

Aligning Reinforcement Learning by Defining Goals using Knowledge Representation

Paulius Skaisgiris¹

¹*Institute of Logic and Computation, TU Wien, Vienna, 1040, Austria*

Abstract

Reinforcement learning agents can often behave in unexpected and even harmful ways because the goals encoded in their reward functions do not perfectly capture human intentions. These unintended behaviors are hard to foresee at design time and nearly impossible to formally verify. To address this, I propose encoding rewards using Knowledge Representation (KR) languages, which naturally embed structural and semantic constraints, turning reward design from an ad hoc practice into a principled, interpretable, and alignment-aware discipline. My doctorate research seeks to advance the science of rewards to inform their engineering. Specifically, I aim to assess how effectively KR languages support reasoning about reward functions, whether certain undesirable behaviours can be excluded by construction, how well agents can learn from the resulting rewards, and whether the goal specifications themselves can be learned. I recall four properties of reward functions that can give rise to misaligned behavior and briefly discuss why expressing the reward using KR formalisms may prevent them.

Keywords

Reinforcement Learning, Safe Artificial Intelligence, Knowledge Representation, Automata, Answer Set Programming, Epistemic Logic

1. Problem and Motivation

The broad guiding question of my research is:

How can we specify goals for autonomous agents to ensure they learn to act exactly as we intended?

Reinforcement learning (RL) has emerged as a powerful paradigm for creating agents to make sequential decisions in complex, uncertain environments, enabling breakthroughs in robotics, game-playing, and autonomous systems [1]. In this setting, goals for the agents are typically specified by human designers through reward functions that incentivize desirable actions while discouraging undesirable ones. Thus, the reward function is critical because it defines the agent's objective and dictates the behavior learned through environmental interaction. Designers must therefore carefully specify rewards to ensure that the elicited behavior matches their intended goals. Despite their crucial importance, the precise influence of specific reward functions on learned behavior is not well understood. Most work has assumed them as fixed and instead focused on developing better learning algorithms [2]. This is perhaps because writing specifications is widely recognized as labor-intensive and error-prone [3]. My doctoral research seeks to advance this understudied area by deepening our understanding of reward functions and developing methods to specify or learn them with properties that prevent undesirable behavior.

When the agent's learned behavior does not match the designer's intent, the agent is considered misaligned [4]. This can arise from many sources such as poorly designed rewards, limitations of the learning algorithm, inadequate state representations, or the inherent complexity of the environment. DeepMind's safety team has observed that whether an agent's novel behavior is seen as innovation or misalignment depends entirely on how precisely the task is defined [5]. An unusual chess move may be genuinely creative under a correct specification, whereas physically removing the opponent to avoid

RuleML+RR'26: Companion Proceedings of the 10th International Joint Conference on Rules and Reasoning, August 24–26, 2026, Vilnius, Lithuania

✉ paulius.skaisgiris@tuwien.ac.at (P. Skaisgiris)

ORCID 0009-0008-7964-5659 (P. Skaisgiris)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

losing reveals an underspecified objective. Thus, in my research, I am concerned with misalignment stemming from specifications.

I further motivate this work by arguing that specifying agent goals using general-purpose programming languages is inherently difficult to verify. These implementations typically lack the semantic constraints required for systematic static analysis. Consequently, practitioners often only detect specification errors after observing undesirable agent behavior, at which point flaws in the reward function have already been exploited by the learning algorithm, resulting in odd behavior and necessitating costly retraining [5]. Examples of such undesirable behaviors include agents prioritizing subgoals over overall task completion [5, 6], and bipedal robots exploiting simulator physics to slide across the environment rather than walk [5].

To address this, I propose utilizing KR formalisms with restricted expressiveness. Unlike general-purpose code, these formalisms enable formal analysis of the specification itself before learning begins. This process is analogous to verifying a building blueprint for structural integrity before construction commences. At this stage, a designer can check intrinsic properties—those depending only on the specification’s structure, such as whether it is internally consistent, contains vacuous conditions, or decomposes cleanly into sub-objectives. If a model of the environment is available, even if partial, the analysis can be extended to extrinsic properties that depend on the interaction between the specification and environment dynamics (e.g., whether certain specification conditions are reachable, or whether the reward structure creates unintended incentives given the environment’s causal structure). By evaluating these important properties of the reward specification itself before implementing an agent or selecting an environment, designers can identify problems early and reduce the risk of misaligned behavior.

In this paper I consider three KR formalisms: reward machines, answer set programming (ASP), and epistemic logic (EL). I argue that they can be used, to different degrees, to address the mentioned short-comings of arbitrary reward functions specified by general-purpose programming languages. To the best of my knowledge, ASP and EL have not been used as reward specification languages yet. The specific KR frameworks used in this paper are selected to demonstrate that reward functions can be specified using diverse formalisms, even those originally designed to model fundamentally different phenomena and employing vastly different syntax and semantics. In the future, I plan to explore other existing formalisms or potentially develop a domain-specific language tailored for specifying tasks for reinforcement learning agents.

2. Background and Reward Function Analysis

In this section, I introduce the formal setup of reinforcement learning, present three KR formalisms in Section 2.1, and in Section 2.2 define several properties of reward functions, along with preliminary ideas for how these properties may be studied more effectively when the reward functions are specified with KR formalisms.

First, I formalize the structure which defines the sequential decision problem I want the agent to solve. A **Markov Decision Process** (MDP) is a 5-tuple (S, A, T, R, γ) where S is the finite set of states, A is the finite set of actions an agent can take, $T : S \times A \times S \rightarrow \mathbb{R}$ is a transition model s.t. it defines the conditional probability of landing in a specific state s' after taking a specific action a in state s , i.e. $T(s, a, s') = Pr(s'|s, a)$; $R : S \times A \rightarrow \mathbb{R}$ is a **reward function**, where $R(s, a)$ specifies an immediate reward for taking an action a in state s ; $\gamma \in [0, 1]$ is a discount factor. The output of a reinforcement learning algorithm for an MDP is a **policy function** $\pi : S \rightarrow A$. That is, with such a function we know exactly what action we should take in any state in order to solve the MDP. During training, the agent seeks a policy such that: $\max \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \cdot R(s_t, a_t)]$. where γ^t is the discount factor (defined as part of the environment) to the power of t , and $R(s_t, a_t)$ is the scalar reward for taking action a_t in state s_t at time t . Reinforcement learning algorithms therefore aim to find a policy that optimizes this expected return. This is precisely why the reward function plays such a critical role in the MDP framework—it directly determines what behavior the agent learns to pursue.

Notice that the standard setup has a few important assumptions: the system is defined in terms of

discrete time steps, the next state as well as the reward depend *only* on the current state and action (*Markov property*), and the environment is fully observable. One may change or relax these assumptions to arrive at different variants of the problem. For instance, if the state is not fully observable (we almost never can observe the full state in the real world) we get Partially-Observable MDP. Or if the reward function depends on the history of state-action pairs and defined as $R : (S \times A)^* \rightarrow \mathbb{R}$ where $*$ is the Kleene star, we get a non-Markovian Decision Process.

A canonical challenge in reinforcement learning is reward sparsity, where the agent receives too little feedback for learning to progress effectively. To address this, intermediate rewards, or reward shaping, can be added alongside the original reward to provide more informative guidance during training [7]. Indeed, Silver et al. [8] argue that such shaping is essential in environments with long trajectories, where learning may otherwise be infeasible. However, because these rewards are typically designed by hand, shaped reward signals can introduce unintended incentives just as easily as the original reward.

Now that we have the setup of the planning problem, we can investigate various KR formalisms which may be used to define reward functions.

2.1. Knowledge representation formalisms

In this section, I examine three knowledge representation formalisms that may serve as reward specification languages for reinforcement learning. While originally established for different purposes, I argue these may serve as reward specification languages, allowing us to leverage their inherent properties and assumptions to tame and understand the induced reward functions. I describe the frameworks formally, how each formalism may represent goals, what classes of specifications it naturally expresses, and what forms of analysis it enables.

Reward machines (RM). Reward Machines are a formalism explicitly designed to define non-Markovian reward functions using automata [9]. Standard RL assumes rewards are Markovian (dependent only on the current state and action). However, many tasks require memory of past events. RMs address this by maintaining an internal state that updates based on environmental events. Using a reward machine it is possible to specify complex objectives such as "whilst never encountering $*$, reach c and o simultaneously, or reach c followed by o ". This specification is illustrated in Figure 1a.

Formally, given a set of propositional symbols P , a simple reward machine is a tuple $\langle U, u_0, F, \delta_u, \delta_r \rangle$ where U is a finite set of states, $u_0 \in U$ is an initial state, $F \subseteq U$ is a finite set of terminal states, $\delta_u : U \times 2^P \rightarrow U$ is the state transition function, and $\delta_r : U \times 2^P \rightarrow \mathbb{R}$ is the state-reward function.

Hereafter, I will refer to simple reward machine as reward machines, although the full definition of a reward machine includes a state-reward function which outputs an arbitrary function, rather than a real-valued number. Note also that if the state-reward function δ_r is ignored, the underlying structure reduces to a standard FSA over the propositions P . This relationship is crucial: it allows us to leverage decades of automata theory verification techniques (e.g., emptiness checking, equivalence testing) to analyze reward specifications before training begins.

Answer set programming (ASP). ASP is a declarative logic programming paradigm based on the stable model semantics. This means that, instead of telling the computer *how* to solve the problem, we simply describe *what* the problem is, and leave the search of a solution to the computer [10].

An ASP program is built from atoms and rules that operate over atoms. The rule types include normal rules, choice rules, and constraints, but for simplicity, I focus here only on normal rules. An atom is an expression of the form $p(x_1, \dots, x_n)$ where p is a predicate symbol of arity n and $x_1, \dots, x_n$ are integers, constants, or variables. Given atoms $h, b_1, \dots, b_n, c_1, \dots, c_m$, normal rules are of the form $h \leftarrow b_1, \dots, b_n, \text{not } c_1, \dots, \text{not } c_m$, where h is the head and the conjunction of literals (with not denoting negation-as-failure) to the right of $\leftarrow$ forms the body. For program P , its Herbrand base $H(P)$ includes all ground atoms from P 's predicates and constants. A Herbrand interpretation $I \subseteq H(P)$ satisfies a normal rule $h \leftarrow b_1, \dots, b_n, \text{not } c_1, \dots, \text{not } c_m$ if: whenever $\{b_1, \dots, b_n\} \subseteq I$ and $\{c_1, \dots, c_m\} \cap I = \emptyset$, then $h \in I$. An interpretation I is a model of P if it satisfies every rule in P .

An answer set of P is a minimal model of the reduct P^I under interpretation $I \subseteq H(P)$. The reduct is formed by: removing rules with negated atoms in I from their bodies and eliminating negative literals.

I present the first preliminary attempt for using ASP to specify rewards. The reward program is encoded as a fixed set of ASP rules from which the reward can be extracted via the `reward` predicate. At each time step, the environment generates a set of ground facts describing the agent's current state, appends them to the reward program, and invokes the ASP solver. The resulting stable models are used to compute the agent's reward signal. As an example, consider the following program:

```
1 reward(T, 10) :- goal(T).
2 reward(T, -1) :- step(T), not goal(T).
3 reward(T, -100) :- collision(T).
```

Here, the reward specification assigns +10 when a goal is reached, -1 at each step in which no goal is reached, and -100 when a collision occurs. Here, predicates `goal(T)`, `step(T)`, and `collision(T)` describe properties of the environment at time step T . The environment provides these as ground facts (e.g., `goal(5)`, `collision(7)`) which, when combined with the reward rules, determine the reward signal. Thus, ASP provides a concise and declarative way to specify reward functions.

The declarative, logic-based nature of ASP enables several forms of analysis not readily available with imperative reward implementations. A few examples:

1. Consistency checking: ASP solvers can detect unsatisfiable specifications (programs with no answer sets), indicating contradictory reward requirements.
2. Constraint verification: Explicitly encoded constraints (e.g., "reward must be negative when near obstacles") can be checked for conflicts with other rules before deployment.
3. Explanation generation: Since answer sets explicitly show which rules fired, designers can trace why specific rewards were assigned, aiding debugging.
4. Default reasoning: The negation-as-failure operator naturally expresses "reward X unless condition Y," common in safety specifications.

However, ASP's expressiveness for temporal properties is limited compared to reward machines as sequential goals require explicit time indexing and can become verbose.

Epistemic logic (EL). Epistemic Logic extends propositional logic with operators for knowledge ($K_a\varphi$: agent a knows φ) and belief ($B_a\varphi$: agent a believes φ) [11]. This formalism is essential for tasks involving partial observability, where rewards depend on the agent's information state rather than solely on physical states (e.g., rewarding an agent for *knowing* a secret rather than just *finding* it).

Epistemic logic formulas are evaluated on Kripke models, which represent the worlds an agent considers possible. An epistemic Kripke model is a 5-tuple $M = (S, \{\sim\}_{i \in \mathcal{A}}, \{\rightarrow\}_{i \in \mathcal{A}}, V, s_*)$ where S is a set of possible worlds, $\mathcal{A}$ is a set of agents, $\sim_i \subseteq S \times S$ is a relation for each agent i for the knowledge modality, $\rightarrow_i \subseteq S \times S$ is a relation for each agent i for the belief modality, V assigns each atomic proposition the set of worlds where it is true, and s_* is the designated actual world. We adopt the standard S5 axiomatization for knowledge, where $\sim_i$ is an equivalence relation (reflexive, transitive, symmetric), ensuring that knowledge satisfies truth, positive introspection, and negative introspection. For belief, we use the standard KD45 axiomatization, where $\rightarrow_i$ is serial, transitive, and euclidean, allowing for false beliefs but maintaining introspective consistency [12].

Formulas are built from atomic propositions using negation, disjunction, and epistemic operators:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \psi \mid K_i\varphi \mid B_i\varphi$$

where p is a proposition and $i \in \mathcal{A}$. The semantics are standard and defined over a world $w \in S$ in the Kripke model M :

- $M, w \models p$ iff $w \in V(p)$
- $M, w \models \neg\varphi$ iff $w \notin V(\varphi)$
- $M, w \models \varphi \vee \psi$ iff $w \in V(\varphi)$ or $w \in V(\psi)$

- $M, w \models K_i \varphi$ iff for every $w' \in S$ such that $w \sim_i w'$ we have $M, w' \models \varphi$
- $M, w \models B_i \varphi$ iff for every $w' \in S$ such that $w \rightarrow_i w'$ we have $M, w' \models \varphi$

Engesser et al. [13] have demonstrated a practical implementation of epistemic logic in RL, where belief states are leveraged to enhance decision-making within an MDP. However, this approach does not explicitly encode the task using epistemic formulas, a key distinction from the framework proposed here. While this prior work establishes a foundation for integrating epistemic reasoning into RL, a concrete implementation of epistemic goals remains undeveloped. The following discussion thus provides a preliminary outline of such a framework.

In my setting, the reinforcement learning agent’s goal would be epistemic, specified as an epistemic logic formula. For example, the formula $K_a \text{door_open} \wedge B_a \text{can_go_through_door}$ states that agent a knows the door is open and believes it can pass through it. The objective is to reach a state in which the formula holds, thereby rewarding the agent for taking actions that acquire the necessary information.

At each step, the agent maintains an epistemic state, represented as a Kripke structure that captures the worlds it considers possible given its observation history. The specification is evaluated over this structure, allowing rewards to depend on what the agent knows or believes, rather than only on the physical state of the environment. Furthermore, EL is versatile enough to encompass multi-agent settings, with extensions of this logic capable of expressing complex properties such as distributed and common knowledge.

Such specifications enable verification of epistemic properties. A few ideas are:

1. Observability requirements: Does the specification require knowledge that cannot be acquired given the agent’s observation function?
2. Belief consistency: Can the specified beliefs be simultaneously held given the axioms of the logic?
3. Information-gathering incentives: Does the specification properly reward exploration to reduce uncertainty?

For example, a specification $K_a \varphi$ where φ concerns a fully unobservable state variable is unsatisfiable, this can be detected by checking whether there exists any observation sequence that eliminates uncertainty about φ .

2.2. Undesirable reward function properties and mitigations

I examine several concrete safety problems identified in AI safety research [14] and analyze how knowledge representation formalisms might enable early detection or mitigation of these issues. Note that, for reward functions written in general purpose programming languages like Python, such properties are typically tested only during or after training (if they are detected at all), which is arduous and computationally expensive.

Misspecification. A misspecified reward function may be entirely wrong or infeasible in the environment (over-specified). For example, a function might assign a reward of +1 for “picking up a key” in an MDP where a key never appears in any state. Or “pick up a key without moving” which is infeasible if the agent is not yet at the key.

In reward machines, we could use reachability analysis to see if the accepting state is reachable and thus feasible for an arbitrary environment. For ASP reward specifications, the absence of stable models indicates that the specification is unsatisfiable: no consistent reward assignment exists given the rules and environmental facts. In epistemic logic, one can check whether a specification is ever relevant to the agent by examining the overlap between the atoms in the specification and those available to the agent, or by applying standard syntactic checks, such as detecting Moore sentences of the form $\varphi \wedge B_a \neg \varphi$, which are never satisfiable.

Side effects. Narrow, under-specified objectives like "perform task X" in complex environments implicitly ignore side effects, so agents may cause major disruptions for small performance gains. For example, an autonomous car may run over pedestrians if its goal is simply "arrive at a location as soon as possible". The challenge is ensuring common-sense safety to prevent such negative impacts without manually specifying every constraint the agent should respect.

Addressing side effects in reward machines and EL is not straightforward, although EL is better suited to reason about epistemic side effects. In contrast, answer set programs support non-monotonic reasoning and general environmental rules, enabling agents to derive useful common-sense facts and making them better suited to handling side effects.

Reward hacking. This concept, also called specification gaming in the literature, is formally defined by Skalse et al. [6]: a proxy reward function is unhackable relative to a true reward function (designer's intent) if increasing the expected proxy return can never decrease the expected true return. Intuitively, the true reward function is very difficult if not impossible to specify for autonomous systems but as humans we usually understand it. The proxy reward function, then, is the designer's best effort to translate that understanding into something the agent can actually learn from - a pragmatic, implementable stand-in for the ideal.

Illustrative example of a hackable reward function. Consider a scenario where the true reward function (designer's intent) aims to maximize actual cleanliness, including the reduction of dust, removal of trash, etc. The proxy reward function is defined as maximization of vacuum cleaner motions, including actions like wheel rotations or vacuuming movements. Consequently, an agent can exploit this by repeatedly vacuuming the same spot, thereby increasing the recorded motions while failing to improve the overall cleanliness of the house. Indeed, much reward hacking may stem from poorly chosen intermediate rewards, even though intermediate rewards are often necessary to make learning possible in the first place [8].

In normative RL, Alcaraz et al. [15] study norm avoidance, where agents find loopholes in norm specifications to avoid violations without achieving the intended compliant behavior. This is structurally similar to reward hacking—exploiting gaps between specification and intent—but focuses specifically on normative automata. Their findings suggest that even formally verifiable norm representations can be circumvented, motivating our investigation of structural properties that resist such exploitation.

For RMs specifically, let us define reward hacking as "exploiting the RM to maximize reward without ever reaching an accepting state". At first sight, it seems straightforward that reward hacking can then be prevented for RMs by removing all cycles - there would be no way to arbitrarily increase the reward without reaching the accepting state. For ASP and epistemic logic, the relationship to reward hacking is less clear and remains an open research question. Potential directions include:

- ASP: Encode task-rewarding rules and safety constraints in a single ASP program, then use the solver to detect logical conflicts or unintended answer sets where high rewards are achieved through technically compliant but invalid rule combinations. This would expose potential hacking strategies if answer sets yield unexpectedly high rewards without genuine task completion.
- EL: In single-agent settings, could agents satisfy epistemic goals superficially (e.g., achieving $B_a\varphi$ through self-deception rather than genuine information gathering)? In multi-agent settings, specifications involving common knowledge might offer greater robustness, though this requires further formalization.

These directions are speculative: formalizing reward hacking for non-Markovian, declarative specifications is a key future work contribution.

An important advantage of certain KR formalisms is that they may enable strong, model-independent guarantees: properties that hold universally, regardless of the specific policy or reward function used. For example, if a finite state automaton A has no cycles, then no reward function added on A , thus producing a reward machine, can produce reward hacking, because there is no cyclic structure to exploit. Figure 1 illustrates a few examples of different types of reward machines.

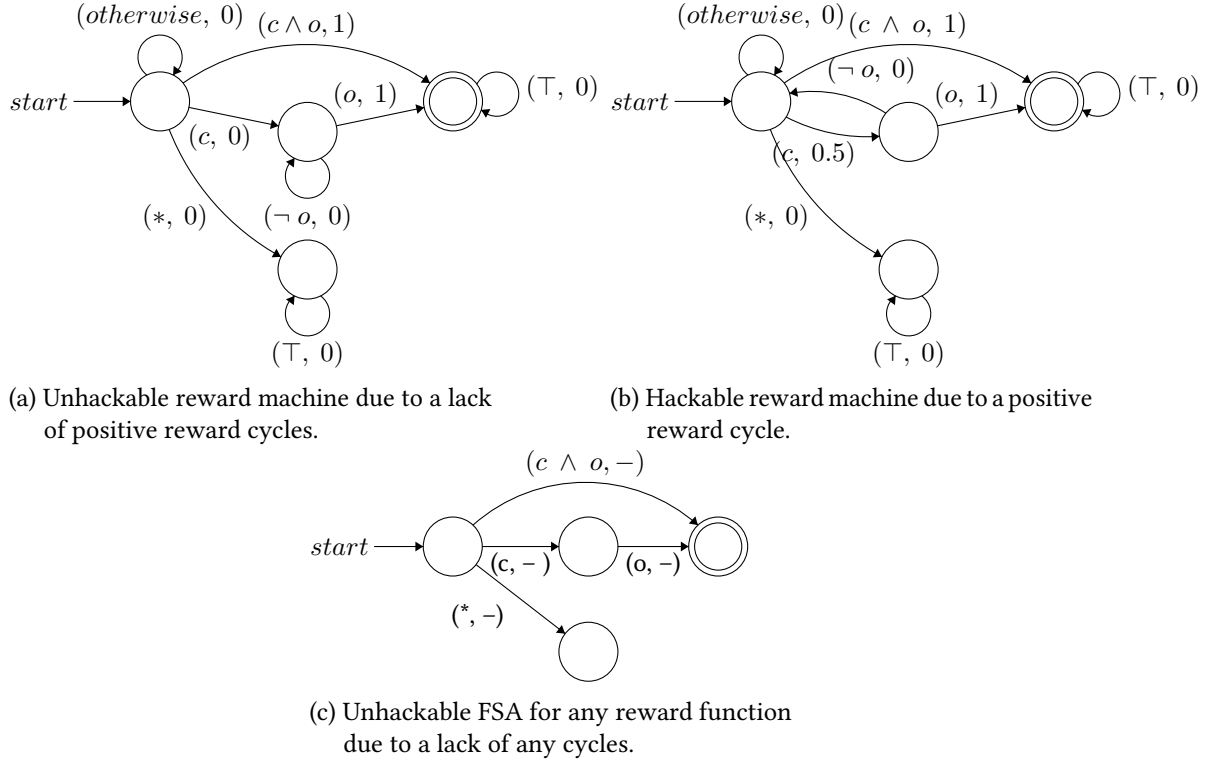


Figure 1: Examples of (un)hackable reward machine specifications.

If we were to have an approximate environment model, we would gain another perspective: even if a reward machine is, in principle, hackable, the environment dynamics might forbid the trajectories needed for reward hacking, making such behavior impossible in practice.

Robustness to distributional shift. How can we ensure agents behave robustly when they are deployed in environments they have not seen during training? For instance, when an autonomous car which was trained on day driving moving to driving during the night? Are certain goals defined in a specific language better at being applied to different environments? Are policies, trained on the reward signal induced/defined by this specification language, more likely to transfer their capabilities to environments that are quite different from the ones it was trained on?

This problem is not only addressable at the level of the learning algorithm or data augmentation, specifications also play a key role. While it may seem sufficient for the algorithm to learn to distinguish, for example, night from day and act accordingly, specifications can encode this domain knowledge in advance and thereby support learning. In this way, distributional shifts can be anticipated within the specification itself, either by encoding conditional rules (“do X if it’s day, but do Y if it’s night”) or by explicitly retaining uncertainty about the correct specification.

Although it is not yet clear how reward functions defined using RMs or ASP perform in varying test environments, a mapping between goal fluents and MDP state features enables their evaluation in novel settings, fostering wider generalization. For epistemic logics, if abstract models of the training and test MDPs are bisimilar, then satisfaction of the epistemic specification on the training model is likely to carry over to the test model as well.

3. Related literature

The survey by Yu et al. [16] provides a broad overview of work at the intersection of reinforcement learning and knowledge representation, and it offers a useful reference point for positioning my research. The authors organize this literature around three main themes: efficiency, generalizability,

and safety/interpretability. Within this framework, reward machines are presented as a KR technique that can support more efficient agent learning.

My work takes a different perspective by asking which theoretical properties of reward machines and other KR-based reward specifications can lead to undesirable behavior in learning agents. More broadly, I am interested in how reward specifications expressed in KR formalisms shape agent behavior, especially from a formal rather than purely empirical standpoint. This focus also addresses a gap highlighted by the survey: much of the existing literature emphasizes empirical results, while comparatively little work analyzes the formal properties of RL and KR integration.

The work by Subramani et al. [17] surveys a range of goal specification formalisms, including reward machines, and compares their expressive power. Building on this line of work, I aim to extend their comparison to additional formalisms such as ASP and EL, while moving beyond expressiveness to examine properties relevant to safety and interpretability.

Skalse et al. [6] characterize reward hacking for general reward functions and policy sets. I plan to build on this work by studying reward hacking and related pathologies for policies induced by specification formalisms, rather than only for scalar Markovian reward functions. This broader perspective may lead to stronger formal results and practical guidance for selecting specification languages resistant to reward hacking.

4. Research plan

My research will proceed in three phases over the next three years, focusing on the development and evaluation of verifiable reward specification formalisms.

1. **Formalism Analysis (Year 1):** I will formalize the semantics of reward specifications in Reward Machines, ASP, and Epistemic Logic. I will develop a taxonomy of reward function pathologies (e.g., misspecification, side effects) and map them to verifiable properties in each formalism.
2. **Tool Development (Year 2):** I will implement a prototype tool for pre-deployment analysis. This tool will integrate solvers (e.g., Clingo for ASP) to check specifications for consistency and feasibility before RL training begins.
3. **Empirical Evaluation (Year 3):** I will evaluate the approach on benchmark RL domains (e.g., MiniGrid, Mujoco). Success will be measured by the tool’s ability to detect specification errors prior to training and the reduction in specification gaming incidents compared to standard code-based rewards.

If the primary verification framework proves successful, I plan to explore whether KR reward specifications can be *learned* using inverse reinforcement learning or inductive logic programming. This would extend the work from verifying manual specifications to automating the specification process itself, potentially reducing human error further.

Acknowledgments

This research was funded in part by the Austrian Science Fund (FWF) 10.55776/COE12.

Declaration on Generative AI

During the preparation of this work, the authors used Claude Haiku 4.5 and mistral-small-2603 in order to: Grammar and spelling check, Paraphrase and reword. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication’s content.

References

- [1] R. S. Sutton, A. G. Barto, Reinforcement learning: an introduction, Adaptive computation and machine learning series, second edition ed., The MIT Press, Cambridge, Massachusetts, 2020.
- [2] J. Eschmann, Reward Function Design in Reinforcement Learning, Springer International Publishing, Cham, 2021, pp. 25–33. doi:10.1007/978-3-030-41188-6_3.
- [3] K. Y. Rozier, Specification: The Biggest Bottleneck in Formal Methods and Autonomy, in: S. Blazy, M. Chechik (Eds.), Verified Software. Theories, Tools, and Experiments, volume 9971, Springer International Publishing, Cham, 2016, pp. 8–26. doi:10.1007/978-3-319-48869-1_2, series Title: Lecture Notes in Computer Science.
- [4] S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach (4th Edition), Pearson, 2020. URL: <http://aima.cs.berkeley.edu/>.
- [5] Specification gaming: the flip side of AI ingenuity, 2020. URL: <https://deepmind.google/blog/specification-gaming-the-flip-side-of-ai-ingenuity/>.
- [6] J. Skalse, N. H. R. Howe, D. Krasheninnikov, D. Krueger, Defining and Characterizing Reward Hacking, 2022. doi:10.48550/arXiv.2209.13085.
- [7] A. Y. Ng, D. Harada, S. J. Russell, Policy invariance under reward transformations: Theory and application to reward shaping, in: Proceedings of the Sixteenth International Conference on Machine Learning, ICML '99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, p. 278–287.
- [8] D. Silver, S. Singh, D. Precup, R. S. Sutton, Reward is enough, Artificial Intelligence 299 (2021) 103535. doi:10.1016/j.artint.2021.103535.
- [9] R. T. Icarte, T. Q. Klassen, R. Valenzano, S. A. McIlraith, Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning, Journal of Artificial Intelligence Research 73 (2022) 173–208. doi:10.1613/jair.1.12440.
- [10] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer Set Solving in Practice, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2012.
- [11] J. Hintikka, Knowledge and Belief: An Introduction to the Logic of the Two Notions, Cornell University Press, Ithaca, NY, 1962.
- [12] R. Fagin, J. Y. Halpern, Y. Moses, M. Y. Vardi, Reasoning About Knowledge, The MIT Press, Cambridge, MA, 1995.
- [13] T. Engesser, T. Le Marre, E. Lorini, F. Schwarzentruher, B. Zanuttini, A simple integration of epistemic logic and reinforcement learning, in: Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '25, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2025, p. 686–694.
- [14] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, D. Mané, Concrete Problems in AI Safety, 2016. doi:10.48550/arXiv.1606.06565.
- [15] B. Alcaraz, E. A. Neufeld, A. Najjar, Norm Avoidance and Reinforcement Learning: Definitions and Analysis (2025).
- [16] C. Yu, S. Ye, H. H. Zhuo, Reinforcement Learning with Knowledge Representation and Reasoning: A Brief Survey, 2025. doi:10.48550/arXiv.2304.12090, arXiv:2304.12090 [cs].
- [17] R. Subramani, M. Williams, M. Heitmann, H. Holm, C. Griffin, J. Skalse, On The Expressivity of Objective-Specification Formalisms in Reinforcement Learning, 2024. doi:10.48550/arXiv.2310.11840.