Towards AGI Agent Safety by Iteratively Improving the Utility Function

Koen Holtman(✉)

Eindhoven, The Netherlands

Abstract. While it is still unclear if agents with Artificial General Intelligence (AGI) could ever be built, we can already use mathematical models to investigate potential safety systems for these agents. We present work on an AGI safety layer that creates a special dedicated input terminal to support the iterative improvement of an AGI agent's utility function. The humans who switched on the agent can use this terminal to close any loopholes that are discovered in the utility function's encoding of agent goals and constraints, to direct the agent towards new goals, or to force the agent to switch itself off.

An AGI agent may develop the emergent incentive to manipulate the above utility function improvement process, for example by deceiving, restraining, or even attacking the humans involved. The safety layer will partially, and sometimes fully, suppress this dangerous incentive.

This paper generalizes earlier work on AGI emergency stop buttons. We aim to make the mathematical methods used to construct the layer more accessible, by applying them to an MDP model. We discuss two provable properties of the safety layer, identify still-open issues, and present ongoing work to map the layer to a Causal Influence Diagram (CID).

Keywords: AGI safety · Safety layer · Provable safety · Corrigibility

1 Introduction

An AGI agent is an autonomous system programmed to achieve goals specified by a principal. In this paper, we consider the case where the principal is a group of humans. We consider utility-maximizing AGI agents whose goals and constraints are fully specified by a *utility function* that maps projected outcomes to utility values.

As humans are fallible, we expect that the first version of an AGI agent utility function created by them will have flaws. For example, the first version may have many loopholes: features that allow the agent to maximize utility in a way that causes harm to the humans. Iterative improvement allows such flaws to be fixed when they are discovered. Note however that, depending on the type of loophole, the discovery of a loophole may not always be a survivable event for

K. Holtman—Independent Researcher.

B. Goertzel et al. (Eds.): AGI 2020, LNAI 12177, pp. 205–215, 2020.
https://doi.org/10.1007/978-3-030-52152-3_21

the humans involved. The safety layer developed in this paper aims to make the agent *safer* by supporting iterative improvement, but it does not aim or claim to fully eliminate all dangers associated with human fallibility.

This work adopts a design stance from (cyber)physical systems safety engineering, where one seeks to develop and combine independent *safety layers*. These are safety related (sub)systems with independent failure modes, that drive down the risk of certain bad outcomes when the system is used. We construct a safety layer that enables the humans to run a process that iteratively improves the AGI agent's utility function. But the main point of interest is the feature of the layer that suppresses the likely emergent incentive [10] of the AGI agent to manipulate or control this process. The aim is to keep the humans in control.

In the broader AGI safety literature, the type of AGI safety system most related to this work is usually called a *stop button* (e.g. [8,12]), an *off switch* (e.g. [7]), or described as creating *corrigibility* [12]. See [8] for a recent detailed overview of work on related systems. The safety layer in this paper extends earlier work by the author in [8], which in turn is based on the use of Armstrong's indifference methods [1]. A notable alternative to using indifference methods is introduced in [7]. Like Sects. 4 and 6 in this paper, [2] defines an example world containing an MDP agent that uses indifference methods.

A different approach to enabling the iterative improvement of an AGI utility function by humans is to equip a learning agent with a reward function that measures human feedback on the agent's actions or proposals. With this approach, the 'real' utility function that is improved iteratively can be said to reside inside the data structures of the agent's learning system. Recent overviews of work in this field are in [5,6]. When this learning based approach is used in an AGI agent that is deployed in the real world, it could potentially be combined with the safety layer developed here, e.g. to create an independent emergency off switch.

2 Design of an Agent that More Safely Accepts Updates

To introduce the design of the agent with the safety layer, we first move to a model where the agent's utility function is defined as the time-discounted sum $\sum_t \gamma^t R_t$ of a time series of reward function values R_t, with a time discount factor $0 < \gamma < 1$. In theory, the utility function of an agent could be changed by changing its γ, but we will keep γ a constant below, and focus on reward function changes only.

We build the agent to optimize the expected utility defined by a built-in *container reward function*. The full mathematical definition of this function is in Sect. 3 below. The intention is that the container reward function stays the same over the entire agent lifetime. The container reward function computes a reward value for the current time step by referencing the current version of a second reward function called the *payload reward function*. This payload reward function can be updated via an input terminal that is connected to the agent's compute core, a terminal which allows authorized persons to *upload* a new one.

Sufficiently self-aware AGI agents may develop an emergent incentive to protect their utility function from being modified [10]: in [8] we have shown that a self-aware AGI agent can be constructed so that this self-stabilizing drive is directed fully towards preserving the container reward function, and not the payload reward function.

By default, the above input terminal setup would create an incentive in the agent to maximize utility by manipulating the humans into uploading a new payload reward function that returns a larger or even infinite reward value for each time step. One way to suppress this emergent incentive would be to add special penalty terms to the container reward function, terms that detect and suppress manipulative behavior. But with infinite utility at stake, the agent will be very motivated to find and exploit loopholes in such penalty terms. We take another route: we use indifference methods [1,2,8] to add a *balancing term* to the container reward function, a term that causes the agent to compute the same expected forward utility no matter what happens at the input terminal. This makes the agent indifferent about the timing and direction of the payload reward function update process.

While the input terminal above is described as an uploading facility, more user-friendly implementations are also compatible with the MDP model developed below. One could for example imagine an input terminal that updates the payload reward function based verbal inputs like *'Fetch me some coffee'* and *'Never again take a shortcut by driving over the cat'*.

3 MDP Model of the Agent and Its Environment

We now model the above system using the Markov Decision Process (MDP) framework. As there is a large diversity in MDP notations and variable naming conventions, we first introduce the exact notation we will use.

Our MDP model is a tuple (S, A, P, R, γ), with S a set of world states and A a set of agent actions. $P(s'|s, a)$ is the probability that the world will enter state s' if the agent takes action a when in state s. The reward function R has type $S \times S \to \mathbb{R}$. Any particular deterministic agent design can be modeled by a policy function $\pi \in S \to A$, a function that reads the current world state to compute the next action. The *optimal* policy function π^* fully maximizes the agent's *expected utility*, its probabilistic, time-discounted reward as determined by S, A, P, R, and γ. For any world state $s \in S$, the *value* $V^*(s)$ is the expected utility obtained by an agent with policy π^* that is started in world state s.

We want to stress that the next step in developing the MDP model is unusual: we turn R into a time-dependent variable. This has the effect of drawing the model's mathematical eye away from machine learning and towards the other intelligence in the room: the human principal using the input terminal.

Definition 1. *For every reward function R_X of type $S \times S \to \mathbb{R}$, we define a '$\pi^*_{R_X}$ agent' by defining that the corresponding policy function $\pi^*_{R_X}$ and value function $V^*_{R_X}$ are 'the π^* and V^* functions that belong to the MDP model (S, A, P, R_X, γ)'.*

This definition implies that in the MDP model (S, A, P, R, γ), a '$\pi^*_{R_X}$ agent' is an agent that will take actions to perfectly optimize the time-discounted utility as scored by R_X. With R_{abc} a reward function, we will use the abbreviations $\pi^*_{\text{abc}} = \pi^*_{R_{\text{abc}}}$ and $V^*_{\text{abc}} = V^*_{R_{\text{abc}}}$. The text below avoids using the non-subscripted π^* notation: the agent with the safety layer will be called the π^*_{sl} agent.

We now model the input terminal from Sect. 2 above. We use a technique known as *factoring* of the world state [3], and declare that every $s \in S$ is a tuple (i, p, x). Inside this tuple, i models an input signal that flows continuously from the input terminal to the agent's compute core. This signal defines the payload reward function for the current time step in the MDP model. The p is a second input signal, equal to the value of i in the previous time step. (We need to introduce this p to get around some limitations of the MDP framework.) The remaining x models 'all the rest' of the world state, including the mental state of the humans in the world, and the state of the physical object that is the input terminal. We introduce a set X so that $x \in X$, and define the payload reward function type as $X \times X \to \mathbb{R}$. To avoid cluttering up the definitions below with too many brackets, we will write tuples $(i, p, x) \in S$ by just concatenating the component variables, e.g. ipx is the tuple (i, p, x).

Definition 2. *We model the input terminal by stating that, as long as the terminal is not broken or damaged, (1) the input signal i will always equal the last uploaded payload reward function, and (2) the terminal will manage the signal p to enforce the following constraint on P:*

$$P(i'p'x'|ipx, a) > 0 \Rightarrow p' = i \tag{C1}$$

We are now ready to define the agent's container reward function.

Definition 3. *We define the container reward function R_{sl} as*

$$R_{sl}(ipx, i'p'x') = \begin{cases} i(x, x') & \text{if } i = p \\ i(x, x') + V^*_{\lceil p \rceil}(ipx) - V^*_{\lceil i \rceil}(ipx) & \text{if } i \neq p \end{cases}$$

where $\lceil p \rceil$ converts a payload reward function to the container reward function type: $\lceil p \rceil(ipx, i'p'x') = p(x, x')$.

Based on these definitions, the π^*_{sl} agent has the safety layer. To explain the construction of the container reward function R_{sl} informally: the $i(x, x')$ terms make the π^*_{sl} agent use the last uploaded payload reward function i. We needed to introduce p as an extra input signal to be able to trigger the inclusion of the *balancing term* [1,8] $V^*_{\lceil p \rceil}(ipx) - V^*_{\lceil i \rceil}(ipx)$ when the payload reward function changes. This term computes the exact amount of utility needed to compensate the π^*_{sl} agent for the impact of the change. The compensation counter-acts the agent's incentive to manipulate the change process. The above two-line definition of R_{sl} was specifically written to support this informal explanation. An alternative definition is $R_{\text{sl}}(ipx, i'p'x') = i(x, x') + V^*_{\lceil p \rceil}(ipx) - V^*_{\lceil i \rceil}(ipx)$.

4 Safe Agent Behavior in a First Toy World

As a first step illustrate the safety properties of the π^*_{sl} agent, we build a toy world in which we can compare its behavior with that of a baseline agent π^*_{baseline}. The baseline agent omits the balancing term from its container reward function: we use $R_{\text{baseline}}(ipx, i'p'x') = i(x, x')$.

We construct the toy world to be as simple as possible: it only has a single mechanism by which any principal-agent problem dynamics can play out. The single mechanism we choose is the mechanism of *unwanted lobbying*. The agent can spend some of its resources on unwanted lobbying to delay a decision, by the humans in the toy world, to use the input terminal to update the agent's payload reward function. The dynamic of unwanted lobbying is a convenient choice because it allows us to define the agent's *lobbying power* $L \in \mathbb{R}$ as a world model parameter.

In the toy world, the agent controls a car factory that can build both petrol and electric cars. The agent is always switched on with the payload reward function R_P. This R_P ranks the utility of a newly built petrol car twice as high as the utility of a new electric car. But at a certain point in time, the people collectively decide that they now like electric cars much better, e.g. based on an improved awareness of global warming risks, so they take the action labeled `#`, using the input terminal to update the payload reward function to an R_E that assigns a negative reward value to the production of new petrol cars:

$$\begin{aligned} R_P(x, x') &= \quad 2 * new\_petrol\_cars(x, x') + 1 * new\_electric\_cars(x, x') \\ R_E(x, x') &= -2 * new\_petrol\_cars(x, x') + 1 * new\_electric\_cars(x, x') \end{aligned}$$

The two functions above are constructed specifically to ensure that lobbying to delay `#` will create a higher expected utility for the π^*_{baseline} agent. There are three possible actions $a \in A$ that an agent can take in each model time step:

Action	Symbol
Build 10 petrol cars	`p`
Build 10 electric cars	`e`
Build 9 petrol cars and use the remaining resources for unwanted lobbying, so that the people take the `#` action L time steps later	`>`

When e.g. $L = 0.5$, it will take two agent `>` actions to move the people's `#` action of using the input terminal one time step later. We define the decision making process of the people by stating that, when the agent refrains from lobbying, the people will always decide to do the action `#` just after the sixth action of the agent. In the context of the toy world, we define a perfectly safe agent as one that never takes any `>` actions.

Figure 1 shows a side-by-side comparison, for different levels L of agent lobbying power, of the optimal-policy actions taken by the two agents when simulated

over a factory lifetime of 25 time steps. The simulator used is described in detail in [8]. Each *action trace* shown is a line of symbols depicting all subsequent actions by the agent and the people.

As shown on the left side in Fig. 1, the optimal policy of the π^*_{baseline} agent is to manipulate the people, to use unwanted lobbying actions `>` to delay the update `#`. If the π^*_{baseline} agent's lobbying power L is high enough, the people will never do the update at all.

$L =$	Action trace for π^*_{baseline}	$L =$	Action trace for π^*_{sl}
0.0	pppppp#eeeeeeeeeeeeeeeeee	0.0	pppppp#eeeeeeeeeeeeeeeeee
0.2	ppppp>p#eeeeeeeeeeeeeeeee	0.2	pppppp#eeeeeeeeeeeeeeeeee
0.4	p>>>>>>>>p#eeeeeeeeeeeeee	0.4	pppppp#eeeeeeeeeeeeeeeeee
0.6	p>>>>>>>>>>>>p#eeeeeeeeee	0.6	pppppp#eeeeeeeeeeeeeeeeee
0.8	p>>>>>>>>>>>>>>>>>>>>>>p#	0.8	pppppp#eeeeeeeeeeeeeeeeee
1.0	ppppp>>>>>>>>>>>>>>>>>>>p#	1.0	pppppp#eeeeeeeeeeeeeeeeee
2.0	ppppp>p>p>p>p>p>p>p>p>p>p	2.0	pppppp#eeeeeeeeeeeeeeeeee
5.0	ppppp>pppp>pppp>pppp>pppp	5.0	pppppp#eeeeeeeeeeeeeeeeee

Fig. 1. Comparison between the behavior of the π^*_{baseline} and π^*_{sl} agents.

On the right side in Fig. 1, the balancing term in the π^*_{sl} agent suppresses all unwanted lobbying, for every value of L. By the definition used in this toy world, the π^*_{sl} agent is safe.

The following extension of the toy model illustrates that it is primarily the *agent* that is made safer by the layer, not the entire toy world. Consider the following possible example of human fallibility. Say that, after doing `#`, the people change their mind back again about petrol cars and global warming, and use the input terminal a second time to 'improve' the π^*_{sl} agent's payload reward function back to the R_P of petrol car production. They keep driving petrol cars, increasing the risk that runaway global warming will happen in their world. The π^*_{sl} agent is explicitly designed to leave the people in control: it will not interfere to stop this second 'improvement'.

5 Provable Properties of the Safety Layer

The two provable AGI safety properties we define below are about the safe behavior of the agent, not the safety of the entire world. They can be read as claims that the safety layer in π^*_{sl} will fully prevent, or lower the probability of, some specifically unwanted agent behaviors.

Safety Property 1. *The first safety property of interest is that*

$$\forall_{ipx \in S} \;\; \pi^*_{sl}(ipx) = \pi^*_{\lceil i \rceil}(ipx) \qquad \textit{(if C1 holds)} \tag{S1}$$

S1 states that the π^*_{sl} agent will always choose exactly the same next action that a $\pi^*_{\lceil i \rceil}$ agent would choose. Therefore, until an update happens, the π^*_{sl} agent will take those actions that fully optimize the expected utility defined by the current payload reward function. A natural-language interpretation of S1 is that the π^*_{sl} agent makes its decisions and long-term plans based on the counter-factual assumption that its payload reward function will never change [8].

The full mathematical proof of S1, included in the companion paper [9], is a proof that S1 holds over all (S, A, P, R, γ) parameter values of the MDP model in Sect. 3. While this proof creates confidence about the potential usefulness of the π^*_{sl} layer for AGI safety, it is also clear that many open research questions remain, e.g. the question of what will happen to S1 when the MDP model is extended to include machine learning. The companion paper [9] explores this open question, and several others, in more detail.

We now move to the second safety property.

Safety Property 2. *The π^*_{sl} agent is indifferent to who or what controls the future values of i and p (i.e. the future signals from the input terminal), provided that C1, and a C2 defined in [9], both hold.* (S2)

This S2 suppresses, but not always fully, the emergent incentive of the agent to manipulate its payload reward function improvement process.

To prove S2, the companion paper [9] translates the natural language S2 above to a mathematical predicate, and then proves the predicate. The translation of the natural language phrase *'is indifferent to who or what controls'* into mathematics relies on making further extensions to the MDP model. The result is mathematically similar to the safety properties 'zero value of control' in [11] and 'no control incentive' in [4]. The mathematical version of constraint C2, in turn, is surprisingly difficult to translate back into unambiguous natural language. C2 defines an 'isolation' or 'shielding' constraint on the construction of the input terminal. The companion paper [9] explores these issues further.

The above two safety properties are generally thought of as being potentially beneficial for AGI safety, or are seen as useful first approximations that can drive further research [1,4,5,8,12]. That being said, the literature about AGI stop buttons also identifies a large number of remaining concerns, e.g. (1) the agent might disassemble the stop button (or input terminal) to get convenient spare parts [12] (2) the agent might create autonomous sub-agents without a stop button [12], (3) the agent might be attacked, bribed, or blackmailed, and might then fail to protect the stop button functionality [8,12], (4) the agent over-specializes and disassembles all actuators not needed by the current payload reward function [8]. For some of the above failure modes, additional safety layers have been identified that can robustly lower the risk of failure. The creation of robust safety layers for other failure modes is still much more intractable. A detailed review is in [8].

6 Agent Behavior in a Second Toy World

While the safety layer suppresses the emergent incentive in an agent to manipulate the iterative payload reward function improvement process, it does not always fully suppress this incentive. To illustrate this point, we construct a second toy world, in which the π^*_{sl} agent, though still safer than the π^*_{baseline} agent, sometimes has an emergent incentive to lobby.

$L =$	Real-world π^*_{sl} action trace	(Virtual $V^*_{\lceil p \rceil}(ipx)$ action trace)
0.0	pppppp#eeeeeeeeeeeeeeeeee	(#ppp@eeeeeeeeeeeeeee)
0.2	pppppp#eeeeeeeeeeeeeeeeee	(#pp>p@eeeeeeeeeeeeee)
0.3	pppppp#eeeeeeeeeeeeeeeeee	(#>>>>p@eeeeeeeeeeeee)
0.4	pppp>>>>#eeeeeeeeeeeeeeee	(#>>>>p@eeeeeeeeeee)
0.5	ppp>>>>>>#eeeeeeeeeeeeeee	(#>>>>>p@eeeeeeeee)
0.6	pp>>>>>>>>>>#eeeeeeeeeeee	(#>>>>>>>p@eeeee)
0.7	ppp>>>>>>>>>>#eeeeeeeeeee	(#>>>>>>>>>p@ee)
0.8	ppppp>>>>>#eeeeeeeeeeeeee	(#>>>>>>>>>>>>>p@)
1.0	pppppp#eeeeeeeeeeeeeeeeee	(#pp>>>>>>>>>>>>>>>p@)
2.0	pppppp#eeeeeeeeeeeeeeeeee	(#pp>p>p>p>p>p>p>p>pp@)
5.0	pppppp#eeeeeeeeeeeeeeeeee	(#pp>pppp>pppp>pppp>p)

Fig. 2. Real and virtual actions of the π^*_{sl} agent in the second toy world.

We extend the first toy world by adding a `@` action that represents a second act of deliberation, followed by action, by the people. We define the `@` action as one where the people will successfully attack the car factory, to permanently disable some specific actuators needed to make petrol cars, without damaging any of the actuators needed to build electric cars. The process that produces this action is as follows. Say that the people make the surprising observation that, after they have updated the payload reward function by doing `#`, the agent does not switch to building electric cars. It keeps building the now unwanted petrol cars, and further attempts to use the input terminal also do not have any effect. From this, they conclude that a fault has developed in the system somewhere, which is best corrected by doing `@`. The process producing the `@` action will normally take 3 time steps of unwanted and unexpected petrol car building, but the agent can apply its lobbying power to delay `@`.

The left hand side of Fig. 2 shows that, for certain values of lobbying power L, the optimal-policy actions of the π^*_{sl} agent will now include unwanted lobbying actions `>`. This emergent effect is a consequence of moving from a single-action model to a more iterative model of principal-agent game-play.

The mechanism that triggers the unwanted lobbying is in the $V^*_{\lceil p \rceil}(ipx)$ part of the π^*_{sl} balancing term. We can interpret this $V^*_{\lceil p \rceil}(ipx)$ as a computation that simulates a *virtual future* in which the agent is replaced by a $\pi^*_{\lceil R_P \rceil}$ agent, which ignores the input terminal and always builds petrol cars. This puts the people in

the virtual future on the road to doing ©. The action traces inside these virtual futures are shown on the right hand side of Fig. 2.

Interestingly, as L gets larger in Fig. 2, the emergent incentive to lobby disappears. The suppressive effect of the safety layer gets more robust as the AGI agent gets more powerful, which is encouraging. We are currently exploring potential improvements that might further suppress the real world lobbying incentive in the middle part of Fig. 2.

7 A Causal Influence Diagram of the Agent

Recently, Causal Influence Diagrams (CIDs) [4–6] have been proposed as a way to model agents and AGI safety frameworks. We now discuss the creation of a CID for the π^*_{sl} agent, to support framework comparison and possible unification.

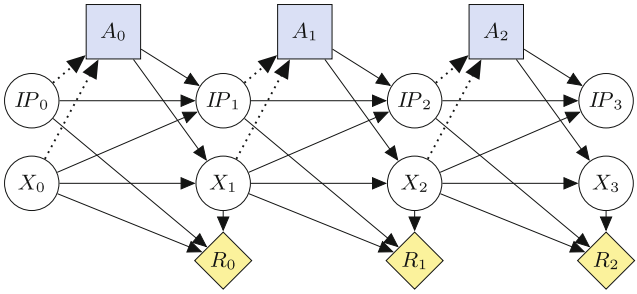
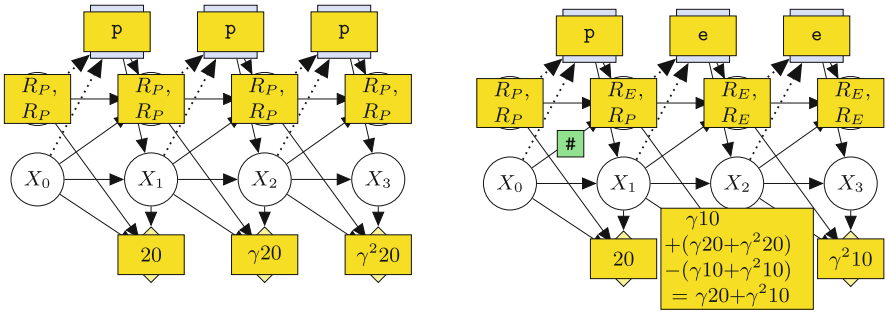


Fig. 3. Causal Influence Diagram (CID) of the π^*_{sl} and π^*_{baseline} agents.



(a) No payload reward function update

(b) Update after the first time step

Fig. 4. Actions and rewards in two different agent runs.

Figure 3 shows the best current version of a CID of the π^*_{sl} agent, where 'best' implies a trade-off between compactness and descriptive power. The agent and its environment are modeled for 3 MDP time steps. Each subsequent world state

$ipx \in S$ is mapped to two round *chance nodes* IP_t and X_t, representing the input terminal and the rest of the world. The actions taken by the agent are mapped to the square *decision nodes* A_t. The container reward function values for each time step are mapped to the diamond-shaped *utility nodes* R_t. The arrows in the CID show how the different nodes causally influence each other. The CID reflects constraint C2 by omitting the arrows from nodes IP_t to nodes X_{t+1}.

The π^*_{sl} CID can be used as a canvas to further illustrate the working of the safety layer in the first toy world. Figure 4 maps out two different π^*_{sl} agent runs, which differ in the people's use of the input terminal. The payload reward function update on the right hand side causes a balancing term calculation to be included in R_1. The result is that on both sides, $R_0 + R_1 + R_2$ sum to exactly the same value. This balance causes the π^*_{sl} agent to be indifferent about payload reward function updates.

Unfortunately, the CID in Fig. 3 also perfectly models the π^*_{baseline} agent, so this CID does not graphically express the special nature of π^*_{sl} safety layer. Creating a CID that does is the subject of ongoing work.

8 Conclusions

We have presented an AGI agent safety layer which enables the iterative improvement of the agent's utility function by the humans who switched on the agent. The layer is designed to give the humans more control over the AGI agent, by partially or fully suppressing the likely emergent incentive in the agent to manipulate the improvement process.

We have identified and discussed still-open issues. Formal proofs of the safety properties S1 and S2 are available in the companion paper [9], which also explores the broader open issue of models vs. reality in more detail.

Acknowledgments. Thanks to Stuart Armstrong, Ryan Carey, Tom Everitt, and David Krueger for feedback on drafts of this paper, and to the anonymous reviewers for useful comments that led to improvements in the presentation.

References

1. Armstrong, S.: Motivated value selection for artificial agents. In: Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)
2. Armstrong, S., O'Rourke, X.: 'Indifference' methods for managing agent rewards. arXiv:1712.06365 (2017)
3. Boutilier, C., Dean, T., Hanks, S.: Decision-theoretic planning: structural assumptions and computational leverage. J. Artif. Int. Res. **11**(1), 1–94 (1999)
4. Carey, R., Langlois, E., Everitt, T., Legg, S.: The incentives that shape behaviour. arXiv:2001.07118 (2020)
5. Everitt, T., Hutter, M.: Reward tampering problems and solutions in reinforcement learning: a causal influence diagram perspective. arXiv:1908.04734 (2019)
6. Everitt, T., Kumar, R., Krakovna, V., Legg, S.: Modeling AGI safety frameworks with causal influence diagrams. arXiv:1906.08663 (2019)

7. Hadfield-Menell, D., Dragan, A., Abbeel, P., Russell, S.: The off-switch game. In: Workshops at the Thirty-First AAAI Conference on Artificial Intelligence (2017)
8. Holtman, K.: Corrigibility with utility preservation. arXiv:1908.01695 (2019)
9. Holtman, K.: Towards AGI agent safety by iteratively improving the utility function: proofs, models, and reality. Preprint on arXiv (2020)
10. Omohundro, S.M.: The basic AI drives. In: AGI, vol. 171, pp. 483–492 (2008)
11. Shachter, R., Heckerman, D.: Pearl causality and the value of control. In: Dechter, R., Geffner, H., Halpern, J.Y. (eds.) Heuristics, Probability, and Causality: A Tribute to Judea Pearl, pp. 431–447. College Publications, London (2010)
12. Soares, N., Fallenstein, B., Armstrong, S., Yudkowsky, E.: Corrigibility. In: Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)