Proceedings of Machine Learning Research vol 242:694–706, 2024

Pointwise-in-Time Diagnostics for Reinforcement Learning During Training and Runtime

Noel Brindise NBRINDI2@ILLINOIS.EDU
Department of Aerospace Engineering, University of Illinois Urbana-Champaign, USA

Andres Felipe Posada-Moreno ANDRES.POSADA@DSME.RWTH-AACHEN.DE
Institute for Data Science in Mechanical Engineering (DSME), RWTH Aachen University, Germany

Cedric Langbort LANGBORT@ILLINOIS.EDU
Coordinated Science Laboratory, University of Illinois Urbana-Champaign, USA

Sebastian Trimpe TRIMPE@DSME.RWTH-AACHEN.DE
Institute for Data Science in Mechanical Engineering (DSME), RWTH Aachen University, Germany

Abstract

Explainable AI Planning (XAIP), a subfield of xAI, offers a variety of methods to interpret the behavior of autonomous systems. A recent "pointwise-in-time" explanation method, called Rule Status Assessment (RSA), characterizes an agent's behavior at individual time steps in a trajectory using linear temporal logic (LTL) rules. In this work, RSA is applied for the first time in a reinforcement learning (RL) context. We first demonstrate RSA diagnostics as a substantial supplement to the basic RL reward curve, tracking whether and when specified subtasks are accomplished. We then introduce a novel "Interactive RSA" which provides the user with detailed diagnostic information automatically at any desired point in a trajectory. We apply RSA to an advanced agent at runtime and show that RSA and its novel interactive variant constitute a promising step towards explainable RL.

Keywords: Explainable AI Planning, Explainable Reinforcement Learning, Linear Temporal Logic, Markov Decision Processes

1. Introduction

While reinforcement learning (RL) agents have proven invaluable for automation, complex agent behavior often defies human understanding, causing difficulties during agent training and inference (Heuillet et al. (2021)). To remedy this, a recent subfield of Explainable Artificial Intelligence, Explainable Reinforcement Learning (XRL), has proposed many approaches to explain RL agent behavior (Wells and Bednarz (2021)). Milani et al. (2022) organize XRL into three major areas: *feature importance* (FP), which determines the influence of input state features on agent actions; *learning process and MDP* (LPM), which identifies past events that led to the current state; and *policy level* (PL), which describes overall agent behavior. Here, FP and LPM methods rely on the internal model of the agent and environment to provide the human with success probabilities (Cruz et al. (2019)), reward decomposition (Juozapaitis et al.), or important junctures in training (Gottesman et al. (2020)). Other methods learn an "explanatory" model to replace the complex agent, finding causal links between variables and actions (Madumal et al. (2020b,a)), reworking complex policies to fit "interpretability" criteria (Hein et al. (2018)), and describing plans in terms of abstract human-interpretable states (Sreedharan et al. (2020)).

While these methods are promising, none compare the agent behavior explicitly to a *desired* behavior, such as known tasks to be completed or rules to be followed; this is a concern in Explainable AI Planning (XAIP), which focuses on autonomous trajectory-planning systems from drones to deep-space robots (Sakai and Nagai (2022); Barkouki et al. (2023); Sreedharan et al. (2022)). We specifically consider the XAIP approach to rule-based explanation, where time- and order-dependent tasks are often expressed in Linear Temporal Logic (LTL), a syntax appearing in formal methods (Baier and Katoen (2008)) and niche controls contexts (Bemporad and Morari (1999)). In XAIP, LTL *inference* has been applied to describe and compare agent behavior by searching for a set of LTL "rules" which are true on agent trajectories, answering the question "what does the agent always, or usually, do?" (Kim et al. (2019); Camacho and McIlraith (2019); Roy et al. (2023); Chou et al. (2020)). Recently, a "pointwise-in-time" framework was proposed to bring analysis to a more local level; here, a set of rules gathered from inference or expert knowledge can be tracked at individual time steps in an agent trajectory, answering the question "what progress has the agent made at time t?" (Brindise and Langbort (2023)).

In this work, we apply this framework, called rule status assessment (RSA), for the first time in RL with the help of our novel explanatory algorithm, Interactive RSA. **During training**, we show how the notion of LTL formula and subformula "status" illuminates the learning progress of an agent on tasks and subtasks. **During inference**, we introduce Interactive RSA to automatically explain a deployed agent's behavior at moments of interest. We use Gymnasium-based Griddly (Bamford (2021)) for training demonstrations and a Video Pre-Training agent (Baker et al. (2022)) on the MineRL environment (MinerlLabs (2022)) to demonstrate benefits of Interactive RSA for developers and users of RL at runtime.

2. Problem Formulation

We consider a setting in which a human user wishes to assess the progress of an RL agent at individual moments in a trajectory, where progress is measured in terms of tasks that the agent should accomplish or rules it should follow. Here, we suppose that these "rules" are expressible in LTL and acquired a priori, e.g. through expert knowledge or LTL inference. In particular, we seek an adaptation of the RSA framework which tracks pointwise-in-time diagnostics for RL training and inference. During training, these diagnostics should provide trajectory **time step(s)** when each task or requirement begins and is completed to assess the agent's policy development; during inference, the diagnostics should describe a **status of each task** at any time step of interest to the human observer. Intuitively, suppose an agent should collect a key to open a door. Our RL diagnostics should then track the key was first collected (if it was) and when the door was opened; they should also allow us to select any time step and ask the question "what is the status of the key-door task?".

To formalize our context, we begin by introducing **Reinforcement Learning**, which refers to a branch of machine learning where an agent improves its behavior (learns) by continuously interacting with an environment and receiving feedback in the form of rewards or penalties. This agent-environment dynamic can be systematically represented using the framework of a Markov Decision Process (MDP). In the context of RL, a policy π is a strategy that the agent employs over the MDP to decide what actions to take in a given state. Specifically, a policy is a mapping from states to actions, where for each state s, $\pi(a \mid s)$ gives the probability distribution over the set of available actions a. A primary objective in many RL scenarios is to identify the best policy to maximize the expected cumulative reward over time. Thus, the objective of most algorithms is to learn

a policy π^*, or equivalently, its associated optimal state-value function V^* or action-value function Q^*. Various algorithms, including Q-learning, proximal policy optimization (PPO) (Schulman et al. (2017a)), and MuZero (Schrittwieser et al. (2020)), have been developed to attain this objective. For our diagnostic goal, we are interested in how and when the trajectories produced by a policy comply or compare with our desired behaviors.

Now, given an RL agent, we must represent the agent's behavior and our requirements (tasks, etc.) such that we may evaluate the former subject to the latter. This is possible by extending the agent MDP with a *labeling function*, resulting in a *labeled Markov decision process* (LMDP).

Definition 1 (Labeled Markov Decision Process) *A labeled MDP is a tuple $\mathcal{M} = \langle S, A, T, R, P, \mathcal{L} \rangle$, where S is a finite set of discrete states, A is a finite set of actions, $T : S \times A \to (S \to \mathbb{R})$ is a stochastic transition function, and $R : S \times A \to \mathbb{R}$ is a reward function. P is a finite set of atomic propositions, or labels, and $\mathcal{L} : S \to 2^P$ is the labeling function.*

The labels produced by $\mathcal{L}$ may represent any states or conditions that are relevant to our tasks; for example, reaching the key may be represented with the label `key`. We briefly note that the set of labels P may be acquired in multiple ways. For an RL agent in a basic simulated environment, labels may be simply be a known list of possible state features or object types. In more complicated state spaces, such as a self-driving car with image observations, labels may be inferred for example via a state abstraction process (Behl et al. (2020)).

As the agent progresses, an LMDP produces a sequence of labels called a *trace*, which may be shortened into *segments*:

Definition 2 (Trace) *For any state $s_t \in S$, $T_0 \leq t \leq T_f$, let $\mathcal{L}(s_t) = L_t$, where L_t is a set of labels. A trace is the sequence $\rho = (L_{T_0}, ..., L_{T_f})$ produced by system trajectory $(s_{T_0}, ..., s_{T_f})$ from time step T_0 to T_f.*

Definition 3 (Trace Segment) *The trace segment $\rho^{t\cdots} = (L_t, ..., L_{T_f})$ is the part of the full trace ρ beginning at t and ending at T_f $(T_0 \leq t \leq T_f)$.*

Now we discuss our *rules*, introducing **Linear Temporal Logic** (LTL), a formal language incorporating both logical and temporal operators which is well suited for expression of time- and process-dependent tasks (Pnueli (1977)). To connect the LMDP to our specifications, we require that our rules depend solely on labels in P:

Definition 4 (Linear Temporal Logic Formula) *For the set of LTL formulas φ over a finite set P of propositions,*

- *if $\alpha \in P$, then α is itself an LTL formula*
- *if φ_1 and φ_2 are LTL formulas, then $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, $\mathbf{X}\varphi_1$, and $\varphi_1 \mathbf{U} \varphi_2$ are LTL formulas.*

All LTL formulas can be constructed from atomic propositions (the labels of our LMDP); standard logical operators $\vee$ (or), $\wedge$ (and), $\rightarrow$ (implication); and temporal operators $\mathbf{X}$, $\mathbf{U}$. The temporal operators are given in Table 1, with additional common operators defined for convenience.

Given an LTL formula, a trace may now be checked for rule satisfaction, defined below; note that we assume finite agent trajectories, employing the finite version of LTL (LTL_f).

Operator	Meaning
$\mathbf{G}\varphi_1$	**Global**: φ_1 is always true.
$\mathbf{F}\varphi_1$	**Eventual**: φ_1 eventually occurs.
$\mathbf{X}\varphi_1$	**Next**: φ_1 must occur at the next time step.
$\varphi_1\mathbf{U}\varphi_2$	**Until**: φ_1 remains true until φ_2 occurs (and φ_2 must occur)
$\varphi_1\mathbf{R}\varphi_2$	**Release**: φ_1 remains true (1) until and including the step when φ_2 occurs or (2) forever

Table 1: LTL operators, where φ denotes an LTL formula. "Occurence" of φ_i at t means $\rho^{t\cdots} \models \varphi_i$.

Definition 5 (LTL Formula Satisfaction) *LTL formula φ is satisfied by trace ρ, denoted $\rho \models \varphi$, in the following cases:*

- $\rho \models \alpha$ *where* $\alpha \in P$ *iff* $\alpha \in L_0$
- $\rho \models \neg\varphi$ *iff* $\rho \not\models \varphi$
- $\rho \models \varphi_1 \vee \varphi_2$ *iff* $\rho \models \varphi_1$ *or* $\rho \models \varphi_2$

and for the temporal operators Next $\mathbf{X}$ *and* Until $\mathbf{U}$*,*

- $\rho \models \mathbf{X}\varphi$ *iff* $\rho^{1\cdots} \models \varphi$
- $\rho \models \varphi_1\mathbf{U}\varphi_2$ *iff* $\exists i \geq 0$ *s.t.* $\rho^{i\cdots} \models \varphi_2$ *and* $\rho^{k\cdots} \models \varphi_1$ *for all* $0 \leq k < i$

Recalling our original agent progress-tracking motive, we will show that the stages of a task can be represented within an LTL formula by the formula's *arguments*:

Definition 6 (Arguments of an LTL formula) *Consider the LTL order of operations: (1) grouping symbols; (2) $\neg, \mathbf{X}$, and other unary operators; (3) $\mathbf{U}$ and other temporal binary operators; and (4) $\vee, \wedge, \rightarrow$. For a given LTL formula φ, the one or more φ_j bound by the weakest operator are the arguments of φ.*

The formula $\varphi = \mathbf{G}\alpha_1 \vee \alpha_2$, for instance, has arguments $\varphi_1 = \mathbf{G}\alpha_1$, $\varphi_2 = \alpha_2$. For practical purposes, we treat the argument of "atomic" formulas $\varphi = \alpha$ as the formula itself. Establishing such a system, we may formalize our problem statement.

Problem Statement. Consider an RL agent following policy π and its LMDP representation, which produces a trace ρ of labels from a vocabulary P for each trajectory of the agent. Define also a list of LTL rules $\varphi_0, \varphi_1 ...$ which describes desired or expected behavior for the agent. For this setting, we seek an RSA-based diagnostic system which

(1) uses times of rule status changes on ρ to track learning progress, and

(2) automatically selects explanatory data based on the status of φ and its arguments ("interactive RSA") given a trace ρ and any selected time step t^*.

In this work, we will outline such a system and demonstrate its use in both a training and an inference context.

3. Methods

To produce diagnostics for LTL rules at individual time steps in a trajectory, we begin with the RSA framework as defined by Brindise and Langbort (2023), with Figure 1 depicting the explanatory

context. As shown, rules may stem from knowledge of the agent or an inference process, both of which would rely on a known set of labels. Formally, $\mathcal{L}(\cdot)$ must be defined on S such that all labels $\alpha \in L_t$ are in P for the LTL rules; thus, if labels are generated through an abstraction process, we assume that the abstractions are sufficiently expressive to formulate meaningful rules. Within the explanatory context, this present work specifically seeks useful "query heuristics" to extract RL-relevant diagnostics from RSA.

We begin by examining the existing framework. To formalize notions of task progress, RSA proposes that any LTL rule may be classified as *active, satisfied, inactive,* or *violated* at individual time steps in a trace. These notions depend on additional notions of *arbitrariness* and *preconditions*:

Definition 7 (Arbitrariness of suffix) *For LTL formula φ, its argument(s) φ_j, trace $\rho^{T_0\cdots}$, and associated segment $\rho^{t_0\cdots}$ where $T_0 \leq t_0 \leq T_f$, we say that $\rho^{t\cdots}$ is* arbitrary *with respect to φ if $\rho^{t_0\cdots} \models \varphi$ regardless of the truth of $\rho^{t\cdots} \models \varphi_j$ for all t where $t_0 \leq t \leq T_f$.*

Definition 8 (Precondition) *The* precondition *of an LTL formula φ is defined as (1) φ_1 if φ has the form $\varphi_1 \rightarrow \varphi_2$ and (2) $\top$ otherwise.*

Definition 9 (Status of LTL formula) *We define the following statuses for an LTL formula φ given a trace $\rho^{T_0\cdots}$ and times $t_0, t \in \{T_0, ..., T_f\}$, $t_0 \leq t$:*

- *φ is* active (a) *at t iff (1) $\rho^{t_0\cdots} \models \varphi$, (2) $\rho^{t_0\cdots} \models \psi$ for precondition ψ of φ, and (3) $\rho^{t\cdots}$ is not arbitrary.*
- *φ is* satisfied (s) *at t iff (1) φ* active *at t and (2) $t = T_f$ or φ not active at $t+1$.*
- *φ is* inactive (i) *at t iff (1) $\rho^{t_0\cdots} \models \varphi$ and (2) φ is neither* active *nor* satisfied *at t.*
- *φ is* violated (v) *at t iff $\rho^{t_0\cdots} \not\models \varphi$.*

Finally, we store the status information for each node in *timesets* τ, where

$$\tau^q = \{t \in \{t_0, ..., T_f\} \mid t \text{ has status } q\}. \tag{1}$$

Example 1 provides a basic system structure to which RSA could be applied. For a demonstration of specific notions of status, we refer the reader to the examples in Brindise and Langbort (2023).

Example 1 (Autonomous Collection Drone) *Consider an autonomous drone which must complete tasks while adhering to the following safety and operational constraints.*

(1) First collect a sample from location Q_1, Q_2, or Q_3 and then deposit it at location D_1.

(2) If battery is low, visit the charging station C_1 before any collection or deposit location.

(3) If an obstacle is detected, reroute to avoid the obstacle.

The system state low battery has label $\texttt{lb}$, obstacle detected has $\texttt{ob}$, and performing reroute has $\texttt{rr}$. Similarly, Q_1 corresponds to label q_1, Q_2 to q_2, and so on, forming the vocabulary $P = \{q_1, q_2, q_3, d_1, c_1, \texttt{lb}, \texttt{ob}, \texttt{rr}\}$.

In LTL syntax, our three requirements can be written

- $\varphi_1 = \mathbf{F}((q_1 \vee q_2 \vee q_3) \wedge \mathbf{F}d_1)$
- $\varphi_2 = \mathbf{G}(\texttt{lb} \rightarrow (\mathbf{F}c_1 \wedge (\neg(q_1 \vee q_2 \vee q_3 \vee d_1)\mathbf{U}c_1))$
- $\varphi_3 = \mathbf{G}(\texttt{ob} \rightarrow \mathbf{X}\texttt{rr})$

On this system, RSA would determine which rules are "in progress" at specific time steps within a trace, e.g. $\rho = L_0, L_1, \underline{L_2}, L_3 \quad \rightarrow \quad$ *At $t = 2$, φ_1 is active on $\rho^{0\cdots}$; φ_2, φ_3 are not.* Moreover,

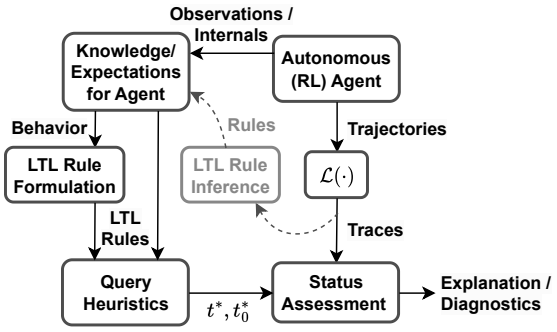

Figure 1: Proposed use of LTL rule status assessment for RL agent diagnostics.

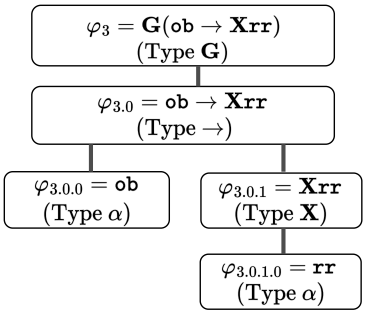

Figure 2: Formula tree for φ_3.

RSA can provide status for each argument of any φ; these arguments may be treated as nodes in a formula "tree," a common decomposition where each node φ has its arguments as children (see Figure 2). Notably, the status of a parent formula depends solely on the truth values of its children; for instance, a formula $\varphi = \varphi_1 \mathbf{U} \varphi_2$ changes from active to satisfied at t when φ_2 true on $\rho^{t\cdots}$. However, the selection of "intuitively helpful" segments $\rho^{t_0\cdots}$ to query for each φ_k node is non-trivial; this selection process and its applications are the main concern of this work.

Finally we note that, as RSA assesses status at all $t_0 \in \{T_0, ..., T_f\}, t \geq t_0$ for all nodes of a given formula tree, the upper bound on complexity is $\mathcal{O}((2|\rho|)^{L-1})$, where L is tree depth.

RSA for Training: the Global Trigger Heuristic. We first propose the use of status changes to track learning progress. One simple method is our novel *global trigger* heuristic, which determines when a task is "triggered" and when it is "done" using status notions from RSA. For formulas of format $\varphi = \mathbf{G}(\varphi_0 \rightarrow \varphi_1)$, where $\rho \models \varphi$, the heuristic identifies a *trigger* time at $t_{tt} = \text{argmin}_{t \geq t_0}\{t \mid \rho^{t\cdots} \models \varphi_0\}$. Intuitively, the rule is triggered when its precondition is true for the first time and *done* at $t_d = \text{argmin}_{t \geq t_{tt}}\{t \mid \varphi_1 \text{ satisfied or inactive at } t\}$ with status evaluated on suffix $\rho^{t_{tt}\cdots}$. This heuristic is demonstrated in Section 4.

RSA for Inference: Interactive RSA. Our problem statement also calls for relevant diagnostic information given a time step of interest t^*. To this end, we introduce *Interactive RSA*. For a given t^*, this novel algorithm returns one status assessment and accompanying description for each rule argument, continuing to a prespecified depth in the rule tree. Interactive RSA, detailed in Algorithm 1, is demonstrated in Section 4.2 on an agent which has been trained to play a game of Minecraft.

4. Numerical Experiments

We present two experiments on RL agents, the first of which is a training scenario on a tabular Griddly environment (Bamford (2021)) based on OpenAI Gym (Brockman et al. (2016)) and trained using PPO (Schulman et al. (2017b)) via RLLib (Liang et al. (2018)). The second is an inference scenario on a MineRL environment (Guss et al. (2019)) with a Video Pre-Training agent (Baker et al. (2022)). Both simulations were run in Python 3.9 on an AMD Ryzen 5 4600H. We refer the reader to `https://github.com/n-brindise/live_expl`.

Input : time of interest t^*, explanation depth D
Output: status and text explanation for φ as generated below
Data: Full trace $\rho^{T_0 \cdots}$ and rules φ_k
Initialize nodeList $\leftarrow \{\varphi_0, ...\varphi_K\}$; nodeDict$[\varphi_k][t_0] \leftarrow T_0$ for all k; $d \leftarrow 0$
while $d \leq D$ **do**
  for $\varphi_x \in$ *nodeList* **do**
    $t_0 \leftarrow$nodeDict$[\varphi_x][t_0]$; childNodeList $\leftarrow \{\}$
    $\tau_a, \tau_s, \tau_i, \tau_v \leftarrow$ `getTaus` (φ_x, ρ^{t_0})
    if φ_x *active at* ρ^{t_0}, t^* **then**
      for $\varphi_{x.y}$ *of* φ_x **do**
        append $\varphi_{x.y}$ to childNodeList
        nodeDict$[\varphi_{x.y}][t_0] \leftarrow t^*$
      end
      print "φ_x is active at ρ^{t_0}, t^*"
    else if φ_x *satisfied at* ρ^{t_0}, t^* **then**
      for... `// same` $\varphi_{x.y}$ `loop as active case`
      print "φ_x is satisfied at ρ^{t_0}, t^*"
    else if φ_x *inactive at* ρ^{t_0}, t^* *and* φ_x *satisfied at some* $t' < t^*$ **then**
      for... `//` $\varphi_{x.y}$ `loop, store` t' `in nodeDict[`$\varphi_{x.y}$`][`t_0`]`
      print "φ_x is inactive at ρ^{t_0}, t^*; previously satisfied at t'"
    else if φ_x *inactive at* ρ^{t_0}, t^* **then**
      for... `//` $\varphi_{x.y}$ `loop, store` t^* `in nodeDict[`$\varphi_{x.y}$`][`t_0`]`
      print "φ_x inactive on all ρ^{t_0}"
    else
      for... `//` $\varphi_{x.y}$ `loop, store` t_0 `in nodeDict[`$\varphi_{x.y}$`][`t_0`]`
      print "φ_x violated on all ρ^{t_0}"
    end
  end
  $d \leftarrow d + 1$; nodeList $\leftarrow$ nodeChildList
end

Algorithm 1: Interactive Rule Status Assessment.

4.1. Training: Treasure Hunt Environment

The "Treasure Hunt" is shown in Figure 3. Here, the agent is rewarded for collecting a key ($R(a_t) = +50$), opening a door ($R(a_t) = +50$), and going to the treasure chest ($R(a_t) = +100$). During training, we store a model checkpoint at each iteration; to produce traces for diagnostic analysis, we then perform inference 100 times for each model. An example reward curve is shown in Figure 5, with notable improvements at iterations 20 and 45 to be investigated further. We begin with a basic RSA status diagnostic checking satisfaction times which requires no additional heuristics.

Average Satisfaction Times. We set V to contain all variables in R and establish LTL rules:

- $\varphi_0 = \mathbf{F}$`key` (eventually go to key)
- $\varphi_1 = \mathbf{F}$`open_door` (eventually open door)
- $\varphi_2 = \mathbf{F}$`treasure_chest` (eventually go to treasure chest)

We use RSA to plot average rule satisfaction times for each iteration (Figure 4). At iteration 20, the agent begins to satisfy $\mathbf{F}$`open_door`, becoming more consistent through iteration 40. The

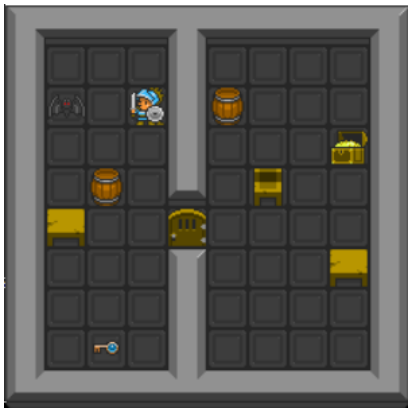

Figure 3: Treasure Hunt tabular environment.

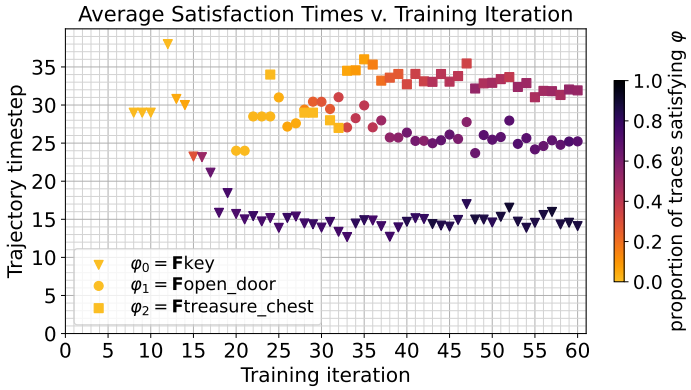

Figure 4: Average times until satisfaction for three rules, with 100 traces per iteration.

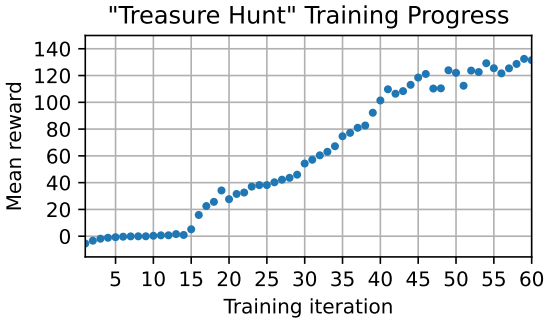

Figure 5: Learning curve for the Treasure Hunt environment.

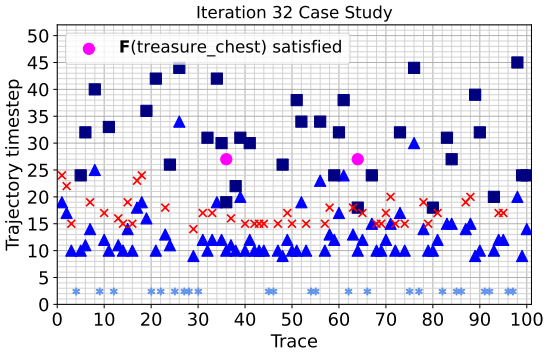

Figure 6: Trigger plot for Iteration 32.

$\mathbf{F}$`treasure_chest` behavior also appears at Iteration 24 and grows in frequency after 35, further explaining the increase in the reward curve.

Global Trigger. Now we consider the more complex subtask `key`-then-`open_door`: $\varphi_3 = \mathbf{G}(\texttt{key} \rightarrow \mathbf{F}\texttt{open\_door})$. From Figure 4, we identify 3 checkpoints of interest.

Iteration 20: The $\mathbf{F}$`open_door` behavior first appears. We analyze 25 traces at this iteration in Figure 7. The majority show triggering of φ_3 (agent reaches the key); however, all but one of these instances ends in rule violation (the red 'x'). Traces 7, 8, 15, and 19 fail to trigger at all, and 17 is the only instance to successfully complete the key-door subtask by opening the door at time step 24.

Iteration 40: From Figure 4, both $\mathbf{F}$`key` and $\mathbf{F}$`open_door` are satisfied much more consistently. Indeed, the trigger plot in Figure 8 shows that 11 of 25 traces successfully complete the key-door subtask; of these successes, 7 take ≤ 10 timesteps from trigger to finish, suggesting a relatively direct path to the door following key acquisition.

Iteration 32: Figure 4 reveals that the average satisfaction time for $\mathbf{F}$`treasure_chest` is lower than that of $\mathbf{F}$`open_door`. This is perhaps unexpected, since the treasure is not reachable

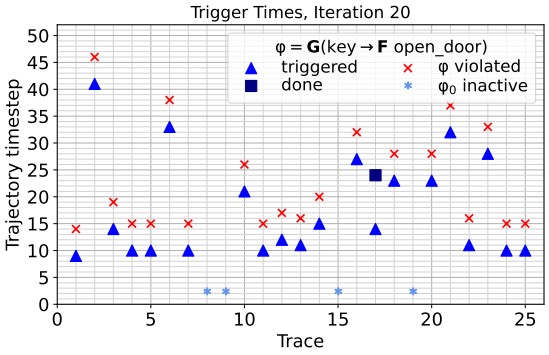

Figure 7: Trigger plot for Iteration 20.

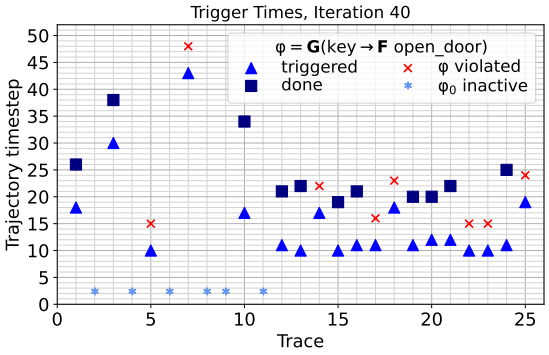

Figure 8: Trigger plot for Iteration 40.

until after the door is open. We produce Figure 6 on 100 traces, overlaying satisfaction times for $\mathbf{F}$`treasure_chest`. We observe that the vast majority of traces triggering φ_3 do not finish until time step 25 or significantly later; meanwhile, `treasure_chest` is reached in only two traces, but it does this comparatively early (around step 26), explaining the discrepancy in the two averages.

Using our enhanced RSA-based analysis, **we are able to see changes in consistency and satisfaction time of subtasks as RL training progresses,** as well as **individual trajectory behaviors** contributing to our statistics.

4.2. Inference: Minecraft Environment and Interactive RSA

Rule status assessment also shows promise for explanation of complex agents during inference, such as a Video Pre-Training MineRL agent (Baker et al. (2022)). Figure 9 demonstrates the application of Interactive RSA to an agent trajectory of length $|\rho| = 3400$ on the MineRL environment subject to five rules:

- $\varphi_0 = \neg($`wooden_pickaxe` $\vee$ `stone_pickaxe`$)$ $\mathbf{R}$ `crafting_table`
- $\varphi_1 = \neg$`wooden_pickaxe` $\mathbf{U}$ (`oak_planks` $\vee$ `spruce_planks`...)
- $\varphi_2 = \neg$`cobblestone` $\mathbf{U}$ `wooden_pickaxe`
- $\varphi_3 = \neg$`stone_pickaxe` $\mathbf{U}$ (`cobblestone` $\wedge$ `stick`)
- $\varphi_4 =$ $\mathbf{F}$ (`iron_ore` $\vee$ `coal`)

Here, rules 0-3 correspond to gameplay mechanics, which determine how tools are crafted and materials are mined. Rule 4 is a goal: we wish to eventually collect either `iron_ore` or `coal`.

Interactive RSA was run with explanation depth 3 at 5 individual time steps selected during an episode of gameplay, with average runtime 2.45s per query. Selected results are shown in Figure 9. We find that Interactive RSA generates diagnostics which are interpretable in intuitive terms. For instance, at time step $t^* = 600$, we have rules 0-2 inactive, meaning that the agent has already satisfied these requirements. In this case, this implies that a `crafting_table`, `oak_planks`, and a `wooden_pickaxe` have all been acquired. The first arguments of 2 are highlighted, informing the user that the pickaxe was first acquired at $t = 592$ ("satisfied on $\rho^{592\ldots}$"). We further examine rule 3, which remains active; here, we see that argument `cobblestone` $\wedge$ `stick` is violated, in particular because `cobblestone` is missing.

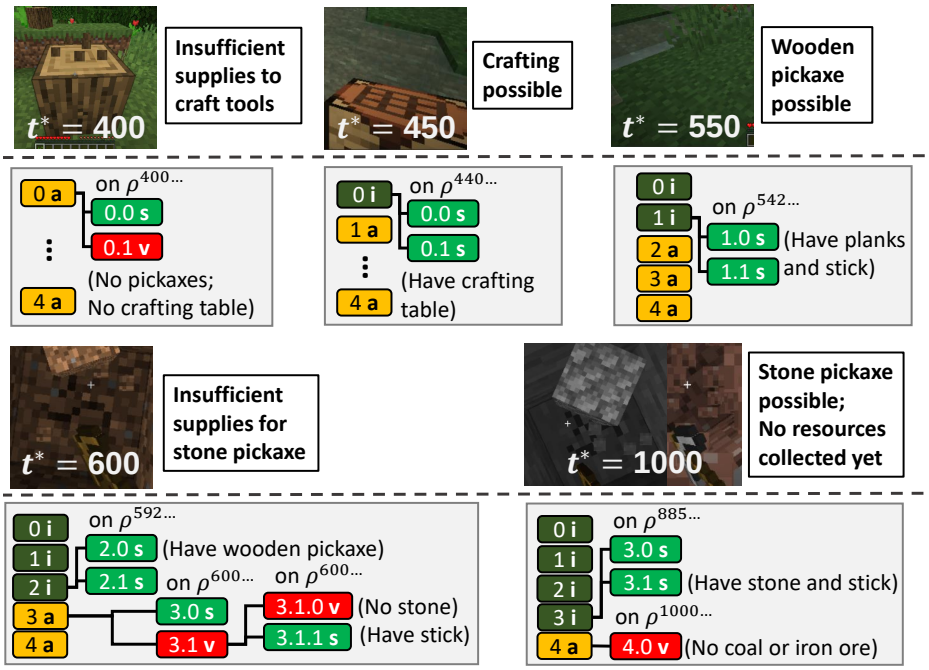

Figure 9: Interactive RSA diagnostics on a MineRL trace.

We note here that, in contrast to the examples in Brindise and Langbort (2023), Interactive RSA requires only a depth specification and a **single time of interest** to automatically identify a small, relevant set of status information. This makes Interactive RSA very well suited for users observing an agent at runtime, as a user who is familiar with the LTL specifications can quickly query and interpret the progress of each rule at any time step. In this way, RSA provides a precise human-defined proxy for understanding behavior in settings where the length of traces or the dimensionality of the state complicates a straightforward diagnosis.

5. Conclusion

In this work, we extend rule status assessment (RSA) diagnostics to reinforcement learning agents for the first time, developing heuristics which automatically analyze agent behavior with respect to predefined tasks and rules. In experiments, RSA explanation provides diagnostics which supplement the training reward curve, tracking the agent's progress in performing specific tasks as training progresses. Moreover, our novel Interactive RSA automatically provides relevant context to a user at any desired time step, as demonstrated on a long trajectory of a complex agent.

Limitations of the RSA analysis include its dependence on full agent trajectories; future work may adapt the framework to handle uncertainty, extending the use case to live agent performance where trajectories are incomplete. Additional study is also necessary to determine the practical utility for users.

Altogether, our results suggest that RSA-based diagnostics are an illuminating supplement to current explainable RL methods, providing a novel contextualization of agent behavior within the desired tasks and rules it should follow.

Acknowledgments

This research was funded in part by a National Defense Science and Engineering Graduate Fellowship. It was also funded in part by an Advanced Research Opportunities Program (AROP) scholarship from the RWTH Aachen University and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy -– EXC-2023 Internet of Production -– 390621612.

References

Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022.

Christopher Bamford. Griddly: A platform for ai research in games. *Software Impacts*, 8:100066, 2021.

Tammer Barkouki, Ziquan Deng, John Karasinski, Zhaodan Kong, and Stephen Robinson. Xai design goals and evaluation metrics for space exploration: A survey of human spaceflight domain experts. In *AIAA SCITECH 2023 Forum*, page 1828, 2023.

Aseem Behl, Kashyap Chitta, Aditya Prakash, Eshed Ohn-Bar, and Andreas Geiger. Label efficient visual abstractions for autonomous driving. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2338–2345. IEEE, 2020.

Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

Noel Brindise and Cédric Langbort. Pointwise-in-time explanation for linear temporal logic rules. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 4387–4392. IEEE, 2023.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Alberto Camacho and Sheila A McIlraith. Learning interpretable models expressed in linear temporal logic. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 621–630, 2019.

Glen Chou, Necmiye Ozay, and Dmitry Berenson. Explaining multi-stage tasks by learning temporal logic formulas from suboptimal demonstrations. *arXiv preprint arXiv:2006.02411*, 2020.

Francisco Cruz, Richard Dazeley, and Peter Vamplew. Memory-based explainable reinforcement learning. In *AI 2019: Advances in Artificial Intelligence: 32nd Australasian Joint Conference, Adelaide, SA, Australia, December 2–5, 2019, Proceedings 32*, pages 66–77. Springer, 2019.

Omer Gottesman, Joseph Futoma, Yao Liu, Sonali Parbhoo, Leo Celi, Emma Brunskill, and Finale Doshi-Velez. Interpretable off-policy evaluation in reinforcement learning by highlighting influential transitions. In *International Conference on Machine Learning*, pages 3658–3667. PMLR, 2020.

William H Guss, Cayden Codel, Katja Hofmann, Brandon Houghton, Noboru Kuno, Stephanie Milani, Sharada Mohanty, Diego Perez Liebana, Ruslan Salakhutdinov, Nicholay Topin, et al. Neurips 2019 competition: the minerl competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:1904.10079*, 1(8), 2019.

Daniel Hein, Steffen Udluft, and Thomas A Runkler. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence*, 76:158–169, 2018.

Alexandre Heuillet, Fabien Couthouis, and Natalia Díaz-Rodríguez. Explainability in deep reinforcement learning. *Knowledge-Based Systems*, 214:106685, 2021.

Zoe Juozapaitis, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI Workshop on explainable artificial intelligence*.

Joseph Kim, Christian Muise, Ankit Jayesh Shah, Shubham Agarwal, and Julie A Shah. Bayesian inference of linear temporal logic specifications for contrastive explanations. International Joint Conferences on Artificial Intelligence, 2019.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International conference on machine learning*, pages 3053–3062. PMLR, 2018.

Prashan Madumal, Tim Miller, Liz Sonenberg, and Frank Vetere. Distal explanations for explainable reinforcement learning agents. *arXiv preprint arXiv:2001.10284*, 2020a.

Prashan Madumal, Tim Miller, Liz Sonenberg, and Frank Vetere. Explainable reinforcement learning through a causal lens. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 2493–2500, 2020b.

Stephanie Milani, Nicholay Topin, Manuela Veloso, and Fei Fang. A survey of explainable reinforcement learning. *arXiv preprint arXiv:2202.08434*, 2022.

MinerlLabs. Minerl v1, 2022. URL https://github.com/minerllabs/minerl.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. ieee, 1977.

Rajarshi Roy, Jean-Raphaël Gaglione, Nasim Baharisangari, Daniel Neider, Zhe Xu, and Ufuk Topcu. Learning interpretable temporal properties from positive examples only. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6507–6515, 2023.

Tatsuya Sakai and Takayuki Nagai. Explainable autonomous robots: A survey and perspective. *Advanced Robotics*, 36(5-6):219–238, 2022.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nat.*, 588(7839):604–609, 2020. doi: 10.1038/s41586-020-03051-4. URL https://doi.org/10.1038/s41586-020-03051-4.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017a. URL http://arxiv.org/abs/1707.06347.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017b.

Sarath Sreedharan, Siddharth Srivastava, and Subbarao Kambhampati. Tldr: Policy summarization for factored ssp problems using temporal abstractions. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 272–280, 2020.

Sarath Sreedharan, Anagha Kulkarni, and Subbarao Kambhampati. Explainable human–ai interaction: A planning perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 16(1):1–184, 2022.

Lindsay Wells and Tomasz Bednarz. Explainable ai and reinforcement learning—a systematic review of current approaches and trends. *Frontiers in artificial intelligence*, 4:550030, 2021.