Action Understanding as Inverse Planning
Appendix

Chris L. Baker, Rebecca Saxe & Joshua B. Tenenbaum
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology

A Markov decision problems

This section formalizes the encoding of an agent's environment and goal into a Markov decision problem (MDP), and describes how this MDP can be solved efficiently by algorithms for rational planning. Let π be an agent's plan, referred to here (and in the MDP literature) as a *policy*, such that $P_\pi(a_t|s_t, g, w)$ is a probability distribution over actions a_t at time t, given the agent's state s_t at time t, the agent's goal g and world state w. This distribution formalizes $P(\mathsf{Actions}|\mathsf{Goal}, \mathsf{Environment})$, the expression for probabilistic planning sketched in the main text. The policy π encodes all goal-dependent plans the agent could make in a given environment.

We assume that agents' policies follow the principle of rationality. Within a goal-based MDP, this means that agents choose action sequences that minimize the expected cost to achieve their goals, given their beliefs about the environment. Let $C_{g,w}(a, s)$ be the environment- and goal-dependent cost to an agent of taking action a in state s. The expected cost to an agent of executing policy π starting from state s is given by the agent's *value function*, which sums the costs the agent is expected to incur over an infinite horizon:

$$V^\pi_{g,w}(s) = E_\pi\left[\sum_{t=1}^{\infty}\sum_{a_t} P_\pi(a_t|s_t, g, w)C_{g,w}(a_t, s_t)\middle| s_1 = s\right]. \quad (1)$$

In general, cost functions may differ between agents and environments. For the environments we consider, action costs are assumed to be proportional to the negative length of the resulting movement, and the Stay action incurs a small cost as well. We assume that agents stop incurring costs once they reach their goals, implying that rational agents will try to reach their goals as quickly as possible.

The state-action value function, or Q, defines the expected cost of taking action a_t from state s_t and executing policy π afterwards by averaging possible outcomes s_{t+1} caused by a_t:

$$Q^\pi_{g,w}(s_t, a_t) = \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t, w)V^\pi_{g,w}(s_{t+1}) + C_{g,w}(a_t, s_t). \quad (2)$$

Here, $P(s_{t+1}|s_t, a_t, w)$ is the state transition distribution, which specifies the probability of moving to state s_{t+1} from state s_t, as a result of action a_t, in world w. For simplicity, we assume state transitions are deterministic in our experiments.

The optimal action from any state is computed by greedily choosing the action that maximizes the Q-function. However, instead of this deterministic policy, we assume that agents have a probability distribution over actions associated with policy π:

$$P_\pi(a_t|s_t, g, w) \propto \exp\big(\beta Q^\pi_{g,w}(s_t, a_t)\big). \quad (3)$$

This type of policy is called a *Boltzmann policy*, which takes the "soft-max" of the Q-function, yielding an approximate principle of rationality, where β controls the amount of noise in the agent's actions.

The optimal Boltzmann policy π^* and the value function of π^* satisfy the Bellman equation [1] for all states:

$$V^{\pi^*}_{g,w}(s) = \sum_{a_t} P_{\pi^*}(a_t|s_t, g, w) Q^{\pi^*}_{g,w}(s_t, a_t). \tag{4}$$

These equations can be solved efficiently using the value iteration algorithm [2], which iteratively updates the left- and right-hand sides of Equation 4 for all states until convergence to a unique fixed point.

Given an agent's situation-specific policy, goal inferences and goal-based action predictions depend crucially on the prior over goals, corresponding to $P(\mathsf{Goal}|\mathsf{Environment})$ in the main text. This prior is instantiated by different models within the inverse planning framework. In the next section, we describe three inverse planning models based on different hypotheses about people's prior knowledge about goals, denoted M1(β), M2(β,γ), and M3(β,κ), which roughly correspond to the three kinds of explanations we offered for the woman's anomalous behavior in the introductory vignette in the main text. In addition to these models, we also consider a simple heuristic alternative, denoted H(β), that people might apply in action understanding. For each model, we will describe the computations involved in each of our three key tasks: online goal inference, retrospective goal inference and goal-based action prediction.

These three kinds of inferences all depend on applying Bayes' rule, in which we compare all possible goal hypotheses against each other, in terms of how well they explain an observed action sequence. Each goal hypothesis yields a different MDP that must be solved. In this paper, we exhaustively enumerate the full hypothesis space of goals under each model, solving the corresponding MDP for each one. This is computationally feasible for the simple environments and the restricted hypothesis spaces of goals that we consider here. Real world settings will likely require more complex goal priors which allow goals to be selected from a very large or infinite hypothesis space. In this case, full enumeration will not be possible, and efficient and accurate approximate inference schemes will be required [7, 9]. Likewise, in MDPs defined over complex environments with many state variables, the state space quickly grows too large to solve the MDP exactly. In these cases, approximate rational planning algorithms are necessary, and this is an active area of research [5, 4, 10, 8] within the field of artificial intelligence.

B Inverse planning models

B.1 Model 1: single underlying goal

Our first model, denoted M1(β), is our most basic instantiation of inverse planning. M1 assumes that agents have one underlying goal in each action sequence that must be inferred. A graphical model of M1 is shown in Fig. 1(a). Observations of agents are assumed to begin at time $t = 1$, with the agent occupying state s_1 in environment w. The agent is assumed to have an invariant goal g, which generates actions a_1 through a_{T-1} according to the policy from Equation 3, where the parameter β determines the agent's level of determinism. At high values of β, agents rarely deviate from the optimal path to their goals, but at low β values, agents' behavior is noisy, becoming a random walk at $\beta = 0$. Agents' actions generate state transitions, producing the state sequence $s_1, s_2, \ldots, s_T$. The objective of inverse planning is to invert this model, yielding inferences of agents' goals g, given observations of the state sequence $s_{1:T}$.

Given an observed state sequence and the environment, the distribution over the agent's goal in M1 is computed using Bayes' rule:

$$P(g|s_{1:T}, w) \propto P(s_{2:T}|s_1, g, w) P(g|w), \tag{5}$$

I constructing the environment: → class
- .length
- .width
- initial state
- goal
- cost Dictionary } implicitly contains all info about obstacles, available action space for each state in state space
- beta

input: grid length
grid width
number of obstacles

output: random goal
random initial state

(cost) Dictionary of Dictionaries: key = (x,y) coordinates on board {state set}
value = Dictionary of possible actions
key = {action set}
value = cost of each action

{ −1, $-1/\sqrt{2}$ −1, −100, 0 }
(horiz/vert, diagonal, stay cost, obstacle, goal)

II. Representations:

Value (S) — Dictionary
key = (x,y) state coordinate
value = $\sqrt{\pi}(s)$

Q(s,a) — Dict of Dicts
key = (x,y)
value = Dict of possible actions in state (x,y)
key: action
value = ①-value

Policy $P_\pi(\vec{a}|s)$ — Dict of Dicts
key = (x,y)
value = Dict of possible actions @ (x,y)
key = action
value = probability of taking that action

III functions

- getNextState (currentState, action):
  return (nextState)

- getActionSet (State):
  return (actionset of state)

- calculate QValue (state, action, valueTable)

```
nextState = getNextState (State, action)
ValueNextstate = valueTable [nextState]
  cost= self.costTable [State][action]
Qvalue =
        valueNextState + cost

return (Qvalue)
```

- Calculate QTable (valueTable)

```
for all state action pairs
    QTable [State][action] = calculate QValue (state,action, valueTable)
return QTable.
```

- calculate Policy PDF (State, QTable)

```
for each action in actionset (state):
    p ~ exp{ self.beta • QTable[state][action]}
p = p / sum(p)   } normalized to be a proper distribution
```

- calculate policyTable (QTable)

```
for all states in environment:
    calculate Policy PDF (current state)
```

- Calculate StateValue (policy PDF, QTable)state)

```
for all actions in getActionSet(State):
    StateValue (state) += Policy PDF [State][action] qTable[State][action]
return state value
```

- calculate ValueTable (policy Table, QTable)

```
∀ States
    calculate State value
return (valueTable)
```