On the Expressivity of Markov Reward

David Abel
DeepMind
dmabel@deepmind.com

Will Dabney
DeepMind
wdabney@deepmind.com

Anna Harutyunyan
DeepMind
harutyunyan@deepmind.com

Mark K. Ho
Department of Computer Science
Princeton University
mho@princeton.edu

Michael L. Littman
Department of Computer Science
Brown University
mlittman@cs.brown.edu

Doina Precup
DeepMind
doinap@deepmind.com

Satinder Singh
DeepMind
baveja@deepmind.com

Abstract

Reward is the driving force for reinforcement-learning agents. This paper is dedicated to understanding the expressivity of reward as a way to capture tasks that we would want an agent to perform. We frame this study around three new abstract notions of "task" that might be desirable: (1) a set of acceptable behaviors, (2) a partial ordering over behaviors, or (3) a partial ordering over trajectories. Our main results prove that while reward can express many of these tasks, there exist instances of each task type that no Markov reward function can capture. We then provide a set of polynomial-time algorithms that construct a Markov reward function that allows an agent to optimize tasks of each of these three types, and correctly determine when no such reward function exists. We conclude with an empirical study that corroborates and illustrates our theoretical findings.

1 Introduction

How are we to use algorithms for reinforcement learning (RL) to solve problems of relevance in the world? Reward plays a significant role as a general purpose signal: For any desired behavior, task, or other characteristic of agency, there must exist a reward signal that can incentivize an agent to learn to realize these desires. Indeed, the expressivity of reward is taken as a backdrop assumption that frames RL, sometimes called the reward hypothesis: "...all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward)" [52, 28]. In this paper, we establish first steps toward a systematic study of the reward hypothesis by examining the expressivity of reward as a signal. We proceed in three steps.

1. An Account of "Task". As rewards encode tasks, goals, or desires, we first ask, "what *is* a task?". We frame our study around a thought experiment (Figure 1) involving the interactions between a designer, Alice, and a learning agent, Bob, drawing inspiration from Ackley and Littman [2], Sorg [49], and Singh et al. [45]. In this thought experiment, we draw a distinction between how Alice thinks of a task (TASKQ) and the means by which Alice incentivizes Bob to pursue this task (EXPRESSIONQ). This distinction allows us to analyze the expressivity of reward as an answer to the latter question, conditioned on how we answer the former. Concretely, we study three answers to the TASKQ in the context of finite Markov Decision Processes (MDPs): A task is either (1) a set of

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

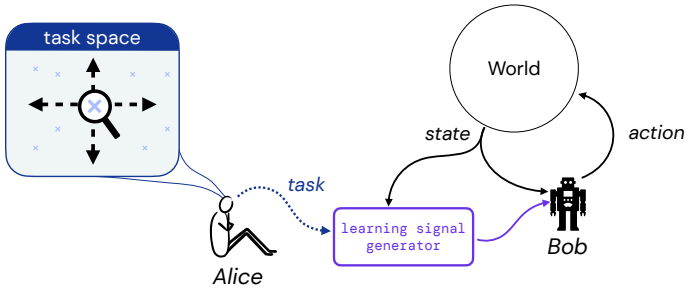


Figure 1: Alice, Bob, and the artifacts of task definition (blue) and task expression (purple).

acceptable behaviors (policies), (2) a partial ordering over behaviors, or (3) a partial ordering over trajectories. Further detail and motivation for these task types is provided in Section 3, but broadly they can be viewed as generalizations of typical notions of task such as a choice of goal or optimal behavior. Given these three answers to the TASKQ, we then examine the *expressivity* of reward.

2. Expressivity of Markov Reward. The core of our study asks whether there are tasks Alice would like to convey—as captured by the answers to the TASKQ—that admit no characterization in terms of a Markov reward function. Our emphasis on Markov reward functions, as opposed to arbitrary history-based reward functions, is motivated by several factors. First, disciplines such as computer science, psychology, biology, and economics typically rely on a notion of reward as a numerical proxy for the *immediate* worth of states of affairs (such as the financial cost of buying a solar panel or the fitness benefits of a phenotype). Given an appropriate way to describe states of affairs, Markov reward functions can represent immediate worth in an intuitive manner that also allows for reasoning about combinations, sequences, or re-occurrences of such states of affairs. Second, it is not clear that general history-based rewards are a reasonable target for learning as they suffer from the curse of dimensionality in the length of the history. Lastly, Markov reward functions are the standard in RL. A rigorous analysis of which tasks they can and cannot convey may provide guidance into when it is necessary to draw on alternative formulations of a problem. Given our focus on Markov rewards, we treat a reward function as accurately *expressing* a task just when the value function it induces in an environment adheres to the constraints of a given task.

3. Main Results. We find that, for all three task types, there are environment–task pairs for which there is no Markov reward function that realizes the task (Theorem 4.1). In light of this finding, we design polynomial-time algorithms that can determine, for any given task and environment, whether a reward function exists in the environment that captures the task (Theorem 4.3). When such a reward function does exist, the algorithms also return it. Finally, we conduct simple experiments with these procedures to provide empirical insight into the expressivity of reward (Section 5).

Collectively, our results demonstrate that there are tasks that cannot be expressed by Markov reward in a rigorous sense, but we can efficiently construct such reward functions when they do exist (and determine when they do not). We take these findings to shed light on the nature of reward maximization as a principle, and highlight many pathways for further investigation.

2 Background

RL defines the problem facing an agent that learns to improve its behavior over time by interacting with its environment. We make the typical assumption that the RL problem is well modeled by an agent interacting with a finite Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, R, T, \gamma, s_0)$. An MDP gives rise to deterministic behavioral policies, $\pi : \mathcal{S} \to \mathcal{A}$, and the value, $V^\pi : \mathcal{S} \to \mathbb{R}$, and action–value, $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, functions that measure their quality. We will refer to a Controlled Markov Process (CMP) as an MDP without a reward function, which we denote E for environment. We assume that all reward functions are deterministic, and may be a function of either state, state-action pairs, or state-action-state triples, but *not* history. Henceforth, we simply use "reward function" to refer to a deterministic Markov reward function for brevity, but note that more sophisticated settings beyond MDPs and deterministic Markov reward functions are important directions for future work. For more on MDPs or RL, see the books by Puterman [40] and Sutton and Barto [53] respectively.

2.1 Other Perspectives on Reward

We here briefly summarize relevant literature that provides distinct perspectives on reward.

Two Roles of Reward. As Sorg [49] identifies (Chapter 2), reward can both define the task the agent learns to solve, and define the "bread crumbs" that allow agents to efficiently learn to solve the task. This distinction has been raised elsewhere [2, 45, 46], and is similar to the extrinsic-intrinsic reward divide [44, 65]. Tools such as reward design [33, 50] or reward shaping [35] focus on offering more efficient learning in a variety of environments, so as to avoid issues of sparsity and long-term credit assignment. We concentrate primarily on reward's capacity to express a *task*, and defer learning dynamics to an (important) stage of future work.

Discounts, Expectations, and Rationality. Another important facet of reward is how it is used in producing behavior. The classical view offered by the Bellman equation (and the reward hypothesis) is that the quantity of interest to maximize is expected, discounted, cumulative reward. Yet it is possible to disentangle reward from the expectation [5], to attend only to ordinal [59] or maximal rewards [25], or to adopt different forms of discounting [60, 10]. In this work, we take the standard view that agents will seek to maximize *value* for a particular discount factor γ, but recognize that there are interesting directions beyond these commitments, such as inspecting the limits of reward in constrained MDPs as studied by Szepesvári [55]. We also note the particular importance of work by Pitis [39], who examines the relationship between classical decision theory [58] and MDPs by incorporating additional axioms that account for stochastic processes with discounting [23, 34, 47, 48]. Drawing inspiration from Pitis [39] and Sunehag and Hutter [51], we foresee valuable pathways for future work that further makes contact between RL and various axioms of rationality.

Preferences. In place of numerical rewards, preferences of different kinds may be used to evaluate agent's behaviors, drawing from the literature on preference-learning [24] and ordinal dynamic programming [7, 34, 47]. This premise gives rise to *preference-based reinforcement learning* (PbRL) in which an agent interacts with a CMP and receives evaluative signals in the form of preferences over states, actions, or trajectories. This kind of feedback inspires and closely parallels the task types we propose in this work. A comprehensive survey of PbRL by Wirth et al. [63] identifies critical differences in this setup from traditional RL, categorizes recent algorithmic approaches, and highlights important open questions. Recent work focuses on analysing the sample efficiency of such methods [64, 37] with close connections to learning from human feedback in real time [22, 31, 6].

Teaching and Inverse RL. The inverse RL (IRL) and apprenticeship learning literature examine the problem of learning directly from behavior [36, 1]. The classical problem of IRL is to identify which reward function (often up to an equivalence class) a given demonstrator is optimizing. We emphasize the relevance of two approaches: First, work by Syed et al. [54], who first illustrate the applicability of linear programming [21] to apprenticeship learning; and second, work by Amin et al. [4], who examine the *repeated* form of IRL. The methods of IRL have recently been expanded to include variations of *cooperative* IRL [13], and *assistive* learning [42], which offer different perspectives on how to frame interactive learning problems.

Reward Misspecification. Reward is also notoriously hard to specify. As pointed out by Littman et al. [29], "putting a meaningful dollar figure on scuffing a wall or dropping a clean fork is challenging." Along these lines, Hadfield-Menell et al. [15] identify cases in which well-intentioned designers create reward functions that produce unintended behavior [38]. MacGlashan et al. [32] find that human-provided rewards tend to depend on a learning agent's entire policy, rather than just the current state. Further, work by Hadfield-Menell et al. [14] and Kumar et al. [26] suggest that there are problems with reward as a learning mechanism due to misspecification and reward tampering [9]. These problems have given rise to approaches to *reward learning*, in which a reward function is inferred from some evidence such as behavior or comparisons thereof [19].

Other Notions of Task. As a final note, we highlight alternative approaches to task specification. Building on the Free Energy Principle [12, 11], Hafner et al. [16] consider a variety of task types in terms of minimization of distance to a desired target distribution [3]. Alternatively, Littman et al. [29] and Li et al. [27] propose variations of *linear temporal logic* (LTL) as a mechanism for specifying a task to RL agents, with related literature extending LTL to the multi-task [57] and multi-agent

[17] settings, or using reward machines for capturing task structure [18]. Jothimurugan et al. [20] take a similar approach and propose a task specification language for RL based on logical formulas that evaluate whether trajectories satisfy the task, similar in spirit to the logical task compositions framework developed by Tasse et al. [56]. Many of these notions of task are more general than those we consider. A natural direction for future work broadens our analysis to include these kinds of task.

3 An Account of Reward's Expressivity: The TaskQ and ExpressionQ

Consider an onlooker, Alice, and an earnest learning agent, Bob, engaged in the interaction pictured in Figure 1. Suppose that Alice has a particular task in mind that she would like Bob to learn to solve, and that Alice constructs a reward function to incentivize Bob to pursue this task. Here, Alice is playing the role of "all of what we mean by goals and purposes" for Bob to pursue, with Bob playing the role of the standard reward-maximizing RL agent.

Two Questions About Task. To give us leverage to study the expressivity of reward, it is useful to draw a distinction between two stages of this process: 1) Alice thinks of a task that she would like Bob to learn to solve, and 2) Alice creates a reward function (and perhaps chooses γ) that conveys the chosen task to Bob. We inspect these two separately, framed by the following two questions. The first we call the *task-definition question* (TaskQ) which asks: What *is* a task? The second we call the *task-expression question* (ExpressionQ) which asks: Which learning signal can be used as a mechanism for expressing any task to Bob?

Reward Answers The ExpressionQ. We suggest that it may be useful to treat reward as an answer to the ExpressionQ rather than the TaskQ. On this view, reward is treated as an expressive language for incentivizing reward-maximizing agents: Alice may attempt to translate any task into a reward function that incentivizes Bob to pursue the task, no matter which environment Bob inhabits, which task Alice has chosen, or how she has represented the task to herself. Indeed, it might be the case that Alice's knowledge of the task far exceeds Bob's representational or perceptual capacity. Alice may know every detail of the environment and define the task based on this holistic vantage, while Bob must learn to solve the task through interaction alone, relying only on a restricted class of functions for modeling and decision making.

Under this view, we can assess the expressivity of reward as an answer to the ExpressionQ conditioned on how we answer the TaskQ. For example, if the TaskQ is answered in terms of natural language descriptions of desired states of affairs, then reward may fail to convey the chosen task due to the apparent mismatch in abstraction between natural language and reward (though some work has studied such a proposal [30, 61]).

3.1 Answers to the TaskQ: What is a Task?

In RL, tasks are often associated with a choice of goal, reward function (R), reward-discount pair (R, γ), or perhaps a choice of optimal policy (alongside those task types surveyed previously, such as LTL). However, it is unclear whether these constructs capture the entirety of what we mean by "task".

For example, consider the Russell and Norvig [41] grid world: A 4×3 grid with one wall, one terminal fire state, and one terminal goal state (pictured with a particular reward function in Figure 4a). In such an environment, how are we to think of task? A standard view is that the task is to reach the goal as quickly as possible. This account, however, fails to distinguish between the *non*-optimal behaviors, such as the costly behavior of the agent moving directly into the fire and the neutral behavior of the agent spending its existence in the start state. Indeed, characterizing a task in terms of choice of π^* or goal fails to capture these distinctions. Our view is that a suitably rich account of task should allow for the characterization of this sort of preference, offering the flexibility to scale from specifying only the desirable behavior (or outcomes) to an arbitrary ordering over behaviors (or outcomes).

In light of these considerations, we propose three answers to the TaskQ that can convey general preferences over behavior or outcome: 1) A set of acceptable policies, 2) A partial ordering over policies, or 3) A partial ordering over trajectories. We adopt these three as they can capture many kinds of task while also allowing a great deal of flexibility in the level of detail of the specification.

Name	*Notation*	*Generalizes*	*Constraints Induced by* $\mathscr{T}$
SOAP	Π_G	task-as-π^*	equal: $V^{\pi_g}(s_0) = V^{\pi_{g'}}(s_0) > V^{\pi_b}(s_0), \forall_{\pi_g, \pi_{g'} \in \Pi_G, \pi_b \in \Pi_B}$ range: $V^{\pi_g}(s_0) > V^{\pi_b}(s_0), \forall_{\pi_g \in \Pi_G, \pi_b \in \Pi_B}$
PO	L_Π	SOAP	$(\pi_1 \oplus \pi_2) \in L_\Pi \implies V^{\pi_1}(s_0) \oplus V^{\pi_2}(s_0)$
TO	$L_{\tau,N}$	task-as-goal	$(\tau_1 \oplus \tau_2) \in L_{\tau,N} \implies G(\tau_1; s_0) \oplus G(\tau_2; s_0)$

Table 1: A summary of the three proposed task types. We further list the constraints that determine whether a reward function *realizes* each task type in an MDP, where we take $\oplus$ to be one of '$<$', '$>$', or '$=$', and G is the discounted return of the trajectory.

3.2 SOAPs, POs, and TOs

(SOAP) Set Of Acceptable Policies. A classical view of the equivalence of two reward functions is based on the optimal policies they induce. For instance, Ng et al. [35] develop potential-based reward shaping by inspecting which shaped reward signals will ensure that the optimal policy is unchanged. Extrapolating, it is natural to say that for any environment E, two reward functions are equivalent if the optimal policies they induce in E are the same. In this way, a task is viewed as a choice of optimal policy. As discussed in the grid world example above, this notion of task fails to allow for the specification of the quality of other behaviors. For this reason, we generalize task-as-optimal-policy to a *set of acceptable policies*, defined as follows.

Definition 3.1. *A set of acceptable policies (SOAP) is a non-empty subset of the deterministic policies,* $\Pi_G \subseteq \Pi$*, with* Π *the set of all deterministic mappings from* $\mathcal{S}$ *to* $\mathcal{A}$ *for a given* E*.*

With one task type defined, it is important to address what it means for a reward function to properly *realize*, *express*, or *capture* a task in a given environment. We offer the following account.

Definition 3.2. *A reward function is said to realize a task* $\mathscr{T}$ *in an environment* E *just when the start-state value (or return) induced by the reward function exactly adheres to the constraints of* $\mathscr{T}$*.*

Precise conditions for the realization of each task type are provided alongside each task definition, with a summary presented in column four of Table 1.

For SOAPs, we take the start-state value $V^\pi(s_0)$ to be the mechanism by which a reward function realizes a SOAP. That is, for a given E and Π_G, a reward function R is said to *realize* the Π_G in E when the start-state value function is optimal for all good policies, and strictly higher than the start-state value of all other policies. It is clear that SOAP strictly generalizes a task in terms of a choice of optimal policy, as captured by the SOAP $\Pi_G = \{\pi^*\}$.

We note that there are two natural ways for a reward function to realize a SOAP: First, each $\pi_g \in \Pi_G$ has *optimal* start-state value and all other policies are sub-optimal. We call this type *equal*-SOAP, or just SOAP for brevity. Alternatively, we might only require that the acceptable policies are each *near*-optimal, but are allowed to differ in start-state value so long as they are all better than *every* bad policy $\pi_b \in \Pi_B$. That is, in this second kind, there exists an $\epsilon \geq 0$ such that every $\pi_g \in \Pi_G$ is ϵ-optimal in start-state value, $V^*(s_0) - V^{\pi_g}(s_0) \leq \epsilon$, while all other policies are worse. We call this second realization condition *range*-SOAP. We note that the range realization generalizes the equal one: Every equal-SOAP is a range-SOAP (by letting $\epsilon = 0$). However, there exist range-SOAPs that are expressible by Markov rewards that are *not* realizable as an equal-SOAP. We illustrate this fact with the following proposition. All proofs are presented in Appendix B.

Proposition 3.1. *There exists a CMP,* E*, and choice of* Π_G *such that* Π_G *can be realized under the range-SOAP criterion, but cannot be realized under the equal-SOAP criterion.*

One such CMP is pictured Figure 2b. Consider the SOAP $\Pi_G = \{\pi_{11}, \pi_{12}, \pi_{21}\}$: Under the equal-SOAP criterion, if each of these three policies are made optimal, any reward function will *also* make π_{22} (the only bad policy) optimal as well. In contrast, for the range criterion, we can choose a reward function that assigns lower rewards to a_2 than a_1 in both states. In general, we take the equal-SOAP realization as canonical, as it is naturally subsumed by our next task type.

(PO) Partial Ordering on Policies. Next, we suppose that Alice chooses a *partial ordering* on the deterministic policy space. That is, Alice might identify a some great policies, some good, and some bad policies to strictly avoid, and remain indifferent to the rest. POs strictly generalize equal SOAPs, as any such SOAP is a special choice of PO with only two equivalence classes. We offer the following definition of a PO.

Definition 3.3. *A policy order (PO) of the deterministic policies* Π *is a partial order, denoted* L_Π.

As with SOAPs, we take the start-state value $V^\pi(s_0)$ induced by a reward function R as the mechanism by which policies are ordered. That is, given E and L_Π, we say that a reward function R *realizes* L_Π in E if and only if the resulting MDP, $M = (E, R)$, produces a start-state value function that orders Π according to L_Π.

(TO) Partial Ordering on Trajectories. A natural generalization of goal specification enriches a notion of task to include the details of how a goal is satisfied—that is, for Alice to relay some preference over trajectory space [62], as is done in preference based RL [63]. Concretely, we suppose Alice specifies a partial ordering on length N trajectories of (s, a) pairs, defined as follows.

Definition 3.4. *A trajectory ordering (TO) of length* $N \in \mathbb{N}$ *is a partial ordering* $L_{\tau,N}$*, with each trajectory* τ *consisting of* N *state–action pairs,* $\{(s_0, a_0), \ldots, (a_{N-1}, s_{N-1})\}$*, with* s_0 *the start state.*

As with PO, we say that a reward function realizes a trajectory ordering $L_{\tau,N}$ if the ordering determined by each trajectory's cumulative discounted N-step return from s_0, denoted $G(\tau; s_0)$, matches that of the given $L_{\tau,N}$. We note that trajectory orderings can generalize goal-based tasks at the expense of a larger specification. For instance, a TO can convey the task, "Safely reach the goal in less than thirty steps, or just get to the subgoal in less than twenty steps."

Recap. We propose to assess the expressivity of reward by first answering the TASKQ in terms of SOAPs, POs, or TOs, as summarized by Table 1. We say that a task $\mathscr{T}$ is *realized* in an environment E under reward function R if the start-state value function (or return) produced by R imposes the constraints specified by $\mathscr{T}$, and are interested in whether reward can always realize a given task in any choice of E. We make a number of assumptions along the way, including: (1) Reward functions are Markov and deterministic, (2) Policies of interest are deterministic, (3) The environment is a finite CMP, (4) γ is part of the environment, (5) We ignore reward's role in shaping the *learning process*, (6) Start-state value or return is the appropriate mechanism to determine if a reward function realizes a given task. Relaxation of these assumptions is a critical direction for future work.

4 Analysis: The Expressivity of Markov Reward

With our definitions and objectives in place, we now present our main results.

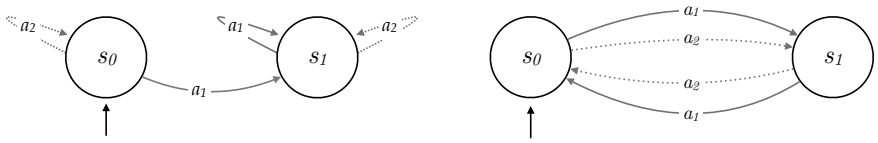
4.1 Express SOAPs, POs, and TOs

We first ask whether reward can always realize a given SOAP, PO, or TO, for an arbitrary E. Our first result states that the answer is "no"—there are tasks that cannot be realized by any reward function.

Theorem 4.1. *For each of SOAP, PO, and TO, there exist* $(E, \mathscr{T})$ *pairs for which no Markow reward function realizes* $\mathscr{T}$ *in* E.

Thus, reward is incapable of capturing certain tasks. What tasks are they, precisely? Intuitively, inexpressible tasks involve policies or trajectories that *must* be correlated in *value* in an MDP. That is, if two policies are nearly identical in behavior, it is unlikely that reward can capture the PO that places them at opposite ends of the ordering. A simple example is the "always move the same direction" task in a grid world, with state defined as an (x, y) pair. The SOAP $\Pi_G = \{\pi_\leftarrow, \pi_\uparrow, \pi_\rightarrow, \pi_\downarrow\}$ conveys this task, but no Markov reward function can make these policies strictly higher in value than all others.

Example: Inexpressible SOAPs. Observe the two CMPs pictured in Figure 2, depicting two kinds of inexpressible SOAPs. On the left, we consider the SOAP $\Pi_G = \{\pi_{21}\}$, containing only the policy that executes a_2 in the left state (s_0), and a_1 in the right (s_1). This SOAP is inexpressible through reward, but only because reward cannot distinguish the start-state value of π_{21} and π_{22} since the policies differ only in an unreachable state. This is reminiscent of Axiom 5 from Pitis [39], which



(a) Steady State Case (b) Entailment Case

Figure 2: Two CMPs in which there is a SOAP that is not expressible under any Markov reward function. On the left, $\Pi_G = \{\pi_{21}\}$ is not realizable, as π_{21} can not be made better than π_{22} because s_1 is never reached. On the right, the XOR-like-SOAP, $\Pi_G = \{\pi_{12}, \pi_{21}\}$ is not realizable: To make these two policies optimal, it is entailed that π_{22} and π_{11} must be optimal, too.

explicitly excludes preferences of this sort. On the right, we find a more interesting case: The chosen SOAP is similar to the XOR function, $\Pi_G = \{\pi_{12}, \pi_{21}\}$. Here, the task requires that the agent choose each action in exactly one state. However, there cannot exist a reward function that makes *only* these policies optimal, as by consequence, both policies π_{11} and π_{22} *must* be optimal as well.

Next, we show that Theorem 4.1 is not limited to a particular choice of transition function or γ.

Proposition 4.2. *There exist choices of $E_{\neg T} = (\mathcal{S}, \mathcal{A}, \gamma, s_0)$ or $E_{\neg\gamma} = (\mathcal{S}, \mathcal{A}, T, s_0)$, together with a task $\mathscr{T}$, such that there is no (T, R) pair that realizes $\mathscr{T}$ in $E_{\neg T}$ or (R, γ) in $E_{\neg\gamma}$.*

This result suggests that the scope of Theorem 4.1 is actually quite broad—even if the transition function or γ are taken as part of the reward specification, there are tasks that cannot be expressed. We suspect there are ways to give a precise characterization of *all* inexpressible tasks from an axiomatic perspective, which we hope to study in future work.

4.2 Constructive Algorithms: Task to Reward

We now analyze how to determine whether an appropriate reward function can be constructed for any $(E, \mathscr{T})$ pair. We pose a general form of the reward-design problem [33, 50, 8] as follows.

Definition 4.1. *The* REWARDDESIGN *problem is: **Given** $E = (\mathcal{S}, \mathcal{A}, T, \gamma, s_0)$, and a $\mathscr{T}$, **output** a reward function R_{alice} that ensures $\mathscr{T}$ is realized in $M = (E, R_{alice})$.*

Indeed, for all three task types, there is an efficient algorithm for solving the reward-design problem.

Theorem 4.3. *The* REWARDDESIGN *problem can be solved in polynomial time, for any finite E, and any $\mathscr{T}$, so long as reward functions with infinitely many outputs are considered.*

Therefore, for *any* choice of finite CMP, E, and a SOAP, PO, or TO, we can find a reward function that perfectly realizes the task in the given environment, if such a reward function exists. Each of the three algorithms are based on forming a linear program that matches the constraints of the given task type, which is why reward functions with infinitely many outputs are required. Pseudo-code for SOAP-based reward design is presented in Algorithm 1. Intuitively, the algorithms compute the discounted expected-state visitation distribution for a collection of policies; in the case of SOAP, for instance, these policies include Π_G and what we call the "fringe", the set of policies that differ from a $\pi_g \in \Pi_G$ by exactly one action. Then, we use these distributions to describe linear inequality constraints ensuring that the start-state value of the good policies are better than those of the fringe.

As highlighted by Theorem 4.1 there are SOAPs, POs, and TOs that are *not realizable*. Thus, it is important to determine how the algorithms mentioned in Theorem 4.3 will handle such cases. Our next corollary illustrates that the desirable outcome is achieved: For any E and $\mathscr{T}$, the algorithms will output a reward function that realizes $\mathscr{T}$ in E, or output '$\perp$' when no such function exists.

Corollary 4.4. *For any task $\mathscr{T}$ and environment E, deciding whether $\mathscr{T}$ is expressible in E is solvable in polynomial time.*

Together, Theorem 4.1 and Theorem 4.3 constitute our main results: There are environment–task pairs in which reward cannot express the chosen task for each of SOAPs, POs, and TOs. However, there are efficient algorithms for deciding whether a task is expressible, and for constructing the realizing reward function when it exists. We will study the use of one of these algorithms in Section 5, but first attend to other aspects of reward's expressivity.

Algorithm 1 SOAP Reward Design

INPUT: $E = (\mathcal{S}, \mathcal{A}, T, \gamma, s_0), \Pi_G$.
OUTPUT: R, or $\perp$.

$\Pi_{\text{fringe}} = \texttt{compute\_fringe}(\Pi_G)$
for $\pi_{g,i} \in \Pi_G$ **do** ▷ Compute state-visitation distributions.
    $\rho_{g,i} = \texttt{compute\_exp\_visit}(\pi_{g,i}, E)$

for $\pi_{f,i} \in \Pi_{\texttt{fringe}}$ **do**
    $\rho_{f,i} = \texttt{compute\_exp\_visit}(\pi_{f,i}, E)$

$C_{\text{eq}} = \{\}$ ▷ Make Equality Constraints.
for $\pi_{g,i} \in \Pi_G$ **do**
    $C_{\text{eq}}.\texttt{add}(\rho_{g,0}(s_0) \cdot X = \rho_{g,i}(s_0) \cdot X)$

$C_{\text{ineq}} = \{\}$ ▷ Make Inequality Constraints.
for $\pi_{f,j} \in \Pi_{\text{fringe}}$ **do**
    $C_{\text{ineq}}.\texttt{add}(\rho_{f,j}(s_0) \cdot X + \epsilon \leq \rho_{g,0}(s_0) \cdot X)$

$R_{\text{out}}, \epsilon_{\text{out}} = \texttt{linear\_programming}(\text{obj.} = \max \epsilon, \text{constraints} = C_{\text{ineq}}, C_{\text{eq}})$ ▷ Solve LP.

if $\epsilon_{\text{out}} > 0$ **then** ▷ Check if successful.
    return R_{out}
else
    return $\perp$

4.3 Other Aspects of Reward's Expressivity

We next briefly summarize other considerations about the expressivity of reward. As noted, Theorem 4.3 requires the use of a reward function that can produce infinitely many outputs. Our next result proves this requirement is strict for efficient reward design.

Theorem 4.5. *A variant of the* REWARDDESIGN *problem with finite reward outputs is NP-hard.*

We provide further details about the precise problem studied in Appendix B. Beyond reward functions with finitely-many outputs, we are also interested in extensions of our results to multiple *environments*. We next present a positive result indicating our algorithms can extend to the case where Alice would like to design a reward function for a single task across multiple environments.

Proposition 4.6. *For any SOAP, PO, or TO, given a finite set of CMPs, $\mathcal{E} = \{E_1, \ldots, E_n\}$, with shared state–action space, there exists a polynomial time algorithm that outputs one reward function that realizes the task (when possible) in all CMPs in $\mathcal{E}$.*

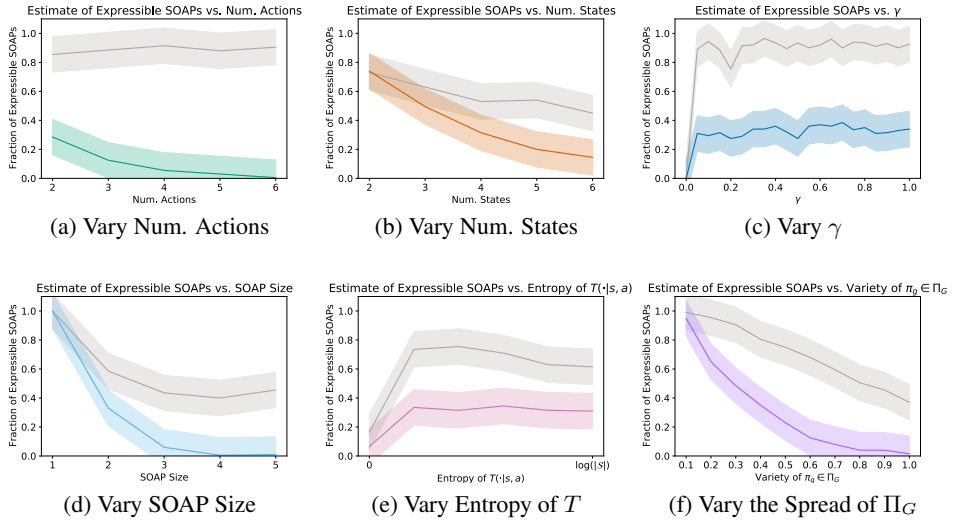
A natural follow up question to the above result asks whether task realization is *closed* under a set of CMPs. Our next result answers this question in the negative.

Theorem 4.7. *Task realization is not closed under sets of CMPs with shared state-action space. That is, there exist choices of $\mathscr{T}$ and $\mathcal{E} = \{E_1, \ldots, E_n\}$ such that $\mathscr{T}$ is realizable in each $E_i \in \mathcal{E}$ independently, but there is not a single reward function that realizes $\mathscr{T}$ in all $E_i \in \mathcal{E}$ simultaneously.*

Intuitively, this shows that Alice must know precisely which environment Bob will inhabit if she is to design an appropriate reward function. Otherwise, her uncertainty over E may prevent her from designing a realizing reward function. We foresee iterative extensions of our algorithms in which Alice and Bob can react to one another, drawing inspiration from repeated IRL by Amin et al. [4].

5 Experiments

We next conduct experiments to shed further light on the findings of our analysis. Our focus is on SOAPs, though we anticipate the insights extend to POs and TOs as well with little complication. In the first experiment, we study the fraction of SOAPs that are expressible in small CMPs as we vary aspects of the environment or task (Figure 3). In the second, we use one algorithm from Theorem 4.3



(a) Vary Num. Actions (b) Vary Num. States (c) Vary γ

(d) Vary SOAP Size (e) Vary Entropy of T (f) Vary the Spread of Π_G

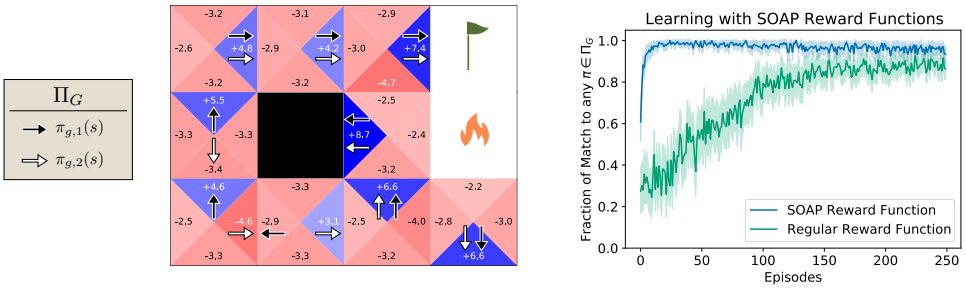
Figure 3: The approximate fraction of SOAPs that are expressible by reward in CMPs with a handful of states and actions, with 95% confidence intervals. In each plot, we vary a different parameter of the environment or task to illustrate how this change impacts the expressivity of reward, showing both equal (color) and range (grey) realization of SOAP.

to design a reward function, and contrast learning curves under a SOAP-designed reward function compared to standard rewards. All experiments take a few minutes to run on a single CPU. Full details about the experiments are found in Appendix C.

SOAP Expressivity. First, we estimate the fraction of SOAPs that are expressible in small environments. For each data point, we sample 200 random SOAPs and run Algorithm 1 described by Theorem 4.3 to determine whether each SOAP is realizable in the given CMP. We ask this question for both the equal (in color) variant of SOAP realization and the range (in grey) variant. We inspect SOAP expressivity as we vary six different characteristics of E or Π_G: The number of actions, the number of states, the discount γ, the number of good policies in each SOAP, the Shannon entropy of T at each (s, a) pair, and the "spread" of each SOAP. The spread approximates average edit distance among policies in Π_G determined by randomly permuting actions of a reference policy by a coin weighted according to the value on the x-axis. We use the same set of CMPs for each environment up to any deviations explicitly made by the varied parameter (such as γ or entropy). Unless otherwise stated, each CMP has four states and three actions, with a fixed but random transition function, sampled from a Dirichlet-Multinomial distribution with parameter $1/|\mathcal{S}|$.

Results are presented in Figure 3. We find that our theory is borne out in a number of ways. First, as Theorem 4.1 suggests, we see the SOAP expressivity is strictly less than one in nearly all cases. This is evidence that inexpressible tasks are not only found in manufactured corner cases, but rather that expressivity is quite a spectrum. We further observe—as predicted by Theorem 3.1—clear separation between the expressivity of range-SOAP (grey) vs. equal-SOAP (color); there are many cases where we can find a reward function that makes the good policies *near* optimal and better than the bad, but cannot make those good policies all *exactly* optimal. Additionally, several trends emerge as we vary the parameter of environment task, though we note that such trends are likely specific to the underlying choice of CMP and may not hold across all environment choices. Perhaps the most striking trend is in Figure 3f, which shows a downward trend in expressivity as the SOAPs become more *spread out*. This is entirely sensible: A more spread out SOAP is likely to lead to more entailments of the kind discussed in Figure 2b.

Learning with SOAP-designed Rewards. Next, we contrast the learning performance of Q-learning under a SOAP-designed reward function (visualized in Figure 4a) with that of the regular goal-based reward in the Russell and Norvig [41] grid world. In this domain, there is 0.35 slip probability such that, on a 'slip' event, the agent randomly applies one of the two orthogonal action



(a) Grid World SOAP Reward

(b) Grid World Learning

Figure 4: A SOAP-designed reward function (left) and the resulting learning curves (right) for Q-learning compared to the traditional reward function for the Russell and Norvig [41] grid world. Each series presents average performance over 50 runs of the experiment with 95% confidence intervals.

effects. The regular goal-based reward function provides $+1$ when the agent enters the terminal flag cell, and -1 when the agent enters the terminal fire cell. The bottom left state is the start-state, and the black cell is an impassable wall.

Results are presented in Figure 4. On the right, we present a particular kind of learning curve contrasting the performance of Q-learning with the SOAP reward (blue) and regular reward (green). The y-axis measures, at the end of each episode, the average (inverse) minimum edit distance between Q-learning's greedy policy and any policy in the SOAP. Thus, when the series reaches 1.0, Q-learning's greedy policy is identical to one of the two SOAP policies. We first find that Q-learning is able to quickly learn a $\pi_g \in \Pi_G$ under the designed reward function. We further observe that the typical reward does not induce a perfect match in policy—at convergence, the green curve hovers slightly below the blue, indicating that the default reward function is incentivizing different policies to be optimal. This is entirely sensible, as the two SOAP policies are extremely cautious around the fire; they choose the orthogonal (and thus, safe) action in fire-adjacent states, relying on slip probability to progress. Lastly, as expected given the amount of knowledge contained in the SOAP, the SOAP reward function allows Q-learning to rapidly identify a good policy compared to the typical reward.

6 Conclusion

We have here investigated the expressivity of Markov reward, framed around three new accounts of task. Our main results show that there exist choices of task and environment in which Markov reward cannot express the chosen task, but there are efficient algorithms that decide whether a task is expressible *and* construct a reward function that captures the task when such a function exists. We conclude with an empirical examination of our analysis, corroborating the findings of our theory. We take these to be first steps toward understanding the full scope of the reward hypothesis.

There are many routes forward. A key direction moves beyond the task types we study here, and relaxes our core assumptions—the environment might not be a finite CMP, Alice may not know the environment precisely, reward may be a function of history, or Alice may not know how Bob represents state. Along similar lines, a critical direction incorporates how reward impacts Bob's *learning dynamics* rather than start-state value. Further, we note the potential relevance to the recent *reward-is-enough* hypothesis proposed by Silver et al. [43]; we foresee pathways to extend our analysis to examine this newer hypothesis, too. For instance, in future work, it is important to assess whether reward is capable of inducing the right kinds of attributes of cognition, not just behavior.

Acknowledgments and Disclosure of Funding

The authors would like to thank André Barreto, Diana Borsa, Michael Bowling, Wilka Carvalho, Brian Christian, Jess Hamrick, Steven Hansen, Zac Kenton, Ramana Kumar, Katrina McKinney, Rémi Munos, Matt Overlan, Hado van Hasselt, and Ben Van Roy for helpful discussions. Michael Littman was supported in part by funding from DARPA L2M, ONR MURI, NSF FMitF, and NSF RI.

References

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the International Conference on Machine learning*, 2004.

[2] David Ackley and Michael L. Littman. Interactions between learning and evolution. *Artificial Life II*, 1992.

[3] Sundararaman Akshay, Nathalie Bertrand, Serge Haddad, and Loic Helouet. The steady-state control problem for Markov decision processes. In *Proceedings of the International Conference on Quantitative Evaluation of Systems*, 2013.

[4] Kareem Amin, Nan Jiang, and Satinder Singh. Repeated inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

[5] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2017.

[6] Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, 2017.

[7] Gerard Debreu. Representation of a preference ordering by a numerical function. *Decision Processes*, 3:159–165, 1954.

[8] Daniel Dewey. Reinforcement learning and the reward engineering principle. In *Proceedings of the AAAI Spring Symposium Series*, 2014.

[9] Tom Everitt, Victoria Krakovna, Laurent Orseau, Marcus Hutter, and Shane Legg. Reinforcement learning with a corrupted reward channel. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2017.

[10] William Fedus, Carles Gelada, Yoshua Bengio, Marc G. Bellemare, and Hugo Larochelle. Hyperbolic discounting and learning over multiple horizons. *arXiv preprint arXiv:1902.06865*, 2019.

[11] Karl J. Friston. The free-energy principle: a unified brain theory? *Nature reviews neuroscience*, 11(2):127–138, 2010.

[12] Karl J. Friston, Jean Daunizeau, and Stefan J. Kiebel. Reinforcement learning or active inference? *PloS One*, 4(7):e6421, 2009.

[13] Dylan Hadfield-Menell, Anca Dragan, Pieter Abbeel, and Stuart Russell. Cooperative inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, 2016.

[14] Dylan Hadfield-Menell, Anca Dragan, Pieter Abbeel, and Stuart Russell. The off-switch game. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2017.

[15] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart Russell, and Anca Dragan. Inverse reward design. In *Advances in Neural Information Processing Systems*, 2017.

[16] Danijar Hafner, Pedro A. Ortega, Jimmy Ba, Thomas Parr, Karl J. Friston, and Nicolas Heess. Action and perception as divergence minimization. *arXiv preprint arXiv:2009.01791*, 2020.

[17] Lewis Hammond, Alessandro Abate, Julian Gutierrez, and Michael Wooldridge. Multi-agent reinforcement learning with temporal logic specifications. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2021.

[18] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2018.

[19] Hong Jun Jeon, Smitha Milli, and Anca Dragan. Reward-rational (implicit) choice: A unifying formalism for reward learning. In *Advances in Neural Information Processing Systems*, 2020.

[20] Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A composable specification language for reinforcement learning tasks. In *Advances in Neural Information Processing Systems*, 2020.

[21] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, 1984.

[22] W. Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The TAMER framework. In *Proceedings of the International Conference on Knowledge Capture*, 2009.

[23] Tjalling C. Koopmans. Stationary ordinal utility and impatience. *Econometrica: Journal of the Econometric Society*, pages 287–309, 1960.

[24] David Kreps. *Notes on the Theory of Choice*. Westview Press, 1988.

[25] Sai Krishna Gottipati, Yashaswi Pathak, Rohan Nuttall, Raviteja Chunduru, Ahmed Touati, Sriram Ganapathi Subramanian, Matthew E. Taylor, and Sarath Chandar. Maximum reward formulation in reinforcement learning. In *NeurIPS Workshop on Deep Reinforcement Learning*, 2020.

[26] Ramana Kumar, Jonathan Uesato, Richard Ngo, Tom Everitt, Victoria Krakovna, and Shane Legg. REALab: An embedded perspective on tampering. *arXiv preprint arXiv:2011.08820*, 2020.

[27] Xiao Li, Cristian-Ioan Vasile, and Calin Belta. Reinforcement learning with temporal logic rewards. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2017.

[28] Michael L. Littman. The reward hypothesis, 2017. URL https://www.coursera.org/lecture/fundamentals-of-reinforcement-learning/michael-littman-the-reward-hypothesis-q6x0e.

[29] Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via GLTL. *arXiv preprint arXiv:1704.04341*, 2017.

[30] James MacGlashan, Monica Babes-Vroman, Marie desJardins, Michael L. Littman, Smaranda Muresan, Shawn Squire, Stefanie Tellex, Dilip Arumugam, and Lei Yang. Grounding English commands to reward functions. In *Proceedings of Robotics: Science and Systems*, 2015.

[31] James MacGlashan, Michael L. Littman, David L. Roberts, Robert Loftin, Bei Peng, and Matthew E. Taylor. Convergent actor critic by humans. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2016.

[32] James MacGlashan, Mark K. Ho, Robert Loftin, Bei Peng, Guan Wang, David L. Roberts, Matthew E. Taylor, and Michael L. Littman. Interactive learning from policy-dependent human feedback. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2017.

[33] Maja J. Mataric. Reward functions for accelerated learning. In *Proceedings of the International Conference on Machine Learning*, 1994.

[34] L. G. Mitten. Preference order dynamic programming. *Management Science*, 21(1):43–46, 1974.

[35] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the International Conference on Machine Learning*, 1999.

[36] Andrew Y. Ng, Stuart J. Russell, et al. Algorithms for inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2000.

[37] Ellen Novoseller, Yibing Wei, Yanan Sui, Yisong Yue, and Joel Burdick. Dueling posterior sampling for preference-based reinforcement learning. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2020.

[38] Pedro A. Ortega, Vishal Maini, and the DeepMind Safety Team. Building safe artificial intelligence: specification, robustness, and assurance, 2018. URL https://medium.com/@deepmindsafetyresearch/building-safe-artificial-intelligence-52f5f75058f1.

[39] Silviu Pitis. Rethinking the discount factor in reinforcement learning: A decision theoretic approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[40] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[41] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1994. ISBN 0-13-103805-2.

[42] Rohin Shah, Pedro Freire, Neel Alex, Rachel Freedman, Dmitrii Krasheninnikov, Lawrence Chan, Michael D. Dennis, Pieter Abbeel, Anca Dragan, and Stuart Russell. Benefits of assistance over reward learning, 2021. URL https://openreview.net/forum?id=DFIoGDZejIB.

[43] David Silver, Satinder Singh, Doina Precup, and Richard S. Sutton. Reward is enough. *Artificial Intelligence*, page 103535, 2021.

[44] Satinder Singh, Andrew G. Barto, and Nuttapong Chentanez. Intrinsically motivated reinforcement learning. Technical report, University of Massachusetts at Amherst Department of Computer Science, 2005.

[45] Satinder Singh, Richard L Lewis, and Andrew G Barto. Where do rewards come from? In *Proceedings of the Annual Conference of the Cognitive Science Society*, 2009.

[46] Satinder Singh, Richard L. Lewis, Jonathan Sorg, Andrew G. Barto, and Akram Helou. On separating agent designer goals from agent goals: Breaking the preferences–parameters confound, 2010.

[47] Matthew J. Sobel. Ordinal dynamic programming. *Management science*, 21(9):967–975, 1975.

[48] Matthew J. Sobel. Discounting axioms imply risk neutrality. *Annals of Operations Research*, 208(1):417–432, 2013.

[49] Jonathan Sorg. *The Optimal Reward Problem: Designing Effective Reward for Bounded Agents*. PhD thesis, University of Michigan, 2011.

[50] Jonathan Sorg, Richard L. Lewis, and Satinder Singh. Reward design via online gradient ascent. *Advances in Neural Information Processing Systems*, 2010.

[51] Peter Sunehag and Marcus Hutter. Axioms for rational reinforcement learning. In *Proceedings of the International Conference on Algorithmic Learning Theory*, 2011.

[52] Richard S. Sutton. The reward hypothesis, 2004. URL http://incompleteideas.net/rlai.cs.ualberta.ca/RLAI/rewardhypothesis.html.

[53] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[54] Umar Syed, Michael Bowling, and Robert E. Schapire. Apprenticeship learning using linear programming. In *Proceedings of the International Conference on Machine Learning*, 2008.

[55] Csaba Szepesvári. Constrained MDPs and the reward hypothesis, 2020. URL http://readingsml.blogspot.com/2020/03/constrained-mdps-and-reward-hypothesis.html.

[56] Geraud Nangue Tasse, Steven James, and Benjamin Rosman. A Boolean task algebra for reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

[57] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Teaching multiple tasks to an RL agent using LTL. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2018.

[58] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1953.

[59] Paul Weng. Markov decision processes with ordinal rewards: Reference point-based preferences. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2011.

[60] Martha White. Unifying task specification in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2017.

[61] Edward C. Williams, Nakul Gopalan, Mine Rhee, and Stefanie Tellex. Learning to parse natural language to grounded reward functions with weak supervision. In *Proceedings of the International Conference on Robotics and Automation*, 2018.

[62] Aaron Wilson, Alan Fern, and Prasad Tadepalli. A Bayesian approach for policy learning from trajectory preference queries. In *Advances in Neural Information Processing Systems*, 2012.

[63] Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *The Journal of Machine Learning Research*, 18(1):4945–4990, 2017.

[64] Yichong Xu, Ruosong Wang, Lin Yang, Aarti Singh, and Artur Dubrawski. Preference-based reinforcement learning with finite-time guarantees. *Advances in Neural Information Processing Systems*, 33, 2020.

[65] Zeyu Zheng, Junhyuk Oh, Matteo Hessel, Zhongwen Xu, Manuel Kroiss, Hado van Hasselt, David Silver, and Satinder Singh. What can learned intrinsic rewards capture? In *Proceedings of the International Conference on Machine Learning*, 2020.

A Anticipated Questions

We first address questions that might arise in response to the main text.

*(Q1) What does it mean for Bob to *solve* one of these tasks? That is, if Alice chooses a SOAP, PO, or TO for Bob to learn to solve, when can Alice determine Bob has solved the task?*

A: Bob can be said to be doing better on a given task if his behavior improves, as is typical in evaluating behavior under reward. The difference with SOAPs, POs, and TOs is that we measure improvement relative to the task rather than reward. For instance, given a SOAP, we might say that Bob has solved the task once he has found one of the good policies, and we might measure Bob's progress on a task in terms of the distance of his greedy policy to one of the good policies (as done in our learning experiments). The same reasoning applies to POs and TOs: Bob is doing better on a task in so far as his greedy policy (or trajectories) is (are) higher up the ordering.

(Q2) These notions of inexpressibility all come about due to the Markov restriction on reward functions. That is, the studied reward functions must be a function of s, (s, a), or (s, a, s'). But, what about history-based reward functions?

A: Indeed, as discussed in our introduction, our goal is to examine the expressivity of Markov rewards in the context of finite MDPs. We assume the environment is fixed and given to Alice with the state and action spaces already determined. While it is sensible to consider history-based rewards, this opens up new considerations: Must the state space also change so as to retain the Markov property? Instead, we suggest that for a given CMP, it is natural to be interested in Markov rewards, but acknowledge the importance of going beyond such functions. As discussed in the main text, we suspect that there is a coherent account of which tasks are and are not expressible as a consequence of some of the axioms for rationality. We hope to study these directions in future work.

(Q3) Why restrict attention to SOAPs, POs, and TOs?

A: First, we recognize these do not necessarily capture all of what we hope to convey to learning agents. It is an important next step in our work to enrich the analysis with more general objectives. Still, we believe that these each represent an interesting, relatively general, and concrete template for what a task might look like. They are quite flexible: SOAPs can be simple while POs and TOs can be complex.

(Q4) Why restrict to the start-state value?

A: We adopt start-state value due to its simplicity. Other considerations might be: (1) The *expected* value under some chosen distribution, or (2) That the constraint hold over all states (so, for SOAPs, each π_g is better than each π_b in value for all states). We note that the former case is identical to start-state value, as we can always add a start-state to any CMP where all actions lead to the desired next-state distribution in T. The latter case is slightly more complicated, so we chose not to focus on it as we prefer the simplicity of the start-state case. However, we note that many inexpressible tasks under the start-state criterion remain inexpressible under the "all-state" criterion (such as the XOR example from Figure 2b).

.....

B Proofs

We next restate each central result, and present its proof.

Proposition 3.1. *There exists a CMP, E, and choice of Π_G such that Π_G can be realized under the range-SOAP criterion, but cannot be realized under the equal-SOAP criterion.*

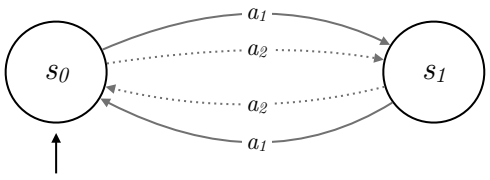


Figure 5: A CMP that separates the two kinds of SOAP realizations.

Proof of Proposition 3.1.

Consider the example in Figure 5 and the SOAP $\Pi_G = \{\pi_{11}, \pi_{21}, \pi_{12}\}$. This Π_G indicates that all policies are acceptable except the policy that always takes a_2. That is, the policy subscripts denote which actions each policy takes in each state (π_{12} means a_1 in s_0, a_2 in s_1).

First, let us note that range SOAP is realizable: The listed rewards allow for each policy in Π_G to obtain $\frac{\text{RMAX}/2}{1-\gamma}$ value or better, while π_{22} achieves zero. Letting $\varepsilon = \text{VMAX}/2$, this choice of rewards satisfies the criteria, and the Π_G is ε-realized in the given MDP.

Next, note that there can exist no other choice of rewards that realize the equal SOAP. That is, such that $V^{\pi_{11}}(s_0) = V^{\pi_{12}}(s_0) = V^{\pi_{21}}(s_0) > V^{\pi_{22}}(s_0)$. This fact is a consequence of the tie in values between the policies. Here, we see that any choice of rewards that makes $V^{\pi_{12}}(s_0) = V^{\pi_{21}}(s_0)$ will also give π_{22} that same value. Thus, the given Π_G is unrealizable under equal SOAP. □

Theorem 4.1. *For each of SOAP, PO, and TO, there exist $(E, \mathscr{T})$ pairs for which no reward function realizes $\mathscr{T}$ in E.*

Proof of Theorem 4.1.

We proceed by proving the existence of a pair $(E, \mathscr{T})$, for each of $\mathscr{T}$ as a SOAP, PO, or TO. Indeed, we find that the simple XOR case is inexpressible for all three task types.

SOAP. For SOAP, we consider the same CMP, E, with two states and two actions, and the SOAP $\Pi_G = \{\pi_{12}, \pi_{21}\}$. That is, the chosen task is for the learning agent to find a policy that chooses each action in at least one state, but not more. Here, we find that

PO. Since the given SOAP is a special case of a PO, we have already identified a given inexpressible PO.

TO. For TO, for simplicity we consider the same CMP. We let $N = 2$, and suppose that the desired trajectory ordering is over state-action pairs, giving rise to a set of good trajectories, and a set of bad trajectories:

$$L_{\tau,N} := \{\tau_G, \tau_B\}, \tag{B.1}$$

$$\tau_G = \{\{(s_0, a_1), (s_1, a_2)\}, \{(s_0, a_2), (s_1, a_1)\}\}, \tag{B.2}$$

$$\tau_B = \{\{(s_0, a_1), (s_1, a_1)\}, \{(s_0, a_2), (s_1, a_2)\}\}. \tag{B.3}$$

The same reasoning from the above cases applies: We cannot make the good trajectories strictly higher in return than the bad trajectories. □

Proposition 4.2. *There exist choices of $E_{\neg T} = (\mathcal{S}, \mathcal{A}, \gamma, s_0)$ or $E_{\neg\gamma} = (\mathcal{S}, \mathcal{A}, T, s_0)$, together with a task $\mathscr{T}$, such that there is no (T, R) pair that realizes $\mathscr{T}$ in $E_{\neg T}$ or (R, γ) in $E_{\neg\gamma}$.*

Proof of Proposition 4.2.

The running XOR example is actually inexpressible for *all* choices of T, or of γ. That is, there is no way to make π_{12} and π_{21} strictly better than both π_{22} *and* π_{11} by varying γ or T. Such examples likely exist for any choice of $\mathcal{S}$ and $\mathcal{A}$ of size greater than one. □

Theorem 4.3. *The* REWARDDESIGN *problem can be solved in polynomial time, for any finite E, and any SOAP, PO, or TO, so long as a reward-function family with infinitely many outputs is used.*

Proof of Theorem 4.3.

We proceed by providing constructive algorithms for each of SOAP, PO, or TO. All three are based on similar applications of a linear program (LP), though there is nuance that separates them. We present each as a Lemma (Lemma C.1, Lemma C.2, Lemma C.3), which together constitute the proof of this Theorem. □

Lemma C.1. *The SOAP variant of* REWARDDESIGN *can be solved in polynomial time.*

Proof of Lemma C.1.

We again proceed by constructing a linear program (LP) whose solution is the desired reward function (or correctly outputs that there is no such reward function). Specifically, note that we want to choose a reward function so that all the policies in Π_G have strictly higher start-state value than all the policies not in the set. We present the proof through five observations.

First, observe that any reward function that will induce the desired ordering ensures that the optimal policy π^* is in the set Π_G. This is true since π^* (under the chosen reward function) is better than all policies. So it is better than all policies not in Π_G.

Second, note that the set Π_G is well connected in the following sense. Let a step in policy space from some reference policy π_{ref} to be a move to any other deterministic policy that differs from π_{ref} in exactly one state. Then, for any pair of policies in Π_G, there must be a sequence of policies in Π_G, each one step apart from the next, from one to the other. This follows from the policy-improvement theorem: we can get from any policy to an optimal policy in a sequence of policies such that each policy (1) is one step from the previous one and (2) strictly dominates the previous one. Since any policy that strictly dominates a policy in Π_G must be better than the policy in Π_G, it must also be in Π_G (if the problem constraint is satisfied). That means if we choose two policies in Π_G, π_1 and π_2, both can reach π^* in a sequence of single steps while staying within Π_G. Since steps are symmetric, Π_G is connected.

The connected set of policies in Π_G has a "fringe" Π_{fringe}—a set of policies not in Π_G that are one step from a policy in Π_G.

Third, for the constraints of the problem to be satisfied, every policy $\pi \in \Pi_G$ must be strictly better than every policy $\pi_f \in \Pi_{\text{fringe}}$.

Fourth, observe that $|\Pi_{\text{fringe}}| <= |\mathcal{A}||\Pi_G|$, so Π_{fringe} is polynomial sized.

Fifth, we can construct a polynomial-sized LP that expresses that every policy $\pi \in \Pi_G$ is strictly better than every policy $\pi_f \in \Pi_{\text{fringe}}$. Note that the direct way to build this LP has a "strictly better than" comparison between each policy $\pi_f \in \Pi_{\text{fringe}}$ and each policy $\pi \in \Pi_G$. That's at most $|\mathcal{A}||\Pi_G|^2$ inequalities.

We now tie the above observations together to show that the solution to this LP solves the constraints of the problem. That is, the reward function returned makes it so every policy $\pi \in \Pi_G$ is strictly better than every policy not in Π_G, *and* no valid reward function is excluded (so, if a solution exists, it will be found).

The argument that no valid reward function is excluded is simply because the set of constraints in the LP is a subset of the defining constraints of the problem. Specifically, the LP constrains the policies inside Π_G to be strictly better than the ones on the fringe instead of all policies not in Π_G.

The argument that only constraining the values on the fringe automatically constrains all the others proceeds as follows. First, with respect to the returned reward function, there is some optimal policy π^*. That policy π^* must be in Π_G. To see why, let us assume it is not. That means there is a sequence of improving steps that turn a policy in Π_G to π^* (currently assumed to be out of Π_G). But, any such sequence must go through the fringe, and we constrained the fringe so that all of the policies in Π_G are strictly better than them. So, π^* must be in Π_G. Next, we know that all policies in Π_G must be strictly better than the policies not in Π_G. To see why, consider an "improving" path from some policy $\pi_b \notin \Pi_G$ to π^*. Since π^* is in Π_G, we know this path must go through some policy $\pi_f \in \Pi_{\text{fringe}}$. Since it's an improving path, that means π_f is better than π_b. But, every policy in Π_G is strictly better than π_f, so it must also be strictly better than π_n. □

Lemma C.2. *The PO variant of* REWARDDESIGN *can be solved in polynomial time.*

Proof of Lemma C.2.

We proceed by constructing a procedure that calls a linear program whose answer is the reward function that induces the given L_Π in M, or the procedure correctly outputs that there is no such R.

Consider the set of policies in Π, numbered $\pi_1, \ldots, \pi_i, \ldots, \pi_N$. Note that the value of π_i in s_0 can be computed in terms of the expected reward under the policy's discounted expected state-action visitation distribution. That is, for each π_i, let

$$\rho_i(s, a) := \sum_{t=0}^{\infty} \gamma^t \Pr(s_t = s, a_t = a \mid s_0, \pi_i). \tag{B.4}$$

Since the given MDP is assumed to have finite state-action space, note that ρ_i may be interpreted as a vector whose elements correspond to $\rho_i(s_0, a_0), \rho_i(s_0, a_1)$, and so on.

The value of π_i under a given reward function R (which may also be interpreted as a vector) is then produced by the dot product $R \cdot \rho_i$.

Given L_Π, we want to find an R that ensures a set of linear constraints hold:

$$R \cdot \rho_0 \geq R \cdot \rho_1 \geq \ldots. \tag{B.5}$$

Note that the trivial reward function, $R_\emptyset : s \mapsto 0$, is a solution to the above linear program. However, we can ensure some minimal increment improvement of ϵ for non-tying policies, where

$$R \cdot \rho_0 \geq R \cdot \rho_1 + \epsilon \geq \dots. \tag{B.6}$$

This ϵ minimal increment is sufficient to separate policies with tying scores and avoids the degenerate solution of $R_\emptyset$, so long as there are infinitely many reward outputs feasible.

Note that the LP above will by default output a vector of rationals for the reward function. We note that this is still suitable for both $\mathcal{R}_\mathbb{N}$ and $\mathcal{R}_\mathbb{Z}$ by scaling.

Note that the input is of size N, where N is the number of constraints imposed on the policy ordering. In the worst case, $N = |L_\Pi| \leq |\mathcal{A}|^{|\mathcal{S}|}$. If there are fewer constraints than either $|\mathcal{A}|$ or $|\mathcal{S}|$, then $N = \max\{|\mathcal{S}|, |\mathcal{A}|\}$. The amount of computational work required is split across two steps:

1. $\tilde{O}(N)$: Compute ρ_i for each policy $\pi_1, \dots \pi_N$.
2. $O(N^3)$: Formulate and solve the above linear program.

Thus, since the described linear program can be constructed in polynomial time outputs a reward function that induces the given L_Π, we conclude the PO case. □

Lemma C.3. *The TO variant of* REWARDDESIGN *can be solved in polynomial time.*

Proof of Lemma C.3.

The algorithm follows the same construction as those catered toward SOAPs and POs, but is in fact much simpler.

We can form linear inequality constraints on the return of two trajectories as follows. First recall that the N-step discounted return of a trajectory τ is

$$G(\tau; s_0) = \sum_{i=0}^{N-1} \gamma^i R(s_i, a_i), \tag{B.7}$$

assuming reward is a function of state and action for simplicity. Note that because reward functions are assumed to be deterministic, the quantity $G(\tau; s_0)$ is not a random variable.

Now, given two trajectories, $\tau_i = \{(s_0^{(i)}, a_0^{(i)}), \dots, (s_{N-1}^{(i)}, a_{N-1}^{(i)})\}$ and $\tau_j = \{(s_0^{(j)}, a_0^{(j)}), \dots, (s_{N-1}^{(j)}, a_{N-1}^{(j)})\}$, note that we can express linear inequality constraints as follows,

$$\tau_i \cdot R - \tau_j \cdot R \leq \epsilon, \tag{B.8}$$

where R is a length N state-action vector, and $\epsilon \in \mathbb{Q}_{\geq 0}$ is a slack variable to be maximized as part of the optimization. Inequality constraints follow the same structure, only simpler. By the same reasoning that underlies the construction of the SOAP and PO based algorithms, the above set of constraints define a linear program whose solution is the realizing reward, if it exists. □

Corollary 4.4. *For any task $\mathscr{T}$ and environment E, deciding whether $\mathscr{T}$ is expressible in E is solvable in polynomial time*

Proof of Corollary 4.4.

For each of PO and TO, the constraints we construct define precisely the space of constraints that constitute the given task. Thus, since linear programming will find a solution for the given

constraint set, we know that these two forms of the algorithm can also correctly identify when no reward function exists.

The SOAP case is slightly more involved, but still relatively straightforward. We note that the policy fringe, $\mathcal{F}$, is a subset of Π_B, since $\mathcal{F}$ consists of some policies *not* in Π_G by construction. This means that the constraints produced that separate each good policy from a fringe policy in value are a subset of the true constraints (those that separate each $\pi_g \in \Pi_G$ from each $\pi_b \in \Pi_B$). Hence, since constraint relaxations of this kind have the property that they do not exclude solutions, we conclude that our proposed linear program will correctly determine when no satisfying reward function exists. □

Next, we provide further details on Theorem 4.3 that examines reward design when only finitely many reward outputs may be used. As noted in the main text, Theorem 4.3 requires that Alice is allowed to design a reward function that can produce infinitely many outputs. It is natural to wonder whether this requirement is strict. Theorem 4.5 answers this question in the negative, by proving that the following decision problem is hard.

Definition B.1. *The* FINITE-PO-REWARDDECISION *problem is defined as follows:* ***Given*** $E = (\mathcal{S}, \mathcal{A}, T, \gamma, s_0)$*, and a set of* k *policy inequalities* ($\pi_{x_i} < \pi_{y_i}$)*,* ***output*** *True iff there is a reward function* $R(s')$ *that induces the given policy inequalities.*

Note that this formulation focuses on POs, and on reward as a function of next-state. Unfortunately, we find this problem is NP-hard, showing that for reward design to be efficient, infinitely many reward outputs are needed.

Theorem 4.5. *The* FINITE-PO-REWARDDECISION *problem is NP-hard.*

Proof of Theorem 4.5.

We assume every $T(s' \mid s, a)$ is expressed as a rational number. We also assume that all policies are deterministic, Markov policies, although results should extend to stochastic policies with rational probabilities as well.

The binary PO problem is the same, but it insists that every $R(s') \in \{0, 1\}$ for all s' in the returned reward function.

Observation 1: The binary PO decision problem is in NP. We can guess an assignment of $R(s, a)$ to either 0 or 1, then evaluate in inequality using linear equation solving as policy evaluation.

We show that the binary PO decision problem can be used to decide the NP-hard monotone clause 3-SAT problem with a polynomial reduction.

A monotone clause 3-SAT problem consists of a set of n variables, and m clauses. Each clause consists of three variables and is either a positive clause or a negative clause. In a positive clause, all three variables appear as literals. In a negative clause, all three variables appear as negated literals. The problem is the same as the standard 3-SAT problem except for the restriction that we cannot mix positive and negative literals in a clause.

We can convert an instance of monotone clause 3-SAT to the binary PO problem as follows. There is only one state where decisions are possible. It is the initial state of the MDP. Each action from this state results in an action-specific probabilistic transition to a set of terminal states, each of which is associated with a terminal reward value.

Because of the simple structure of this MDP, each policy corresponds to an action and vice versa. And, each terminal state responds to a reward and vice versa. So, each policy can be viewed as a convex combination of rewards.

We create two terminal states s_0 and s_1 and create one action (a_0) that transitions directly to s_0 and one (a_1) that transitions directly to s_1. We then add a policy constraint that says the $a_0 < a_1$. Because all rewards are in $\{0, 1\}$, that forces the reward for s_0 to be 0 and for s_1 to be 1. Those become our logical primitives, in a sense.

Next, we add $2n$ states, one for each positive and negative literal in the 3-SAT problem. For each variable v, we add an action a_v with a 50-50 transition to s_v and $s_{\overline{v}}$, along with two constraints: $a_0 < a_v < a_1$. These constraints ensure that the reward assignment to s_v and $s_{\overline{v}}$ can be interpreted as an assignment to the literals where one gets a 1 and the other gets a zero. There is no other way to satisfy these constraints.

Now, for each positive clause c consisting of variables v_1, v_2, and v_3, we create an action a_c that transitions to s_{v_1}, s_{v_2}, and s_{v_3} with equal probability. We add a policy constraint that $a_c > a_0$, forcing the assignment of rewards to the variables to correspond to a satisfying assignment for that clause. (At least one of the rewards needs to be set to 1.)

For the each negative clause c consisting of variables $\overline{v_1}$, $\overline{v_2}$, and $\overline{v_3}$, we create an action a_c that transitions to $s_{\overline{v_1}}$, $s_{\overline{v_2}}$, and $s_{\overline{v_3}}$ with equal probability. We add a policy constraint that $a_c < a_1$, forcing the assignment of rewards to the variables to correspond to a satisfying assignment for that clause. (At least one of the rewards needs to be set to 0.)

By the way the MDP is constructed, the constraints are satisfied if and only if the rewards represent a satisfying assignments for the given monotone clause 3-SAT formula. Therefore, an efficient solution to the binary PO decision problem would provide an efficient solution to the NP-hard monotone clause 3-SAT problem. □

Proposition 4.6. *For any SOAP, PO, or TO, given a finite set of CMPs $\mathcal{E} = \{E_1, \ldots, E_n\}$ with shared state–action space, there exists a polynomial time algorithm that outputs a single reward function that realizes the task (when possible) in each CMP in $\mathcal{E}$.*

Proof of Proposition 4.6.

From Theorem 4.3, we know that there is an algorithm to solve the reward design problem for any task and a single environment, in polynomial time. We form the multi-environment algorithm by simply combining the constraints formed by each individual linear program. By the properties of linear programming, the resulting solution will either satisfy *all* of the given constraints, as desired, or will correctly identify that no such satisficing solution exists. □

Theorem 4.7. *Task realization is not closed under sets of CMPs with shared state-action space. That is, there exist choices of $\mathscr{T}$ and $\mathcal{E} = \{E_1, \ldots, E_n\}$ such that $\mathscr{T}$ is realizable in each $E_i \in \mathcal{E}$ independently, but there is not a single reward function that realizes $\mathscr{T}$ in all $E_i \in \mathcal{E}$ simultaneously.*

Proof of Theorem 4.7.

We consider a pair of CMPs with the same three states and two actions. We will show that there exists choice of $\mathscr{T}$ such that $\mathscr{T}$ is realizable in E_X and E_Y independently, but *not* in both simultaneously. Our result assumes we restrict to reward functions that are only a function of state, but we suspect similar cases exist for reward functions on (s, a) pairs and (s, a, s') triples.

Concretely, consider the pair of CMPs in Figure 6. These two CMPs share a state-action space and start-state, but not a transition function (and, say, a $\gamma > 0.5$). Let us further suppose we are interested in the SOAP $\Pi_G = \{\pi_{112}, \pi_{212}\}$, that is, the policies that take a_1 in s_1, and a_2 in s_2. In both CMPs, the transition function from s_0 transitions to s_1 with probability 0.5 and s_2 with probability 0.5, for both actions.

We first show that this Π_G is realizable in both CMPs. For the CMP on the left, note that the reward function $R(s_1) = 1, R(s_2) = -1$, with $\gamma = 0.95$ will ensure $V^{\pi_{112}}(s_0) = V^{\pi_{212}}(s_0)$, and that both policies are strictly better than all others. Next, note that the same is true for the example on the right where $R(s_1) = -1, R(s_2) = 1$. Thus, Π_G is independently realizable in both of these CMPs.

However, there cannot exist a reward function that makes π_{112} and π_{212} strictly better than all other policies in both CMPs—it is either better to stay in s_1, or to stay in s_2, but it cannot be the case that both are true simultaneously. □

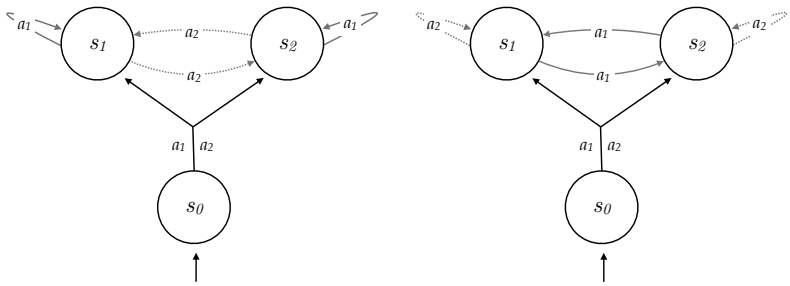


Figure 6: A pair of CMPs with opposite action effects: On the left, a_1 keeps the agent in the same place, while a_2 flips the state. On the right, the effects are exactly inverted.

.....

C Experimental Details

Next, we provide further details about our experiments.

C.1 Expressibility Experiments

First, we provide additional information about the first batch of experiments that focus on the fraction of SOAPs that are expressible in small CMPs.

Six Variants. In each figure, we vary one aspect of the environment or task along the x-axis. Most of these are self-explanatory (a: number of actions, b: number of states, c: γ), though the plots in (d), (e) and (f) are slightly more nuanced. In (d), we vary the *size* of each sampled SOAP, corresponding to the number of good policies in the SOAP. That is, a SOAP of size one consists only of a single good policy, $\Pi_G = \{\pi^*\}$. In (e), we vary the *Shannon entropy* of the transition function on a per state-action basis as per: $H(T) = -\sum_{s' \in \mathcal{S}} \log_2 T(s' \mid s, a)$. This is accomplished by interpolating between the fully deterministic transition function that only transitions to a single next-state and the uniform random transition function through a simple soft-max distribution in which one next-state is the intended transition, while each other next-state receives a small amount of probability mass depending on the given entropy. In plot (f), we vary the *spread* of the SOAP, which is a measure of how different the good policies in the SOAP are on average. The x-axis corresponds to an approximate edit-distance between policies, where each point on the x-axis defines the parameter of a coin that we flip to determine whether to change a chosen reference policy's action for each state. So, we first randomly sample one policy, say π_1. Then, we construct the next policy for the SOAP as follows: For a given coin weight θ, we flip a coin at each state of the CMP. If the coin lands heads (the trial is successful), then we change the action of the new policy to a fixed action chosen uniformly at random. Thus, when θ is zero, the SOAP will only contain π_1. When θ approaches one, the SOAP will likely contain many different policies.

Environment Details. In each case, unless otherwise specified, the underlying environment is a four state, three action CMP with $\gamma = 0.95$, and a transition function that is a multinomial over next-states sampled from a Dirichilet-multinomial distribution with α parameters set to $\frac{1}{|\mathcal{S}|}$. When not specified, the size of each SOAP is two. We varied many aspects of these parameters and found little change in the nature of the plots, though trends will be shifted up or down in many cases. For instance, given the downward trend of Figure (3d) as the SOAP size increases, we know that the remaining plots will each be scaled downward if we were to run the same experiment for a SOAP size larger than two. We sample random SOAPs by first sampling a SOAP size randomly between 1 and $|\Pi|$. Then, we sample N SOAPS of the chosen size uniformly at random (unless otherwise specified, as in the case of Figure (3f)).

C.2 Learning Experiments

In the grid environment, we set slip probability of 0.35 for all (non-terminal) states. When a "slip" event occurs, the action effect is orthogonal to the intended direction. For instance, in the bottom left cell, if the up action is executed, there is a 0.175 chance the agent will execute `left` (thus staying in the bottom right cell), and a 0.175 chance the agent will execute `right`, and a 0.65 chance the agent will move up a cell. We experiment with tabular Q-learning with ϵ-greedy exploration, with $\epsilon = 0.2$ and learning rate $\alpha = 0.1$ and no annealing. Each episode consists of 10 steps in the environment, with 250 episodes per algorithm. We repeat the experiment 50 times and report 95% confidence intervals. The y-axis measures, at the end of each episode, the (inverse) minimum edit distance between Q-learning's greedy policy and any of the policies in the SOAP along the trajectory taken by Q-learning's greedy policy. Thus, when the series reaches 1.0, Q-learning's greedy policy is identical to one of the SOAP policies in the states that the greedy policy will reach. We observe that the gap between the blue and green curves is due to the different kinds of policies that the SOAP reward and the regular reward promote—one is not necessarily better or worse than the other, they just convey different kinds of objectives.

.....

D NeurIPS Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] The vast majority are in the main paper, but a few small details are deferred to Appendix B.
   (b) Did you include complete proofs of all theoretical results? [Yes] In Appendix B.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] Our experiments are small in scope, but we are looking into making our code open source.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We mention in Section 5 that all of our experiments can be run in a few minutes with a single CPU (our experiments are extremely small in scale).
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects... [N/A]