Stochastic Actions, Probabilities, and Markov Decision Processes in the Situation Calculus

Ray Reiter
Department of Computer Science
University of Toronto

Motivation and Background

- The Toronto dialect of the sitcalc relies on *successor state axioms* to solve the frame problem.

  $$F(\vec{x}, do(a, s)) \equiv \phi_F(\vec{x}, a, s).$$

- In other words, F's truth value in the next situation $do(a, s)$ is uniquely determined by the current situation s.
- Therefore, these presuppose *deterministic* actions.
- So how can this framework support nondeterministic actions, like flipping a coin, with their associated outcome probabilities?
- There are many approaches (e.g. Bacchus et al, Pirri et al, Pinto et al, Poole, etc).

The Toronto Trick

- Consider a coffee and mail delivery robot.
- Introduce a $stochastic$ action $giveCoffee(p)$—the robot gives a cup of coffee to person p.
- This may succeed or fail (e.g., the robot spills the coffee) with known probabilities.
- Decompose $giveCoffee(p)$ into two $deterministic$ actions
  1. $giveCoffeeS(p)$—the action succeeds.
  2. $giveCoffeeF(p)$—it fails.
- View $giveCoffee(p)$ to be under the robot's control; it can choose to perform, or withhold this action.
- If it elects to perform it, nature steps in and does exactly one of the deterministic actions $giveCoffeeS(p)$, or $giveCoffeeF(p)$.

$$choice(giveCoffee(p), a) \stackrel{def}{=} a = giveCoffeeS(p) \vee a = giveCoffeeF(p).$$

- Associate probabilities with which of these she actually does.
- Introduce one other stochastic action $go(l)$—the robot goes to location l. It too can succeed or fail:

$$choice(go(l), a) \stackrel{def}{=} a = endUpAt(l) \vee a = getLost(l).$$

- Introduce two deterministic agent actions:

$$choice(puMail, a) \stackrel{def}{=} a = puMail.$$

$$choice(giveMail(p), a) \stackrel{def}{=} a = giveMail(p).$$

- Introduce some fluents:
  - $mailPresent(p, s)$.
  - $carryingMail(p, s)$.
  - $loc(s)$.
  - $coffeeRequested(p, s)$.
- Because nature's actions are deterministic, it is predictable how they change the state of the world.

$$mailPresent(p, do(a, s)) \equiv mailPresent(p, s) \wedge a \neq puMail,$$

$$carryingMail(p, do(a, s)) \equiv a = puMail \wedge mailPresent(p) \vee carryingMail(p, s) \wedge a \neq giveMail(p),$$

$$coffeeRequested(p, do(a, s)) \equiv coffeeRequested(p, s) \wedge a \neq giveCoffeeS(p),$$

$$\begin{aligned} loc(do(a, s)) = l \equiv\ & a = endUpAt(l) \vee \\ & (\exists l')a = getLost(l') \wedge l = BlackHole \vee \\ & loc(s) = BlackHole \wedge l = BlackHole \vee \\ & loc(s) \neq BlackHole \wedge \\ & \quad \neg(\exists l')[a = endUpAt(l') \wedge loc(s) \neq l'] \wedge \\ & \quad \neg(\exists l')a = getLost(l') \wedge l = loc(s). \end{aligned}$$

- Nature's actions have preconditions:

$$Poss(endUpAt(l), s) \equiv$$
$$l \neq BlackHole \wedge loc(s) \neq BlackHole \wedge loc(s) \neq l,$$

$$Poss(getLost(l), s) \equiv$$
$$l \neq BlackHole \wedge loc(s) \neq BlackHole \wedge loc(s) \neq l,$$

$$Poss(puMail, s) \equiv loc(s) = MR \wedge (\exists p) mailPresent(p, s),$$

$$Poss(giveMail(p), s) \equiv carryingMail(p, s) \wedge loc(s) = office(p),$$

$$Poss(giveCoffeeS(p), s) \equiv$$
$$coffeeRequested(p, s) \wedge loc(s) = office(p),$$

$$Poss(giveCoffeeF(p), s) \equiv$$
$$coffeeRequested(p, s) \wedge loc(s) = office(p).$$

- **The main point:**

This is a standard Toronto sitcalc theory with action precondition and successor state axioms, and deterministic actions.

Representing Probabilities

- It's unknown which of nature's choices actually gets done, but their probabilities are known.
- $prob(a, \beta, s)$: the probability that nature selects deterministic action a as the outcome of stochastic action β in situation s.

  $$prob(a, \beta, s) = p \stackrel{def}{=} choice(\beta, a) \wedge Poss(a, s) \wedge p = prob_0(a, \beta, s) \vee [\neg choice(\beta, a) \vee \neg Poss(a, s)] \wedge p = 0.$$

- $prob_0(a, \beta, s)$: the probability that nature selects a as the outcome of stochastic action β, given that a is one of nature's choices for β and, moreover, *that a is possible in s.*

  This modularizes the axiomatizer's job.

- For the delivery robot:

  $$prob_0(giveCoffeeS(p), giveCoffee(p), s) \stackrel{def}{=} 0.95,$$
  $$prob_0(giveCoffeeF(p), giveCoffee(p), s) \stackrel{def}{=} 0.05,$$
  $$prob_0(puMail, puMail, s) \stackrel{def}{=} 1,$$
  $$prob_0(giveMail(p), giveMail(p), s) \stackrel{def}{=} 1,$$
  $$prob_0(endUpAt(l), go(l), s) \stackrel{def}{=} \frac{1000}{1000 + dist(loc(s), l)},$$
  $$prob_0(getLost(l), go(l), s) \stackrel{def}{=} \frac{dist(loc(s), l)}{1000 + dist(loc(s), l)}.$$

Derived Probabilities

- What is the probability of some state of affairs after an agent performs a sequence of stochastic actions?
- **Example:** Consider the sequence

  $go(office(Sue))\,;\, giveMail(Sue)\,;\, giveCoffee(Sue)\,;$
  $giveCoffee(Sue).$

  - What is the probability that Sue has coffee after this program is performed?

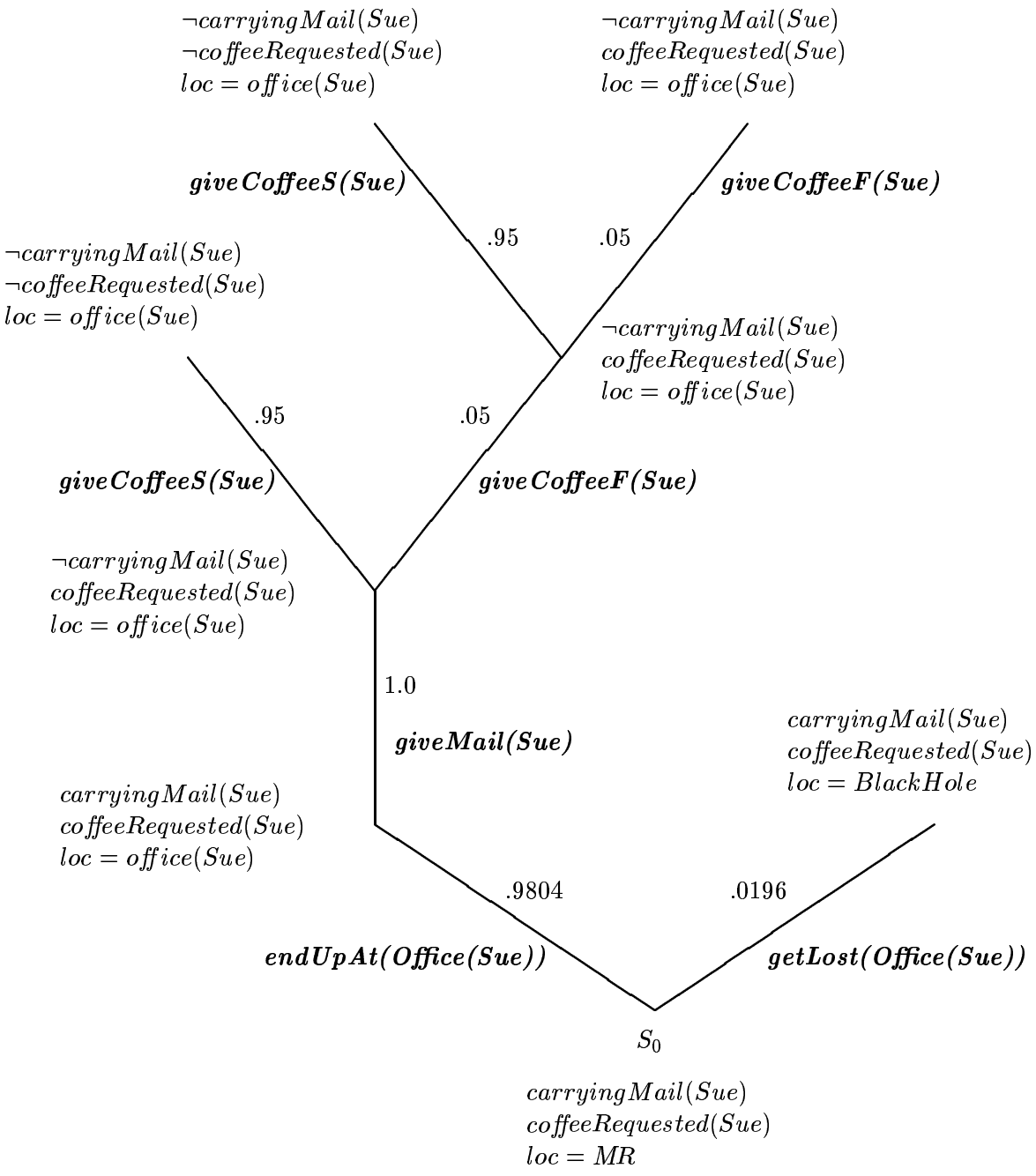

Figure 1: The tree of possible outcomes for stochastic action sequence $go(office(Sue))$; $giveMail(Sue)$; $giveCoffee(Sue)$; $giveCoffee(Sue)$).

stGolog: Stochastic Golog

- Generalize the above account of stochastic action sequences to include *programs*.
- Get Golog, with stochastic actions instead of deterministic primitive actions, and without nondeterministic choice.

  $go(office(Sue))$;
  if $loc = BlackHole$ **then** $stop$
  else if $carryingMail(Sue)$ **then** $giveMail(Sue)$;
  if $coffeeRequested(Sue)$ **then** $giveCoffee(Sue)$;
  if $hasCoffee(Sue)$ **then** $stop$
  else $giveCoffee(Sue)$

- $stDo(\gamma, p, s, s')$: s' is one of the situations that can be reached by executing the stGolog program γ starting in situation s, and p is the probability of ending up in s'.
- **Probabilistic projection problem:** What is the probability of ψ after executing the stGolog program γ?

  $$probF(\psi, \gamma) \stackrel{def}{=} \sum_{\{(p,\sigma)|\mathcal{D} \models stDo(\gamma,p,S_0,\sigma) \wedge \psi[\sigma]\}} p.$$

- Can give a sitcalc specification for the semantics of stGolog, and an implementation for an interpreter, and therefore, for solving probabilistic projection.

Two Spinoffs

1. Symbolic dynamic programming for solving MDPs.

   C. Boutilier, R. Reiter and R. Price. Symbolic dynamic programming for first-order Markov decision processes. *Proc. IJCAI'01*.

2. Decision theoretic Golog.

   C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. *Proc. AAAI'00*, 2000, 355–362.

Symbolic Dynamic Programming for First-Order MDPs

Craig Boutilier
Ray Reiter
Robert Price

Department of Computer Science
University of Toronto

Motivation

- The standard story for MDPs is *state-based*. Requires:
  1. Explicit enumeration of the state space +
  2. State transitions under stochastic actions +
  3. Probabilities for state transitions +
  4. Reward function for all states.
- Doesn't capture domain regularities.
- Huge state spaces.
- An improvement: Equivalence classes of identically behaving states are represented by propositional *sentences*.
- But propositionalizing a domain destroys regularities expressible with quantification.
- Moreover, the number of propositions and actions grows quickly with the domain size.
- Solution: First-order MDPs.
- How in a nutshell:
  1. Decompose stochastic actions into deterministic primitives.
  2. Axiomatize the domain's probabilities, reward function and deterministic primitives in the situation calculus.
  3. Value iteration = regression of value function through stochastic actions.

Example: A Simple Logistics Domain

- **Nature's Choices for Stochastic Actions:** For each stochastic action specify the deterministic choices available to nature.

  The stochastic $load(b,t)$ action can succeed—$loadS(b,t)$—or fail—$loadF(b,t)$:

  $$choice(load(b,t),a) \stackrel{def}{=} a = loadS(b,t) \vee a = loadF(b,t)$$

  Similarly for unloading and driving:

  $$choice(unload(b,t),a) \stackrel{def}{=} a = unloadS(b,t) \vee a = unloadF(b,t)$$

  $$choice(drive(t,c),a) \stackrel{def}{=} a = driveS(t,c) \vee a = driveF(t,c)$$

- These deterministic outcomes have known probabilities:

  $$prob(loadS(b,t), load(b,t), s) = 0.99$$

  $$prob(loadF(b,t), load(b,t), s) = 0.01$$

  $$prob(unloadS(b,t), unload(b,t), s) = p \equiv Rain(s) \wedge p = 0.7 \vee \neg Rain(s) \wedge p = 0.9$$

  $$prob(unloadF(b,t), unload(b,t), s) = 1 - prob(unloadS(b,t), unload(b,t), s)$$

  $$prob(driveS(t,c), drive(t,c), s) = 0.99$$

  $$prob(driveF(t,c), drive(t,c), s) = 0.01$$

Logistics Example Continued

- Because these choices by nature are deterministic, we can predict how they affect the world. They have "classical" successor state and action precondition axioms:

  Successor State Axioms:

  $$On(b,t,do(a,s)) \equiv a = loadS(b,t) \vee On(b,t,s) \wedge a \neq unloadS(b,t)$$

  $$TIn(t,c,do(a,s)) \equiv a = driveS(t,c) \vee TIn(t,c) \wedge \neg(\exists c')a = driveS(t,c')$$

  $$BIn(b,c,do(a,s)) \equiv (\exists t)[TIn(t,c,s) \wedge a = unloadS(b,t)] \vee BIn(b,c,s) \wedge \neg(\exists t)a = loadS(b,t)$$

  Action Preconditions for Deterministic Actions:

  $$Poss(loadS(b,t),s) \equiv (\exists c).BIn(b,c,s) \wedge TIn(t,c,s)$$

  $$Poss(loadF(b,t),s) \equiv (\exists c).BIn(b,c,s) \wedge TIn(t,c,s)$$

  $$Poss(unloadS(b,t),s) \equiv On(b,t,s)$$

  $$Poss(unloadF(b,t),s) \equiv On(b,t,s)$$

  $$Poss(driveS(t,c),s) \equiv true$$

  $$Poss(loadF(b,t),s) \equiv (\exists c).BIn(b,c,s) \wedge TIn(t,c,s)$$

  $$Poss(unloadS(b,t),s) \equiv On(b,t,s)$$

  $$Poss(unloadF(b,t),s) \equiv On(b,t,s)$$

  $$Poss(driveS(t,c),s) \equiv true$$

$$Poss(driveF(t, c), s) \equiv true$$

- *Nowhere do these axioms restrict the domain of discourse to some prespecified set of trucks, boxes, or cities.* There are models of these axioms with finitely many, infinitely many—even uncountably many—individuals.

Logistics Example Continued

- Reward functions are easily expressed in the sitcalc:

$$R(s) = r \equiv (\exists b) BIn(b, Paris, s) \wedge r = 10 \vee \\ \neg(\exists b) BIn(b, Paris, s) \wedge r = 0.$$

- This completes the sitcalc specification of the logistics MDP.

  - Compact.
  - Declarative.
  - Exploits the full relational and quantificational abilities of FOL.
  - Independent of domain size, and therefore, of the size of the state space.
  - No states!

- Next question: How to solve MDPs within this sitcalc representation.

Dynamic Programming for FOMDPs

To give the sitcalc analogue of classical, state-based dynamic programming, we need to:

1. Determine sitcalc expressions for the n-stage-to-go Q-functions for stochastic actions.
2. Determine the sitcalc expression for the n-stage-to-go value function. This is the max, over all stochastic actions, of these Q-functions.
3. Iterate until convergence.

Determining Q-functions for Stochastic Actions

- Suppose $V^n(s)$ is the n-stage-to-go value function, $R(s)$ is a reward function, and $A(\vec{x})$ is a stochastic action, with nature's choices $C_1(\vec{x}), \ldots, C_k(\vec{x})$. Here, variable s ranges over situations (= finite sequences of actions), *not* states.
- In the sitcalc representation for MDPs, the discounted $(n+1)$-stage-to-go Q-function for A in situation s is defined to be:

$$Q^{n+1}(A(\vec{x}), s) = R(s) + \gamma \cdot \sum_j prob(C_j(\vec{x}), A(\vec{x}), s) \cdot V^n(do(C_j(\vec{x}), s))$$

- Problem: The RHS depends on s—the current situation—and $do(C_j(\vec{x}), s))$, which are successor situations to s.
- Want an expression for $Q^{n+1}(A(\vec{x}), s)$ that depends only on s.
- Solution: *Regress* $V^n(do(C_j(\vec{x}), s))$ to a logically equivalent formula that depends only on s.
  1. Suppose $V^n(do(C_j(\vec{x}), s))$ mentions the fluent $F(\vec{t}, do(C_j(\vec{x}), s))$.
  2. Suppose F's successor state axiom is

     $$F(\vec{y}, do(a, s)) \equiv \phi(\vec{y}, a, s).$$

     s is the only situation term mentioned by $\phi(\vec{y}, a, s)$.
  3. Replace $F(\vec{t}, do(C_j(\vec{x}), s))$ in $V^n(do(C_j(\vec{x}), s))$ by $\phi(\vec{t}, C_j(\vec{x}), s)$.
  4. Repeat for all fluents with situation argument $do(C_j(\vec{x}), s)$ mentioned by $V^n(do(C_j(\vec{x}), s))$.
- Get a RHS that depends only on the current situation s.

Determining Value-functions

- This is easy. Suppose we have determined n-stage-to-go Q-functions $Q^n(A(\vec{x}), s)$, one for each stochastic action $A(\vec{x})$.
- Then the n-stage-to-go value function $V^n(s)$ is the max, over all stochastic actions $A(\vec{x})$, of these $Q^n(A(\vec{x}), s)$.
- Expressing this max in FOL:

$$V^n(s) = v \equiv (\exists a) Q^n(a, s) = v \wedge (\forall b) Q^n(b, s) \leq v.$$

A purely logical description, in the sitcalc, of the n-stage-to-go value function.

Implementing Value Iteration

$$V^n(s) = v \equiv (\exists a) Q^n(a, s) = v \wedge (\forall b) Q^n(b, s) \leq v.$$

- Not a very promising expression for implementation purposes.
- Introduce the *case notation* for value functions.

$$V(s) = case \begin{bmatrix} \sigma_1(s) & V_1 \\ \sigma_2(s) & V_2 \\ & \vdots \\ \sigma_k(s) & V_k \end{bmatrix}$$

- *There is a systematic, equivalence-preserving sequence of transformations applied to the expressions for the n-stage-to-go Q-functions and value function that yields a case expression for the n-stage-to-go value function.*
- These transformations and the resulting case expression are the basis for our implementation.
- See paper (IJCAI'01) for details.

Optimal Value Function for Logistics Example

$$V(s) =$$

$$
\begin{array}{ll}
 & \left[\begin{array}{ll}
\exists b BIn(Paris, b, s) & 10.0 \\
\neg Rain(s) \wedge \neg \exists b BIn(Paris, b, s) \wedge & \\
\quad \exists b, t. On(b, t, s) \wedge TIn(t, Paris, s) & 5.56 \\
Rain(s) \wedge \neg \exists b BIn(Paris, b, s) \wedge & \\
\quad \exists b, t. On(b, t, s) \wedge TIn(t, Paris, s) & 4.29 \\
\neg Rain(s) \wedge \exists b, t On(b, t, s) \wedge \neg \exists b BIn(Paris, b, s) \wedge & \\
\quad \neg \exists b, t. On(b, t, s) \wedge TIn(t, Paris, s) & 2.53 \\
\neg Rain(s) \wedge \exists b, t, c[BIn(c, s) \wedge TIn(c, s)] \wedge & \\
\quad \neg \exists b, t On(b, t, s) \wedge \neg \exists b BIn(Paris, b, s) & 1.52 \\
Rain(s) \wedge \exists b, t On(b, t, s) \wedge \neg \exists b BIn(Paris, b, s) \wedge & \\
\quad \neg \exists b, t. On(b, t, s) \wedge TIn(t, Paris, s) & 1.26 \\
\neg \exists b BIn(Paris, b, s) \wedge \neg \exists b, t On(b, t, s) \wedge & \\
\quad [Rain(s) \vee \neg \exists b, t, c. BIn(c, s) \wedge TIn(c, s)] & 0.0
\end{array}\right] \\
case &
\end{array}
$$

Implementation

- We have a preliminary implementation that solves simple problems, namely, MDPs whose optimal policies require only a modest number of distinct state descriptions.
- Relies heavily on first-order formula simplification. Uses the leanTap theorem-prover enhanced with equality.
- Complex MDPs require too many state space abstractions for our simple-minded implementation that relies on explicitly enumerating these abstractions.
- We are now reformulating our approach. First-order BDDs? To be followed by an implementation.

Summary

- We have proposed a sitcalc specification for MDPs, and a logical characterization of dynamic programming in which first-order sentences serve as the mechanism for state space abstraction.
- In this framework, there are no states, only *state descriptions*.

A bridge between UAI and classical, logic-based KR

Decision Theoretic Golog

Craig Boutilier
Ray Reiter
Misha Soutchanski

Department of Computer Science
University of Toronto

Sebastian Thrun

Department of Computer Science
Carnegie-Mellon University

Motivation

- Golog:
  - Primitive actions are deterministic.
  - High degree of nondeterministic program constructs.
  - No criteria to distinguish one execution trace from another. All execution traces of a given program are equivalent.
- When primitive actions are stochastic, and world states have decision-theoretic values, different execution traces do matter.
- We want nondeterministic program choices to be made in a way that maximizes expected value.
- We also want to give the robot programmer the ability to control the computation in directions she knows are optimal, thereby reducing the decision-theoretic planning involved in executing the program.
- DTGolog.

DTGolog: An Interpreter

- $bestDo(prog, s, policy, val, prob, h)$.
  - $prog$: current DTGolog program.
  - s: the current situation.
  - $policy$: the current optimal policy for the program. A conditional (if-then-else) statement, with sense actions.
  - val: the expected value of this policy.
  - $prob$: the probability of successful execution of $prog$ under $policy$.
  - h: the current horizon (how far we're prepared to look ahead).
- It's relatively straightforward to inductively define $bestDo$, and therefore, to implement a DTGolog interpreter.
- For recent theoretical and empirical results in controlling an RWI B21, see Misha Soutchanski's forthcoming Ph.D. thesis.