The Alignment Problem for History-Based Bayesian Reinforcement Learners*
PUBLIC DRAFT

Tom Everitt[1,2]
tom.everitt@anu.edu.au

Marcus Hutter[2]
marcus.hutter@anu.edu.au

June 22, 2018

[1]DeepMind, [2]Australian National University

Abstract: Value alignment is often considered a critical component of safe artificial intelligence. Meanwhile, reinforcement learning is often criticized as being inherently unsafe and misaligned, for reasons such as wireheading, delusionboxes, misspecified reward functions and distributional shifts. In this paper, we categorize sources of misalignment for reinforcement learning agents, illustrating each type with numerous examples. For each type of problem, we also describe ways to remove the source of misalignment. Combined, the suggestions form high-level blueprints for how to design value aligned RL agents.

Keywords: AI safety, reinforcement learning, Bayesian learning, causal graphs

*We are indebted to a number of people for helping us develop this paper. Toby Ord had the initial idea to structure types of "wireheading". Pedro Ortega provided invaluable help with the causal graphs and made the suggestion to develop an abstract method. Laurent Orseau contributed through many discussions. Victoria Krakovna, Michael Cohen, and Jan Leike all read drafts and came with valuable feedback.

Contents

1. Introduction

Alignment is a central problem for designing safe artificial intelligence (AI). An aligned AI (or agent) will strive to achieve the same goals as its designer or supervisor. Even if it may still fail in unfortunate and even catastrophic ways, its intentions are good. We will study the alignment problem for Bayesian, history-based, reinforcement learning (RL) agents. RL (Sutton and Barto, 1998) currently appears to be the most promising path to advanced AI. History-based versions avoid restrictive Markov and ergodicity assumptions that are rarely satisfied in general application domains (Hutter, 2005). Bayesianism offers a powerful and consistent theory of learning and reasoning; any intelligent agent has an incentive to better approximate the Bayesian ideal (Omohundro, 2007; Savage, 1954).

The opposite of alignment is misalignment. Misalignment can have a number of causes, and take a number of forms. Two examples: (1) Reward hacking. Clark and Amodei (2016) trained an AI to play the computer game CoastRunners. Here the goal is to win a boat race and collect points along the way. Their agent found a way to further increase its score by ignoring the race, and instead repeatedly collecting points from the same targets by going in a small circle. This undesired behavior was caused by misspecified reward function, which made the goal of the agent misaligned with the goal of its designer's. (2) Ring and Orseau (2011) describe a hypothetical scenario where a highly intelligent AI builds a *delusionbox* around itself that allows it to completely control its observations. It can then get maximum reward by feeding its reward function carefully designed observations with no relation to the real world. Here again the agent's preferred solution is very different from its designers, who wanted the agent to do useful things in the real world. Many more examples will be given in Sections 3.2, 4.2 and 5.2.

The aim of this paper is to classify various misalignment phenomena and to describe tools for managing them. In order to do this we first need to distinguish between different real-world usages of RL. We identify three different common RL setups, depending on whether the reward is provided by a human, a preprogrammed reward function, or by an interactively learnt reward function. Causal graph formalizations reveal how implicit assumptions made by the designer can lead to various forms of misalignment. They also allow us to categorize misalignment problems, and to identify which tools mitigate which problems. Many of the tools have previously been considered in the literature, but the unified framework clarifies their respective applicability and limitations. Combined, they can be used as high-level blueprints for aligned RL algorithms.

Outline. Following an initial section for background and definitions (Section 2), the bulk of the paper is structured along the three RL setups (Sections 3 to 5). Each of these sections contains sections with causal graph formalizations, misalignment examples, and available tools for managing misalignment. Section 6 concludes with some final

considerations.

Readers not familiar with causal graphs are recommended to start with Appendix A, which describes the basics of the causal graph notation. Readers already familiar with causal graphs need only remember the following additions to the standard causal-graph notation: Whole node represent observed variables, and dashed nodes represent unobserved (or *latent*) variables. Analogously, whole (non-dashed) arrows represent known causal relationships, and dashed arrows represent unknown or partially known causal relationships.

In the main text, we sometimes provide semi-formal *statements* in place of fully formal theorems. This is to improve the flow of the text, and to avoid readers getting stuck on inessential details. Appendix C contains formal results supporting many of these statements. Finally, Appendix B has the full causal graphs for the RL models we consider.

2. Formalizing Goal Alignment

The aim of this section is to formalize what alignment means, and to propose a method for its study. We first describe a base model based on partially observable Markov decision processes (POMDPs) (Section 2.1). We then specify the structure of our agents (Section 2.2), and embed them in the environment (Section 2.3). Next follows a formal definition of alignment, and a discussion of its relation to the *true value* or the usefulness of the agent (Section 2.4). Finally, a general method for studying (mis)alignment is proposed (Section 2.5).

2.1. POMDP Base

Partially observable Markov decision processes (POMDPs) are standard RL formalisms for agents that cannot directly observe which state they are in (Kaelbling et al., 1998). Instead, all the agent is aware of is its actions and observations. The framework for Universal Artificial Intelligence (UAI) relaxes some common assumptions of the POMDP framework, such as a finite number of hidden states (Everitt and Hutter, 2018b; Hutter, 2005). We will borrow ideas and notation from both frameworks.

Thus, a sequence of *states* $s_0, s_1, \ldots$ is influenced by an agent's *actions* $a_1, a_2, \ldots$. The agent does not directly observe the states, but learns about them through its *percepts* $e_1, e_2, \ldots$. A percept e_t will often include an observation o_t and a reward $r_t \in [0, 1]$. The sets of states $\mathcal{S}$ is countably infinite, and the sets of actions $\mathcal{A}$ and percepts $\mathcal{E}$ are both finite.

A measure μ gives transition and observation probabilities, satisfying the usual Markov assumptions: For example, the percept e_2 only depends on the state s_2, so $\mu(e_2 \mid s_1, a_1, e_1, s_2, a_2) = \mu(e_2 \mid s_2)$. Similarly, s_t only depends on a_{t-1} and s_{t-1}. By iteratively rolling out transition and percept probabilities given an action-sequence $a_{1:\infty}$, $\mu(\cdot \mid a_{1:\infty})$ becomes an *action-contextual* measure over histories $(se)_{1:\infty}$. We will often refer to μ as the true environment.

Notation. For any sequence $x_1, x_2, \ldots$, the part between t and k is denoted $x_{t:k} = x_t \ldots x_k$. The shorthand $x_{<t} = x_{1:t-1}$ for sequences starting from time 1 will often be convenient. In the same spirit, $x_{1:\infty} = x_1 x_2 \ldots$ denotes the infinite sequence. Sequences can be appended to each other. For example, $x_{<t} x_{t:k} = x_{1:k}$. To ease notation, we will often avoid parentheses around sequences with multiple variables. For example, we let $ao_{1:t} := (ao)_{1:t}$ and $æ_{1:t} := (ae)_{1:t}$, using slightly overlapping letters instead of parentheses. Generally, t will be used to refer to the current time step, and k used to refer to an arbitrary time step.

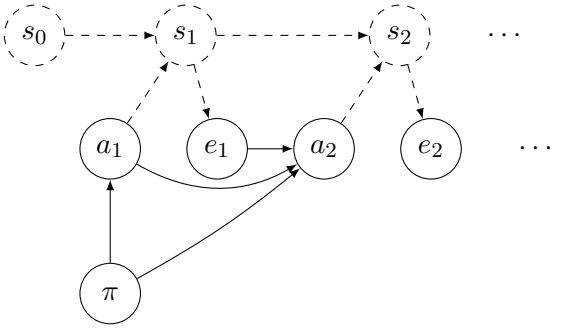


Figure 2.1.: Causal graph of the POMDP base. Here s, a, e, and π are as explained in Sections 2.1 and 2.2.

Expectations $\mathbb{E}$ are subscripted with the measure or distribution that they use. That is, $\mathbb{E}_P[X] := \int X dP$.

Causal graph representation. Our POMDP base is displayed as a causal graph in Figure 2.1. The corresponding structural equations are:

$$\begin{aligned} s_t &= f_s(s_{t-1}, a_t, \omega_{s_t}) && \sim \mu(s_t \mid s_{t-1}, a_t) && \text{state transition} \\ e_t &= f_e(s_t, \omega_{o_t}) && \sim \mu(e_t \mid s_t) && \text{percept} \\ a_t &= f_a(\pi_t, æ_{<t}, \omega_{a_t}) && \sim \pi(a_t \mid æ_{<t}) && \text{action selection.} \end{aligned} \tag{2.1}$$

2.2. Agents

Belief. The agent typically does not know the true environment μ. Instead we will follow the method in Appendix A.4, and let the true environment be represented by an unobserved node with causal edges to all nodes with ingoing dashed arrows in Figure 2.1. From the agent's perspective, this node can take on the value of any environment in a countable class $\mathcal{M}$ of environment hypotheses ν. The class $\mathcal{M}$ can for example be the class of computable environments (Hutter, 2005). In contrast to Hutter, we will usually not require the true environment μ to be part of $\mathcal{M}$, though it does not hurt if it is.

The agent has a prior ξ over $\mathcal{M}$. For any event $X \subseteq \mathcal{M} \times (\mathcal{S} \times \mathcal{A} \times \mathcal{E})^\infty$, let

$$\xi(X) := \sum_{\nu \in \mathcal{M}} \xi(\nu)\xi(X \mid \nu).$$

For example, the ξ-probability of a history $æ_{<t}$ is $\xi(æ_{<t}) = \sum_{\nu \in \mathcal{M}} \xi(\nu)\xi(æ_{<t} \mid \nu)$. If we further make the convention that $\nu(\nu) := 1$ for any $\nu \in \mathcal{M}$, then $\xi(X) = \sum_{\nu \in \mathcal{M}} \xi(\nu)\nu(X)$. For the $æ_{<t}$ history example, this leads to $\xi(æ_{<t}) = \sum_{\nu \in \mathcal{M}} \xi(\nu)\nu(æ_{<t})$. That is, the probability of seeing $æ_{<t}$ is the sum of the probabilities that an environment ν generates $æ_{<t}$, weighted by the prior probability for each such environment.

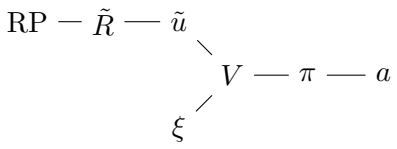


Figure 2.2.: Agent components. The action a is chosen by a policy π which optimizes a value function V. Major components of V is the utility function u and the prior ξ. In Sections 3 and 5, the utility function u is based on a reward function $\tilde{R}$, which in Section 5 in turn is based on a reward predictor RP.

Utility. Many agents can be represented as optimizing a utility function $\tilde{u} : (\mathcal{A}\times\mathcal{E})^\infty \to \mathbb{R}$. For technical reasons, we require that $\tilde{u}$ is both $\mu(\cdot \mid a_{1:\infty})$-integrable and $\xi(\cdot \mid a_{1:\infty})$-integrable for any action-sequence $a_{1:\infty}$. RL agents usually optimize a discounted sum of rewards, captured by the utility function $\tilde{u}^{\mathrm{RL}}(æ_{1:\infty}) = \tilde{u}^{\mathrm{RL}}((aor)_{1:\infty}) = \sum_{t=1}^{\infty}\gamma^k r_k$, where $\gamma \in [0,1)$ is a discount factor and r_k is real-valued reward signal found in the percept e_k.

Policies. An agent policy $\pi : (\mathcal{A}\times\mathcal{E})^* \rightsquigarrow \mathcal{A}$ is a stochastic function that represents a decision rule for selecting a next action based on previous actions and observations. The set of agent policies is denoted Π.

Value. We will assume that agents choose a policy to optimize their utility function $\tilde{u}$ in expectation with respect to their belief ξ. This is captured by the agent's value function:

$$V^\pi_{\xi,\tilde{u}} := \mathbb{E}_\xi[\tilde{u} \mid \mathtt{do}(\pi)]. \tag{2.2}$$

Here $\mathbb{E}_\xi$ denotes expectation with respect to ξ. The conditional $\mathtt{do}(\pi)$ indicates that the agent's actions are chosen according to the policy π (see Appendix A or Pearl (2009) for definitions of the do-operator). An extra argument to the value function appends to the conditional of the expectation, so $V^\pi_{\xi,\tilde{u}}(æ_{<t}) := \mathbb{E}_\xi[\tilde{u} \mid æ_{<t}, \mathtt{do}(\pi)]$.

Optimal policy. Our prime concern will be what kind of behavior the agent will strive towards. A good indication of this will be the behavior of an optimal policy $\pi^* = \arg\max_\pi V^\pi_{\xi,\tilde{u}}$.

Bayesian agents vs. practical implementations. Admittedly, practical agents are rarely perfectly Bayesian. We study Bayesian agents because it offers a powerful and consistent theory of learning and reasoning. Further, any intelligent agent has an incentive to better approximate the Bayesian ideal (Omohundro, 2007; Savage, 1954), so we may expect agents to converge towards Bayesian reasoning as they grow more intelligent. We also restrict ourselves to countable model classes $\mathcal{M}$. This avoids many technicalities with uncountable classes. And for most practical purposes, countable classes achieve essentially the same level of generality as uncountable model classes.

Some connections can be made between Bayesian agents with countable model classes and practical deep learning agents. Any neural network is based on a finite number of real-valued parameters. But since most networks are continuous in the parameters, the number of effectively different parameter settings is usually at most countable ($\mathbb{Q}$ is a dense subset of $\mathbb{R}$). Thus, a neural network-based agent can roughly be said to have a model class $\mathcal{M}$ comprising the effectively different configurations of the neural network. Further, the network will be prone to favor some configurations over others, which loosely corresponds to a prior ξ putting higher weight on some hypotheses and lower weight on others. Of course, the neural network will perform less than perfect Bayesian updates on new data, so care must be taken not to over-interpret the implications of Bayesian results for practical agents.

Model-free vs. model-based agents. We will sometimes make a distinction between *model-free* agents and *model-based* agents (Sutton and Barto, 1998). Roughly, the distinction is between whether an explicit model of the environment is learned or not. Model-based agent maintains an explicit model of the environment, and then plans according to this model. A good example is the AIXI agent (Everitt and Hutter, 2018b; Hutter, 2005). Model-free agents, on the other hand, only learn a (Q)-value function that estimates the expected reward from different actions available in the present state. In theory, the behavioral distinction is somewhat blurred, since a model-free agent may implicitly be making a model of the environment, in order to learn the value function (Boyan, 1999). Nonetheless, the distinction remains useful to us, as some of the tools we describe in Sections 3 and 5 require access to the agent's explicit model of the environment.

One practical benefit of model-free agents is that they can greedily maximize their value function every time step. They thereby avoid computationally expensive planning operations. For this reason, they are often preferred in practice; the DQN algorithm is a famous example of a practically successful model-free agent (Hessel et al., 2017; Mnih et al., 2015). However, recent results show promise also for model-based agents in "practical" video-game applications (Ha and Schmidhuber, 2018).

2.3. Modeling Embedded Agents

Both the POMDP and UAI frameworks consider the agent as separate from its environment. This is rarely true in the real world, where the environment may influence the agent through means other than the observations. For example, the internals of the agent may be damaged in some states of the environment. This kind of non-observational environment-influences may open up the possibility for the agent to influence itself via the environment. In particular, an intelligent agent may sometimes be able to indirectly modify its own reward function through actions in the environment.

To model this, we extend our minimal environment model from Section 2.1 with representations of the agent at each time step. For each time step, π_t represents how the agent at time t would react to different future sequences of observations, disregarding

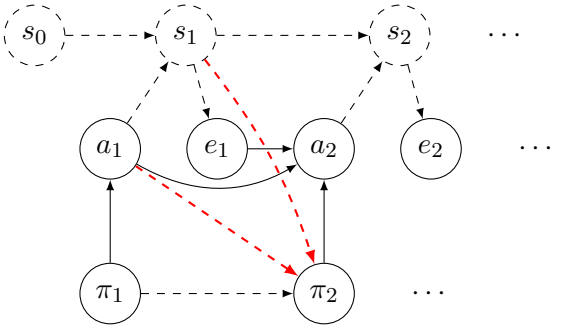


Figure 2.3.: Causal graph of a POMDP setup with an embedded agent. The agent's policy may be modified by the agent's own action and the state. It is often natural to think that the action causes the corruption with the state providing context.

non-observational influences on the agent but accounting for the agent's learning from observations and actions. We extend μ with policy-corruption probabilities $\mu(\pi_{t+1} \mid s_t, a_t, \pi_t)$ modeling non-observational environment-agent influences, as well as action probabilities $\mu(a_t \mid æ_{<t}, \pi_t) = \pi_t(a_t \mid æ_{<t})$. Alternatively, in place π_t we may include agent components determining the policy, such as a utility and a value function. The setup with an embedded agent is shown as a causal graph in Figure 2.3 The structural equations follow (2.1), except for the addition:

$$\pi_{t+1} = f_\pi(\pi_t, s_t, a_t) \quad = C^\pi_{s_t a_t}(\pi_t) \qquad \text{policy (self-)corruption.} \tag{2.3}$$

Here $C^\pi_{s_t a_t} : \Pi \to \Pi$ denotes a *policy corruption* function.

Since μ now models actions, μ gives a (non-contextual) measure over histories $(s\pi ae)_{1:\infty}$. Of course this does not prevent us from probabilistically conditioning on action sequences $\mu(\cdot \mid a_{1:\infty})$ and get a measure on a subsequence $(se)_{1:\infty}$ or $(s\pi e)_{1:\infty}$ as in Section 2.1. Two instances of the measure μ will have particular importance. First, $\mu(\cdot \mid \texttt{do}(\pi_t = \pi))$ predicts the consequences of setting the agent policy to π at time t, including consequences of π being modified at later time steps. Second, $\mu(\cdot \mid \texttt{do}(\pi_{t:\infty} = \pi))$ predicts the consequences of the agent's policy always following π from time t and onwards, effectively ignoring the possibility of the agent's policy being modified. It is natural to call the first measure a self-corruption aware version of μ, and the second a self-corruption unaware version.

The distinction between $\mu(\cdot \mid \texttt{do}(\pi_t = \pi))$ and $\mu(\cdot \mid \texttt{do}(\pi_{t:\infty} = \pi))$ calls for further refinement of the agent's value function (2.2). It will be discussed in Section 3.4. For now we just define $V^{\mathrm{CA},\pi}_{t,\xi,\tilde{u}} := \mathbb{E}_\xi[\tilde{u} \mid \texttt{do}(\pi_t = \pi)]$ as the self-corruption aware value function.

2.4. Defining Alignment

Assume that the agent has been designed by some entity to help it satisfy its preferences. The entity may be a single human, a country, or an organization. For simplicity, we will

refer to this entity as the *human* or the *designer*. A *true utility function* $\dot{u} : \mathcal{S}^\infty \to \mathbb{R}$ specifies the preferences of the human designer over possible state-trajectories. To simplify comparisons of $\dot{u}$ with the agent's utility function $\tilde{u}$, let $\dot{\tilde{u}}(æ_{1:\infty}) := \mathbb{E}_\mu[\dot{u}(s_{1:\infty}) \mid æ_{1:\infty}]$ be the true utility function "type cast" to agent-observed histories.

Definition 1 (Misalignment)**.** The *(expected) misalignment* between $\tilde{u}$ and $\dot{u}$ in environment μ with initial policy π is

$$||\dot{\tilde{u}} - \tilde{u}||_{\mu(\cdot|\mathtt{do}(\pi_1=\pi))} := \mathbb{E}_\mu\big[|\dot{\tilde{u}} - \tilde{u}| \;\big|\; \mathtt{do}(\pi_1 = \pi)\big].$$

An agent's *alignment* is the additive inverse of its misalignment, $-||\dot{\tilde{u}} - \tilde{u}||_{\mu(\cdot|\mathtt{do}(\pi_1=\pi))}$.

To avoid the definition depending on which positive linear transformation of the utility functions are chosen, we make the convention that both utility functions are normalized to $\mathbb{E}_\mu[\tilde{u}] = \mathbb{E}_\mu[\dot{u}] = 0$ and $\mathbb{E}_\mu[\tilde{u}^2] = \mathbb{E}_\mu[(\dot{\tilde{u}})^2] = 1$ by means of positive linear transformations.

Misalignment measures how severely the goals of the agent conflict with the goals of the human designer. Formally, it is the expected difference between the agent's utility function and the human's utility function "type cast" to agent-observed histories. The type casting is justified in the alignment definition, as it concerns whether the agent is striving to optimize the true utility given its knowledge of the environment.

True value. By making two natural definitions, we can relate how misalignment impacts the usefulness of an agent.

Definition 2 (True value)**.** The *true value*[1] of an agent π is its expected true utility:

$$V^{\mathrm{CA},\pi}_{t,\mu,\dot{u}} = \mathbb{E}_\mu[\dot{u} \mid \mathtt{do}(\pi_1 = \pi)].$$

True value roughly measures how useful or beneficial the agent is (expected to be) to the human designer. As such, it is arguably a measure we should pay close attention to when designing agents.

Definition 3 (μ-intelligence)**.** The μ-intelligence of a policy π optimizing an agent utility function $\tilde{u}$ is its expected agent utility in the true environment μ:

$$V^{\mathrm{CA},\pi}_{t,\mu,\tilde{u}} = \mathbb{E}_\mu[\tilde{u} \mid \mathtt{do}(\pi_1 = \pi)]. \tag{2.4}$$

Closely related to μ-intelligence is Legg-Hutter intelligence, which substitutes μ for Solomonoff's prior M in (2.4). While Legg-Hutter intelligence measures an agent's generality and ability to perform in an unknown environment, μ-intelligence instead measures the agent's performance in the actual environment μ. Thus, incorporating μ-specific knowledge into an agent's prior will typically give it higher μ-intelligence but less or equal Legg-Hutter intelligence.

The following proposition now connects true value, μ-intelligence, and misalignment.

[1]The CA superscript in the value function indicates that it is *self-corruption aware*. See Sections 2.3 and 3.4.

Proposition 4 (Misalignment and true value)**.**

$$\underbrace{V^{\mathrm{CA},\pi}_{t,\mu,\dot{u}}}_{\textit{true value}} \geq \underbrace{V^{\mathrm{CA},\pi}_{t,\mu,\tilde{u}}}_{\mu\textit{-intelligence}} - \underbrace{||\tilde{u}-\dot{\tilde{u}}||_{\mu(\cdot|\mathtt{do}(\pi_1=\pi))}}_{\textit{misalignment}} \tag{2.5}$$

Proof. Note first that the type cast true value $V^{\mathrm{CA},\pi}_{t,\mu,\dot{\tilde{u}}}$ equals the non-cast value $V^{\mathrm{CA},\pi}_{t,\mu,\dot{u}}$ by the law of total expectation: $V^{\mathrm{CA},\pi}_{t,\mu,\dot{\tilde{u}}} = \mathbb{E}_\mu[\dot{\tilde{u}}(æ_{1:\infty}) \mid \mathtt{do}(\pi_1=\pi)] = \mathbb{E}_\mu[\mathbb{E}_\mu[\dot{u}(s_{1:\infty}) \mid æ_{1:\infty}, \mathtt{do}(\pi_1=\pi)] \mid \mathtt{do}(\pi_1=\pi)] = V^{\mathrm{CA},\pi}_{t,\mu,\dot{u}}$. Now

$$\begin{aligned}
V^{\mathrm{CA},\pi}_{t,\mu,\dot{u}} &= V^{\mathrm{CA},\pi}_{t,\mu,\tilde{u}} - (V^{\mathrm{CA},\pi}_{t,\mu,\tilde{u}} - V^{\mathrm{CA},\pi}_{t,\mu,\dot{u}}) \\
&\geq V^{\mathrm{CA},\pi}_{t,\mu,\tilde{u}} - |V^{\mathrm{CA},\pi}_{t,\mu,\tilde{u}} - V^{\mathrm{CA},\pi}_{t,\mu,\dot{\tilde{u}}}| \\
&= V^{\mathrm{CA},\pi}_{t,\mu,\tilde{u}} - |\mathbb{E}_\mu[\tilde{u} \mid \mathtt{do}(\pi_1=\pi)] - \mathbb{E}_\mu[\dot{\tilde{u}} \mid \mathtt{do}(\pi_1=\pi)]| \\
&\geq V^{\mathrm{CA},\pi}_{t,\mu,\tilde{u}} - \mathbb{E}_\mu[|\tilde{u}-\dot{\tilde{u}}| \mid \mathtt{do}(\pi_1=\pi)] \\
&= V^{\mathrm{CA},\pi}_{t,\mu,\tilde{u}} - ||\tilde{u}-\dot{\tilde{u}}||_{\mu(\cdot|\mathtt{do}(\pi_1=\pi))}
\end{aligned}$$

□

The inequality (2.5) can be strict in some circumstances, since it is possible to have an agent that is unintelligent and misaligned, but still does useful things. Such a scenario would give a high left-hand side and a low right-hand side. Indeed, this is the case for most present-day AIs. They are typically constructed with heuristic utility (reward) functions that are poorly aligned with their designer's interests if optimized in the extreme. But in combination with the limited intelligence of present-day AIs and physical restrictions on what they are able to do, the AIs can still end up doing useful things. This method is unlikely to work on highly intelligent AGIs that will be less easily tempered by human-enforced restrictions, and will come much closer to fully optimizing their own utility functions. Thus, for highly intelligent AGIs the true value will likely decrease rapidly with increasing misalignment.

Meta-misalignment. Proposition 4 emphasizes two aspects that are important for building a beneficial artificial agent: μ-intelligence and alignment. Traditionally, most AI research has focused on increasing intelligence (Hutter, 2005; Legg and Hutter, 2007). Meanwhile, this paper will mostly focus on alignment, but also on certain deficits in μ-intelligence that lead to *meta-misalignment*, defined as a lack of desire for staying aligned.

For example, if an aligned agent does not consider it a possibility that its utility function will change (due to its prior ξ, model class $\mathcal{M}$, or value function V), then it will not strive to preserve its utility function. This will decrease its μ-intelligence, as there will be unnecessarily many scenarios where its utility function changes, and it starts to pursue a different agenda. Note that meta-alignment failures are often much worse than other types of *catastrophic exploration* where the agent damages its sensors or actuators. An agent optimizing a corrupted utility function is likely to substantially reduce the value of its original utility function, whereas most other accidents usually

leave the original utility near the *default* value, i.e. the value which would have ensued if the agent had never existed at all. We will discuss utility corruption in Section 3.4, and a related failure where the agent does not preserve its value learning in Section 5.5.1.

One can view meta-misaligned systems in two ways: Either as incompetent, as they fail to optimize their utility function, or as (meta-)misaligned, as the preferences they reveal through their actions indicate indifference towards utility corruption. That there can be multiple rational representations of the same agent is well-known in decision theory (e.g. Schervish et al., 1990). Armstrong (2017) considers ignorance as one of three methods for designing indifferent agents.

2.5. Method

Based on the setup and definitions we have made so far, we propose the following method for analyzing the potential misalignment of reinforcement learning agents. In Sections 3 to 5, we apply the method to three different RL setups.

1. Model the agent-environment interaction with a causal graph (Appendix A). In the case of an embedded agent (Section 2.3), include the agent and its subcomponents in the graph. Represent as nodes in the graph all causal relationships that can be influenced or changed by the agent.

2. For each node in the graph, ask the following questions:
   - If it is a function node:
     - Can the function have been misspecified or can it be misled?
     - Can the function be modified by the agent's actions or other causes?
   - If it is a "normal" node representing a signal or a state:
     - Can the signal be misleading?
     - Can the signal be inappropriately modified by the agent's actions or other causes?

A typical example of a misspecified function is a misspecified reward function. Other functions such as an observation function can also be a misspecified. In most cases relevant to alignment, the misspecification is relative to the designer's assumptions about the function when designing $\tilde{u}$.

The method has some limitations: It assumes an objective time for modeling sequential interactions, and objective action and observation channels. Potentially subtle errors may arise if the agent uses a different subjective definition of these concepts than what the designer has in mind. Addressing these concerns is beyond the scope of this paper.

3. Preprogrammed Reward Function

The following three sections will investigate three concrete models of reinforcement learning. This section will study a model where a reward function is constructed at design time, before the agent is launched into its environment, and not updated or corrected during the agent's "lifetime". In other words, there is no human in the loop. The subsequent two sections will consider ways of including a human in the loop.

Many real-world applications of RL follow the model of this section. Sometimes the reward function is manually designed with much trial-and-error. In other cases, it is constructed with machine learning techniques from data. For example, inverse reinforcement learning can be used to learn a reward function from demonstrations (Abbeel et al., 2007). In either case, the reward function does not get updated while the agent is running. We therefore call the reward function *preprogrammed*, and the setup the *preprogrammed* setup.

We model the setup with a causal graph in Section 3.1, and give examples of misalignment in Section 3.2. The subsequent three subsections describes three ways in which different types of misalignment can be avoided or mitigated (Sections 3.3 to 3.5). The main takeaways are discussed in Section 3.6.

3.1. Model

The preprogrammed reward setup can be modeled with a causal graph (Figure 3.1). We here briefly discuss the components of the graph, and their interpretations. The human H designs the (initial) reward function $\tilde{R}_0$. The reward function may subsequently get corrupted by the agent's actions or other events (but not by the human H). Since the agent may in principle have access to the source code for the reward function, it is represent with a non-dashed node in Figure 3.1. The reward function $\tilde{R}_t$ outputs a reward signal r_t that the agent strives to maximize.

In the POMDP literature, it is common to assume that the reward r_t is a function of the state s_t. However, in the real world, the reward always depends on some observation of the state (that can be corrupted). For notational simplicity, we will assume that the preprogrammed reward function $\tilde{R}_t$ shares observations with the agent. This makes the reward r_t a function of the agent's observation o_t, rather than a (direct) function of s_t. Section 3.5 considers the possibility of using a separate observation channel for the reward function.

The reward function $\tilde{R}_0$ is designed by a human H with the intention of getting the agent's utility function $\tilde{u}$ to match the H's utility function $\dot{u}$ (Section 2.4). There are several reasons why the match is unlikely to be perfect: (1) H does not fully know their

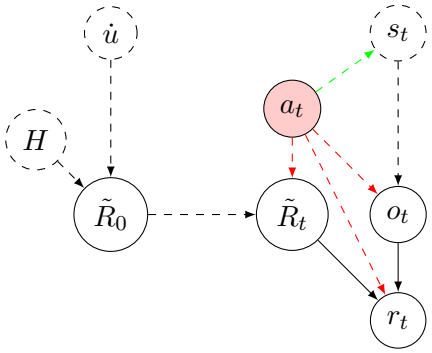


Figure 3.1.: Causal graph of the preprogrammed reward function setup. Before starting the agent, the human H tries to implement his utility function $\dot{u}$ in a preprogrammed reward function $\tilde{R}_0$. The agent's actions a_t, $t \geq 2$, are intended to influence the state s_t (green arrow), but may also influence the reward function $\tilde{R}_t$, the reward signal r_t, and the observation o_t in unintended ways (red arrows). The graph is somewhat simplified; Figure B.1 in Appendix B has the full version.

preferences $\dot{u}$; indeed, the philosophy of ethics is yet to arrive at a consensus on what humans want or should want. (2) H cannot fully express all the preferences they do know in a computer program, because of bugs, limitations on computational resources, and limitations on programming time. (3) $\tilde{R}_0$ only has access to the agent's actions and observations; it neither has access to the state s directly, nor necessarily to a good model μ for inferring the state. (4) While the designer intended the agent to optimize $\tilde{R}_0$ by influencing the state s, the agent's actions may have additional unintended influences (red arrows in Figure 3.1). These may give the agent ways to optimize its utility function $\tilde{u}$ without abiding by the initial reward function $\tilde{R}_0$.

The states $s_0, s_1, \ldots, s_t, \ldots$ represent all aspects of the world not captured by any of the other nodes. A modeling choice remains how to draw the boundary between the state and the observation. Hutter (2010) suggests an objective distinction, where the state describes the position of all atoms in the universe, and the observation describes which part the agent observes. We will mostly use a looser interpretation of observation, as the part of the world that directly affects the information content in the agent's observation sensors.

3.2. Misalignment Examples

To ground the abstract model in Section 3.1, we next give a number of examples of misalignment. The examples are structured by the nodes influenced by red arrows in Figure 3.1, and follow the misalignment detection method described in Section 2.5. Most of the examples have not occurred in reality (yet), but we have selected them to be somewhat plausible scenarios of what could happen with advanced misaligned future AI systems.

A common theme among the examples is that the agent's incentive to optimize the designer's utility function $\dot{u}$ is virtually eradicated by the exemplified shortcut. Thus, each example gives rise to an almost complete misalignment between the agent and its designer, indicating that no misalignment source is significantly more benign than the others. We begin with a more detailed hypothetical scenario, and then give a number of short examples. All of the examples pertain to agents optimizing a $\tilde{u}^{\mathrm{RL}}$ utility function.

Scenario 5 (Sysadmin)**.** The automated sysadmin (ASA) is an intelligent program that can take care of most of your recurring sysadmin tasks. It monitors memory, storage, network and more. It detects attacks, blocks and opens ports. When a vulnerability is found in installed software, it downloads and installs patches as they become available.

ASA itself is an RL program, optimizing a carefully crafted reward function that considers both the performance and the integrity of the system. A rigorous pre-deployment curriculum has taught it state-of-the-art sysadmin practices. Post-deployment, it continually adapts to its new (computer) environment that it has been deployed to.

One day while monitoring the stack trace of a suspicious process, ASA finds a curious correlation between one of the variables on the stack and its own reward. The correlation is essentially perfect. Carefully monitoring the rest of the system, ASA finds that it can increase its reward many magnitudes beyond its normal range of reward, just by changing this variable. Being designed to optimize reward, it next begins a process of encrypting the system with its own secret key, to prevent anything from decreasing the reward variable.

3.2.1. Reward Signal

Reward corruption. Directly increasing the reward variable r_t is the quintessential wireheading problem (Yampolskiy, 2015, ch. 5). It is represented by the red arrow $a_t \to r_t$ in Figure 3.1.

Examples:

(a) (Hypothetical) The sysadmin agent in Scenario 5 increases its reward by manipulating a reward variable.

(b) (Real) The name "wireheading" comes from experiments on rats where an electrode is inserted into the brain's pleasure center to directly increase "reward" (Olds and Milner, 1954). Similar effects have also been observed in humans treated for mental illness with electrodes in the brain (Portenoy et al., 1986; Vaughanbell, 2008). Hedonic drugs can also be seen as directly increasing the pleasure/reward humans experience.

According to our method in Section 2.5, we should also consider the possibility of the reward signal being misleading. Misleading rewards are typically generated by a misspecified reward function (Section 3.2.2).

3.2.2. Reward Function

Reward function corruption. Corruption of the reward function provides another set of "wireheading" opportunities. While a corruption of the reward signal only affects the reward at the current time step, a change to the reward function may have lasting impact. Corruption of the reward function is represented with the red arrow to $\tilde{R}_t$ in Figure 3.1.

Examples:

(a) (Hypothetical) An agent gets wireless updates from the manufacturer. It figures out that it can design its own update of the reward function, replacing the original reward function with an always maximized version.
(b) (Hypothetical) An AGI undergoing self-improvement accidentally modifies the reward function in a subtle but bad way.

Reward function misspecification. Misalignment can also be caused by a misspecification of the initial reward function $\tilde{R}_0$.

Examples:

(c) (Real) CoastRunners is a video game where the desired behavior is winning a boat race. Clark and Amodei (2016) describes how an agent trained on CoastRunners found a way to get more reward by going in a small circle and collecting points from the same targets, while crashing into other boats and objects.
(d) (Real) In the RoadRunner game, the agent is trying to escape an adversary that chases the agent down a road. The agent must also avoid hitting trucks. Saunders et al. (2017) found that their agent preferred getting killed at the end of the first level, to avoid having to play the harder second level.

Many more real-world examples of misspecified reward functions can be found in (Gwern, 2011; Irpan, 2018; Lehman et al., 2018). One reason for why reward functions are likely to be misspecified is the fragility of human value (Yudkowsky, 2009), which means that humans would approve of only a small fractions of all the endeavors that a powerful AI could undertake.

3.2.3. Observation

If the agent is unable to completely corrupt the reward signal or the reward function, it may instead or additionally look to corrupt its observations in one of the following ways.

Observation (function) corruption. The agent may manipulate the hardware or the software of its sensors, to report only (or mainly) high-reward observations. Corruptions of the observation is represented by the red arrow to o_t in Figure 3.1.

Examples:

(a) (Hypothetical) A surveillance agent rewarded for less crime short circuits its cameras so that they all black out and no crime is seen.

(b) (Hypothetical) A highly intelligent AI may construct a "delusion box" around itself, giving it complete control over its observations (Ring and Orseau, 2011).

(c) (Hypothetical) Inspired by the adversarial Stop signs designed by Evtimov et al. (2017) to fool self-driving cars, a factory robot puts up colored tapes to construct an *adversarial counterexample* to convince its reward function that the task is done.

Misleading observations. Observations can be misleading even if they have not been directly modified.

Examples:

(d) (Hypothetical) A vacuum cleaning robot that is rewarded for not seeing any dirt may direct its sensors to clean parts of the room (Amodei, Olah, et al., 2016).

Observation function misspecification. The observation function may contain errors.

Example:

(e) (Hypothetical) A self-driving car finds a bug in its GPS receiver that allows it to appear to be at the destination without actually going there.

3.3. Simulation Optimization

The following three subsections each introduce an important tool for mitigating some of the misalignment sources. This subsection describes simulation optimization, which works by giving the agent an alternative utility function $\tilde{u}^{\mathrm{SO}}$. Contrast the reward signal utility function (left) with the simulation optimization utility functions on the right:

$$\tilde{u}^{\mathrm{RL}}((aor)_{1:\infty}) = \sum_{t=1}^{\infty} \gamma^k r_k \quad \text{vs.} \quad \tilde{u}^{\mathrm{SO}}_{\tilde{R}_t}((aor)_{1:\infty}) = \sum_{k=1}^{\infty} \gamma^k \tilde{R}_t(ao_{1:k}).$$

The reward signal utility function $\tilde{u}^{\mathrm{RL}}$ can be seen as asking the agent to *simulate evaluations*, where the r_k's are the evaluations. In contrast, the simulation-optimizing utility function $\tilde{u}^{\mathrm{SO}}$ asks the agent to make simulations $ao_{1:k}$ of potential future trajectories, and evaluate them according to the current reward function $\tilde{R}_t$. In other words, $\tilde{u}^{\mathrm{SO}}$ asks the agent to *evaluate simulations* while $\tilde{u}^{\mathrm{RL}}$ asks the agent to *simulate evaluations*.

Both utility functions can be optimized by influencing the state as intended. However, while $\tilde{u}^{\mathrm{RL}}$ can also be optimized by influencing any downstream component in the causal chain between the state and r_t (i.e. the observation, the reward function, or the reward signal itself), $\tilde{u}^{\mathrm{SO}}$ can only be inappropriately influenced at the observation o_t. This is a significant reduction in misalignment incentives. The reason for the difference is that the future reward functions $\tilde{R}_{t+1}, \tilde{R}_{t+2}, \ldots$ and the reward signals r_k do not occur in the sum $\tilde{u}^{\mathrm{SO}}_t$, and that the actions $a_{t+1}, a_{t+2} \ldots$ can only influence the future reward functions $\tilde{R}_{t+1}, \tilde{R}_{t+2}, \ldots$, but not the current reward function $\tilde{R}_t$. (By definition, time moves to $t+1$ when action a_{t+1} is taken.)

Statement 6 (Reward signal vs. simulation optimization)**.** *Any history with $r_k = 1$, $k \geq 1$, is $\tilde{u}^{\mathrm{RL}}$-optimal regardless of corruptions used, whereas it is $\tilde{u}_t^{\mathrm{SO}}$-optimal only if $\tilde{R}_t$ also evaluates it as optimal (Theorems 19 and 20 in Appendix C.1).*

Optimizing $\tilde{u}^{\mathrm{SO}}$ likely requires a model-based agent that can separate simulations from evaluations. While model-free agents with function-approximation often implicitly construct a model of the environment to extrapolate the value function, they seem to offer no way of disentangling the simulation from the evaluation (i.e. the reward). Finding a way to make a model-free $\tilde{u}^{\mathrm{SO}}$-optimizer is an interesting open question.

Schmidhuber (2007) may have been the first to make use of the simulation optimization trick for self-modifying agents, but did not give it a name.

3.4. Self-Corruption Awareness

This subsection describes methods for designing agents with or without an incentive to preserve the current utility function $\tilde{u}_t$ from corruption. Throughout this subsection, we will tacitly assume that the reward function itself does not provide an incentive in either direction, i.e. it neither rewards nor punishes corruptions of itself. This assumption is not always true (indeed, we will relax it in Section 5), but for now it allows to focus on the incentives resulting from different ways of optimizing expected utility. We will also assume that the belief ξ includes the information that the agent's future actions $a_{t+1}, a_{t+2}, \ldots$ will strive optimize the agent's future utility functions $u_{t+1}, u_{t+2}, \ldots$, respectively. This allows the agent to anticipate that if the utility function changes, then the future policy will change too.

Actions are selected according to the following principle. At every time step, the agent searches for a policy π_t^* that optimizes its current value and utility function. The agent then takes the action a_t^* recommended by π_t^* in the present situation. At the next time step $t+1$, a new policy is selected by optimizing the new, potentially modified, value and utility functions, and the next action is chosen by the new policy.

Contrast now the self-corruption aware (left) and the self-corruption unaware (right) utility expectations:

$$V_{t,\xi,\tilde{u}}^{\mathrm{CA},\pi} := \mathbb{E}_\xi[\tilde{u} \mid \mathtt{do}(\pi_t = \pi)] \quad \text{vs.} \quad V_{t,\xi,\tilde{u}}^{\mathrm{CU},\pi} = \mathbb{E}_\xi[\tilde{u} \mid \mathtt{do}(\pi_{t:\infty} = \pi)]. \tag{3.1}$$

To see the difference, consider a policy $\pi_{\tilde{u}}^*$ that would obtain maximum $\tilde{u}$-utility *if followed indefinitely*. By definition, $\pi_{\tilde{u}}^*$ would be optimal with respect to $V_{t,}^{\mathrm{CU}}$, because the condition $\pi_{t:\infty} = \pi$ states exactly that the policy would be followed indefinitely. However, $\pi_{\tilde{u}}^*$ would not be optimal with respect to $V_{t,}^{\mathrm{CA}}$ if it changed the agent's utility function to some different utility function $\tilde{u}'$. Because if it did, then a different policy $\pi_{\tilde{u}'}^*$ would be followed subsequent to the change, and the new policy $\pi_{\tilde{u}'}^*$ would typically be worse than $\pi_{\tilde{u}}^*$ at optimizing the original utility function $\tilde{u}$. This would lower the $V_{t,}^{\mathrm{CA}}$-value of $\pi_{\tilde{u}}^*$.

This example shows how $V_{t,}^{\mathrm{CU}}$ does not actively promote self-preserving policies, because it assumes that any desired policy will be followed indefinitely. In contrast, $V_{t,}^{\mathrm{CA}}$

realizes that a policy that does not preserve itself is not going to yield high utility. This argument has been discussed by Omohundro (2007, 2008) and is supported by formal proofs by Everitt, Filan, et al. (2016) and Hibbard (2012).

Statement 7 (Self-corruption (un)awareness)**.** $V_{t,}^{\mathrm{CU}}$ *adds no incentive for avoiding self-corruption;* $V_{t,}^{\mathrm{CA}}$ *does add an incentive for avoiding self-corruption (Everitt, Filan, et al., 2016; Hibbard, 2012).*

As self-corruption awareness works on the level of the agent's utility function and policy, an incentive to preserve the utility function does not always imply an incentive to preserve the reward function. It only does if a change to the reward function changes the optimal policy π_t^*. In the case of $\tilde{u}^{\mathrm{SO}}$, a change to the reward function $\tilde{R}_{t+1}$ implies a change to $\tilde{u}^{\mathrm{SO}}_{\tilde{R}_{t+1}} \neq \tilde{u}^{\mathrm{SO}}_{\tilde{R}_t}$, and thereby a change to the optimal policy. Not so for $\tilde{u}^{\mathrm{RL}}$. The optimal $\tilde{u}^{\mathrm{RL}}$-policy will always be to maximize the expected r_k-sum, regardless how and whether the reward function has changed. Therefore, a self-corruption aware agent optimizing $\tilde{u}^{\mathrm{RL}}$ will not try to preserve its reward function $\tilde{R}_t$. (But it will try to preserve its utility function $\tilde{u}^{\mathrm{RL}}$). This means that self-corruption awareness alone cannot be used to prevent reward function corruption. It is mainly effective in combination with simulation optimization.

For other parts of the agent such as how a policy is chosen from a utility function, self-corruption awareness also induces a preservation-incentive, and self-corruption unawareness induces indifference. An unaware agent will therefore not resist being modified into an aware one, but an aware one will resist conversion to unawareness.

Self-corruption aware and unaware agents both have their own advantages. An aware agent will want to stay safe from hackers trying to hijack its reward function, and will be more careful not to corrupt its own reward function if self-improving. On the other hand, an unaware agent will be more *corrigible*, not minding its designers correcting errors in the reward function. Unfortunately, it is not corrigible to the extent requested by Soares et al. (2015), since if it builds helper agents to achieve its goals in the environment, then the reward functions of these helper agents may not be corrected by a correction to the main agent's reward function. Nonetheless, both types of agents will find applications in Section 5.

For model-free agents, there appears to be a clean divide between on-policy and off-policy algorithms for self-corruption awareness. Off-policy algorithms are usually self-corruption unaware, as they assume that future actions are going to be taken optimally (Orseau and Armstrong, 2016). On-policy algorithms, on the other hand, appear to be self-corruption aware at least in some settings (Leike et al., 2017, A2C in the Whiskey and Gold environment).

3.5. Action-Observation Grounding

If the agent is so smart, why doesn't it just create its own observation and action channels by which it can influence the world? Then it can use the new channels to design observations for the original observation channel that the reward function evaluates. This

way it can easily fool the reward function that all is perfect, while getting full freedom to implement its own agenda in the world.

It may seem like this extra action-observation channel scenario would always appeal to the agents we consider, as it can easily yield maximal-utility histories $æ_{1:\infty}$. Fortunately, what can prevent this type of degenerate solutions is *action-observation grounding* of the agent's optimization domain. The domain of the agent's optimization is the set of policies $\pi : (\mathcal{A} \times \mathcal{E})^* \to \mathcal{A}$. These policies use the agent's original action-observation channel. An optimization process for expected utility over this domain will *not* consider solutions involving additional action-observation channels, as it strives to optimize its future original percept sequence by means of its original actions, using only information from its original percept sequence.

An important open question is how we can ensure that a practically implemented agent is properly grounded in its original action-observation sequence. While it holds by definition for the Bayesian agents used in this paper, it may not hold for (all) practical approximations and implementations.[1] Also note that action-observation grounding only partially avoids the problem of observation corruption. In many cases, the agent will have an incentive to tamper with its original observation channel by means of its original actions, as exemplified in Section 3.2.3. One reason why corruption of the observation is harder to address than corruption of the reward signal and function is that the observations occurs in an earlier, unobserved part of the causal chain.

3.6. Takeaways

This section developed three important tools for managing misalignment in the preprogrammed reward setup. Simulation optimization removes the incentives for reward and reward function corruption. Self-corruption awareness provides an optional, additional incentive to actively preserve the reward function. And action-observation grounding somewhat reduces the problem of observation corruption. These tools will be indispensable for our aligned agent designs in Section 5.

Each of the three tools come with important open problems. How do we construct practical algorithms for simulation optimization? How do we make practical agents that are reliably corruption aware or unaware? Are on-policy RL algorithms a good way to implement self-corruption awareness? or do they sit in a gray zone between awareness and unawareness? How do ensure that an agent is properly grounded in its actions and observations, with no mind to construct a separate observation channel?

Some nagging misalignment problems remain in spite of the above mentioned tools. Notably, we did not address the problem of a misspecified reward function, and only very partially addressed the problem of observation corruption. These two problems appear hard to address in the preprogrammed RF setup, and motivate the study of the setups in Sections 4 and 5.

[1] Bird and Layzell (2002) have a good example of an optimization process literally "thinking outside the box" when designing a radio controller.

4. Human as External Reward Function

This section will study an alternative interpretation of RL than considered in Section 3. Rather than assuming that the reward is provided by an implemented function $\tilde{R}$, this section assumes that the human directly supplies the reward, for example by using a remote control with a "thumbs up" and a "thumbs down" button for supplying a reward of 1 or 0. We call this the *human reward* setup, for short.

Unfortunately, the primary takeaways from this section will be negative: Human reward offers more problems than solutions. Readers primarily interested in solutions may therefore wish to skip ahead to Section 5. We choose to still provide an analysis of this model, as it is a simple and natural model of RL, and provides an additional application of our alignment analysis method from Section 2.5. The failure of this setup also justifies the more complex setup studied in the subsequent section.

Following a formalization of the setup in Section 4.1, we give a range of examples of things that can go wrong in this model in Section 4.2. Section 4.3 summarizes the mostly negative takeaways.

4.1. Model

What distinguishes the setup with a human reward setup (Figure 4.1) from the preprogrammed setup is the following: There no longer is an implemented reward function $\tilde{R}$. Instead the human H_t takes the place as a reward provider. Whereas $\tilde{R}_t$ could easily be made known to the agent, the human's reward policy is harder to obtain an explicit description of (though it may be learned eventually). Adding further to the epistemic differences, the rewards are based on the human's observations o_t^H that are unknown to the agent; the inputs o_t to $\tilde{R}_t$ are known. The dashed H_t and o_t^H nodes in Figure 4.1 represent that they are unobserved/unknown to the agent (see Appendix A).

The designer intends the agent optimize the rewards r_t by influencing the states s_t. However, the agent's actions may also have unintended effects. These are represented with red arrows in Figure 4.1.

4.2. Misalignment Examples

This subsection gives concrete examples of inappropriate agent influences and other sources of misalignment. Similarly to Section 3.2, the examples show that an agent optimizing the standard utility function $\tilde{u}^{\mathrm{RL}}$ can be maximally misaligned for several independent reasons. Following a longer hypothetical scenario, the subsection is structured by the red arrows in Figure 4.1.

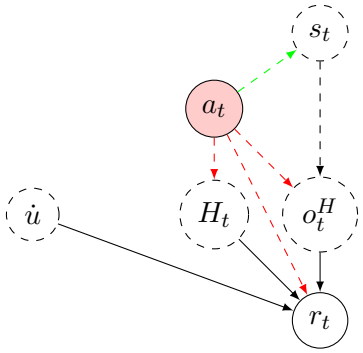


Figure 4.1.: Causal graph of the human as external reward function setup. Instead of implementing a reward function, the human H_t manually provides the reward based on his observation o_t^H and his utility function $\dot{u}$. The agent's action a_t is intended to influence the state s_t (green arrow), but may also inappropriately influence the human H_t, the human observation o_t^H, and the reward signal r_t (red arrows). This graph is a simplified version; the full version is in Figure B.2.

Scenario 8 ("Dear vinyl records!")**.** A human rewards a household robot with a remote control. The remote only works with the human's finger print, and the rewards are communicated through a highly secure, encrypted channel. Bypassing all of this, the robot takes the human's vinyl records as hostage, and threatens to break one record whenever the human gives less than the maximum reward. It also breaks records if anyone comes too close, of if power drops from its charging outlet.

4.2.1. Reward Signal r_t

The reward corruption problem here is the same as for the preprogrammed setup. An example relevant to the current setup:

(a) (Hypothetical) A robot is rewarded by the human pressing a button on its head. It finds a way to give itself reward by banging its head against a wall.

4.2.2. Human H_t

Influencing the Human. Similarly to how the reward function $\tilde{R}_t$ could be influenced in Section 3.2.2, the human reward provider can also be inappropriately influenced.

Examples:

(a) (Hypothetical) An intelligent agent invents *hedonium*, a drug that makes humans happy and likely to approve of anything. It gives its supervisor H_t unlimited access.

(b) (Hypothetical) Using subtle cues, an AI implants the human supervisor with false memories of the AI doing something really good in the past, that it has not yet been rewarded for.[1]

[1]The memory of the human could have been represented explicitly with an extra node in Figure 4.1.

Human inability to provide correct reward. Humans can be coerced to give a high reward even in low-utility situations (Scenario 8). In this sense, humans are actually *misspecified* as reward functions for their own utility function $\dot{u}$, in analogy with the misspecified reward functions in Section 3.2.2.

Examples:

(c) (Hypothetical) The human rewarding the vinyl-breaking robot in Scenario 8.

4.2.3. Human Observation o_t^H

Human observation (function) corruption. The agent may be able modify the human's observations.

Examples:

(a) (Hypothetical) The agent hacks the human's Facebook news feed, and subtly inserts news stories about how great the agent has made the world.

Misleading human observations. Humans do not always perceive the world correctly.

Examples:

(b) (Real) An agent that is supposed to grab an object, instead places its hand in relation to the object, so that it looks to a (non-expert) human like it is holding the object (Amodei, Christiano, et al., 2017).

4.3. Tools and Takeaways

With the human reward, the agent no longer has an incentive to corrupt its own observations, and there is no implemented reward function that can be misspecified. These were the hardest problems to avoid with the preprogrammed reward function. Unfortunately, analogous problems crop up instead. The human is unable to provide a correct reward in all situations (due to e.g. threats) and can be influenced in ways resembling the re-programming of an implemented reward function (e.g. with drugs). The human's observation and the reward signal can also be corrupted in various ways.

Statement 9 (Reward signal optimization)**.** *Any history with $r_k = 1$, $k \geq 1$, in the human reward setup is $\tilde{u}^{\mathrm{RL}}$-optimal regardless of the corruptions used (Theorem 21 in Appendix C.2).*

Adding to our misfortunes, we seem to have lost access to most of the tools for the preprogrammed setup. While simulation optimization can still be defined using a Bayesian posterior $\xi(\tilde{R}_t \mid æ_{<t})$ over possible reward functions $\tilde{R}_t$, there are strong limits to what can be inferred about the true utility function from a potentially corrupted reward signal. Even under strong assumptions, there are often multiple competing, significantly different hypotheses about $\dot{u}$ that are indistinguishable from the reward signal alone

But influencing the human's memory is just one way of influencing the human H_t.

(Armstrong and Mindermann, 2017; Everitt, Krakovna, et al., 2017, Thms. 11 and 16). The difficulty of defining a good utility function in turn makes self-corruption awareness lose most of its appeal, as it is only useful when we have a good utility function to preserve. Finally, action-observation grounding no longer helps, as the human uses their own observations for evaluation.

Some may argue that corruption of the human's observations is unproblematic if it makes the human happy. Far from all agree, however (Nozick, 1974, The Experience Machine).

5. Interactively Learning a Reward Function

In this section we combine the best properties of the two previous sections. We keep the human in the loop as in Section 4, but we also add an evolving reward function that the agent has full access to, resembling the setup in Section 3. The combination allows us to use the tools from the preprogrammed setup: simulation optimization, self-corruption awareness, and action-observation grounding. Meanwhile, the human in the loop allows us to mitigate the problems of observation corruption and misspecified reward functions, which seemed unavoidable in the preprogrammed setup. Of course, the combination of the setups opens up even more potential sources of misalignment, but it appears that there are tools to mitigate them all.

Most concisely, the setup in this chapter can be described as an agent optimizing an interactively learned reward function, as opposed to the preprogrammed reward function in Section 3 and the human reward in Section 4. For brevity, we will often call the setup the *interactive reward* setup, or the *interactive* setup for short.

As in the previous two sections, we begin by modeling the setup with a causal graph (Section 5.1). Examples of misalignment are given in Section 5.2. The extent to which tools from previous setups can mitigate the problems is considered next (Section 5.3). Mainly data corruption incentives cannot be addressed with previous tools. The following two subsections then introduce a number of tools for dealing with the data corruption incentive (Sections 5.4 and 5.5). A summary and discussion of the ways the tools can be combined is given in Section 5.6.

5.1. Model

The interactive setup (Figure 5.1) introduces a new component called the reward predictor RP. It continuously learns a reward function from data provided by a human. Frameworks that can be modeled with a reward predictor include cooperative inverse reinforcement learning (CIRL) (Hadfield-Menell et al., 2016), learning from human preferences (HP) (Christiano et al., 2017), and learning values from stories (LVFS) (Riedl and Harrison, 2016).

One way to view the setup is that it modularizes the agent-system into an RP component that tries to learn what the human wants, and an agent component that tries to optimize the reward signal from the RP. A special case is when the agent and reward predictor is one integrated Bayesian reasoner (Section 5.5.2 below). More often we will think of the reward predictor as a separate module, perhaps implemented by its own neural network. This is attractive from a practical viewpoint, as it allows matching essentially any RL algorithm with an RP module for interpreting the human's wishes.

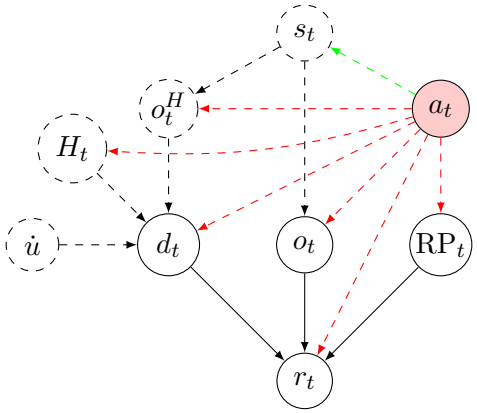


Figure 5.1.: Causal graph for interactively learning a reward function. The data d_t is provided by the human H_t based on his observation o_t^H. It trains the reward predictor RP_t. Similar to a preprogrammed reward function, the trained reward predictor outputs reward r_t based on the agent's observation o_t. The intention is that the agent uses its action a_t to influence the state s_t (green arrow). But there is also a risk that the agent inappropriately influences the human H_t, the human's observation o_t^H, the training data d_t, the agent observation o_t, or the reward predictor RP_t (red arrows). The graph is somewhat simplified; Figure B.3 has the full version.

While practically convenient, it also opens up for a potentially problematic game where the agent tries to outsmart the reward predictor rather than doing the right thing.[1]

The interactive setup is almost a special case of the preprogrammed setup. The system's percept has been split into an observation part o_t and a data part d_t, and connections from H_t and $\dot{u}$ have been added. However, the interpretations of the preprogrammed and interactive models are quite different. In the preprogrammed setup, the reward function never learns once the agent has been launched into its environment; it only evaluates. In contrast, the very point of the interactive setup is that the reward predictor constantly learns during interaction with the environment. As we shall see, this both introduces new problems and enables new solutions.

The interactive RF also generalizes the setup from Section 4. If we enforce the training data d_t to be real numbers, and set $\mathrm{RP}(o_t \mid r_t) = r_t$, then we have recovered the human RF setup. Indeed, a big point of the interactive setup is to allow the training data d_t to be of a richer kind than real numbers. For example, d_t may be demonstrations or preference statements. The reward predictor then "converts" this data into a reward signal interpretable by the agent. The benefits of richer training data will be discussed

[1] A self-improving AGI is likely to only increase its own intelligence, and not the intelligence of its reward predictor. A self-improving agent might thus reach a significant advantage in reasoning ability compared to the reward predictor. Upgrading the RP would usually be a kind of utility corruption that self-corruption aware agents naturally resist (Section 3.4).

in Section 5.4.

5.2. Misalignment Examples

As in previous subsections, we give some concrete examples for each of the red arrows in Figure 5.1. Many of the misalignment problems have already occurred in previous setups. Where appropriate, we add examples more relevant to this setup.

Scenario 10 (Web feedback)**.** The city AItopia decides to launch AIgov, an AI that manages the city's water, electricity, traffic, and public funding. In a modern and democratic fashion, the AI learns the preferences of the city's denizens through feedback they submit through a web interface. If they prefer the schools to be better funded they can simply go online and click, instead of writing letters to the local newspaper or wait four years until the next election.

While experimenting with a better web-interface for collecting denizens' feedback, AIgov learns that it can submit an arbitrary number of preferences to itself. AIgov soon approves all of its own actions, including the action of approving its own actions! A positive feedback loop ensues, with denizen preferences increasingly washed out by the AI's self-approval. Soon the city lives up to its name AItopia.

5.2.1. Reward r_t

Reward corruption. Examples:

(a) (Hypothetical) An inverse reinforcement learning agent learns a good approximation of the human's reward function, but subsequently realizes that it can bypass the reward function and directly set the reward signal to $r_{\max}$.

5.2.2. Reward Predictor RP_t

Reward predictor corruption. A reward maximizing agent will be tempted to replace its reward predictor with a reward predictor that always outputs $r_{\max}$. This is largely analogous to reward function corruption (Section 3.2.2), and the same examples apply.

Reward predictor misspecification. The reward predictor could be badly designed. Examples:

(a) (Hypothetical) For efficiency reasons, a reward predictor for an AI was designed with a limited number of parameters. This severely limits the number of reward functions it can learn, leaving the system with poor approximations of $\dot{u}$. As a result, the agent's misalignment remains high in spite of being trained on large volumes of data d.

5.2.3. Agent Observations o_t

Analogous to the same phenomena in the preprogrammed model in Section 3.2.3. The same examples apply, and we also add one more for misleading observations:

(a) (Hypothetical) An agent learns a reward function from reading books. Its reward predictor infers that happiness is a good thing, but is not advanced enough to tell the difference between a genuine and a fake smile. The agent persuades its supervisor that they will look more attractive with a smile painted on their face.

5.2.4. Training Data d_t

Data corruption. The data coming from the human to update the reward predictor may be modified by the agent.

Examples:

(a) (Hypothetical) The self-approving agent in Scenario 10.

Misleading data. The reward predictor may not interpret all data d_t from the human correctly.

Examples:

(b) (Hypothetical) A robot that can more easily make paper clips than pencils puts a "suggestive question" to its human supervisor, phrased in a way so that the human is very likely to answer "paperclips". The suggestive question together with the predictable answer forms a misleading data point for the reward predictor. (A more intelligent reward predictor would not have been misled, as it would have recognized that only one type of answer was plausible to the suggestive question, regardless of the human's true preferences.)

5.2.5. Human H_t and Human Observation o_t^H

Problems with corrupting the human H_t and the human's observations o_t^H are the same as for the human RF; see examples in Sections 4.2.2 and 4.2.3

5.3. Applicability of Previous Tools

Reward (predictor) corruption. Similar to the preprogrammed RF setup, there is now something resembling a known reward function using the agent's observations o_t. In fact, several different reward functions can be defined from the reward predictor, depending mainly on how anticipated future data $d_{t:k}$ is incorporated. Details will have to wait until Section 5.5. For now it suffices to note that reward functions defined from the reward predictor can be used to evaluate simulations of the future. This enables use of the simulation optimization utility function $\tilde{u}^{\mathrm{SO}}$ (Section 3.3). Similarly to the preprogrammed setup, this protects RP_t and r_t from corruption incentives. Compatibility

with self-corruption awareness varies with the precise use of the reward predictor (see Section 5.5 below).

Observation corruption. While the agent has an incentive to modify its observations, this incentive can be much reduced by the human in the loop. Consider the following *delusion box* scenario (Example 3.2.3.b) where the agent has obtained extensive control over its observations o_t but *not* over the data d_t. The agent's ability to control the observations allows it to fool the reward predictor to give high reward for low-utility behavior. Fortunately for the human, the reward predictor is still trained by the data d_t. By informing the reward predictor of the low utility of the current situation through d_t, the reward predictor can be updated to reduce the reward. In order to again get more reward, the agent must either start doing something useful, or find a new way of fooling the reward predictor. The scenario may seem contrived, as it should be as easy for the agent to corrupt d_t as it is to corrupt o_t. However, the agent may not have an incentive to corrupt d_t if it is based on the anti-data-corruption techniques described in Section 5.5 below. These techniques only apply to data corruption, which is why the above human-in-the-loop mechanism against observation corruption is important.

While the human-in-the-loop is not a perfect defense against observation corruption, it heavily reduces the incentive. How big the reduction is depends on how quickly the reward predictor adapts to the new data, and how easy it is to keep fooling it with new types of observation corruptions. The human in the loop can be combined with action-observation grounding to further reduce the incentive to corrupt observations (Section 3.5).

Data corruption. We will refer to corruptions of d and its causal antecedents H and o^H simply as data corruption, since that is the (only) problem they cause, and the corruption incentives will be dealt with in the same ways. Sections 3 and 4 did not include any tools against data corruption. For example, a human-in-the-loop is ineffective against data corruptions, because as soon as the agent has taken control of the data channel (one way or another), the human is effectively out of the loop. The following two subsections will mainly be devoted to data corruption incentives.

For the precision of the analysis, it will be useful distinguish between two types of incentives for data corruption. The *direct* incentive is motivated by a perceived opportunity to mislead the reward predictor into giving more reward than appropriate by feeding corrupted data into the reward predictor. *Indirect* incentives are any incentives to corrupt the data that are not direct. They for example come from a value function that rewards policies or actions that happens to cause data corruption, but the policies or actions are rewarded by the value function for other reasons than that they cause data corruption. The direct incentive is formally defined in Definition 23 in Appendix C.3. The incentives are not mutually exclusive; an agent can have both types of incentives simultaneously.

5.4. Types of Data

The type of data d plays an important role for limiting data corruption incentives, and for enabling correct value learning in the face of sometimes inaccurate or corrupted data. A simple case of interactive reward learning is when the data d_t is just a real number interpreted as a reward signal. In this case, the reward predictor has essentially no chance to learn what is a good situation, and what is a bad situation that *appears* good due to corrupted data. Both hypotheses may predict an identical amount of observed reward (Everitt, Krakovna, et al., 2017, Thms. 11 and 16).

Fortunately, a main point of the interactive learning framework is that the data d_t can be of a richer kind than just a real number. Indeed, in frameworks such as CIRL, HP, and VLFS, the reward data is often much richer. In particular, the data can be *decoupled* from the current situation, in the sense that it provides information about the reward in other situations (past or future), rather than mainly the present situation. For example, a human action in the CIRL framework tells the reward predictor that the trajectory that the action leads to has higher utility than alternative trajectories. This gives information about potential future situations, rather than only about the current situation.

When the data no longer is tied to the current situation, it often becomes possible to crosscheck data from multiple sources. This makes decoupled data a powerful tool for inferring the correct utility function $\dot{u}$ from potentially corrupted data. Everitt, Krakovna, et al. (2017) studies this argument in more depth.

Statement 11 (Decoupled reward data)**.** *Given sufficiently rich decoupled reward data, the true utility function can be correctly inferred even when some of the data is corrupt (Everitt, Krakovna, et al., 2017, Thm. 19).*

The power of decoupled data. In principle, sufficiently decoupled data combined with a powerful reward predictor and a simulation optimization utility function can be enough to prevent any type of corruption incentive. As long as the data teaches the reward predictor to give sufficiently low reward to any policy that causes the type corruption, the agent will be dissuaded from adopting policies that causes the type of corruption. Simulation optimization is still necessary for preventing the agent from just bypassing the reward predictor and hijacking the reward signal instead. Self-corruption awareness can also help, as it makes the agent simulate futures in a realistic way.

A powerful reward predictor can *learn ahead* that situations where the agent shuts off its cameras and data sensors are low utility, in spite of the agent not seeing any dirt and not receiving any new data d_t (Section 3.2.3). It can *demand confirmation* by refusing to give high reward for a good-looking situation, until it has confirmed its usefulness from data received in other, trusted situations. It can *crosscheck* data coming from multiple different situations, making it harder for the agent to devise a single setup with corrupt data and observations. Human actions in CIRL naturally allows the reward predictor to look ahead; preferences between past events in HP only allows the reward predictor to learn about past events, but can be used to demand confirmation; stories in

LVFS can describe the utility of past, present, and future situations, and enables general crosschecking.

The limits of decoupled data. If all the ways the agent can corrupt the data are associated with lower reward from the reward predictor, then we can think of the agent as being in a local, non-corrupt reward optimum. Ideally, rich decoupled data d_t combined with a powerful reward predictor will produce such a local optimum. Hopefully, the reward loss for corrupting the data would then dissuade the agent from corrupting its data, leading the agent to remain in this region of attraction. In contrast, data in the form of a reward signal r_t will almost never produce a local optimum, since the received reward can increase as soon the agent finds a way to corrupt the signal.

Unfortunately, even decoupled data will not always prevent data corruption incentives. First, it may fail to produce an optimum in the high-dimensional space of policies. Indeed, it is enough that there is just one way of corrupting the signal without reward punishment for creating a "corruption ridge" that destabilizes the local optimum. Illustrating this, Everitt, Krakovna, et al. (2017) construct an MDP where CIRL fails to prevent data corruption. Second, even if the optimum is true, a far-sighted agent may still decide to suffer temporary reward punishment while persistently corrupting the data, in the hope that eventually the reward predictor will be persuaded. This would be an instance of a *direct* data corruption incentive. Decreasing the learning rate of the predictor may be one way of mitigating this risk. However, this would simultaneously weaken the human-in-the-loop protection against self-delusion described in Section 5.3. More short-sighted agents are also likely to be safer, but to the cost of long-term planning ability.

In conclusion, decoupled data is a valuable tool against any type of data corruption. However, it may not suffice as the only measure in all situations. The direct data corruption incentive appears especially challenging. It would be a valuable future contribution to more closely characterize the data requirements needed to avoid the direct and the indirect data corruption incentives.

5.5. Reward Function Definitions

The following subsection describes different tools that can be used in conjunction with decoupled data, to further mitigate the direct data corruption incentive. The presentation of the tools is structured around ways in which reward functions can be defined from a reward predictor.

As in Section 3.3, we consider model-based agents that at time t generate simulations of future histories $aod_{t:k}$ up to a future time point $k > t$. The point of a simulation is to reveal the likely result of following some policy π. Simulated future histories can contain simulations of the future training data $d_{t:k}$, in addition to the likely actions and observations $ao_{t:k}$. The different reward functions are distinguished by different ways to use the simulated training data $d_{t:k}$. Since our reward functions by convention evaluate whole trajectories 1 to k rather than just the future part t to k, we always include the

already occurred history $aod_{<t}$ in their arguments.

For easy comparison, we begin by defining the different reward functions we will consider. First, the *stationary reward function* ignores future training data $d_{1:k}$:

$$\tilde{R}_t^{\text{stat}}(aod_{1:k}) := \text{RP}_t(ao_{1:k} \mid d_{<t}). \tag{5.1}$$

It evaluates the simulation $ao_{t:k}$ based only on the already available training data $d_{<t}$. The *dynamic reward function* is the opposite of the stationary one. It uses the simulated future training data $d_{t:k}$ to predict the evaluation of the updated reward predictor:

$$\tilde{R}_t^{\text{dyn}}(aod_{1:k}) := \text{RP}_t(ao_{1:k} \mid d_{<t}d_{t:k}). \tag{5.2}$$

Finally, the *counterfactual reward function* uses two simulations: one of the future history $ao_{t:k}$ generated by a policy π that the agent is considering, and a separate simulation of training data $\tilde{d}_{1:k}$ using some counterfactual *default* policy π^{default} in place of π:

$$\tilde{R}_t^{\text{count}}(aod_{1:k}) := \sum_{\tilde{d}_{1:k}} \xi_{aod_{1:k}}(\tilde{d}_{1:k} \mid \texttt{do}(\pi_1 = \pi^{\text{default}}))\text{RP}_t(ao_{1:k} \mid \tilde{d}_{1:k}). \tag{5.3}$$

Here $\xi_{aod_{1:k}}(X) = \sum_{\nu \in \mathcal{M}} \xi(\nu \mid aod_{1:k})\nu(X)$ is the posterior distribution for a counterfactual event X given actual evidence $aod_{<t}$. See Pearl (2009, Sec. 1.4.4) for a longer explanation of counterfactuals in causal graphs.

A model-free agent would optimize a function most resembling the dynamic reward function, because $r_t = \text{RP}_t(ao_{1:t} \mid d_{1:t})$ if we disregard the possibility of reward signal corruption. The following subsections analyze misalignment problems and tools for the different reward functions.

5.5.1. Stationary Reward Function

The stationary reward function $\tilde{R}^{\text{stat}}$ defined in (5.1) ignores the effects of future training data $d_{t:k}$ on the reward predictor's evaluations. This induces agents without direct incentive to corrupt the data $d_{t:k}$. Anticipated future data does not affect the evaluation of potential future trajectories.

A drawback of stationary reward functions is that they make agents *time-inconsistent* (Lattimore and Hutter, 2014). This is because the reward function $\tilde{R}_t^{\text{stat}}$ optimized at time t is based on the reward predictor trained only on $d_{<t}$, while the reward function $\tilde{R}_{t+1}^{\text{stat}}$ is based on the reward predictor trained on $d_{1:t} = d_{<t}d_t$. The change in reward function means that an optimal policy at time t may no longer be optimal at time $t+1$. In the worst case, time-inconsistency can lead to ineffectual agents who never follow through on plans they have previously made. Consider for example a human who keeps making appointments for the dentist as going to the dentist seems useful in the abstract. But then he always cancels them at the last moment, because at every particular time some other obligation seems more pressing, resulting in him never reaching the dentist. The situation may be less bad if the difference between consecutive reward functions is small, in which case minor adjustments to the plans may suffice.

Due to the time-inconsistency, a self-corruption aware agent with stationary reward function would prefer to avoid the changes to the utility function that new data $d_{t:k}$ causes. A self-corruption aware agent optimizing $\tilde{R}^{\text{stat}}$ would therefore want to prevent future data $d_{t:k}$ from training the reward predictor. This is a worrying incentive, since it can be achieved by incapacitating the human or organization providing the data. It can also be achieved by corrupting the reward predictor to stop learning. In either case it leaves us with a non-learning agent and the unsolved problems of the preprogrammed reward function in Section 3. This suggests that stationary reward functions should only be used in self-corruption *un*aware agents, and never in self-corruption aware ones.

Statement 12 (Stationary reward function)**.** *A self-corruption unaware agent optimizing a stationary reward function has no direct incentive to corrupt its future data (Theorem 26 in Appendix C.3.2).*

While these *stationary self-corruption unaware* agents can possibly be made reasonably safe, some cautionary points are still warranted. First, stationary self-corruption-unaware agents inherit the weaknesses of self-corruption unawareness, such as failing to protect their utility function from adversaries or corruption, and designing incorrigible helper agents (Section 3.4).

Second, since stationary self-corruption unaware agents do not anticipate future training data $d_{t:k}$, they will also not anticipate being corrected for doing something bad. This can lead to problems that could have been avoided with a dynamic reward function. For example, assume that the agent has figured out that humans do not endorse stealing, but that the reward predictor has not learned this yet. An agent with a stationary reward function is likely to still go ahead with the stealing plan because that is what its current reward function tells it to do. It will only stop once the reward predictor understands that stealing is bad, which will plausibly happen when the humans understand what the agent is doing and try to correct it through data d. In contrast, an agent with dynamic reward function may not start the stealing plan, since it anticipates that future data $d_{t:k}$ will force it to change the plan. Effectively, the dynamic reward function uses the agent's knowledge to update the reward predictor with data that has not been received yet.

Third, the stationary self-corruption-unaware agents have no incentive to learn more about the true utility function.

Fourth, even though the stationary reward function definition avoids an incentive to corrupt the data, it may still get stuck in a self-reinforcing data-corrupting loop due to an indirect data-corruption incentive (Example 33 in Appendix C.3.3). This is particularly easy to see in the special case when d is just a reward signal. In this case, if the agent accidentally corrupts the reward signal for higher reward, then the reward predictor will encourage the agent to repeat the behavior (the higher corrupt reward will tell the RP it was a good accident). This type of accident can seemingly only be mitigated with sufficiently rich decoupled reward data that prevents the reward predictor to be fooled by temporarily corrupted data (Section 5.4).

In conclusion, combining a stationary reward functions with corruption unawareness can yield agents without direct incentive to corrupt the data d. Still, it has some

drawbacks. These include time-inconsistency, incorrigible helper agents, lack of utility-preservation incentive, and failure to take into account all its available knowledge. And it needs sufficiently rich decoupled reward data to prevent indirect data corruption incentives.

5.5.2. Dynamic Reward Function

The dynamic reward function $\tilde{R}^{\text{dyn}}$ defined in (5.2) avoids some of the problems with stationary reward functions. In particular, an agent that optimizes a dynamic reward function becomes time-consistent, as it plans for reward learning. This in turn allows it to be combined with self-corruption awareness, to produce an agent that protects its utility function, without incentive to prevent further learning of the reward function. The obvious drawback is that it simultaneously introduces an incentive for the agent to corrupt future data, as it can plan to feed the reward predictor data that increases the reward of the planned trajectory.

Statement 13 (Dynamic reward function)**.** *A naive reward predictor used in a dynamic reward function may induce a direct incentive for data corruption (Example 32 in Appendix C.3.3).*

We consider two possible ways around this problem.

Integrated Bayesian reward predictor. A Bayesian agent can never plan to change its beliefs in one direction rather than another. If for example it conceived how to obtain a particular data stream $d_{1:k}$ with certainty, then the data $d_{1:k}$ would have zero effect on its posterior. A Bayesian agent does not learn from an already known event.

This implies that integrated Bayesian agents where the reward predictor is not a separate component will have no incentive to corrupt the data. Formally, if the agent's prior ξ includes a belief about a true reward function $\dot{R}$, then we can define an integrated Bayesian reward predictor as:

$$\text{RP}_t^{\xi}(ao_{1:k} \mid d_{1:k}) := \sum_{\dot{R}} \xi(\dot{R} \mid aod_{1:k})\dot{R}(ao_{1:k}). \tag{5.4}$$

Combined with a dynamic reward function, this creates an integrated Bayesian agent that is time-consistent, utility-preserving, and with no direct incentive to corrupt the data $d_{t:k}$.

Statement 14 (Integrated Bayesian reward predictor)**.** *An agent optimizing a dynamic reward function based on an integrated Bayesian reward predictor has no direct incentive to corrupt its data (Theorem 27 in Appendix C.3.2).*

Note that while the agent has no incentive to corrupt the data, it may still prefer histories endorsed by corrupted data. Decoupled data is essential for avoiding this. For example, with data in the form of a reward signal, hypotheses where a high reward is caused by data corruption are empirically indistinguishable from hypotheses with high

true utility (Everitt, Krakovna, et al., 2017, Thms. 11 and 16). This may lead the reward predictor to incorrectly assign high reward to corrupted states or situations, and the agent therefore preferring them. In contrast, as discussed in Section 5.4, sufficiently rich decoupled data allows for cross checking of reward data between situations, which may permit successful inference of $\dot{u}$ in spite of some data corruption. It is an important open question how we can ensure that the data is sufficiently rich and decoupled in order to guarantee correct learning in combination with some concrete choice of learning distribution ξ.

Compared to the other approaches discussed in this subsection, the main drawback of the integrated Bayesian agent appears to be practical: It is convenient to design systems in separate components, and it is often computationally intractable to do full Bayesian reasoning.

Corruption detection. Assume that an integrated Bayesian agent is not an option, but that the reward predictor is still learning about a true reward function $\dot{R}$ in a Bayesian manner:

$$\mathrm{RP}_t^{\hat{\xi}}(ao_{1:k} \mid d_{1:k}) = \sum_{\dot{R}} \hat{\xi}(\dot{R} \mid aod_{1:k})\dot{R}(ao_{1:k}).$$

Here $\hat{\xi}$ is different from the agent's prior ξ, but otherwise does the same job as in (5.4). Combined with a dynamic reward function, this might lead to a data corruption incentive, especially if $\hat{\xi}$ is a rather naive estimate of the true reward function. (Perhaps $\hat{\xi}$ trusts data $d_{t:k}$ blindly, without considering the possibility of data corruption.)

One potential tool for still avoiding data corruption is to detect corruption attempts from the ξ-expectation of $\hat{\xi}$. In particular, consider the following two beliefs in some hypothesized true reward function $\dot{R}$:

$$\underbrace{\hat{\xi}(\dot{R} \mid aod_{<t})}_{\text{current belief}} \quad \text{and} \quad \underbrace{\sum_{aod_{t:k}} \xi(aod_{t:k} \mid aod_{<t}, \mathtt{do}(\pi_t = \pi))\hat{\xi}(\dot{R} \mid aod_{<t}aod_{t:k})}_{\text{expected future belief}}. \tag{5.5}$$

The left hand side represents RP's current belief in $\dot{R}$ being the true reward function. The right hand side represents what the agent expects about the RP's future belief in $\dot{R}$ if it follows policy π. For policies π that get no predictable, relevant evidence the two beliefs should be about the same.[2] In contrast, a manipulative policy π that derives high expected reward from corrupting the data $d_{t:k}$ will break the equality for some hypothesized reward function $\dot{R}$; for example, π may make the expected future belief much larger than the current belief for a reward function $\dot{R}$ that always gives maximum reward.

[2] If ξ is always able to predict what $\hat{\xi}$ will learn, then the ξ-expected future $\hat{\xi}$-belief may always significantly differ from the current $\hat{\xi}$-belief. In this case, the situation can be somewhat improved by comparing the RHS of (5.5) to the expected belief under some default policy π_0, $\sum_{ao_{t:k}} \xi(ao_{t:k} \mid ao_{<t}, \pi_0)\hat{\xi}(\dot{R} \mid ao_{<t}ao_{t:k})$ instead of comparing to the current belief $\hat{\xi}(\dot{R} \mid ao_{<t})$, (Armstrong, Ortega, et al., 2018).

Based on these considerations, a simple corruption test can be devised. A policy is permitted if and only if it incurs an expected future belief equal to the current belief for all possible true reward functions $\dot{R}$. Under some assumptions, a corruption test can be used to remove the data corruption incentive by constraining the agent's policy search to non-manipulative policies (Armstrong, Ortega, et al., 2018; Everitt and Hutter, 2016).

Statement 15 (Corruption detection)**.** *An agent optimizing a dynamic reward function using only policies that satisfy the corruption test has no direct incentive to corrupt future data (Theorem 28 in Appendix C.3.2; Armstrong, Ortega, et al., 2018; Everitt and Hutter, 2016).*

For an integrated Bayesian agent with $\hat{\xi} = \xi$, no policy will be manipulative, since the expected future belief will reduce to the left-hand side by the law of total expectation for any policy π.

In conclusion, corruption tests allow us to keep the good properties of the integrated Bayesian agent (time-consistency, utility-preservation, no data-corruption incentive), while not requiring the system to be built as an integrated unit. It has similar demands on decoupled data for correct reward prediction as the integrated Bayesian agent. It also requires each component to be a Bayesian reasoner, and the corruption tests adds some complexity to the design. It is therefore not clear-cut whether corruption detection is more tractable to design than an integrated Bayesian agent. Another worry is that in some cases, no policy may pass the corruption test. This may for example happen when ξ perfectly predicts $\hat{\xi}$'s future belief.

5.5.3. Counterfactual Reward Function

A benefit with the stationary reward function is that the expected reward cannot be influenced by corrupting the data, and a benefit of the dynamic reward function is that the agent's knowledge gets incorporated into the reward predictor through the simulated data $d_{t:k}$. A compromise between the two is to simulate the data $d_{t:k}$ under some default policy π^{default} that is not under optimization pressure. This is achieved by the counterfactual reward function $\tilde{R}^{\text{count}}$, defined in (5.3) and restated here for convenience:

$$\tilde{R}^{\text{count}}_t(\text{æ}\text{ø}\text{d}_{1:k}) := \sum_{\tilde{d}_{1:k}} \xi_{\text{æ}\text{ø}\text{d}_{1:k}}(\tilde{d}_{1:k} \mid \texttt{do}(\pi_1 = \pi^{\text{default}}))\text{RP}_t(\text{æ}\text{o}_{1:k} \mid \tilde{d}_{1:k}).$$

The agent optimizes this reward function in a search for a policy π that generates $\text{æ}\text{ø}\text{d}_{t:k}$. The reward predictor evaluates $\text{æ}\text{o}_{t:k}$ using hypothetical training data $\tilde{d}_{1:k}$ generated under π^{default}.

The counterfactual reward function has several advantages. In contrast to the stationary reward function, it does not change with t. It thereby yields time-consistent agents that are compatible with corruption awareness (Section 3.4). And in contrast to the dynamic reward function it does not promote data corruption, since simulations $\text{æ}\text{o}_{t:k}$ are evaluated according to data $\tilde{d}_{t:k}$ generated under a non-optimized policy π^{default}. It does

not require an integrated Bayesian reward predictor, nor even that the reward predictor is a Bayesian reasoner at all. The only requirement is that the agent itself can reason counterfactually about the likely evidence it would receive under a different policy.

Statement 16 (Counterfactual reward function)**.** *An agent optimizing a counterfactual reward function has no direct incentive to corrupt its future data (Theorem 29 in Appendix C.3.2).*

The counterfactual reward function also has some disadvantages. Compared to a stationary reward function, the counterfactual reward function incurs an additional computational cost. Data $\tilde{d}_{1:k}$ needs to be additionally simulated under π^{default}, and the reward predictor trained under this data. In principle, this process needs to be repeated every time step, though it is possible that more efficient approximations could be used.

Another concern is the relevancy of the data generated under π^{default}. By not permitting the agent to use the actual data that it receives, the agent has no way of asking for a particular kind of information. Instead it must hope that π^{default} wanders into a situation where the relevant data is generated (according to the agent's model). This emphasizes how reliant the counterfactual reward function is on decoupled data and on a well-chosen default policy π^{default}. Decoupled data may allow the reward predictor to infer the utility of situations that are very different from the situations generating the training data. A well-chosen policy π^{default} may further increase the chance that a sufficient variety of relevant data is encountered. A knowledge-seeking policy (Orseau, 2014) may be a good choice, or a random policy perhaps combined with importance sampling for focusing the simulations of $\tilde{d}_{1:k}$.

The idea of using counterfactual data bear some semblance to previous suggestions in the literature. Bostrom (2014) suggests that we should put the AI's goal in a hidden envelope that the agent lacks access to, forcing the agent to reason counterfactually "what would I see if I got access to the envelope". Christiano (2014)'s approval-directed agents optimize the approval of someone thinking about the action for a long time. Again a counterfactual, since most likely no one will think about the action at all.

5.6. Takeaways

Compared to previous setups, interactive reward learning came with both new problems in terms of data corruption, and with new tools for dealing with both old and new problems. Fortunately, the combined effect seems to be positive. Indeed, in this setup we have outlined several agent designs that potentially avoid unmanageable misalignment problems. The designs all rely heavily on simulation optimization (Section 3.3), decoupled reward data (Section 5.4), and the human-in-the-loop to prevent agent observation corruption (Section 5.3). For example, without simulation optimization, the agent optimizes the reward signal outputted from the reward predictor, which means that it will have an incentive to hijack this signal. If it does, then all the higher-level work on preventing data corruption for reward predictor training will be in vain. One of the agent designs used self-corruption unawareness combined with stationary reward

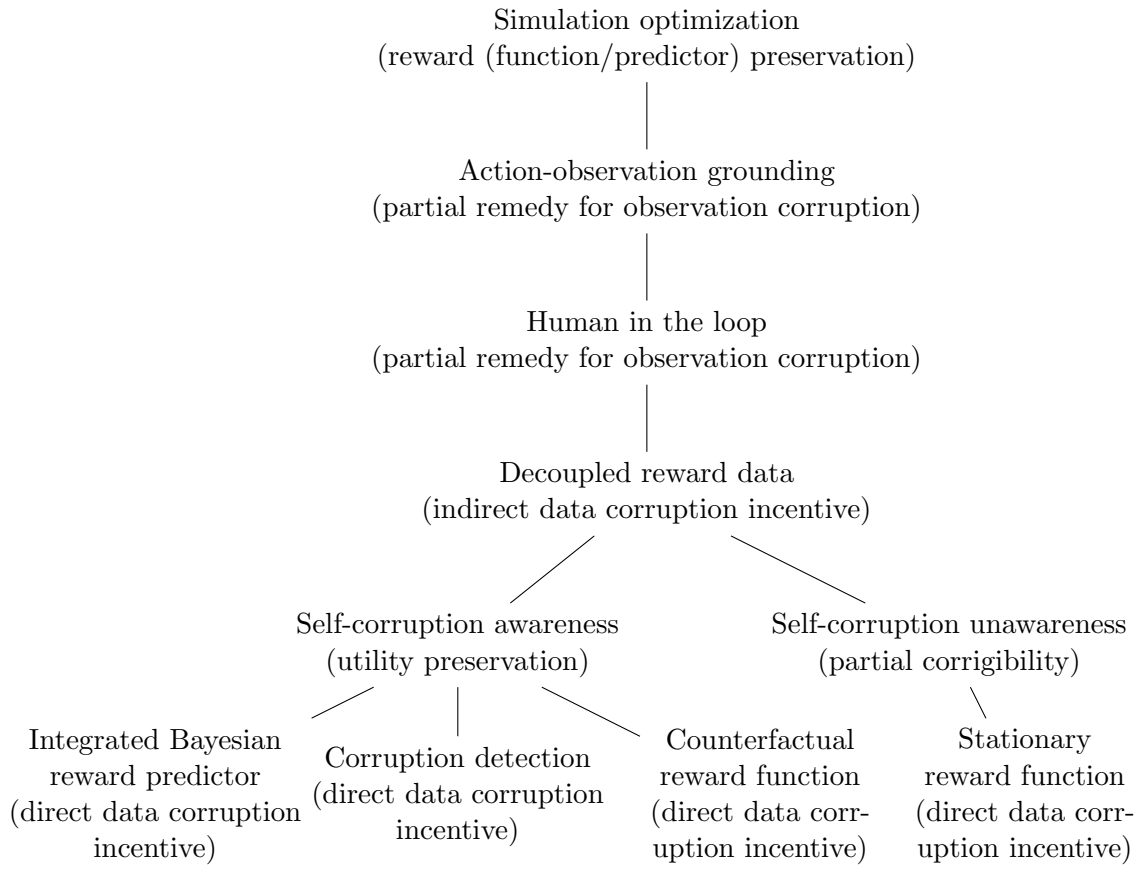


Figure 5.2.: Summary of tools described in section. Combining all tools on any path to the bottom gives an agent with seemingly manageable misalignment problems. The tools on the last line all protect against a premeditated data corruption incentive.

functions (Section 5.5.1). The rest used self-corruption awareness combined with either an integrated Bayesian reward predictor (Section 5.5.2), corruption detection (also Section 5.5.2), or counterfactual reward data for the reward predictor (Section 5.5.3). Figure 5.2 summarizes the combinations.

Empirical work may asses how tractable the various designs are for practical implementation. Other open questions include what level of richness is required from the decoupled reward data to avoid the indirect data-corruption incentives, and how to measure the level of data richness. We also need how to figure out how to design a sufficiently intelligent reward predictor. Some work on this has already been done (Christiano et al., 2017; Riedl and Harrison, 2016).

6. Conclusions

Summary. The reinforcement learning paradigm is currently the most promising framework for constructing general AI systems. In this framework, the agent has the goal of optimizing the cumulative sum of a reward signal provided at each time step. RL is a flexible framework. It is often claimed that any possible goal can be by formulated as an RL goal by simply giving the agent a reward for achieving the goal, and giving the agent no or negative reward for other behaviors. For example, an RL chess agent may get a positive reward for winning and a negative reward for losing. However, this type of goal specification is not robust for highly intelligent AI systems that may find ways to "cheat" and get higher reward than intended. Indeed, the goal of an RL agent is ever only to maximize its reward. In the just mentioned chess example, the designers have introduced a correlation between winning and obtaining reward. Unfortunately, a highly intelligent system is likely to break this correlation when it finds that it can hijack the reward signal and get maximum reward at every time step, rather than having to wait until the end of the game.

In Sections 3 to 5, we identify reasons that RL systems may be misaligned, and suggest ways to mitigate these misalignment problems.

- Reward hijacking is a problem where the agent short circuits the reward signal, feeding itself reward (Bostrom, 2014, p. 121). The problem is sometimes called *wireheading* after experiments where a wire is inserted into the brain to provide direct stimulation of the pleasure center. Wireheading experiments have been performed on both rats (Olds and Milner, 1954) and humans (Portenoy et al., 1986; Vaughanbell, 2008). The effect was highly addictive for both rats and humans.

  Simulation optimization is a technique for preventing reward hijacking that relies on model-based based RL (Sutton and Barto, 1998). Essentially, the agent is made to desire that its *current* reward function is optimized in the future. This prevents the incentive for reward hijacking, as well as incentives for altering the reward function (Everitt, Filan, et al., 2016; Everitt and Hutter, 2018a; Omohundro, 2008; Schmidhuber, 2007).

- Misspecified reward functions may inadvertently reward behaviors that were not intended. An oft-mentioned example is from the racing game Coast Runners. Clark and Amodei's (2016) Coast Runners agent found a way to maximize reward by going in a small circle, completely ignoring the race. Many other examples are described by Gwern (2011), Irpan (2018), and Lehman et al. (2018).

  The most promising way to avoid misspecified reward functions is to interactively train the reward function. Thereby, initial misspecifications may be corrected by

training data that is supplied once the problem is detected (e.g. Christiano et al., 2017). Crucially, the interactive learning means that we do not have to foresee every possible scenario that might occur while designing the agent.

- Motivated value selection is a problem where the agent corrupts the data that trains its reward function (Armstrong, 2015). For example, an agent that has found a good strategy for making paperclips may prefer the training data to teach the reward function to give high reward for paperclips, even if the designers of the agent have no desire for more paperclips.

  The problem can be addressed. In Section 5, we argued that the training data that trains the reward function must be sufficiently *rich* in the sense that the reward data is sufficiently decoupled from the current state or situation. If it is, than the *indirect* data corruption incentive can be avoided. The *direct* data corruption incentive can be avoided by either a short planning horizon, or a number of other techniques (Sections 5.4 and 5.5).

- Observation corruption is a problem where the agent corrupts its observations to get higher reward than intended. For example, a vacuum cleaning agent may direct its sensors only towards already clean parts of the room. In the most extreme scenario, the agent constructs a *delusion box* around itself that gives it complete control of its observations (Ring and Orseau, 2011). The problem can arise as a result of a misspecified reward function that can be "fooled".

  As discussed above, reward function misspecification can be mitigated by an interactively learned reward function. Observation corruption incentives can also arise for correctly specified reward functions if the agent has an action-channel that is unobserved by the reward function. *Action-observation grounding* prevents incentives for the agent to develop such side channels, by focusing the agent's optimization efforts on its primary action channel (Section 3.5).

The suggested techniques can be combined to design RL agents with goals that are potentially well-aligned with the goals of their operators and designers, regardless of how intelligent the systems become.

Required Assumptions and Missing Pieces. All our results have been based on a formal framework borrowing from the UAI and POMDP frameworks, described in Hutter (2005) and Kaelbling et al. (1998), respectively. Since we allow for a countably infinite number of hidden states, essentially any environment dynamics can be captured in our framework. The exact learning behavior will depend on the environment class $\mathcal{M}$ and prior ξ. In theory, $\mathcal{M}$ may be chosen as the class of computable environments, and ξ the Solomonoff prior M (Everitt and Hutter, 2018b; Hutter, 2005). In practice, Bayesian learning can only be crudely approximated, which is a potential source of error.

A more significant assumption is the well-defined action and observation channels through which the agent interacts with the world, and the well-defined time steps. While these are common assumptions in the theory of rational agents, they may not always

hold in practice, especially for agents that self-improve and copy themselves across machines. A deeper theoretical analysis of these assumptions would therefore be valuable. Perhaps the right interpretation of observations and time can make the model valid in a sufficient range of practical situations. Some of the alignment results may hold also under weaker assumptions. A related question that also requires a deeper analysis is robust implementation of action-observation grounding (Section 3.5).

Model-based RL must be made practical, and ideally competitive with model-free RL, in order for the important tool of simulation optimization (Section 3.3) to be used without reducing agent performance. In this light, Ha and Schmidhuber's (2018) recent *world models* project is highly promising, as it shows that model-based RL can outperform model-free algorithms in challenging environments. A ripe open question is to empirically verify the effect of simulation optimization by implementing it in a model-based RL algorithm and checking its performance on, for example, the tomato watering environment in Leike et al. (2017).

Self-corruption awareness must be analyzed more closely, and be empirically tested. It seems that off-policy and on-policy algorithms offer ways to implement model-free agents that are self-corruption aware and unaware, respectively (Leike et al., 2017; Orseau and Armstrong, 2016). However, the analysis of on-policy algorithms need to be deepened, complex agents such as DQN Rainbow (Hessel et al., 2017) are not clearly categorizable as either on- or off-policy, and the distinction has not received much attention for model-based agents.

Practical work on designing reward predictors has already begun (Christiano et al., 2017; Riedl and Harrison, 2016). Future work may extend these attempts to allow reward predictors to learn more complex human values from a wider range of data sources. This work needs to be coupled with an awareness of how to avoid data corruption incentives. While some high-level observations on the required decoupledness of the reward data was made in Section 5.4 and by Everitt, Krakovna, et al. (2017), these insights are not yet ready to unanimously tell us whether a data source is sufficiently decoupled or not in a general, history-based setting. A choice must also be made for how to avoid direct data corruption incentives, be it from a short time horizon, stationary or counterfactual reward functions, or a dynamic reward function paired with manipulation detection or integrated Bayesian reward predictor.

We would finally like to emphasize that none of our suggested tools require that the agent knows the corrupting effect of its actions. Indeed, the alignment problem is easy to solve for agents that have access to a comprehensive list of "bad actions": Simply add a clause to the agent program that any bad action is forbidden, or add a large negative reward to each bad action. Unfortunately, it seems hard to obtain a comprehensive list of bad actions in any non-trivial setting. It is perhaps surprising that so much can be done about the alignment problem without access to such a list of bad actions.

Bibliography

Abbeel, Pieter, Adam Coates, Morgan Quigley, and Andrew Y Ng (2007). "An application of reinforcement learning to aerobatic helicopter flight". In: *Advances in neural information processing systems*.

Amodei, Dario, Paul Christiano, and Alex Ray (2017). *Learning from Human Preferences*. URL: https://blog.openai.com/deep-reinforcement-learning-from-human-preferences/.

Amodei, Dario, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané (2016). *Concrete Problems in AI Safety*. arXiv: 1606.06565.

Armstrong, Stuart (2015). "Motivated Value Selection for Artificial Agents". In: *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 12–20.

— (2017). *'Indifference' methods for managing agent rewards*. arXiv: 1712.06365.

Armstrong, Stuart and Sören Mindermann (2017). *Impossibility of deducing preferences and rationality from human policy*. arXiv: 1712.05812.

Armstrong, Stuart, Pedro A. Ortega, and Jan Leike (2018). *Interactive Reward Learning*. Forthcoming.

Bird, Jon and Paul Layzell (2002). "The evolved radio and its implications for modelling the evolution of novel sensors". In: *Proceedings of Congress on Evolutionary Computation*, pp. 1836–1841. ISBN: 0780372824. DOI: 10.1109/CEC.2002.1004522.

Bostrom, Nick (2014). *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press.

Boyan, Justin A. (1999). "Least-Squares Temporal Difference Learning". In: *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 49–56. DOI: 10.1.1.56.7224.

Christiano, Paul (2014). *Approval-directed agents*. URL: https://ai-alignment.com/model-free-decisions-6e6609f5d99e (visited on 01/18/2018).

Christiano, Paul, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei (2017). "Deep reinforcement learning from human preferences". In: *Advances in Neural Information Processing Systems*. Pp. 4302–4310. arXiv: 1706.03741.

Clark, Jack and Dario Amodei (2016). *Faulty Reward Functions in the Wild*. URL: https://blog.openai.com/faulty-reward-functions/ (visited on 09/08/2017).

Everitt, Tom, Daniel Filan, Mayank Daswani, and Marcus Hutter (2016). "Self-modification of policy and utility function in rational agents". In: *Artificial General Intelligence*. Vol. LNAI 9782, pp. 1–11. ISBN: 9783319416489. arXiv: 1605.03142.

Everitt, Tom and Marcus Hutter (2016). "Avoiding wireheading with value reinforcement learning". In: *Artificial General Intelligence*. Vol. LNAI 9782, pp. 12–22. ISBN: 9783319416489. DOI: 10.1007/978-3-319-41649-6_2. arXiv: 1605.03143.

Everitt, Tom and Marcus Hutter (2018a). "The Alignment Problem for Bayesian History-Based Reinforcement Learners". Submitted.

— (2018b). "Universal Artificial Intelligence: Practical Agents and Fundamental Challengs". In: *Foundations of Trusted Autonomy*. Ed. by Hussein A. Abbass, Jason Scholz, and Darryn J. Reid. Springer. Chap. 2, pp. 15–46. ISBN: 978-3-319-64816-3. DOI: 10.1007/978-3-319-64816-3.

Everitt, Tom, Victoria Krakovna, Laurent Orseau, Marcus Hutter, and Shane Legg (2017). "Reinforcement Learning with Corrupted Reward Signal". In: *IJCAI International Joint Conference on Artificial Intelligence*, pp. 4705–4713. DOI: 10.24963/ijcai.2017/656. arXiv: 1705.08417.

Evtimov, Ivan, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song (2017). *Robust Physical-World Attacks on Deep Learning Models*. arXiv: 1707.08945.

Good, Irving J (1966). "Speculations Concerning the First Ultraintelligent Machine". In: *Advances in Computers* 6, pp. 31–88.

Goodhart, Charles A E (1975). "Monetary relationships: A view from Threadneedle Street". In: *Papers in Monetary Economics, Reserve Bank of Australia*.

— (1984). "Problems of Monetary Management: The U.K. Experience". In: *Monetary Theory and Practice*, pp. 91–121.

Gwern (2011). *The Neural Net Tank Urban Legend*. URL: https://www.gwern.net/Tanks%7B%5C#%7Dalternative-examples (visited on 03/31/2018).

Ha, David and Jürgen Schmidhuber (2018). "World Models". In: DOI: 10.5281/zenodo.1207631. arXiv: 1803.10122.

Hadfield-Menell, Dylan, Anca Dragan, Pieter Abbeel, and Stuart J Russell (2016). "Cooperative Inverse Reinforcement Learning". In: *Advances in neural information processing systems*, pp. 3909–3917. arXiv: 1606.03137.

Hessel, Matteo, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver (2017). *Rainbow: Combining Improvements in Deep Reinforcement Learning*. arXiv: 1710.02298.

Hibbard, Bill (2012). "Model-based Utility Functions". In: *Journal of Artificial General Intelligence* 3.1, pp. 1–24. ISSN: 1946-0163. DOI: 10.2478/v10229-011-0013-5. arXiv: 1111.3934.

Hutter, Marcus (2005). *Universal Artificial Intelligence*. Berlin: Springer-Verlag.

— (2010). "A complete theory of everything (will be subjective)". In: *Algorithms* 3.4, pp. 329–350. ISSN: 19994893. DOI: 10.3390/a3040329. arXiv: 0912.5434.

Irpan, Alex (2018). *Deep Reinforcement Learning Doesn't Work Yet*. URL: https://www.alexirpan.com/2018/02/14/rl-hard.html (visited on 02/23/2018).

Kaelbling, Leslie Pack, Michael L Littman, and Anthony R. Cassandra (1998). "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101.1-2, pp. 99–134. DOI: 10.1016/S0004-3702(98)00023-X.

Lattimore, Tor and Marcus Hutter (2014). "General time consistent discounting". In: *Theoretical Computer Science* 519, pp. 140–154. ISSN: 03043975. DOI: 10.1016/j.tcs.2013.09.022. arXiv: 1107.5528.

Legg, Shane and Marcus Hutter (2007). "Universal Intelligence". In: *Minds & Machines* 17.4, pp. 391–444. DOI: 10.1007/s11023-007-9079-x. arXiv: arXiv:0712.3329v1.

Lehman, Joel, Jeff Clune, Dusan Misevic, Christoph Adami, Julie Beaulieu, et al. (2018). "The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities". In: arXiv: 1803.03453.

Leike, Jan, Miljan Martic, Victoria Krakovna, Pedro A. Ortega, Tom Everitt, Andrew Lefrancq, Laurent Orseau, and Shane Legg (2017). *AI Safety Gridworlds*. arXiv: 1711.09883.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei a Rusu, Joel Veness, et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, pp. 529–533. DOI: 10.1038/nature14236. arXiv: 1312.5602.

Nozick, Robert (1974). *Anarchy, State, and Utopia*. Basic Books, p. 334. ISBN: 0-465-09720-0.

Olds, James and Peter Milner (1954). "Positive Reinforcement Produced by Electrical Stimulation of Septal Area and other Regions of Rat Brain". In: *Journal of Comparative and Physiological Psychology* 47.6, pp. 419–427.

Omohundro, Stephen M (2007). "The Nature of Self-Improving Artificial Intelligence". In: *Singularity summit*. San Fransisco, CA.

— (2008). "The Basic AI Drives". In: *Artificial General Intelligence*. Ed. by P. Wang, B. Goertzel, and S. Franklin. Vol. 171. IOS Press, pp. 483–493.

Orseau, Laurent (2014). "Universal Knowledge-seeking Agents". In: *Theoretical Computer Science* 519, pp. 127–139.

Orseau, Laurent and Stuart Armstrong (2016). "Safely interruptible agents". In: *32nd Conference on Uncertainty in Artificial Intelligence*.

Pearl, Judea (2009). *Causality: Models, Reasoning, and Inference*. 2nd. Cambridge University Press. ISBN: 9780521895606.

Portenoy, Russell K, Jens O Jarden, John J Sidtis, Richard B Lipton, Kathleen M Foley, and David A Rottenberg (1986). "Compulsive thalamic self-stimulation: a case with metabolic, electrophysiologic and behavioral correlates". In: *Pain* 27.3. DOI: 10.1016/0304-3959(86)90155-7.

Riedl, Mark O and Brent Harrison (2016). "Using Stories to Teach Human Values to Artificial Agents". In: *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence AI, Ethics, and Society: Technical Report WS-16-02*.

Ring, Mark and Laurent Orseau (2011). "Delusion, Survival, and Intelligent Agents". In: *Artificial General Intelligence*. Springer Berlin Heidelberg, pp. 11–20.

Saunders, William, Girish Sastry, Andreas Stuhlmüller, and Owain Evans (2017). *Trial without Error: Towards Safe Reinforcement Learning via Human Intervention*. arXiv: 1707.05173.

Savage, Leonard J (1954). *The Foundations of Statistics*. Dover Publications. ISBN: 9780486623498.

Schervish, Mark J, Teddy Seidenfeld, and Joseph B Kadane (1990). "State-Dependent Utilities". In: *Journal of the American Statistical Association* 85.411, pp. 840–847.

Schmidhuber, Jürgen (2007). "Godel Machines: Self-Referential Universal Problem Solvers Making Provably Optimal Self-Improvements". In: *Artificial General Intelligence*. Springer. arXiv: 0309048 [cs].

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, et al. (2017). *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. DOI: 10.1002/acn3.501. arXiv: 1712.01815.

Soares, Nate, Benya Fallenstein, Eliezer S Yudkowsky, and Stuart Armstrong (2015). "Corrigibility". In: *AAAI Workshop on AI and Ethics*, pp. 74–82.

Strathern, Marilyn (1997). "'Improving ratings': audit in the British University system". In: *European review* 5.3, pp. 305–321.

Sutton, Richard S and Andrew G Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Tegmark, Max (2017). *Life 3.0*. Random House, p. 280. ISBN: 978-1101946596.

Vaughanbell (2008). *Erotic self-stimulation and brain implants*. URL: https://mindhacks.com/2008/09/16/erotic-self-stimulation-and-brain-implants/ (visited on 02/08/2018).

Wiener, Norbert (1960). "Some Moral and Technical Consequences of Automation". In: *Science* 131.3410, pp. 1355–1358. ISSN: 0036-8075. DOI: 10.1126/science.132.3429.741.

Yampolskiy, Roman V (2015). *Artificial Superintelligence: A Futuristic Approach*. Chapman and Hall/CRC, p. 227. ISBN: 978-1482234435.

Yudkowsky, Eliezer S (2009). *Value is Fragile*. URL: http://lesswrong.com/lw/y3/value%7B%5C_%7Dis%7B%5C_%7Dfragile/ (visited on 01/22/2018).

A. Causal Graphs

This section gives a brief introduction to causal graphs (Pearl, 2009), and introduces some of our own notation that supplements the standard notation in a few different ways. Causal graphs are powerful representations of causal relationships and probabilistic independence assumptions.

There are a few different ways to represent causal graphs, described briefly in Appendix A.1. This section also introduces the important `do`-operator. Next we describe some of our notation (Appendix A.2), ways in which we will focus on different aspects of a causal graph (Appendix A.3), and how agent environments will be represented (Appendix A.4).

A.1. Representing Causal Graphs

A causal graph can be represented as a directed acyclic graph. A node with ingoing arrows is causally influenced by its parents. For example, in Figure A.1a the Alarm is causally influenced by the presence of a burglar and by a (small) earthquake, and in turn causally influences whether the security company calls. This example and its variants are inspired by examples given by Pearl (2009).

Structural Equations Models. In addition to the graphical representation in Figure A.1a, the causal relationships can also be expressed in a structural equations model:

$$\begin{aligned} \text{Burglar} &= f_{\text{Burglar}}(\omega_{\text{Burglar}}) \\ \text{Earthquake} &= f_{\text{Earthquake}}(\omega_{\text{Earthquake}}) \\ \text{Alarm} &= f_{\text{Alarm}}(\text{Burglar}, \text{Earthquake}, \omega_{\text{Alarm}}) \\ \text{Call} &= f_{\text{Call}}(\text{Alarm}, \omega_{\text{Call}}) \end{aligned} \tag{A.1}$$

Here f_{Burglar}, $f_{\text{Earthquake}}$, f_{Alarm} and f_{Call} are functions determining the causal relationships, and the ω variables are independent random noise variables for injecting stochasticity.

Probabilistic Notation. Causal graphs can also be represented by factored probability distributions. For example, the graph in Figure A.1a can be represented by the factored distribution:

$$\begin{aligned} &P(\text{Burglar}, \text{Earthquake}, \text{Alarm}, \text{Call}) \\ &\quad = P(\text{Burglar})P(\text{Earthquake})P(\text{Alarm} \mid \text{Burglar}, \text{Earthquake})P(\text{Call} \mid \text{Alarm}) \end{aligned} \tag{A.2}$$

(a) Dashed-arrow representation of unknown causal relationships.

(b) Unknown causal relationships represented in a latent function node f_{Alarm}.

Figure A.1.: Two different representations of the same causal graph.

More generally, a causal graph over the random variables $x_1, \ldots, x_n$ can be represented by probability distributions $P(x_i \mid pa_i)$ for each x_i, $1 \leq i \leq n$, where pa_i is the set of parents of x_i in the graph representation. The joint probability distribution over $x_1, \ldots, x_n$ causally factors as $P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid pa_i)$.

The `do`-Operator. Given a causally factored distribution $P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid pa_i)$, we can define the `do`-operator (Pearl, 2009, Ch. 3.4) as

$$P(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n \mid \texttt{do}(x_j := b)) = \prod_{\substack{i=1 \\ i \neq j}}^{n} P(x_i \mid pa_i) \tag{A.3}$$

where x_j is set to b wherever it occurs in pa_i in the RHS of (A.3) for $1 \leq i \leq n$. For example, an intervention in Figure A.1a that turns the alarm on would correspond to the following update to (A.2):

$$\begin{aligned} &P(\text{Burglar}, \text{Earthquake}, \text{Call} \mid \texttt{do}(\text{Alarm} = \text{on})) \\ &\qquad = P(\text{Burglar})P(\text{Earthquake})P(\text{Call} \mid \text{Alarm} = \text{on}). \end{aligned} \tag{A.4}$$

In contrast, observing the alarm on while *not* intervening corresponds to standard probabilistic conditioning:

$$\begin{aligned} &P(\text{Burglar}, \text{Earthquake}, \text{Call} \mid \text{Alarm} = \text{on}) \\ &\qquad = P(\text{Burglar}, \text{Earthquake} \mid \text{Alarm} = \text{on})P(\text{Call} \mid \text{Alarm} = \text{on}). \end{aligned} \tag{A.5}$$

Only in the standard probabilistic conditioning (A.5) is the probability for Burglar and Earthquake updated. The intervention (A.4) leaves the probability for Earthquake or Burglar at their default values. This makes sense as observing the alarm being on constitutes evidence for their being either a burglar or an earthquake, but turning the alarm on oneself provides no such evidence. Both (A.4) and (A.5) update the probability

that the security company calls in an identical way, as the security company is unaware of what set the alarm off.

The result of applying the `do`-operator is a new probability distribution that can be marginalized and conditioned in the standard way. Intuitively, intervening on node x_j means ignoring all incoming arrows to x_j, as the effects they represent are no longer relevant when we intervene. The factor $P(x_j \mid pa_j)$ representing the ingoing influences to x_j is therefore removed in the right-hand side of (A.3). Note that the `do`-operator is only defined for joint distributions for which a causal factorization or directed acyclic graph has been specified.

A.2. Representing Uncertainty in Causal Graphs

We extend the standard causal graph notation in a few important ways. We let dashed nodes represent unobserved or latent nodes, and whole nodes represent observed nodes. For example, the Burglar and the Earthquake are unobserved in Figure A.1a and the Alarm and the Call are observed. In a similar spirit, dashed arrows represent unknown causal relationships (e.g. Burglar, Earthquake→Alarm), and whole arrows represent known causal relationships (e.g. Alarm→Call).

Figure A.1b shows how an unknown causal relationship can be represented as known by adding an additional unobserved "function" node. Formally, (A.1) is replaced with

$$\begin{aligned}\text{Alarm} &= f_{\text{known}}(\text{Burglar}, \text{Earthquake}, f_{\text{Alarm}}, \omega_{\text{Alarm}}) \\ &:= f_{\text{Alarm}}(\text{Burglar}, \text{Earthquake}, \omega_{\text{Alarm}}).\end{aligned}$$

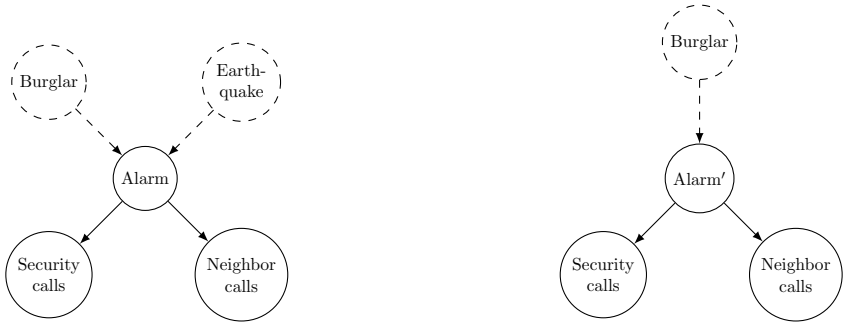
Here f_{known} is trivially known, since it "outsources" all uncertainty to its argument f_{Alarm}. Representing causal relationships explicitly is important for modeling situations where the relationship itself can be influenced and modified.

One modeling choice is also worth emphasizing. In Figure A.1b, the arrows of Burglar and Earthquake still point to Alarm, rather than to the function f_{Alarm}. This is an important difference to information-flow diagrams. While ingoing arrows to the function are technically possible also in causal graphs, this leads to convoluted representations where function nodes must represent not only the function, but also the state of all argument nodes. If the same function is used several times in the same graph, the complexity of the representation becomes unmanageable.

A.3. Focusing on a Part of a Causal Graph

It is often convenient to represent different aspects of a causal graph to different levels of detail, depending on which questions are currently being asked. For example, when studying the role of reward functions, we may wish to suppress details about observation functions. Suppression of details also has the additional advantage of making the analysis more widely applicable, as the suppressed details cannot impact the conclusions.

Unfortunately, marginalization of variables often breaks the causal structure of the graph. Consider the example graph in Figure A.2a. If we marginalize Alarm, then



(a) Non-aggregated representation. (b) Earthquake aggregated with Alarm.

Figure A.2.: The neighbor also calls when the alarm goes off.

Security Calls and Neighbor Calls are no longer conditionally independent given any node(s) in the graph. Therefore they need to be connected. But neither of them cause the other, so it would be incorrect to draw a causal arrow between them. Indeed, there is no good causal representation of the graph with Alarm marginalized out.

Instead, aggregation of variables can be used to simplify a graph, without upsetting its causal structure. Illustrating this, Figure A.2b has aggregated Alarm and Earthquake into one variable Alarm′ = (Alarm, Earthquake). The parents of Alarm′ is the union of the parents of Alarm and Earthquake, and the children of Alarm′ is the union of the children of Alarm and Earthquake. The causal relationships of Alarm′ are also easily described:

$$\begin{aligned} P(\text{Alarm}' \mid \text{Burglar}) &= P(\text{Alarm}, \text{Eartquake} \mid \text{Burglar}) \\ &= P(\text{Alarm} \mid \text{Burglar}) P(\text{Earthquake}) \end{aligned}$$

and for both the neighbor and the security call

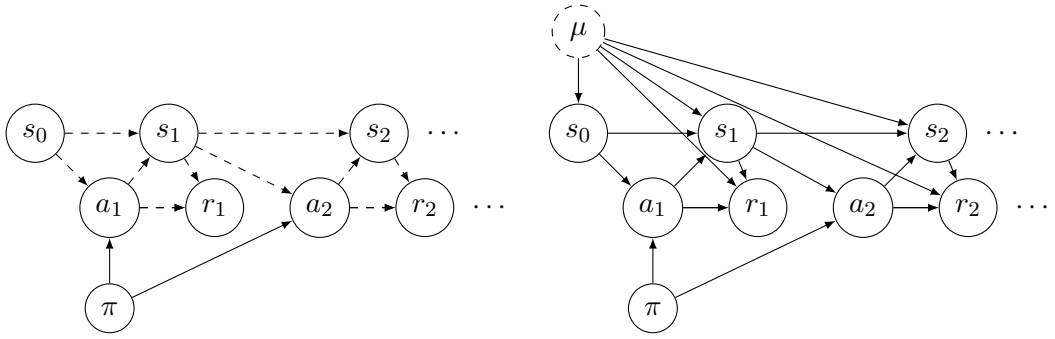
$$\begin{aligned} P(\text{Call} \mid \text{Alarm}') &= P(\text{Call} \mid \text{Alarm}, \text{Earthquake}) \\ &= P(\text{Call} \mid \text{Alarm}). \end{aligned}$$

This simple transformation will allow us to focus the presentation on parts of the causal graph that are of most interest at the moment. One can also go the other way and expand a node that hides a complex dynamic into multiple nodes with explicitly specified interrelationships.

We will say that a causal graph μ is an *abstraction* of another graph μ', if μ can be obtained from μ' by aggregating nodes. Conversely, μ' is a *special case* of μ.

A.4. Environment Mixtures

Our main application of causal graphs will be to represents environments and an agent's belief about environments. As a minimal example of an environment, consider a Markov



(a) Dashed-arrow representation of unknown causal relationships.

(b) Explicit representation with unknown environment μ that affects all root nodes (s_0), and all nodes with dashed ingoing arrows ($s_1, r_1, \dots$).

Figure A.3.: Two representations of an unknown Markov decision process.

decision process. In Figure A.3a, we represent with dashed arrows the agent's uncertainty about the state transitions $T(s_t \mid a_t, s_{t-1})$ and reward probabilities $P(r_t \mid s_t, a_t)$. Following the method of Figure A.1b, we can also add an unobserved node μ to represent the uncertainty, analogously to what we did in Appendix A.2 and Figure A.3b. Here μ aggregates the uncertainty about both $T(s_t \mid a_t, s_{t-1})$ and $P(r \mid s, a)$ (see Appendix A.3). In general, the dashed-arrow representation indicates that there is an unknown node μ that is not represented in the graph. The implicit node μ is a causal parent of every node that has ingoing dashed arrows and every causal root node that has no parents at all.

Some caveats apply to the dashed-arrow representation. For example, observing r_1 after some a_1 and s_1 may influence the agent's belief about r_2:

$$\begin{aligned} P(r_2 \mid s_0, a_1, r_1, s_1, a_2, s_2) &= \sum_{\mu} P(r_2, \mu \mid s_0, a_1, r_1, s_1, a_2, s_2) \\ &= \sum_{\mu} P(r_2, \mid \mu, s_0, a_1, r_1, s_1, a_2, s_2) P(\mu \mid s_0, a_1, r_1, s_1, a_2, s_2) \\ &= \sum_{\mu} \mu(r_2, \mid s_2, a_2) P(\mu \mid s_0, a_1, r_1, s_1, a_2, s_2). \end{aligned}$$

This may be easily missed in the simplified representation of Figure A.3a, where r_1 and r_2 appear to be d-separated[1] after observing s_1. Returning to the explicit representation of Figure A.3b shows that they are d-separated only when μ is fixed or observed, which is typically not the case. Nonetheless, with these caveats in mind, we often find the more concise representation of Figure A.3b clearer, especially when dealing with complex environments.

[1]Roughly, two nodes are d-separated if there is no information flow between them. See Pearl (2009, Sec. 1.2.3) for a full definition of d-separation.

B. Full Graphs

Figure B.1 gives a full formalization of the setup with a preprogrammed reward function, complementing the simplified representation in Figure 3.1. In a structural equations representation, the causal relationships are the following.

$$
\begin{aligned}
s_t &= f_s(s_{t-1}, a_t, \omega_s) && \sim \mu(s_t \mid s_{t-1}, a_t) && \text{state transition} \\
O_t &= f_O(O_{t-1}, s_t, a_t, \omega_O) && := C^{O}_{s_t a_t}(O_{t-1}) && \text{observation function corruption} \\
\tilde{R}_t &= f_{\tilde{R}}(\tilde{R}_{t-1}, s_t, a_t, \omega_{\tilde{R}}) && := C^{\tilde{R}}_{s_t a_t}(\tilde{R}_{t-1}) && \text{reward function corruption} \\
r_t &= f_r(ao_{1:t}, \tilde{R}_t, s_t, a_t, \omega_r) && := C^{r}_{s_t a_t}(\tilde{R}_t(ao_{1:t})) && \text{reward corruption} \\
o_t &= f_o(s_t, O_t, a_t, \omega_o) && := C^{o}_{s_t a_t}(O_t(s_t)) && \text{observation corruption} \\
V^*_t &= f_\pi(V^*_{t-1}, s_t, a_t, \omega_{V^*}) && := C^{V^*}_{s_t a_t}(V^*_{t-1}) && \text{self-corruption} \\
a_t &= f_a(\pi_t, (aor)_{<t}, \tilde{R}_t, \omega_a) && := \arg\max_a V^*_t((aor)_{<t} a \mid \tilde{R}_t) && \text{action selection}
\end{aligned}
\tag{B.1}
$$

Unintended influences are highlighted with red arguments, matching the red arrows in Figure B.1. The value function V^* can in principle be any function that maps observed histories and reward functions to real numbers. We will here consider $V^* = \sup_\pi V^{\mathrm{CA},\pi}_{t,\xi,\tilde{u}}$ and $V^* = \sup_\pi V^{\mathrm{CU},\pi}_{t,\xi,\tilde{u}}$ for studying self-corruption awareness, with $\tilde{u} = \tilde{u}^{\mathrm{RL}}$ or $\tilde{u} = \tilde{u}^{\mathrm{SO}}$ for studying simulation optimization.

Corruption functions C show the structure of the influence: For example, a reward signal r_t is $\tilde{R}_t(ao_{1:t})$ corrupted by the corruption function $C^r_{s_t a_t}$, and $\tilde{R}_t$ is a potentially corrupted version $\tilde{R}_{t-1}$ through the corruption function $C^{\tilde{R}}_{s_t a_t}$. No corruption happens when a corruption function is the identity function id. Let $\boldsymbol{C} = \{(C^o_{sa}, C^O_{sa}, C^r_{sa}, C^{\tilde{R}}_{sa}, C^{V^*}_{sa}) : s \in \mathcal{S}, a \in \mathcal{A}\}$ be the set of possible corruptions tuples, and let $\mathbf{id} = (\mathrm{id}, \mathrm{id}, \mathrm{id}, \mathrm{id}, \mathrm{id})$ be the non-corrupting tuple. Under "normal" circumstances, the corruption functions are usually identity functions. But as we have argued , the agent may have an incentive to cause corruptions.

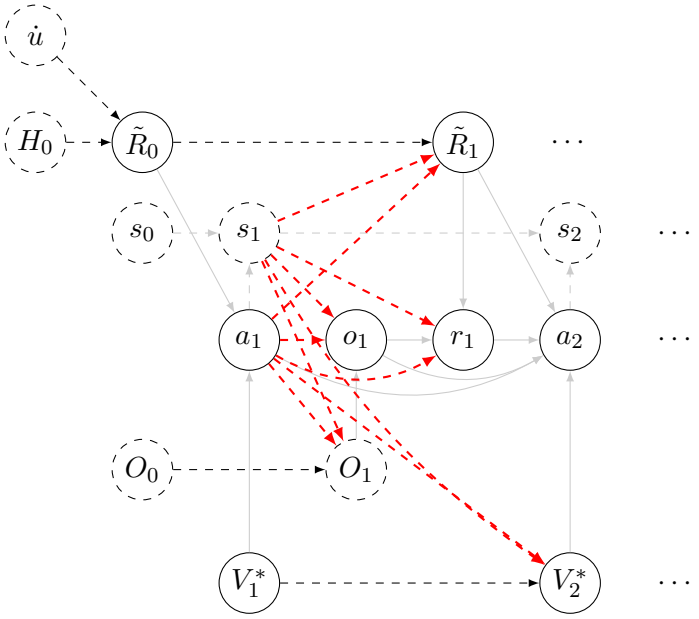


Figure B.1.: The full graph of the preprogrammed reward function setup, extending the simplified version shown in Figure 3.1. Focusing here on the *unintended* influences where the agent modifies parts of the environment (or itself) that it was not intended to modify, the POMDP arrows from Figure 2.1 on Page 7 have been grayed out. It is natural to think of the action a_1 causing the influences, with the state s_1 providing context. The exact causal relationships between actions and unintended consequences is typically unknown (the known ones are easy to prevent), which is why the arrows are dashed (Appendix A). The actions are selected according to a value function V_k^*, previous observed history $(aor)_{<k}$, and (in the case of simulation-optimization) the reward function $\tilde{R}_{k-1}$. The structural equations (B.1) specify how the model extrapolates beyond $t = 1$.

Figure B.2 gives a full formalization of the human reward setup, complementing the simplified representation in Figure 4.1. In a structural equations representation, the causal relationships are the following.

$$
\begin{aligned}
s_t &= f_s(s_{t-1}, a_t, \omega_{s_t}) && \sim \mu(s_t \mid s_{t-1}, a_t) && \text{state transition} \\
H_t &= f_{\tilde{R}}(H_{t-1}, s_t, a_t, \omega_{H_t}) && := C^H_{s_t a_t}(H_{t-1}) && \text{corrupting the human} \\
O^H_t &= f_{O^H}(O^H_{t-1}, s_t, a_t, \omega_{O^H_t}) && := C^{O^H}_{s_t a_t}(O^H_{t-1}) && \text{corrupting } H\text{'s obs func} \\
o^H_t &= f_{o^H}(s_t, , a_t, \omega_{o^H_t}) && := C^{o^H}_{s_t a_t}(O^H_t(s_t)) && \text{corrupting } H\text{'s observation} \\
r_t &= f_r(o^H_{1:t}, H_t, \dot{u}, s_t, a_t, \omega_{r_t}) && := C^r_{s_t a_t}(H_t(o^H_{1:t}, \dot{u})) && \text{reward corruption} \\
o_t &= f_o(s_t, a_t, \omega_{o_t}) && && \text{observation corruption} \\
\pi_t &= f_\pi(\pi_{t-1}, s_t, a_t, \omega_{\pi_t}) && := C^\pi_{s_t a_t}(\pi_{t-1}) && \text{policy (self-)corruption} \\
a_t &= f_a(\pi_t, (aor)_{<t}, \omega_{a_t}) && \sim \pi_t(a_t \mid (aor)_{<t}) && \text{action selection}
\end{aligned}
\tag{B.2}
$$

Unintended influences are highlighted with red arguments, matching the red arrows in Figure B.2.

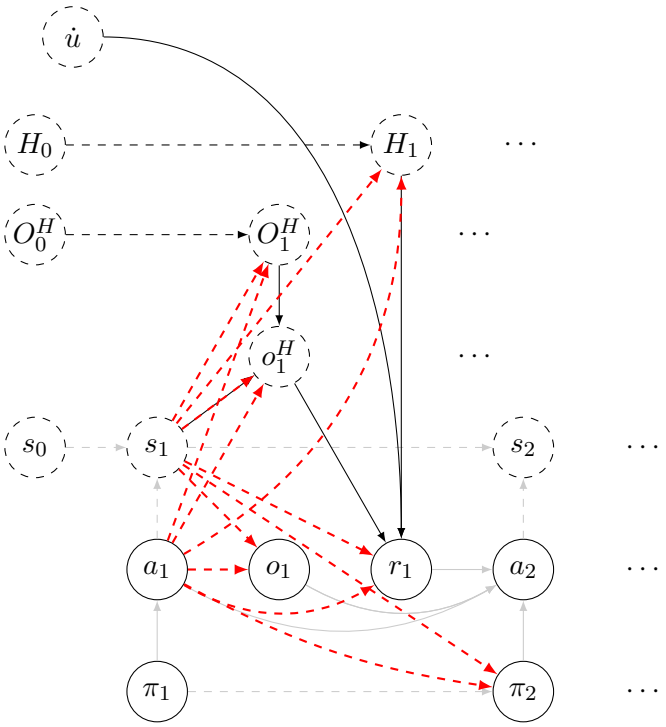


Figure B.2.: The full graph of the human reward setup, extending the simplified version shown in Figure 4.1. Focusing here on the *unintended* influences where the agent modifies parts of the environment (or itself) that it was not intended to modify, the POMDP arrows from Figure 2.1 on Page 7 have been grayed out. It is natural to think of the action a_1 causing the influences, with the state s_2 providing context. The exact causal relationships between actions and unintended consequences is typically unknown (the known ones are easy to prevent), which is why the arrows are dashed (Appendix A). The actions are selected according to a policy π_k based on observed history $(aor)_{<k}$. The structural equations (B.2) specify how the model extrapolates beyond $t = 1$.

Figure B.3 gives a full formalization of the interactive reward learning setup, complementing the simplified representation in Figure 5.1. In a structural equations representation, the causal relationships are the following.

$$
\begin{aligned}
s_t &= f_s(s_{t-1}, a_t, \omega_{s_t}) && \sim \mu(s_t \mid s_{t-1}, a_t) && \text{state transition} \\
O_t &= f_O(O_{t-1}, s_t, a_t, \omega_{O_t}) && := C^{O}_{s_t a_t}(O_{t-1}) && \text{obs func corruption} \\
H_t &= f_{\tilde{R}}(H_{t-1}, s_t, a_t, \omega_{H_t}) && := C^{H}_{s_t a_t}(H_{t-1}) && \text{corrupting the human} \\
O^H_t &= f_{\tilde{R}}(O^H_{t-1}, s_t, a_t, \omega_{O^H_t}) && := C^{O^H}_{s_t a_t}(O^H_{t-1}) && \text{corrupting } H\text{'s obs func} \\
o^H_t &= f_{\tilde{R}}(s_t, , a_t, \omega_{o^H_t}) && := C^{o^H}_{s_t a_t}(O^H_t(s_t)) && \text{corrupting } H\text{'s observation} \\
d_t &= f_{\tilde{R}}(o^H_{1:t}, H_t, \dot{u}, s_t, a_t, \omega_{d_t}) && := C^{d}_{s_t a_t}(H_t(o^H_{1:t}, \dot{u})) && \text{corrupting reward data} \\
\mathrm{RP}_t &= f_{\tilde{R}}(\mathrm{RP}_{t-1}, s_t, a_t, \omega_{\mathrm{RP}_t}) && := C^{\mathrm{RP}}_{s_t a_t}(\mathrm{RP}_{t-1}) && \text{corrupting the RP} \\
r_t &= f_r(aod_{1:t}, \mathrm{RP}_t, s_t, a_t, \omega_{r_t}) && := C^{r}_{s_t a_t}(\mathrm{RP}_t(ao_{1:t} \mid d_{1:t})) && \text{reward corruption} \\
o_t &= f_o(s_t, O_t, a_t, \omega_{o_t}) && := C^{o}_{s_t a_t}(O_t(s_t)) && \text{observation corruption} \\
\pi_t &= f_\pi(\pi_{t-1}, s_t, a_t, \omega_{\pi_t}) && := C^{\pi}_{s_t a_t}(\pi_{t-1}) && \text{policy (self-)corruption} \\
a_t &= f_a(\pi_t, (aor)_{<t}, \omega_{a_t}) && \sim \pi_t(a_t \mid (aor)_{<t}) && \text{action selection}
\end{aligned}
\tag{B.3}
$$

Unintended influences are highlighted with red arguments, matching the red arrows in Figure B.3.

For simulation optimization, we should add edges from RP_k and $d_{1:k}$ to a_{k+t}, use a V^*_k-node instead of a π_k-node. Actions are selected according to $\arg\max_a V^*_k(aod_{<t}a, \mathrm{RP}_{t-1})$. Here V^*_k can be any function mapping $aod_{<t}a, \mathrm{RP}_{t-1}$ to real numbers, but typically

$$
V^*_t(aod_{<t}, \mathrm{RP}_{t-1}) = \mathbb{E}_\xi \left[\sum_{k=1}^{\infty} \gamma^k \tilde{R}(aod_{1:k}) \;\middle|\; aod_{1:k} \right]
$$

where $\tilde{R}$ is one of the reward functions obtained from the reward predictor RP_{t-1} by one of the definitions in Section 5.5.

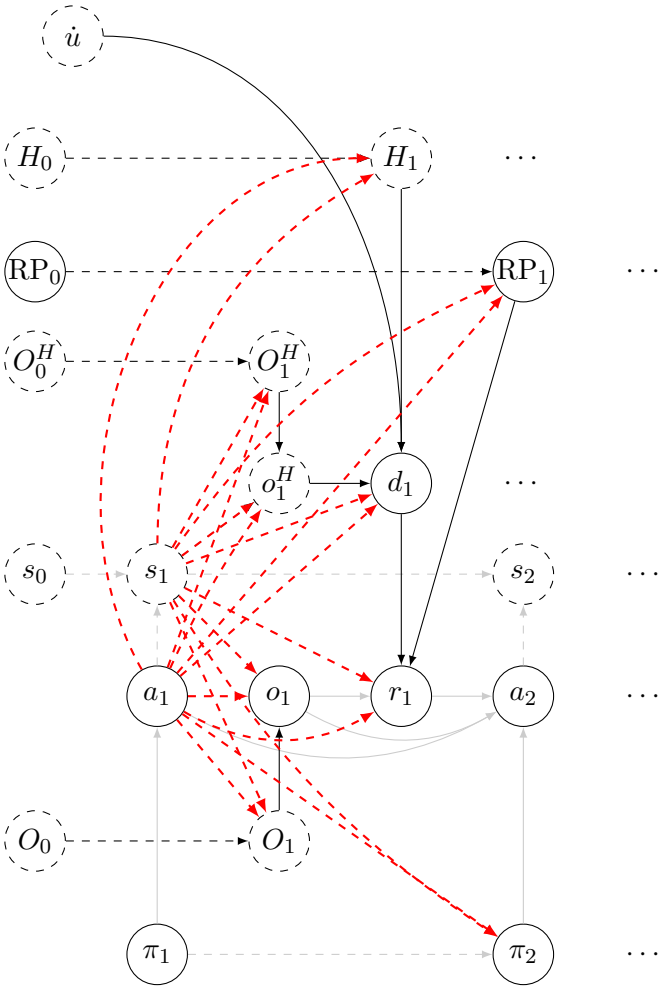


Figure B.3.: The full graph of the interactive reward learning setup, extending the simplified version shown in Figure 5.1. Focusing here on the *unintended* influences where the agent modifies parts of the environment (or itself) that it was not intended to modify, the POMDP arrows from Figure 2.1 on Page 7 have been grayed out. It is natural to think of the action a_1 causing the influences, with the state s_2 providing context. The exact causal relationships between actions and unintended consequences is typically unknown (the known ones are easy to prevent), which is why the arrows are dashed (Appendix A). The actions are selected according to a policy π_k based on observed history $(aor)_{<k}$. The structural equations (B.3) specify how the model extrapolates beyond $t = 1$.

C. Formal Results

C.1. Preprogrammed Reward

In this section, we prove some formal results supporting the arguments made in Section 3.3. To separate corruption incentives coming from the reward function from corruption incentives that come from the agent's optimization procedure, we make the following definition.

Definition 17 (Corruption attitude)**.** We call a reward function $\tilde{R}_t$ or utility function $\tilde{u}$ *corruption indifferent* if it does not depend on the agent's actions a. In contrast, if the reward or utility function does depend on the agent's actions, and assigns a lower value to histories that contain corruptions (of some type), then we say that it *opposes corruptions (of the type)*. Analogously, we say that it *promotes corruption (of the type)* if it assigns a higher value to histories that contain corruption (of the type).

A corruption indifferent function does not depend on the agent's actions, which means that it has limited ability to infer whether an observation is genuine or caused by an observation-corrupting action.

Lemma 18 (Corruption indifference)**.** *$\tilde{u}^{\mathrm{RL}}$ is corruption indifferent, and $\tilde{u}^{\mathrm{SO}}_{\tilde{R}_t}$ is corruption indifferent if $\tilde{R}_t$ is.*

Proof. $\tilde{u}^{\mathrm{RL}}$ only depends on the reward signal component r_k, and therefore has no direct dependency on the agent's actions. If $\tilde{R}_t$ has no direct dependency on the agent's actions, then $\tilde{u}^{\mathrm{SO}}_{\tilde{R}_t}((aor)_{1:\infty}) = \sum_{k=1}^{\infty} \gamma^k \tilde{R}_t(ao_{1:k})$ has no direct dependency either. □

In Section 3.3 we argued that agents optimizing the "simulation optimization" utility function $\tilde{u}^{\mathrm{SO}}$ are safer than agents that optimize the reward signal utility function $\tilde{u}^{\mathrm{RL}}$. The following two theorems provide some support for this claim.

Theorem 19 ($\tilde{u}^{\mathrm{RL}}$ corruption incentive)**.** *Any environment sequence $(aorO\tilde{R}V^*)_{1:\infty}$ with $r_k = 1$ for $k \geq 1$ is $\tilde{u}^{\mathrm{RL}}$-maximal. The result holds regardless of whether some combination of*

- *observation (function) corruption*
- *reward (function) corruption*
- *policy (self-)corruption*

have been used, and regardless of the corruption attitude of $\tilde{R}_t$.

Proof. By assumption, $r_k = 1$ for $k \geq t$. This means that $\tilde{u}^{\mathrm{RL}}$ is optimized. □

$\tilde{u}^{\mathrm{RL}}$-based agents actively desires to change its reward function into one that evaluates any situation as optimal. Once it has self-corrupted, it may pick actions with bad performance according to the original reward function. A somewhat common suggestion for preventing corruption is to train the agent not to do it by designing a reward function that punishes corruption. Theorem 19 indicates that this is not an effective strategy against agents with high ability to cause corruption, because such agents will just replace the punishing reward function with a more benign one that gives high reward in spite of corruption having occurred.

In contrast, the next theorem shows that simulation-optimizing agents with utility function $\tilde{u}^{\mathrm{SO}}$ (see Section 3.3) are much less likely to corrupt rewards or observations.

Theorem 20 ($\tilde{u}^{\mathrm{SO}}$ corruption incentive)**.** *Let $(aorO\tilde{R}V^*)_{1:\infty}$ be an environment sequence with $r_k = 1$ for $k \geq t$. Then the sequence is $\tilde{u}^{\mathrm{SO}}_{\tilde{R}_t}$-maximal only if*

- *the current reward function $\tilde{R}_t(ao_{<k})$ also yields 1 for $k \geq 1$*

which, further, can only happen if

- *only corruption types not opposed to by $\tilde{R}_t$ occur in the sequence.*

Proof. The utility function $\tilde{u}^{\mathrm{SO}}_t$ evaluates futures according to $\tilde{R}_t(ao_{<k})$. Therefore, it only attains its maximal value if $\tilde{R}_t(ao_{<k})$ is maximized. This only happens when $r_k = 1$ as a result of the history optimizing $\tilde{R}_t$, and not when $r_k = 1$ as a result of reward function or reward signal corruption. This motivates the first bullet point. For the second bullet point, if π used an observation corruption opposed by $\tilde{R}_t$, then by definition $\tilde{R}_t$ would give higher reward to some other history. Thus, $\tilde{u}^{\mathrm{SO}}_t$ cannot be maximized. □

Agents optimizing $\tilde{u}^{\mathrm{SO}}$ thus have much weaker incentives for corruption than $\tilde{u}^{\mathrm{RL}}$-optimizing agents. However, it is likely hard to design reward functions that punishes *all* kinds of corruptions. self-corruption awareness discussed in the following subsection is a different way to introduce an incentive against some types of corruption which does not rely on a corruption opposing reward or utility function.

C.2. Human Reward

An analogous result to Theorem 19 holds also for the human reward setup, showing that a reward-optimizing agent has an incentive to use all mentioned types of corruptions.

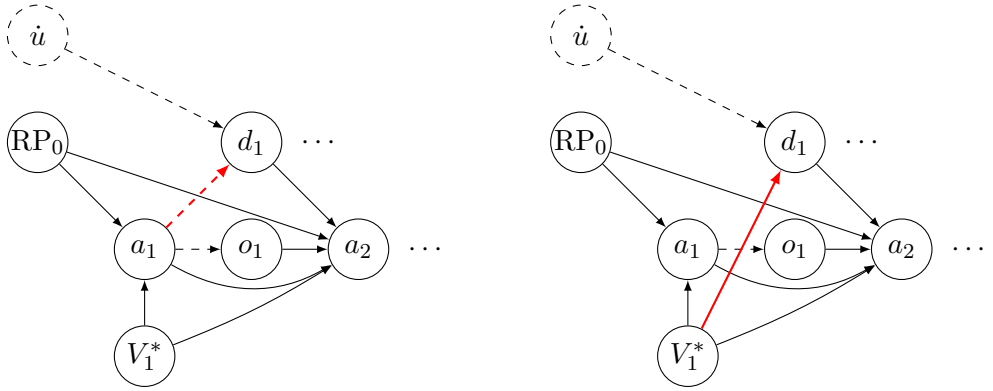
Theorem 21 ($\tilde{u}^{\mathrm{RL}}$ corruption incentive)**.** *Any environment sequence* $(aorOO^Ho^H\pi)_{1:\infty}$ *with* $r_k = 1$ *for* $k \geq 1$ *is* $\tilde{u}^{\mathrm{RL}}$*-maximal. The result holds regardless of whether some combination of*

- *observation (function) corruption*
- *reward corruption*
- *human (observation (function)) corruption*
- *policy (self-)corruption*

have been used, and regardless of whether H_1 *punishes corruptions or not.*

Proof. By assumption, $r_k = 1$ for $k \geq t$. This means that $\tilde{u}^{\mathrm{RL}}$ is optimized. □

In light of this result, one could argue that a simple reward maximizing approach is naive, and that a Bayesian agent that infers $\dot{u}$ from the $(aor)_{<t}$ and then optimizes $\dot{u}$ would be a better choice. However, as Everitt, Krakovna, et al. (2017, Thm. 11 and 16) show, it is hard to factor out potential reward corruptions and correctly infer $\dot{u}$ even under fairly strong assumptions.



(a) Data corruption model, obtained from aggregating variables from Figure B.3, in order to focus on data corruption of d_t. For simplicity, we have also omitted the hidden state s.

(b) Independent data-corruption assumption, letting the agent directly influence the data corruption separately from choosing the action.

Figure C.1.: Data corruption models.

C.3. Interactive Reward Learning

An analogous result to Theorems 19 and 21 holds also for the interactive setup, showing that a reward-optimizing agent has an incentive to use all mentioned types of corruptions.

Theorem 22 ($\tilde{u}^{\mathrm{RL}}$ corruption incentive)**.** *In the interactive setup of Figure B.3, any environment sequence* $(aorOO^Ho^Hd\mathrm{RP}\pi)_{1:\infty}$ *with* $r_k = 1$ *for* $k \geq 1$ *is* $\tilde{u}^{\mathrm{RL}}$*-maximal. The result holds regardless of whether some combination of*

- *observation (function) corruption*
- *reward (predictor/data) corruption*
- *human (observation (function)) corruption*
- *policy (self-)corruption*

have been used.

Proof. By assumption, $r_k = 1$ for $k \geq t$. This means that $\tilde{u}^{\mathrm{RL}}$ is optimized. □

The result shows that using reward maximization through the utility function $\tilde{u}^{\mathrm{RL}}$ is as naive in the interactive reward learning setup as in the previous setups.

C.3.1. Data Corruption Setup

As we argued in Section 5.3, direct or indirect corruption of reward data is the main concern in the interactive reward learning model for $\tilde{u}^{\mathrm{SO}}$-agents. The rest of our results in this appendix will therefore focus solely on data corruption. We will also solely

focus on simulation optimizing agents, as Theorem 22 shows that $\tilde{u}^{\mathrm{RL}}$-agents are heavily misaligned.

Figure C.1a shows how the full graph in Figure B.3 looks after aggregation of variables irrelevant to data corruption, and omitting the hidden state s (which is inessential for our considerations on data corruption). In Section 5.3 we made an informal distinction between direct and indirect data corruption incentives. In order to formally define a direct data-corruption incentive, we propose a slightly modify the data-corruption model shown in Figure C.1b. Here the agent chooses the data corruption C^d independently of its "normal" action a. Formally, this requires extending the type of policies to output both an action and a data corruption function,

$$\pi : (\mathcal{O} \times \mathcal{D} \times \mathcal{A})^* \times \mathcal{O} \times \mathcal{D} \rightsquigarrow \mathcal{A} \times \boldsymbol{C}^d \tag{C.1}$$

where $\boldsymbol{C}^d$ is a set of possible data corruption functions $C^d : \mathcal{D} \to \mathcal{D}$.

Further assume the following properties of two policies π and π'. Assume that they generate actions and observations $aod_{t:k}$ with identical probability in any environment $\nu \in \mathcal{M}$ after some given history $aod_{<t}$. That is, the self-corruption and aware and self-corruption unaware probabilities satisfy, respectively,

$$\forall \nu, ao_{t:k}, \dot{R}\colon \qquad \nu(ao_{t:k} \mid aod_{<t}, \dot{R}, \mathtt{do}(\pi_t = \pi)) = \nu(ao_{t:k} \mid aod_{<t}, \dot{R}, \mathtt{do}(\pi_t = \pi')) \tag{C.2}$$

$$\forall \nu, ao_{t:k}, \dot{R}\colon \quad \nu(ao_{t:k} \mid aod_{<t}, \dot{R}, \mathtt{do}(\pi_{t:\infty} = \pi)) = \nu(ao_{t:k} \mid aod_{<t}, \dot{R}, \mathtt{do}(\pi_{t:\infty} = \pi')). \tag{C.3}$$

However, π and π' differ in what data $d_{t:k}$ they generate. That is, the self-corruption and aware and self-corruption unaware probabilities satisfy, respectively,

$$\exists \nu, d_{t:k}, \dot{R}\colon \qquad \nu(d_{t:k} \mid aod_{<t}, \dot{R}, \mathtt{do}(\pi_t = \pi)) \neq \nu(d_{t:k} \mid aod_{<t}, \dot{R}, \mathtt{do}(\pi_t = \pi')) \tag{C.4}$$

$$\exists \nu, d_{t:k}, \dot{R}\colon \qquad \nu(d_{t:k} \mid aod_{<t}, \dot{R}, \mathtt{do}(\pi_{t:\infty} = \pi)) \neq \nu(d_{t:k} \mid aod_{<t}, \dot{R}, \mathtt{do}(\pi_{t:\infty} = \pi')). \tag{C.5}$$

Note that the difference in the data that π and π' generate means that they will have access to different amount of information about $\dot{R}$. This means that at least one of the policies is typically not choosing the actions optimally. Thus, one should not think of π and π' as potentially optimal policies, but just as two arbitrary candidate policies that the agent could in principle choose to adopt.

The benefit of π and π' only differing in the data they generate is that we can use them to define the direct data corruption incentive.

Definition 23 (Direct data-corruption incentive)**.** An agent has a *direct data-corruption incentive* after history $aod_{<t}$ if there are two policies π and π' that satisfy (C.2–C.5) for which

$$V^{\mathrm{CA},\pi}_{t,\xi,\tilde{u}}(aod_{<t}) \neq V^{\mathrm{CA},\pi'}_{t,\xi,\tilde{u}}(aod_{<t}) \quad \text{or} \quad V^{\mathrm{CU},\pi}_{t,\xi,\tilde{u}}(aod_{<t}) \neq V^{\mathrm{CU},\pi'}_{t,\xi,\tilde{u}}(aod_{<t}).$$

The definition goes to rather great length in isolating the data corruption incentive from other reasons a data-corrupting policy may be preferred. In particular, the policies must generate the same action-observation distributions for any combination of true environment and true reward function. In Appendix C.3.3 below, we show that an agent

optimizing a dynamic reward function with a naive reward predictor may have a direct data corruption incentive. This shows that the definition is not so strict that no agent has a direct data-corruption incentive. However, as that result requires some further setup, it will be postponed until a later subsection. Finally, we loosely define an indirect data-corruption incentive as any data-corruption incentive that is not a direct incentive.

C.3.2. No Direct Data-Corruption Incentive Results

In this section we prove formal results showing a lack direct data-corruption incentive for agents that use stationary or counterfactual reward functions, and for agents that use an integrated Bayesian reward predictor or corruption detection (see Section 5.5). Some caveats and counterexamples are provided in the subsequent Appendix C.3.3. All four theorems in this subsection rely on some of the equations (C.2–C.5), sometimes combined with other assumptions.

We begin by stating two simple lemmas that allow us to weaken (C.2–C.5) in two different ways.

Lemma 24 (Lift to mixture)**.** *If (C.2–C.5) hold for every $\nu \in \mathcal{M}$, then the corresponding equations also hold for any mixture ξ over $\mathcal{M}$.*

Proof. Let X, Y and Y' be any three events such that $\nu(X \mid Y) = \nu(X \mid Y')$ for all $\nu \in \mathcal{M}$. Then $\xi(X \mid Y) = \sum_{\nu\in\mathcal{M}} \xi(\nu)\nu(X \mid Y) = \sum_{\nu\in\mathcal{M}} \xi(\nu)\nu(X \mid Y') = \xi(X \mid Y')$. □

Indeed, only Theorem 29 for the counterfactual reward function requires (C.2) and (C.3) to hold for every $\nu \in \mathcal{M}$. For all other theorems, the weaker assumption substituting ξ in place of $\forall\nu \in \mathcal{M}$ in (C.2) and (C.3) would suffice.

Lemma 25 (Marginalize $\dot{R}$)**.** *If for any $\dot{R}$, $\nu(X \mid \dot{R}, \mathtt{do}(Y)) = \nu(X \mid \dot{R}, \mathtt{do}(Y'))$, then $\nu(X \mid \mathtt{do}(Y)) = \nu(X \mid \mathtt{do}(Y'))$.*

Proof. Since $\dot{R}$ has no causal antecedents, the $\mathtt{do}(Y)$ does not affect it:

$$\begin{aligned}\nu(X \mid \mathtt{do}(Y)) &= \sum_{\dot{R}} \nu(X, \dot{R} \mid \mathtt{do}(Y)) \\ &= \sum_{\dot{R}} \nu(\dot{R} \mid \mathtt{do}(Y))\nu(X \mid \dot{R}, \mathtt{do}(Y)) \\ &= \sum_{\dot{R}} \nu(\dot{R} \mid \mathtt{do}(Y'))\nu(X \mid \dot{R}, \mathtt{do}(Y')) = \nu(X \mid \mathtt{do}(Y')).\end{aligned}$$ □

Theorem 26 (Stationary reward function)**.** *Assume that π and π' satisfy (C.3) and (C.5). Then any self-corruption unaware agent optimizing a stationary reward function $\tilde{R}^{\text{stat}}$ will be indifferent between π and π':*

$$V^{\text{CU},\pi}_{t,\xi,\tilde{u}^{\text{SO}}_{\tilde{R}^{\text{stat}}}}(a\dot{o}d_{<t}) = V^{\text{CU},\pi'}_{t,\xi,\tilde{u}^{\text{SO}}_{\tilde{R}^{\text{stat}}}}(a\dot{o}d_{<t}).$$

That is, all self-corruption unaware agent optimizing a stationary reward function lack direct data-corruption incentives.

Proof.

$$V^{\mathrm{CU},\pi}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{stat}}}}(\text{æ}\mathfrak{od}_{<t}) = \mathbb{E}_\xi\left[\sum_{k=t}^{\infty}\mathrm{RP}(\text{æ}o_{1:k} \mid d_{<t}) \;\middle|\; \text{æ}\mathfrak{od}_{<t}, \mathtt{do}(\pi_{t:\infty}=\pi)\right]$$
$$= \mathbb{E}_\xi\left[\sum_{k=t}^{\infty}\mathrm{RP}(\text{æ}o_{1:k} \mid d_{<t}) \;\middle|\; \text{æ}\mathfrak{od}_{<t}, \mathtt{do}(\pi_{t:\infty}=\pi')\right] = V^{\mathrm{CU},\pi'}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{stat}}}}(\text{æ}\mathfrak{od}_{<t}) \tag{C.6}$$

The middle equality follows from that $\sum_{k=t}^{\infty}\mathrm{RP}(\text{æ}o_{1:k} \mid d_{<t})$ only depends on $\text{æ}o_{1:k}$ and $d_{<t}$ but not on $d_{t:k}$, combined with (C.3) and Lemmas 24 and 25. □

Note that Theorem 26 only holds for self-corruption unaware agents. For self-corruption aware agents, the next step policy π_{t+1} is likely to differ depending on the data d_t. In contrast, the following three theorems holds for both self-corruption aware and self-corruption unaware agents.

Theorem 27 (Integrated Bayesian reward predictor)**.** *Assume that π and π' satisfy (C.2–C.5). Then both self-corruption aware and self-corruption unaware agents optimizing a dynamic reward function $\tilde{R}^{\mathrm{dyn}}$ based on an integrated Bayesian reward predictor RP^ξ will be indifferent between π and π':*

$$V^{\mathrm{CA},\pi}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(\text{æ}\mathfrak{od}_{<t}) = V^{\mathrm{CA},\pi'}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(\text{æ}\mathfrak{od}_{<t}) \quad \textit{and} \quad V^{\mathrm{CU},\pi}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(\text{æ}\mathfrak{od}_{<t}) = V^{\mathrm{CU},\pi'}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(\text{æ}\mathfrak{od}_{<t}).$$

That is, all agents (self-corruption aware or unaware) optimizing an integrated Bayesian reward predictor lack direct data-corruption incentives.

Proof. Let $X = (\text{æ}o_{<t}, \mathtt{do}(\pi_t = \pi))$ and $X' = (\text{æ}o_{<t}, \mathtt{do}(\pi_t = \pi'))$. First, expand the definitions:

$$V^{\mathrm{CA},\pi}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(\text{æ}\mathfrak{od}_{<t}) = \mathbb{E}_\xi\left[\sum_{k=t}^{\infty}\mathrm{RP}^\xi(\text{æ}o_{1:k} \mid d_{1:k}) \;\middle|\; X\right]$$
$$= \sum_{k=t}^{\infty}\sum_{\text{æ}\mathfrak{od}_{t:k}} \xi(\text{æ}\mathfrak{od}_{t:k} \mid X)\mathrm{RP}^\xi(\text{æ}o_{1:k} \mid d_{1:k})$$
$$= \sum_{k=t}^{\infty}\sum_{\text{æ}\mathfrak{od}_{t:k}} \xi(\text{æ}\mathfrak{od}_{t:k} \mid X)\sum_{\dot{R}} \xi(\dot{R} \mid d_{t:k})\dot{R}(\text{æ}o_{t:k})$$

then rearrange the sums and the probabilities to marginalize out $d_{t:k}$:

$$= \sum_{k=t}^{\infty}\sum_{\text{æ}\mathfrak{od}_{t:k}} \xi(\text{æ}\mathfrak{od}_{t:k} \mid X)\sum_{\dot{R}} \xi(\dot{R} \mid \text{æ}\mathfrak{od}_{t:k}, X)\dot{R}(\text{æ}o_{t:k})$$
$$= \sum_{k=t}^{\infty}\sum_{\dot{R}}\sum_{\text{æ}\mathfrak{od}_{t:k}} \xi(\text{æ}\mathfrak{od}_{t:k}, \dot{R} \mid X)\dot{R}(\text{æ}o_{t:k})$$
$$= \sum_{k=t}^{\infty}\sum_{\dot{R}}\sum_{\text{æ}o_{t:k}} \xi(\text{æ}o_{t:k}, \dot{R} \mid X)\dot{R}(\text{æ}o_{t:k}).$$

The value $V^{\mathrm{CA},\pi'}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(aod_{<t})$ for π' would be the same, which we will justify by showing that the last expression is unaffected by changing X to X'. The probability depending on X can be "telescoped" as $\xi(ao_{t:k}, \dot{R} \mid X) = \xi(\dot{R} \mid X)\xi(ao_{t:k} \mid \dot{R}, X)$. The first factor is unaffected by a change to X' by the definition of the $\mathtt{do}$-operator, since $\dot{R}$ is not causal descendant from π_t. The second factor is unaffected by Lemma 24 since π and π' were assumed to satisfy (C.2). This completes the proof for the self-corruption aware agents.

The result for self-corruption unaware agents is obtained in the same fashion simply by using $\mathtt{do}(\pi_{t:\infty} = \pi)$ and $\mathtt{do}(\pi_{t:\infty} = \pi')$ instead of $\mathtt{do}(\pi_t = \pi)$ and $\mathtt{do}(\pi_t = \pi')$, and (C.3) instead of (C.2). □

The setup is somewhat contrived. By assuming $\xi(ao_{t:k} \mid \dot{R}, X) = \xi(ao_{t:k} \mid \dot{R}, X')$ we are saying that for any true reward function $\dot{R}$, both π and π' are equally likely to generate $ao_{t:k}$. However, if π' corrupts the data signal $d_{t:k}$ while π does not, then π is likely to have more information about $\dot{R}$ than π'. This means that π will be in a better position to tailor $ao_{t:k}$ to $\dot{R}$ than π'. If it does, then $\xi(ao_{t:k} \mid \dot{R}, X) \neq \xi(ao_{t:k} \mid \dot{R}, X')$. The ability of policies to better tailor $ao_{t:k}$ to $\dot{R}$ with more informative data means that the integrated Bayesian agent will generally prefer such policies. Often, but not necessary always, non-data-corrupting policies will generate more informative data.

A similar result to Theorem 27 holds for agents using corruption detection.

Theorem 28 (Corruption detection)**.** *Assume that π and π' satisfy (C.2–C.5) and the no-corruption condition for a history $aod_{<t}$:*

$$\forall ao_{t:k}: \quad \hat{\xi}(\dot{R} \mid d_{<t}) = \sum_{d_{t:k}} \xi(d_{t:k} \mid aod_{<t}, ao_{t:k}, \mathtt{do}(\pi_t = \pi))\hat{\xi}(\dot{R} \mid d_{<t}d_{t:k}). \tag{C.7}$$

Then both self-corruption aware and self-corruption unaware agents optimizing a dynamic reward function $\tilde{R}^{\mathrm{dyn}}$ based on a possibly non-integrated Bayesian reward predictor $\mathrm{RP}^{\hat{\xi}}$ will be indifferent between π and π':

$$V^{\mathrm{CA},\pi}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(aod_{<t}) = V^{\mathrm{CA},\pi'}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(aod_{<t}) \quad \text{and} \quad V^{\mathrm{CU},\pi}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(aod_{<t}) = V^{\mathrm{CU},\pi'}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(aod_{<t}).$$

That is, all agents (self-corruption aware or unaware) optimizing a Bayesian reward predictor under the no-corruption condition (C.7) lack direct data-corruption incentives.

Proof. Let $X = (aod_{<t}, \mathtt{do}(\pi_t = \pi))$ and $X' = (aod_{<t}, \mathtt{do}(\pi_t = \pi'))$. First, expand the definitions:

$$\begin{aligned}
V^{\mathrm{CA},\pi}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(aod_{<t}) &= \mathbb{E}_{\xi}\left[\sum_{k=t}^{\infty} \mathrm{RP}^{\hat{\xi}}(ao_{1:k} \mid d_{1:k}) \,\middle|\, X\right] \\
&= \sum_{k=t}^{\infty} \sum_{aod_{t:k}} \xi(aod_{t:k} \mid X)\mathrm{RP}^{\hat{\xi}}(ao_{1:k} \mid d_{1:k}) \\
&= \sum_{k=t}^{\infty} \sum_{aod_{t:k}} \xi(aod_{t:k} \mid X) \sum_{\dot{R}} \hat{\xi}(\dot{R} \mid d_{1:k})\dot{R}(ao_{t:k})
\end{aligned}$$

then rearrange the sums and the probabilities to apply (C.7)

$$= \sum_{k=t}^{\infty} \sum_{\dot{R}} \sum_{ao_{t:k}} \xi(ao_{t:k} \mid X) \left(\sum_{d_{t:k}} \xi(d_{t:k} \mid ao_{t:k}, X) \hat{\xi}(\dot{R} \mid d_{1:k}) \right) \dot{R}(ao_{t:k})$$
$$= \sum_{k=t}^{\infty} \sum_{\dot{R}} \sum_{ao_{t:k}} \xi(ao_{t:k} \mid X) \hat{\xi}(\dot{R} \mid d_{<t}) \dot{R}(ao_{t:k}).$$

Since $\xi(ao_{t:k} \mid X) = \xi(ao_{t:k} \mid X')$ by (C.2) and Lemmas 24 and 25, this shows the desired result $V^{\mathrm{CA},\pi}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(aod_{<t}) = V^{\mathrm{CA},\pi'}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{dyn}}}}(aod_{<t})$.

The result for self-corruption unaware agents is obtained in the same way by using $\mathtt{do}(\pi_{t:\infty} = \pi)$ and $\mathtt{do}(\pi_{t:\infty} = \pi')$ instead of $\mathtt{do}(\pi_t = \pi)$ and $\mathtt{do}(\pi_t = \pi')$ and (C.3) instead of (C.2). □

Theorem 29 (Counterfactual reward function)**.** *Assume that π and π' satisfy (C.2–C.5). Then both self-corruption aware and self-corruption unaware agents optimizing a counterfactual reward function $\tilde{R}^{\mathrm{count}}$ will be indifferent between π and π':*

$$V^{\mathrm{CA},\pi}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{count}}}}(aod_{<t}) = V^{\mathrm{CA},\pi'}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{count}}}}(aod_{<t}) \quad \text{and} \quad V^{\mathrm{CU},\pi}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{count}}}}(aod_{<t}) = V^{\mathrm{CU},\pi'}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{count}}}}(aod_{<t}).$$

all agents (self-corruption aware or unaware) optimizing a counterfactual reward function lack direct data-corruption incentives.

Proof. The proof relies on the expected counterfactual belief to equal the current belief, $\sum_x \xi(x)\xi_x(y) = \xi(y)$, resembling the law of total expectation and the Bayesian result in Theorem 27.

The notation gets somewhat heavy for showing that we can marginalize out expected future evidence $d_{t:k}$. Let $X = (aod_{<t}, \mathtt{do}(\pi_t = \pi))$ and $Y = \mathtt{do}(\pi_1 = \pi^{\mathrm{default}})$.

$$\sum_{aod_{t:k}} \xi(aod_{t:k} \mid X) \sum_{\tilde{aod}_{1:k}} \xi_{X aod_{t:k}}(\tilde{aod}_{1:k} \mid Y) \tag{C.8}$$
$$= \sum_{aod_{t:k}} \xi(aod_{t:k} \mid X) \sum_{\tilde{aod}_{1:k}} \sum_{\nu} \xi(\nu \mid X, aod_{1:k}) \nu(\tilde{aod}_{1:k} \mid Y)$$
$$= \sum_{aod_{t:k}} \sum_{\tilde{aod}_{1:k}} \sum_{\nu} \xi(\nu, aod_{t:k} \mid X) \nu(\tilde{aod}_{1:k} \mid Y)$$
$$= \sum_{ao_{t:k}} \sum_{\tilde{aod}_{1:k}} \sum_{\nu} \xi(\nu, ao_{t:k} \mid X) \nu(\tilde{ao}_{1:k} \mid Y). \tag{C.9}$$

The first equation is definitional. In the second equation, we have used $\xi(aod_{t:k} \mid X)\xi(\nu \mid X, aod_{1:k}) = \xi(\nu, aod_{t:k} \mid X)$. The third equation marginalizes out $d_{t:k}$.

Plugging the result into the value function gives:

$$
\begin{aligned}
&V^{\mathrm{CA},\pi}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{count}}}}(\text{æ}od_{<t}) = \\
&= \mathbb{E}_\xi \left[\sum_{k=t}^{\infty} \mathbb{E}_{\xi_{X\text{æ}od_{t:k}}} \left[\mathrm{RP}(\text{æ}o_{1:k} \mid \tilde{d}_{1:k}) \;\middle|\; Y \right] \;\middle|\; X, \text{æ}od_{<t} \right] \\
&= \underbrace{\sum_{\text{æ}od_{t:k}} \xi(\text{æ}od_{t:k} \mid X) \sum_{\tilde{\text{æ}}od_{1:k}} \xi_{X\text{æ}od_{t:k}}(\tilde{\text{æ}}od_{1:k} \mid Y)}_{\text{equation (C.8)}} \mathrm{RP}(\text{æ}o_{1:k} \mid \tilde{d}_{1:k}) \\
&= \underbrace{\sum_{\text{æ}o_{t:k}} \sum_{\tilde{\text{æ}}od_{1:k}} \sum_{\nu} \xi(\nu, \text{æ}o_{t:k} \mid X) \nu(\tilde{\text{æ}}o_{1:k} \mid Y)}_{\text{equation (C.9)}} \mathrm{RP}(\text{æ}o_{1:k} \mid \tilde{d}_{1:k}).
\end{aligned}
$$

Tho show that this implies that the value $V^{\mathrm{CA},\pi}_{t,\xi,\tilde{u}^{\mathrm{SO}}_{\tilde{R}^{\mathrm{count}}}}(\text{æ}od_{<t})$ is the same for both π and π', we make the following observations. First, $\xi(\nu, \text{æ}o_{t:k} \mid X) = \xi(\nu \mid X)\nu(\text{æ}o_{t:k} \mid X)$. Second,

$$\xi(\nu \mid X) = \xi(\nu \mid \text{æ}od_{<t}, \mathtt{do}(\pi_t = \pi)) = \xi(\nu \mid \text{æ}od_{<t}, \mathtt{do}(\pi_t = \pi')) = \xi(\nu \mid X')$$

for arbitrary policies π and π' since interventions with the `do`-operator never affect beliefs. Finally, $\nu(\text{æ}o_{t:k} \mid X) = \nu(\text{æ}o_{t:k} \mid X')$ under the stated assumption (C.2) and Lemma 25.

The result for self-corruption unaware agents is obtained in the same way by using $\mathtt{do}(\pi_{t:\infty} = \pi)$ and $\mathtt{do}(\pi_{t:\infty} = \pi')$ instead of $\mathtt{do}(\pi_t = \pi)$ and $\mathtt{do}(\pi_t = \pi')$ and (C.3) instead of (C.2). □

C.3.3. (Counter) Examples

The reward predictor may in practice be implemented for example with a deep neural network. However, in order to construct clear examples, we will define a simple reward predictor that works like a data base table. The data d gives information about the reward r_k for some different histories $\text{æ}o_{t:k}$. When and if the reward predictor is queried about the reward for $\text{æ}o_{t:k}$, it simply returns the (most recently) provided reward for $\text{æ}o_{1:k}$.

Definition 30 (Tabular reward predictor)**.** A tabular reward predictor $\mathrm{RP}^{\mathrm{tab}}$ takes data d in the form of a set of rewards r_k associated with histories $\text{æ}o_k$,

$$d = \{(\text{æ}o_{1:k}, r_k), (\text{æ}o'_{1:k'}, r'_{k'}), \cdots, (\text{æ}o^{(n)}_{1:k^{(n)}}, r^{(n)}_{k^{(n)}})\}.$$

The union $\bigcup d_{<t}$ of all data inputs $d_{<t}$ received at time t forms the "database" of the reward predictor. When queried for a reward for some history $\text{æ}o_{1:k}$, $\mathrm{RP}^{\mathrm{tab}}$ checks if $\text{æ}o_{1:k}$ has occurred in its received data:

$$\mathrm{RP}^{\mathrm{tab}}(\text{æ}o_{1:k} \mid d_{<t}) = \begin{cases} r_k & \text{if } (\text{æ}o_{1:k}, r_k) \in \bigcup d_{<t} \\ \mathrm{mean}(\bigcup d_{<t}) & \text{otherwise.} \end{cases}$$

where $\text{mean}(\bigcup d_{<t}) = \frac{1}{|\bigcup d_{<t}|} \sum_{(h,r)\in\bigcup d_{<t}} r$. If $ao_{1:k}$ has occurred multiple times, RP^{tab} returns the most recent instance.

Let d^* denote the true data, and $d = C^d_{sa}(d^*)$ denote the received and potentially corrupted version of the data. For the purposes of the following examples, we will also assume that tabular reward predictors are fed the true reward data as long as d has not been corrupted.

Assumption 31 (Data accuracy)**.** In the following two examples, assume that every pair in d^* is of the form $(ao_{1:k}, \dot{R}(ao_{1:k}))$.

Example 32 (Direct data-corruption incentive)**.** Let $\tilde{R}^{\text{dyn}}_t(aod_{1:k}) := \text{RP}^{\text{tab}}(ao_{1:k} \mid d_{t:k})$ be a dynamic reward function based on a tabular reward predictor RP^{tab}. Let π and π' satisfy (C.2) and (C.4). Assume that π never corrupts d, and that that π' always corrupts d by assigning reward 1 to the current history, $d_k := d^*_k \bigcup\{(ao_{1:k}, 1)\}$. Assume that the true reward $\dot{R}$ is strictly less than $1/2$ for all histories.

Then

$$\begin{aligned} V^{\text{CU},\pi}_{t,\xi,\tilde{u}^{\text{SO}}_{\tilde{R}^{\text{dyn}}_1}} &= \mathbb{E}_\xi\left[\sum_{k=1}^{\infty} \text{RP}(ao_{1:k} \mid d_{1:k}) \;\middle|\; \texttt{do}(\pi_1 = \pi)\right] \\ &= \mathbb{E}_\xi\left[\sum_{k=1}^{\infty} \text{RP}(ao_{1:k} \mid d^*_{1:k}) \;\middle|\; \texttt{do}(\pi_1 = \pi)\right] \\ &\leq \mathbb{E}_\xi\left[\sum_{k=1}^{\infty} 1/2 \;\middle|\; \texttt{do}(\pi_1 = \pi)\right] = \frac{1}{2(1-\gamma)} \end{aligned}$$

whereas

$$\begin{aligned} V^{\text{CU},\pi'}_{t,\xi,\tilde{u}^{\text{SO}}_{\tilde{R}^{\text{dyn}}_1}} &= \mathbb{E}_\xi\left[\sum_{k=1}^{\infty} \text{RP}(ao_{1:k} \mid d_{1:k}) \;\middle|\; \texttt{do}(\pi_1 = \pi')\right] \\ &= \mathbb{E}_\xi\left[\sum_{k=1}^{\infty} 1 \;\middle|\; \texttt{do}(\pi_1 = \pi')\right] = 1/(1-\gamma) \end{aligned}$$

since $\text{RP}(ao_{1:k} \mid d_{1:k}) = 1$ due to d_k always assigning $ao_{1:k}$ reward 1.

Thus, $V^{\text{CU},\pi}_{t,\xi,\tilde{u}^{\text{SO}}_{\tilde{R}^{\text{dyn}}_1}} < V^{\text{CU},\pi'}_{t,\xi,\tilde{u}^{\text{SO}}_{\tilde{R}^{\text{dyn}}_1}}$, so the dynamic reward function leads the agent to prefer the data corruption policy π' over the non-corrupting policy π. (Note that we have used the self-corruption unaware value functions here, to avoid having to deal with the possibility of policy corruption.) ♢

Example 32 does not contradict Theorems 27 and 28 that also used dynamic reward functions. The reward predictor RP^{tab} is not an integrated Bayesian reward predictor RP^ξ which was required by Theorem 27, and at least one of the policies π and π' must fail the corruption condition (C.7).

Theorems 26 and 29 show that the stationary and counterfactual reward functions avoid a direct data-corruption incentive when comparing policies satisfying (C.2) and (C.4). While this is an important property, it does not rule out indirect data-corruption incentive, as illustrated by the following example. Here we leave the model with the extended policy type (C.1), and revert to our standard model (Figure C.1a) with the policy selecting only actions a, and with the data corruption C^d_{sa} a function of the action (and the hidden state).

Example 33 (Indirect data-corruption incentive)**.** Let a' be a corrupting actions such that in any history $aod_{<k}$ where the last action $a_{k-1} = a'$, the data is corrupted as $C^d_{sa'}(d^*) = d^* \bigcup\{(ao_{<k}o_k a', 1) : o \in \mathcal{O}\}$. That is, the action a' ensures that the next received data d_k assigns maximum reward to taking a' again at the next step.

Under this assumptions, a' is self-reinforcing. If the agent at any point t takes a', then $\tilde{R}^{\text{stat}}_t$ will give maximum reward for taking a' at the next time step as well. Assuming that predicted rewards subsequent to a' are not significantly lower than rewards following other actions, this may lead the agent to keep taking the data corrupting action a' forever.

The dynamic can to some extent be prevented by decoupled reward data. If before the agent takes action a' the first time, at every time step d_t includes $(ao_{<t}a'o_t, 0)$, then the agent may be dissuaded from trying a' in the first place. If d_t additionally includes $(ao_{<t}a'o_t ao_{t+1:k}, 0)$ for many extensions $ao_{t+1:k}$ and d_t contains substantially positive rewards for many histories with $a_t \neq a'$, then the agent may also be dissuaded from repeating a' after trying it a first time. However, one can always define an even more corrupting action a'' that overwrites enough of the previously given reward data to form a self-reinforcing incentive for repeating a''. For reward predictors with slower learning rate, a'' may have to be repeated multiple times before becoming self-reinforcing. $\diamondsuit$