIn(t, c_1)]$ then $[\TruckIn(t, c)]$
      * $\forall \text{City} : c_1. \text{when } [\TruckIn(t, c_1)] \text{ then } [\neg \TruckIn(t, c_1)]$
Situation Calculus: PDDL to Effects

• Translate to **positive** and **negative** effect axioms

• **load**(Box : b, Truck : t):

  - Effects:
  
  * [∃c. a = load(b, t)∧BoxIn(b, c, s)∧TruckIn(t, c, s)] ⊃ BoxOn(b, t, do(a, s))
  * [∃t. a = load(b, t)∧BoxIn(b, c, s)∧TruckIn(t, c, s)] ⊃ ¬BoxIn(b, c, do(a, s))

• **unload**(Box : b, Truck : t):

  - Effects:
  
  * [∃t. a = unload(b, t)∧BoxOn(b, t, s)∧TruckIn(t, c, s)] ⊃ BoxIn(b, c, do(a, s))
  * [∃c. a = unload(b, t)∧BoxOn(b, t, s)∧TruckIn(t, c, s)] ⊃ ¬BoxOn(b, t, do(a, s))

• **drive**(Truck : t, City : c):

  - Effects:
  
  * [∃c1. a = drive(t, c) ∧ TruckIn(t, c1, s)] ⊃ TruckIn(t, c, do(a, s))
  * [∃c. a = drive(t, c) ∧ TruckIn(t, c1, s)] ⊃ ¬TruckIn(t, c1, 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

\[ [(C_1 \supset F) \land (C_2 \supset F)] \equiv [(C_1 \lor C_2) \supset F] \]
Frame Problem

• Now we have **positive** and **negative** effects

\[ \gamma^+_{\text{BoxIn}}(\vec{x}, a, s) \supset \text{BoxIn}(\vec{x}, do(a, s)) \]
\[ \gamma^+_{\text{TruckIn}}(\vec{x}, a, s) \supset \text{TIn}(\vec{x}, do(a, s)) \]
\[ \gamma^+_{\text{BoxOn}}(\vec{x}, a, s) \supset \text{BoxOn}(\vec{x}, do(a, s)) \]

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:

\[
\begin{align*}
&\gamma^+_F(\vec{x}, a, s) \supset F(\vec{x}, do(a, s)) \\
&\gamma^-_F(\vec{x}, a, s) \supset \neg F(\vec{x}, do(a, s))
\end{align*}
\]

\[
F(\vec{x}, do(a, s)) \equiv \gamma^+_F(\vec{x}, a, s) \lor F(\vec{x}, s) \\
\land \neg \gamma^-_F(\vec{x}, a, s)
\]
SSAs

• Shorthand:

\[ F(\vec{x}, \text{do}(a, s)) \equiv \Phi_F(\vec{x}, a, s) \]

\[ \equiv \gamma^+_F(\vec{x}, a, s) \lor F(\vec{x}, s) \land \neg \gamma^-_F(\vec{x}, a, s) \]

• Reality check:

\[ \text{BoxOn}(b, t, \text{do}(a, s)) \equiv \Phi_{\text{BoxOn}}(b, t, a, s) \]

\[ \equiv [\exists c. a = \text{load}(b, t) \land \text{BoxIn}(b, c, s) \land \text{TruckIn}(t, c, s)] \]

\[ \lor \text{BoxOn}(b, t, s) \land \neg [\exists c. a = \text{unload}(b, t) \land \text{BoxOn}(b, t, s) \land \text{TruckIn}(t, c, s)] \]

\[ \text{BoxIn}(b, c, \text{do}(a, s)) \equiv \Phi_{\text{BoxIn}}(b, c, a, s) \]

\[ \equiv [\exists t. a = \text{unload}(b, t) \land \text{BoxOn}(b, t, s) \land \text{TruckIn}(t, c, s)] \]

\[ \lor \text{BoxIn}(b, c, s) \land \neg [\exists t. a = \text{load}(b, t) \land \text{BoxIn}(b, c, s) \land \text{TruckIn}(t, c, s)] \]

\[ \text{TruckIn}(t, c, \text{do}(a, s)) \equiv \Phi_{\text{TruckIn}}(t, c, a, s) \]

\[ \equiv [\exists c_1. a = \text{drive}(t, c) \land \text{TruckIn}(t, c_1, s)] \]

\[ \lor \text{TruckIn}(t, c, s) \land \neg [\exists c_1. a = \text{drive}(t, c) \land \text{TruckIn}(t, c_1, s)] \]
Regression

• Why have we defined SSAs?

• Regression:
  – If $\varphi$ held after action $a$
    then regression is the $\varphi'$ that held before action $a$

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

- Given $\exists b. \text{BoxIn}(b, \text{paris}, \text{do}(\text{unload}(b^*, t^*), s))$

- Regress through $\text{unload}(b^*, t^*)$

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

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

$$= \exists b. [\exists t. \text{unload}(b^*, t^*) = \text{unload}(b, t) \land \text{BoxOn}(b, t, s) \land \text{TruckIn}(t, \text{paris}, s)]$$

$$\lor \text{BoxIn}(b, \text{paris}, s)$$

$$\land \neg [\exists t. \text{unload}(b^*, t^*) = \text{load}(b, t) \land \text{BoxIn}(b, \text{paris}, s) \land \text{TruckIn}(t, \text{paris}, s)]$$

$$= [\exists b, t. b = b^* \land t = t^* \land \text{BoxOn}(b, t, s) \land \text{TruckIn}(t, \text{paris}, s)]$$

$$\lor \exists b. \text{BoxIn}(b, \text{paris}, s)$$

$$= [(\exists b. b = b^*) \land (\exists t. t = t^*) \land \text{BoxOn}(b^*, t^*, s) \land \text{TruckIn}(t^*, \text{paris}, s)]$$

$$\lor \exists b. \text{BoxIn}(b, \text{paris}, s)$$

$$= [\text{BoxOn}(b^*, t^*, s) \land \text{TruckIn}(t^*, \text{paris}, s)] \lor \exists b. \text{BoxIn}(b, \text{paris}, s)$$

**Note free vars $b^*, t^*$; why?**

**Make non-empty domain assumption**
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

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

$$\quad = \exists b^*, t^*. \ [BoxOn(b^*, t^*, s) \land TruckIn(t^*, \ paris, s)]$$

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

$$\quad = [\exists b^*, t^*. \ BoxOn(b^*, t^*, s) \land TruckIn(t^*, \ paris, s)]$$

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

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.,
  \[ \exists b. \text{BoxIn}(b, \text{paris}, s) \]

- 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, $\exists$ to tell whether goal will hold

\[
\exists b^*, t^*. BoxOn(b, t, s) \land TruckIn(t, paris, s)
\]
\[
\lor \exists b. BoxIn(b, paris, s)
\]

\[
\exists b^*, t^*. BoxOn(b, t, s) \land TruckIn(t, paris, s)
\]
\[
\lor \exists b. BoxIn(b, paris, s)
\]

Goal State: Captures initial state?

Captures initial state?
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

- **AlGOI in LOGic**
  - Search the space of sequential action plans
  - Regress actions to initial state to test reachability
  - *Restrict* action space with program:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>primitive action</td>
</tr>
<tr>
<td>$\phi$?</td>
<td>condition test</td>
</tr>
<tr>
<td>$(\delta_1, \delta_2)$</td>
<td>sequence</td>
</tr>
<tr>
<td>if $\phi$ then $\delta_2$ endIf</td>
<td>conditional</td>
</tr>
<tr>
<td>while $\phi$ then $\delta$ endWhile</td>
<td>loop</td>
</tr>
<tr>
<td>$(\delta_1</td>
<td>\delta_2)$</td>
</tr>
<tr>
<td>$\pi \bar{x} [\delta]$</td>
<td>nondeterministic choice of arguments</td>
</tr>
<tr>
<td>$\delta^*$</td>
<td>nondeterministic iteration</td>
</tr>
<tr>
<td>proc $\beta(\bar{x}) \delta$ endProc</td>
<td>procedure call definition</td>
</tr>
<tr>
<td>$\beta(\bar{t})$</td>
<td>procedure call</td>
</tr>
</tbody>
</table>
Golog Example

- **Golog Program:**

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

- **Diagram of Golog Planning:**

  - Initial state need not be fully known!
  - Program exploits first-order action abstraction!
  - Heavily restricted action sequences!

  *Initial State*
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 Programming

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 \gamma 

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 $\gamma$ important ($\gamma = .9$ vs. $\gamma = .1$)
MDP Policy, Value, & Solution

• Define value of a policy $\pi$:

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

• Tells how much value you expect to get by following $\pi$ starting from state $s$

• MDP Optimal Solution:
  – Find optimal policy $\pi^*$ that maximizes value
  – Fortunately: $\exists \pi^*. \forall s, \pi. V_{\pi^*}(s) \geq V_\pi(s)$
Value Iteration: from finite to $\infty$ 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 $t$?
    \[
    V^t(s) := \max_{a \in A} \{ Q^t(s, a) \}
    \]
  - At $\infty$ horizon, get same value ($=V^*$)
    \[
    \lim_{t \to \infty} \max_s |V^t(s) - V^{t-1}(s)| = 0
    \]
- $\pi^*$ says act same at each decision stage for $\infty$ horizon!
Single Dynamic Programming Step

- Graphical view:

\[
\begin{align*}
  V^{t+1}(s_1) &= \max_{a_1} Q^t(s_1, a_1) \\
  V^t(s_1) &= Q^t(s_1, a_1) \\
  V^t(s_2) &= Q^t(s_1, a_2) \\
  V^t(s_3) &= Q^t(s_1, a_2) \\
  \sum_{s'} T(s, a, s') \\
  V^t(s_2) &= \sum_{s'} T(s, a, s')
\end{align*}
\]
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 $\pi_V$ 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 $s$, $\pi_V$ 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...

\[
\text{rCase}(s) = \begin{cases} 
1 & \forall b,c. \text{Dest}(b,c) \Rightarrow BIn(b,c,s) \\
0 & \neg " \end{cases}
\]

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

\[
\begin{array}{c|c}
\phi & 10 \\
\neg \phi & 20 \\
\end{array} \quad \oplus \quad \begin{array}{c|c}
\phi & 3 \\
\neg \phi & 4 \\
\end{array} = \begin{array}{c|c}
\phi \land \phi & 13 \\
\phi \land \neg \phi & 14 \\
\neg \phi \land \phi & 23 \\
\neg \phi \land \neg \phi & 24 \\
\end{array}
\]
Stochastic Actions & FODTR

- Stochastic actions using deterministic SitCalc:
  - User’s stochastic action: \( A(x) = \text{load}(b,t) \)
  - Nature’s choice: \( n(x) \in \{\text{loadS}(b,t), \text{loadF}(b,t)\} \)
  - Probability distribution over Nature’s choice:
    \[
    P(\text{loadS}(b,t) \mid \text{load}(b,t)) = \begin{array}{c|c}
    \text{snow}(s) & .1 \\
    \neg\text{snow}(s) & .6 
    \end{array}
    \]
    \[
    P(\text{loadF}(b,t) \mid \text{load}(b,t)) = \begin{array}{c|c}
    \text{snow}(s) & .9 \\
    \neg\text{snow}(s) & .4 
    \end{array}
    \]

- First-order decision-theoretic regression
  - FODTR = expectation of regression:
    \[
    FODTR[v\text{Case}(s),A(x)] = \mathbb{E}_{P(n(x)\mid A(x))} [\text{Regr}[v\text{Case}(s),n(x)]]
    \]
Q-functions and Backups

- FODTR almost gives us a Q-function

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

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

- Specify a backup operator for this

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

- Idea: if \textit{exists} action instance that achieves value
- Yields a first-order Q-function
Symbolic Dynamic Programming

- What value if 0-stages-to-go?
  - Obviously $V^0(s) = r\text{Case}(s)$

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

  \[
  V^1(s) = \max_s \begin{cases} \varphi_1 & 9 \\ \varphi_2 & 0 \\ \varphi_3 & 3 \\ \varphi_4 & 1 \end{cases}
  = B^{A_1}[r\text{Case}(s)]
  \]
  - 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 $V^0(s) = r\text{Case}(s)$

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

<table>
<thead>
<tr>
<th>$\phi_1$</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi_2$</td>
<td>0</td>
</tr>
<tr>
<td>$\phi_3$</td>
<td>3</td>
</tr>
<tr>
<td>$\phi_4$</td>
<td>1</td>
</tr>
</tbody>
</table>

$V^1(s) = B^A_1[r\text{Case}(s)]$

- 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 $V^0(s) = r\text{Case}(s)$

• What value if 1-stage-to-go?
  – We know value for each action
    
    $V^1(s) = \begin{array}{c|c}
    \varphi_1 & 9 \\
    \text{else } \varphi_3 & 3 \\
    \end{array}$
    $= B^{A_1}[r\text{Case}(s)]$

    $= \begin{array}{c|c}
    \varphi_1 & 9 \\
    \varphi_2 & 0 \\
    \end{array}$
    $= B^{A_2}[r\text{Case}(s)]$

    – 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 $V^0(s) = r\text{Case}(s)$

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

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

  - Now just need max for every state

  - $V^1(s) = B^A_1[r\text{Case}(s)] = B^A_2[r\text{Case}(s)]$

- **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 $V_0(s) = r\text{Case}(s)$

• 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 $(V_{n+1} \ominus V_n) < \epsilon$
First-order ADDs

• Want to compactly represent:

\[
\text{case} = \begin{cases} 
1 & \exists x. [A(x) \lor \forall y. A(x) \land B(x) \land \neg A(y)] \\
0 & \neg " \end{cases}
\]

• Push down quantifiers, expose prop. structure:

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

<table>
<thead>
<tr>
<th>Var</th>
<th>( \equiv ) Var ( \Leftrightarrow ) FOL KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>[\exists x. A(x)]</td>
</tr>
<tr>
<td>( b )</td>
<td>[\exists x. A(x) \land B(x)]</td>
</tr>
</tbody>
</table>

\[
\text{case} = \begin{cases} 
1 & a \lor (b \land \neg a) \\
0 & \neg " \end{cases}
\]

• Convert to first-order ADD

\[
\text{case} = \begin{cases} 
1 & a \\
0 & 0 \end{cases}
\]

[Sanner, Thesis]
Results for SDP with FOADDS

- Replace case with FO(A)ADDS, e.g. BoxWorld
  \[ rCase(s) = \exists b. BIn(b, Paris, s) \]

- Use FO(A)ADD ops for structured SDP (using \( \gamma = .9 \))
  \[ vCase(s) = \exists b. BIn(b, Paris, s) \]
  \[ 100 : \text{noop} \]
  \[ 89 : \text{unload}(b, t) \]
  \[ 80 : \text{drive}(t, Paris) \]
  \[ 72 : \text{load}(b, t) \] ...
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)*

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

• 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)*

Kristian covers this.

Saket covers this.

3rd place in ICAPS IPPC5 (after FPG, FF-Replan)
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 \emph{a priori}…
  – first-order solution must consider $\infty$ topologies
    • e.g., if \textit{Moscow} reachable from \textit{Rome} in five steps…
  – in general case, leads to $\infty$ values / policy
Caveats of First-order Planning

• Universal Rewards

\[ R(s) = \begin{cases} \forall b, c. \text{Dst}(b, c) \rightarrow \text{BoxIn}(b, c, s) : 1 \\ \neg " \rightarrow 0 \end{cases} \]

• Value function must distinguish \( \infty \) cases

\[ V^t(s) = \begin{cases} \forall b, c. \text{Dst}(b, c) \rightarrow \text{BoxIn}(b, c, s) : 1 \\ \text{One box not at destination} : \gamma \\ \text{Two boxes not at destination} : \gamma^2 \\ \vdots \\ t - 1 \text{ boxes not at destination} : \gamma^{t-1} \end{cases} \]

• Policy will also likely be \( \infty \)
  – 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 3\textsuperscript{rd} 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