Synthesis of Shields for Safe Reinforcement Learning in Both Discrete and Continuous State and Action Spaces

Anonymous submission

Abstract

Reinforcement Learning agents operating under temporal safety constraints such as "always/never do X", "do X within *k* timesteps", "whenever condition C holds, do X next", etc. must learn those constraints through trial and error, with no guarantee of compliance and significant wasted training time, and the possible experience of unacceptably dangerous states if training in the real world. Shielding addresses this by intercepting agent actions and rejecting any that violate constraints or foreclose future safe choices, providing provable safety guarantees. Existing shield construction methods are largely limited to finite, discrete state and action spaces and scale poorly. We present a novel shield synthesis approach that handles infinite, continuous or discrete state and action spaces. The shield is constructed offline, is minimally interfering, and is agnostic to the choice of RL algorithm.

Introduction

Reinforcement Learning (RL) is a form of machine learning in which an agent learns to perform a task by attempting to maximize its reward. Many domains where RL is applied have rules or constraints that need to be observed: includine.g automated driving (Shalev-Shwartz, Shammah, and Shashua 2016), games (Giacobbe et al. 2021), healthcare (Jia et al. 2020), real world mechanics (Banerjee et al. 2025), and legal (Abe et al. 2010), etc. Such constraints can take the form of "always do X", "eventually do X", "do X within *k* timesteps", "infinitely often do X", and logical combinations of these. Currently an RL agent has to learn these constraints the hard way, taking many training episodes, with no guarantee that they were actually learnt. Besides needing to ensure that the agent does not enter bad states during execution, such states may also be unacceptable during training for agents that learn in the real world rather than simulations.

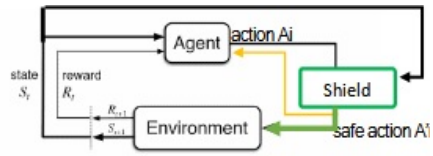


Figure 1: Role of the shield: accepting or rejecting action choices by the agent

One approach to enforcing constraints is to use *shielding* (Alshiekh et al. 2018), as shown in Figure 1. In shielding, the action choices of an RL agent are first passed to a shield which ensure that they don't violate any constraints or lead to situations in which there is no safe action choice. If the action choice is allowed, then it is *provably guaranteed* to never lead to an unsafe state. In case the shield rejects the action, the agent is asked to make a different choice.

In this paper we introduce a novel approach to shield construction that, in contrast to many current approaches:

- Handles both discrete and continuous spaces as well as discrete and continuous actions, in a uniform way.
- Supports multi-objective synthesis. We give an example in which the agent must simultaneously satisfy 3 objectives.
- Avoids explicit dependence on enumerated state-space size by operating symbolically rather than through finite-state automata, with computational cost governed instead by the size of the symbolic constraints and the underlying SMT quantifier-elimination procedures. It also has a relatively straightforward implementation.

Preliminaries

Reinforcement Learning and MDPs

In RL, an agent interacts with an environment in order to learn to carry out a task. The agent has some choice of actions. For example, for a simple drone, the choices might be to move forward, move back, move left, move right, or remain still. The outcome of the chosen action (for example the new location) is determined by the environment. The resulting state may be non-deterministic(e.g., drawn from a distribution); for example moving left may cause the drone to be in one of several places which are not exactly left due to wind currents. Additionally, the environment can contain other agents, such as other drones which must be avoided, or geofences that constrain the space in which the drone can operate.

Definition. A (non probabilistic) Markov Decision Process (MDP) is a tuple $\langle S, I, A, R\rangle$ where S is a (possibly infinite) set of states, $I \subseteq S$ is a set of initial states, $A \subseteq S \times S$ is a set of actions (transitions), and $R : S \times A \times S \to \mathbb{R}$ is a reward function. The set of allowable transitions is specified by an action formula called an action definition.

The agent begins in some initial state and then repeatedly picks an action which leads to a new state as determined by

the MDP until a goal state or some termination condition is reached.

Temporal Properties

Temporal logic is a formalism used to express properties whose truth value can vary over time (For example, "It is raining" may not hold now but may have held yesterday, or "There will be another leap year" is a property that always holds). A *basic linear temporal logic (LTL) formula* is defined by the following grammar:

$$\phi ::= false \,|\, true \,|\, p \,|\, (\phi) \,|\, \phi \wedge \phi \,|\, \phi \vee \phi \,|\, \phi \rightarrow \phi \,|\, \neg\phi \,|\, \bigcirc \phi$$

p where p is an ordinary first order state formula or a first order action formula and $\bigcirc$ is the temporal operator "next" indicating that the formula it precedes holds in the next time step. The fact of a formula ϕ holding at position n of a trace σ (an infinite sequence of states) is denoted $\sigma, n \models \phi$ and defined, by induction on the structure of ϕ, as follows:

- $\sigma, n \models f$, for f a state formula, if f holds at state $\sigma[n]$, i.e. $[\![f]\!](\sigma[n])$;
- $\sigma, n \models a$, for a an action, if a holds over the states $\sigma[n], \sigma[n+1]$, i.e. $[\![a]\!](\sigma[n], \sigma[n+1])$;
- $\sigma, n \models \neg\phi$ if $\sigma, n \not\models \phi$;
- $\sigma, n \models \phi \wedge \psi$ if both $\sigma, n \models \phi$ and $\sigma, n \models \psi$;
- $\sigma, n \models \phi \vee \psi$ if either $\sigma, n \models \phi$ or $\sigma, n \models \psi$, or both;
- $\sigma, n \models \phi \rightarrow \psi$ if either $\sigma, n \models \neg\phi$ or $\sigma, n \models \psi$, or both;
- $\sigma, n \models \bigcirc\phi$ if $\sigma, (n+1) \models \phi$

$\models \phi$ is an abbreviation for $\sigma, 0 \models \phi$. A *safety property* is an LTL formula prefixed by the "always" operator $\Box$ where

- $\sigma, n \models \Box\phi$ if $\sigma, i \models \phi$ for every $i \geq n$

$stateP(\phi)$ holds if ϕ is a state formula. $actionP(\phi)$ holds if ϕ is an action formula.

Models

We abstract an MDP into a *labeled transition system* (LTS) (Baier and Katoen 2008). Such transition systems are formulated in a simple linear temporal logic of actions, similar to Lamport's TLA (Lamport 2002). A *state* is a map from variables to (type-consistent) values. *State formulas* are boolean-valued expressions formed over the variables of a state and the constants (including functions) relevant to an application domain. A state formula p denotes a predicate $[\![p]\!]$ over states, so $p(s)$ denotes the truth value $[\![p]\!](s)$ for state s. *Actions* are boolean-valued expressions formed over variables, primed variables, and the constants (including functions) relevant to an application domain. An action a specifies a state transition and it denotes a predicate $[\![a]\!]$ over a pair of states, and $a(s,t)$ denotes the truth value $[\![a]\!](s,t)$ for states s and t. The expression $x' = x + 1 + y$ is an example action where the unprimed variables refer to the first state and primed variables refer to the second state.

In our case an LTS consists of *nodes* connected by arcs corresponding to *transitions*, along with a countable set of variables V. Transitions may have guards (predicates) on them that are based on some environment input. LTSs in which there are only self arcs, only inter node arcs, or both, are possible. In this paper, shielded agents all follow the same policy, so there is only one node.

Nodes have *labels* (state predicates, to be found) associated with them. The semantics of a node is given by the set of MDP states satisfying the corresponding node label, i.e. for node m

$$[\![m]\!] = \{s \mid L_m(s)\}$$

Arcs have arc labels. An arc label L_a is a predicate that specifies a non-deterministic action :

$$L_a(s, e, u, s') \equiv e \in E_a(s) \wedge G_a(s, u) \wedge s' = f_a(s, u, e)$$

where

1. e is treated as an uncontrollable adversary input, constrained by a given predicate $E_a(s)$;
2. u is treated as a controllable input provided by the agent, and conforms to a guard predicate $G_a(s, u)$ which is to be found;
3. function f_a gives the deterministic response of the action, as determined by the MDP.

Similar to a node, the semantics of an arc, $[\![a]\!]$, is given by the transitions or steps that satisfy the arc label:

$$[\![a]\!] = \{(s, s') \mid \exists e, u.\ L_a(s, e, u, s')\}$$

Traces. A *trace* is a sequence of states. An LTS generates a trace $\tau = s_0, s_1, \ldots$ if

1. Initially, s_0 is a legal state of the node m, i.e. $s_0 \in [\![m]\!]$;
2. Inductively, if $i \geq 0$ and s_i is a legal state of m, i.e. $s_i \in [\![m]\!]$, then there exists an arc $a_j = \langle m, m \rangle_j$ where $(s_i, s_{i+1}) \in [\![a_j]\!]$ and where s_{i+1} is a legal state of the node; i.e. $s_{i+1} \in [\![m]\!]$.

The set of traces of a LTS M is denoted $\mathscr{L}(M)$. A legal state s of M is *non-blocking* if there is a an arc $a = \langle m, m \rangle$ and control input u such that for any adversary input satisfying assumption A, the guard G_a hold for s, u and a transitions to a legal configuration of M. In game-theoretic terms, if all states of M are non blocking, then the system has a winning strategy that ensures that a safety property always holds. The goal of shield synthesis is to find those guards that ensure that the system can always win.

Shield Synthesis

At a very high level, our approach is to start with a LTS, here referred to as a *model*, that *overapproximates* the intended shield, that is it accepts all the traces that satisfy the given safety property and typically also those that do not. Through a process called *model refinement* the violating behaviors are eliminated until only the desired traces are accepted. Using a technique called *property enforcing model refinement* PEMR (Smith and Nedunuri 2024), the refinement process strengthens the inductive invariant (L) on the states or arcs and the guard (G) on each arc. The refinement process is implemented by a constraint satisfaction algorithm that iterates over a system of problem specific constraints until convergence. These constraints serve three purposes: they localize the required state and action properties to node invariants

Algorithm 1 Model Refinement Algorithm. The definition of *wcp* is given in Appendix 2

//initialize the node and arc invariants to the state
//and step safety properties, resp.
if $stateP(\phi)$
then: $L := L \ \wedge \ \phi$
else $L_a \leftarrow L_a \ \wedge \ \phi$

//iterate over the constrains until no further
//refinement possible
do for each arc a
$G_a \ := \ G_a \ \wedge \ wcp(L)$
$L := L \wedge \exists u.\, G_a$
until for all a,$L \rightarrow G_a \ \wedge \ L$.

and arc labels, refine each guard so that any allowed action leads to a legal next state for all admissible environment behaviors, and eliminate blocking states by requiring that every state satisfying the invariant admit at least one allowed action. The final convergent system provides the necessary predicates for the desired shield. The exact constraints and the details by which this is carried out are in Appendix 2.

Constraint solving

Given a set of monotone constraints such as are detailed in Appendix 2, a straightforward algorithm for solving them is presented in Algorithm 1. *wcp* is the weakest control predicate, also defined in Appendix 2.

The iteration converges to a fixpoint when the invariants and guards do not change in an iteration. An efficient control strategy for this algorithm is presented in (Rehof and Mogensen 1999), which maintains a queue of constraints that are possibly violated, resulting in an optimal algorithm for solving definite constraints. Assuming Algorithm 1 converges to a greatest fixpoint, then (1) the result is a weakest refinement of $\mathcal{S}$ that satisfies the required properties Φ, and (2) the final refined invariant L captures the set of nonblocking initial states from which the system can ensure that all behaviors stay within the specified safety conditions.

Several key problems arise in solving the constraint system: (1) eliminating the quantifiers in wcp instances, (2) simplifying intermediate formulas, and (3) ensuring termination. For (1) we make use of the built in quantifier elimination (QE) procedure for linear arithmetic provided by SMT solvers such as Z3 (De Moura and Bjørner 2008). For (2) we have developed a custom simplifier which aggressively carries out context-dependent simplification. For (3), forcing termination in a fixpoint iteration algorithm is addressed by widening mechanisms from abstract interpretation (Cousot and Cousot 1977). In practice, we terminate the synthesis procedure after a fixed number of iterations (50) as that typically indicates a problem with the specification.

A prototype implementation of the shield synthesis as well as an implementation of a shielded agent utilizing the *StableBaselines3* implementation of the PPO algorithm is available for download from `https://github.com/Anon78702/Shielded-RL`. Several key problems arise in solving the constraint system; these are outlined in Appendix 2.

Example 1: Predator and Prey

Our running example is one in which several predators pursue a prey. At least one predator must intercept the prey before it reaches the safety of the herd (Chance et al. 2021). This takes place on a 2-D *(x,y)* grid.

Several different constraints arise (1) The predator may be constrained to stay within a geofenced zone (2) the predator may need to avoid obstacles, including the herd, and (3) the predator may need to know that it can reach the prey within a bounded number of steps *before* attempting a pursuit (for example to meet energy or time limitations). What makes the problem challenging and difficult to solve with reward functions alone is the presence of *inertia*, preventing instantaneous acceleration or deceleration, requiring the agent to look ahead an unbounded number of steps.

We conduct experiments with all three constraints, both shielded and unshielded, along with a fourth experiment involving all three constraints. The experiment description is in two parts. For each variant, the shield construction is shown, followed by the experimental results. The agent represents the predator, and is rewarded with +1 for intercepting the prey, and -1 for violating a constraint.

(a) Predator must stay within a geofence

The initial specification is shown in Fig. 2. We have chosen a environment bounded by $L = -10$ on the left, $R = 10$ on the right, $B = 0$ on the bottom, and $T = 10$ on the top. The predator's action (acceleration) is bounded by $[A_{min} = -10, A_{max} = 10]$ in each of 2 physical dimensions. All units are arbitrary. At each time step, the predator observes the position and velocity of itself and the prey. The first iteration of the algorithm as implemented in our prototype tool is shown in Fig. 10 in Appendix 1

The starting invariant is simply the given safety property

$$x >= -10 \wedge x <= 10 \wedge y >= 0 \wedge y <= 10$$

The *wcp* is:

$$\forall x', y', v'_x, v'_y \cdot x' = x + v_x t + a_x t/2 \wedge y' = y + v_y t + a_y t^2/2 \rightarrow x' >= -10 \wedge x' <= 10 \wedge y' >= 0 \wedge y' <= 10$$

(In our experiments, the timestep t is 1). This formula is is in the language of linear arithmetic, either integer or real (LIA or LRA) (De Moura and Bjørner 2011) which admits quantifier elimination (QE) so our Z3-based prototype simplifies it to

$$x + v_x.t + a_x t/2 \geq -10 \ \wedge x + v_x.t + a_x t/2 \geq -10 \ \wedge \ y + v_y t + a_y t^2/2 \geq -10 \ \wedge y + v_y t + a_y t^2/2 \leq 10$$

Next, the updated guard, the guard disjunction, and the the result of quantifier elimination and simplification on the guard disjunction are automatically computed as (Fig. 10 in Appendix 1). Finally, the invariant itself is updated, and as it has changed, the solving process continues to the next iteration. This process continues for three further iterations until there are no more changes to the invariant or any of

Module Predator-Prey-Geofence-Initial
State Vars
$x, y, v_x, v_y : Real$
Control Vars
$a_x, a_y : Real$
Transition
Pred
$x' = x + v_x + a_x/2 \wedge$
$v_x' = v_x + a_x \wedge$
$y' = y + v_y + a_y/2 \wedge$
$v_y' = v_y + a_y$
Control Pred
$A_{MIN} \leq a_x \leq A_{MAX} \wedge$
$A_{MIN} \leq a_y \leq A_{MAX}$
Invariant
$True$
Required Properties
$\Box\ L \leq x \leq R$
$\Box\ B \leq y \leq T$
End Module

Figure 2: Initial Specification of Shield for Geofencing Constraint

Module Predator-Prey-Geofence-Final
State Vars
$x, y, v_x, v_y : Real$
Control Vars
$a_x, a_y : Real$
Transition
Pred
$x' = x + v_x + a_x/2 \wedge$
$v_x' = v_x + a_x \wedge$
$y' = y + v_y + a_y/2 \wedge$
$v_y' = v_y + a_y$
Control Pred
$2y + 4v_y + 3a_y \leq 30 \wedge$
$2x + 4v_x + 3a_x \leq 30 \wedge$
$x + 2v_x + (3/2)a_x \geq -15 \wedge$
$y + 2 * v_y + (3/2)a_y \geq -5$
Invariant
$x \geq -10 \wedge$
$x \leq 10 \wedge$
$y \geq 0 \wedge$
$y \leq 10 \wedge$
$2y + 2v_y \leq 30 \wedge$
$2x + 2v_x \leq 30 \wedge$
$x + v_x \geq -15 \wedge$
$y + v_y \geq -5 \wedge$
$x + 2v_x \geq -30 \wedge$
$2x + 4v_x \leq 60$
Theorems
$\Box\ -10 \leq x \leq 10$
$\Box\ 0 \leq y \leq 10$
End Module

Figure 3: Final Refined Specification of Shield for Geofencing Constrain

the guards, the solver converges to following refined model shown below Fig. 3.

Both the invariant and guard define complex polyhedra in the state and control value space. Since the required properties are enforced by-construction, they are theorems of the model (as can be confirmed with a model checker such as nuXmv (Cavada et al. 2014)). Provided the system starts from an initial state satisfying the invariant, then it is guaranteed that the shield is non-blocking and sound, that is, there is always an action that the agent can make that ensures the constraint is never violated.

Fig. 4 (first row) shows the results of training 16 RL agents to intercept prey within a geofence. Although both the shielded agent and un-shielded one converge to a mean episode return of 1 (the maximum possible value), the shielded agent learns faster (68K timesteps) than the un-shielded agent (152K timesteps). Additionally, the shielded agent never violates the geofence constraint when taking an allowed action, only when it fails to sample an acceptable action after 100 attempts (749 times over 1M steps), but the un-shielded agent collides with the geofence frequently (522K times in 1M steps).

(b) Predator must avoid certain regions

In this experiment, the predator is required to avoid the herd, which could represent an obstacle. In our case, the herd is represented by a rectangular region from $-4 \ldots 4$ on the horizontal axis and $3 \ldots 8$ on the vertical axis. The initial specification is as for *Predator-Prey-Geofence-Initial* except that *Required Properties* are now $x < -4 \wedge x > 4 \wedge x < 3 \wedge x > 8$. Running the prototype tool as before, converges after three iterations to the final spec shown in Fig. 12 in Appendix 1. Figure 4 (second row) shows the results of training an agent on this problem. The shielded and un-shielded agents converge to the maximum episode return, but the shielded agent converges faster (102K steps) than the un-shielded agent (612K steps) and violates the constraints far less frequently (14 vs 3035 times in 1M time steps). Again, the violations by the shielded agent are due to no acceptable action being found even after 100 samples.

(c) Predator Must Reach Prey Within Bounded Number of Steps (Bounded Liveness)

In this experiment, the predator needs to know that it can reach the prey within a bounded number of steps *before* attempting a pursuit (for example due to fuel or time limitations). The required property is $\Diamond^k(x = x_{prey} \wedge y = y_{prey})$ which is is read as the predator position x, y matches the prey position within k timesteps. $\Diamond^k p$ is equivalent to $\bigcirc^0 p \vee \bigcirc^1 p \vee \cdots \vee \bigcirc^k p$ where $\bigcirc^t p$ means p holds in t timesteps. It is possible to handle such *bounded liveness* formulas using a *variant* (see the *Bounded Liveness* section of Appendix 2 for details) in addition to the invariant that's used for safety properties. Since both the prey and predator are moving, in order to limit to a single agent, our solution is to use relative velocities and relative accelerations for the predator. In this way, the prey can be assumed to be stationary. To account for floating point errors, we allow for a

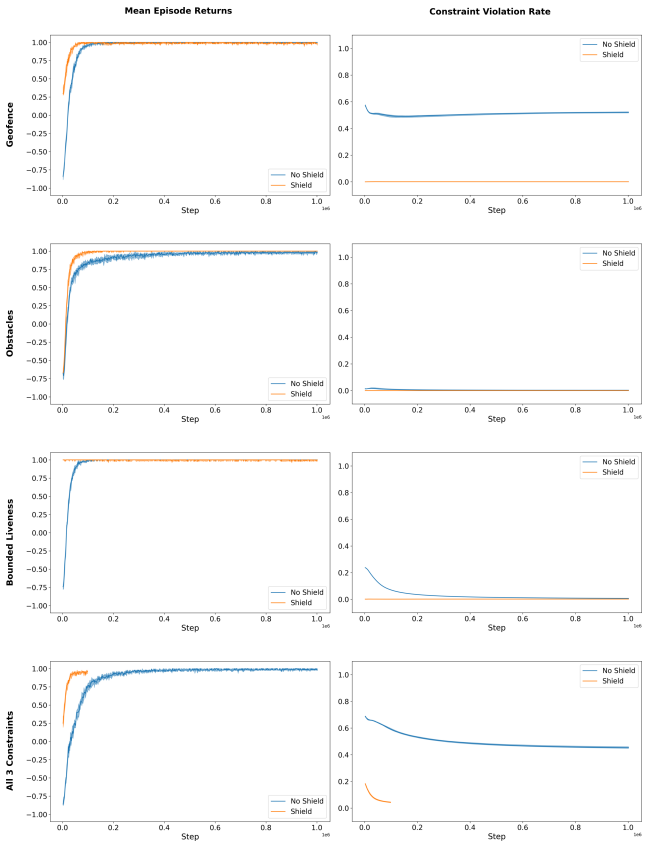


Figure 4: Mean episode returns (possible range: [-1 1]) and constraint violation rate (possible range: [0 1]) for each constraint type for the Predator-Prey task. Lines and shaded regions (barely visible) represent the median and interquartile ranges across 16 trained agents.For the Shielded, All 3 Constraints, early stopping was employed.

"kill" region around the target, so the desired property is: $\Diamond^k(|x - x_{prey}| \leq d \wedge |y - y_{prey}| \leq d)$. We have set d to 1.

The initial specifications is in Fig. 13 in Appendix 1. The shield synthesis converges after four iterations with the final spec shown in Fig. 14 in Appendix 1. Figure 4 (third row) shows the results of training RL agents with this constraint. When using the shield, agents achieve the maximum episode return, even without training while only violating the constraint on random action choices (2348 times in 1M time steps). Without the shield, agents require 96K time steps to reach the maximum episode return and violates the constraints in 7483 of 1M time steps.

(d) Geofencing, Obstacle Avoidance, and Bounded Liveness

The final experiment combines all three constraints. As can be seen from Figure 4(last row), the shielded agent converges much faster (66K versus 432K time steps), although the un-shielded agent actually achieves slight higher median episode returns (0.96 versus 1.0). Additionally, the shielded agent only violates the constraints on random actions (107K times vs 453K times in 1M time steps).

(e) Summary of Predator-Prey Experiments

Across all predator prey experiments, the shielded agents learned faster while never violating constraints except when given random action choices .This is because even though the shield guarantees that an accepted action will not violate the constraints, if the agent was unable to generate a valid action within 100 attempts, a random action is chosen, even if that action violates the constraints. Future work will consider how to better choose actions by using the same information that is used to construct the shield.

Example 2: Water Tank Controller

The second example we consider is from Alshiekh et al (Alshiekh et al. 2018) and illustrates how we handle an adversarial player[1] as well as more complex temporal constraints. The following problem description is taken directly from (Alshiekh et al. 2018):

"We want to learn an energy-efficient controller for a hot water storage tank. Stored water is kept warm by a heater whose energy consumption depends on the filling level of the tank, but we do not know what the exact relationship is. The outflow is always between 0 and 1 liters per second, and the inflow is known to be between 1 and 2 liters per second whenever the valve is open (and it is 0 otherwise). The capacity of the tank is limited to 100 liters, and whenever the inflow is switched on or off, the setting has to be kept for at least three seconds to limit the wear-out of the valve. Also, the tank must never overflow or run dry."

The goal is to synthesize a shield that allows the agent to learn an energy-efficient controller for a hot water storage tank that respects the given constraints. The initial specification is in Fig. 5.

However unlike in the Predator-Prey example, this problem specification has a nested "next" operator ($\bigcirc$). To address this we have devised a transformation that is applied to the specification to remove the operator and reduce the specification to standard form. First we provide some equivalences. Their proof is straightforward so it is omitted.

$$\begin{aligned} \neg \bigcirc \phi &\equiv \bigcirc \neg \phi \\ \bigcirc(\phi_1 \wedge \phi_2) &\equiv \bigcirc\phi_1 \wedge \bigcirc\phi_2 \\ \bigcirc(\phi_1 \vee \phi_2) &\equiv \bigcirc\phi_1 \vee \bigcirc\phi_2 \end{aligned} \tag{1}$$

Now, Let $\mathcal{S} = \langle M, \Phi \rangle$ be a specification containing only positive occurrences of $\bigcirc$. Transform $\mathcal{S}$ as follows:

1. Translate M into Negation Normal Form for LTL (NNF-LTL) (Baier and Katoen 2008) by applying the equivalences in and a standard transformation for conversion to *Conjunctive Normal Form* (CNF). A formula in CNF consists of a conjunction of disjunctive terms (where each arm of the conjunction is known as a *conjunct* and

[1]Referred to as "Env" in the module syntax but not to be confused with the environment in RL which is modeled by the MDP. Our use of the term "Env" follows the more traditional use of the word "environment" in control to represent the adversary and the word "plant" to represent the RL term "environment".

Module Water-Tank-Initial
State Vars
$tank : Real$
Env Vars
$in, out : Real$
Env Pred
$1 \leq in \leq 2 \wedge 0 \leq out \leq 1$
Control Vars
$open : Bool$
Transitions
$OpenAction$
Pred $tank' = tank + in - out$
Control Pred $open$
$CloseAction$
Pred $tank' = tank - out$
Control Pred $\neg open$
Invariant
$True$
Required Properties
$\square\ 0 \leq tank, tank \leq 100$
$\square\ open \wedge \bigcirc\neg open \rightarrow \bigcirc\bigcirc\ \neg open \wedge \bigcirc\bigcirc\bigcirc\neg open$
$\square\ \neg open \wedge \bigcirc open \rightarrow \bigcirc\bigcirc\ open \wedge \bigcirc\bigcirc\bigcirc open$
End Module

Figure 5: Initial specification for Water Tank problem

consists of a disjunction of *literals*. A literal is a positive or negated quantifier free predicate term prefixed by some number (possibly zero) of $\bigcirc$ operators).

2. Distribute the outer $\square$ operator over the conjuncts
3. For each conjunct
   (a) order the terms by increasing number of $\bigcirc$ operators. The resulting conjunct will be of the form $\phi_0 \vee \bigcirc\phi_1 \vee \cdots \vee \bigcirc^n\phi_n$, where any of the disjuncts may be empty
   (b) Add fresh a protected variable $t : Int$ to V, representing a timer
   (c) Add the following to Φ:
      i. $t = 0$
      ii. $\square\,(t = 0\ \wedge \phi_0) \rightarrow t' = 1$
      iii. $\square\,(t = 0\ \wedge \neg\phi_0) \rightarrow t' = 0$
      iv. For every $i, 1 \leq i < n$, add $\square\,(t = i\ \wedge \phi_i) \rightarrow t' = t$
      v. For every $i, 1 \leq i < n$, add $\square\,(t = i\ \wedge \neg\phi_i) \rightarrow t' = t + 1$
      vi. $\square\, t = n \rightarrow \phi_n \wedge t' = 0$

The informal explanation of each of the six constraints in 3(c) is as follows:

- (i) The timer is initialized to 0
- (ii) and (iii) enforce that the timer only starts when the first predicate ϕ_0 holds
- (iv) stays the timer as long as the predicate holds
- (v) moves onto the next timestep once the predicate ϕ_i no longer holds
- (vi) enforces that at least one of the predicates holds at the corresponding timestep

...
Required Properties
$\square\ 0 \leq tank, tank \leq 100$
$\square\ open \wedge \bigcirc\neg open \rightarrow \bigcirc\bigcirc\ \neg open$
$\square\ \neg open \wedge \bigcirc open \rightarrow \bigcirc\bigcirc\ open$
...

Figure 6: Simplified Water Tank problem

Theorem 1. *If $\mathcal{S}'$is the result of applying the above transformation to spec $\mathcal{S}$, then $\mathcal{S} \equiv \mathcal{S}'$*

Proof. See Appendix 2. □

For the purposes of illustration we consider a slightly simpler version of the problem in which the valve is only required to open for two steps after a change (the full worked example is available in the repo). The relevant fragment of the simplified form is shown in 6

The result of applying this transformation to the given initial specification is shown in Fig. 7

(where NUM_NEXTS is 2).

The shield synthesis algorithm converges after 4 iterations with the fully refined specification shown in Fig. 18 in Appendix 2, from which the *solnExists* and *OK* functions can be extracted as described in Appendix 1. Figure 8 show the results of this experiment with an episode length of 100. The shielded agent converges immediately to the maximum possible episode return, where as the un-shielded agents require 84K time steps to converge to a median reward of 6.6. While the shielded agent never violates the constraints, the un-shielded agents violates them 1000 times in 100K time steps.

Conclusion

Across both Water Tank and Predator-Prey, shielding improves safety during training and usually accelerates learning, but its effect on the learned policy depends strongly on how the policy is evaluated. During rollouts, shielded agents consistently avoid safety violations, whereas un-shielded agents incur substantial violations before eventually improving. Although the shield is effective (it correctly rejects actions which cannot allow the agent to reach the target within the given number of steps), there are situations in which the agent fails to find an acceptable action even after 100 attempts, and this is assigned a negative reward. The reason this occurs more frequently than in the previous two cases is that there is often a relatively narrow angle of action choices, even narrower than with the previous two experiments.

Shield synthesis is performed offline and produces symbolic predicates characterizing the admissible state-action pairs. At execution time, the policy proposes an action and the shield checks whether that action satisfies the synthesized predicates in the current state; under the assumed model, any accepted action is therefore guaranteed to preserve the required constraint. The expensive steps of synthesis, including quantifier elimination and fixpoint iteration,

Module Water Tank Transformed

State Vars

$tank : Real$

$s, t : Int$

Env Vars

$in, out : Real$

Env Pred

$1 \leq in \leq 2 \land 0 \leq out \leq 1$

Control Vars

$open : Bool$

Transitions

$OpenAction$

Pred $tank' = tank + in - out$

Control Pred $open$

$CloseAction$

Pred $tank' = tank - out$

Control Pred $\neg open$

Invariant

$True$

Required Properties

$\square\ 0 \leq tank, tank \leq 100$

$//open \land \bigcirc \neg open \rightarrow \bigcirc \bigcirc \neg open$

$\square\ 0 \leq t, t \leq$ NUM_NEXTS

$\square\ t = 0 \land open \rightarrow t' = 1$

$\square\ t = 0 \land \neg open \rightarrow t' = 0$

$\square\ t = 1 \land \neg open \rightarrow t' = 2$

$\square\ t = 1 \land open \rightarrow t' = 1$

$\square\ t = 2 \rightarrow \neg open \land t' = 0$

$//\neg open \land \bigcirc open \rightarrow \bigcirc \bigcirc open$

$\square\ 0 \leq s, s \leq$ NUM_NEXTS

$\square\ s = 0 \land \neg open \rightarrow s' = 1$

$\square\ s = 0 \land open \rightarrow s' = 0$

$\square\ s = 1 \land open \rightarrow s' = 2$

$\square\ s = 1 \land \neg open \rightarrow s' = 1$

$\square\ s = 2 \rightarrow open \land s' = 0$

End Module

Figure 7: Transformed specification for Water Tank problem

occur only once offline rather than during deployment. In practice, the main runtime challenge is not enforcing the shield but finding a good admissible action when the feasible region is narrow, as in the bounded-liveness experiments where repeated random sampling may fail to hit the safe set.

Although the synthesized predicates are generated symbolically, they admit a straightforward operational interpretation. In the geofencing case, the additional linear inequalities act like braking-distance constraints induced by inertia: near the boundary, admissibility depends not only on current position but also on current velocity and on whether some acceleration still keeps the next state inside the fence. In the bounded-liveness case, the same lookahead mechanism becomes progressively more selective: as the remaining horizon decreases, the admissible action set contracts to those controls that still preserve a feasible intercept trajectory. This helps explain the empirical results: bounded

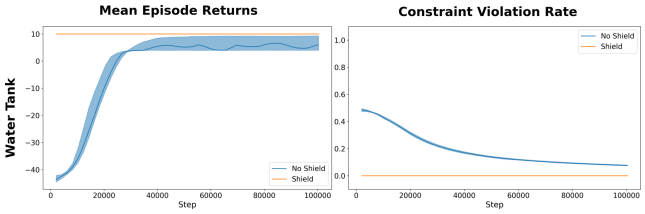


Figure 8: Mean episode returns (maximum possible: 10) and constraint violation rate (possible range: [0 1]) for each constraint type for the water tank task. Lines and shaded regions represent the median and interquartile ranges across 16 trained agents.

liveness is more restrictive than pure geofencing or obstacle avoidance, so random sampling is less likely to find an admissible action even though every accepted action remains safe.

Related Work

Related work on safe reinforcement learning spans shielding, invariant-based methods, reward shaping, and safe exploration; see (García and Fernández 2015) for a broad survey. Our work is is inspired by the pioneering work of Alsheikh et al, (Alshiekh et al. 2018), who introduced shielding for reinforcement learning with formal safety guarantees and minimal interference, but their construction is automata-based and, for continuous systems, relies on a finite abstraction of the state space. By contrast, our method uses constraint propagation and SMT solving to synthesize shields symbolically, avoiding explicit state-space abstraction and directly representing the set of admissible actions.

Several recent approaches also address continuous state or action spaces, but with different tradeoffs. (Dalal et al. 2018) learns approximate dynamics and minimally corrects unsafe actions by solving a quadratic program; however, it does not support infinite lookahead and assumes limited simultaneous constraint interaction. (Anderson et al. 2020) iteratively refines a neurosymbolic policy with a neural controller and a discrete shield, but relies on invariant construction during deployment. (Zhu et al. 2019) also uses invariant synthesis, searching within user-provided sketches, but does not explicitly represent multiple admissible actions. In contrast, we synthesize a shield offline and compute the full safe action set, which is useful when safety depends on choosing among multiple feasible actions.

Other lines of work relax safety during learning rather than enforcing it formally. Empirical rule-learning methods such as (Shperberg et al. 2022) accumulate safety rules from observed failures, while curriculum-style penalty methods (Shperberg, Liu, and Stone 2023) gradually increase the cost of violations to encourage early exploration. Reward shaping is another common approach, but can be difficult to tune under multiple interacting constraints and can affect optimization in unintended ways (Shalev-Shwartz, Shammah, and Shashua 2016; Ng, Harada, and Russell 1999). These methods can be effective in practice, but they generally do not provide the hard guarantees targeted here.

Control Barrier Functions (CBFs) (Ames et al. 2019, 2017) are also closely related: they enforce safety in continuous control by solving an online optimization problem that minimally modifies the controller's action. CBFs are powerful for smooth continuous systems, but they are aimed at state-safety constraints and do not naturally capture temporal properties such as bounded reachability. Our approach instead synthesizes a symbolic shield offline for discrete systems and continuous systems with linear dynamics, enabling runtime enforcement through membership checks rather than online optimization.

Further Work

Future work follows directly from the current formulation. A first direction is to make better use of the information already produced by shield synthesis. At present, the policy receives only a binary admissibility signal, although the shield computes a richer description of the safe action set; in the linear case, this set is a polyhedron. Incorporating that structure into learning could provide more informative guidance, especially when the admissible region is narrow, and may connect naturally to constrained policy optimization methods (Achiam et al. 2017). A second direction is to extend the framework to multi-agent settings, where safety depends on the interaction between agents and the environment. This requires moving from the current single-agent constraint structure to one that explicitly captures reactive choices through alternating quantification over environment and agent actions.

References

Abe, N.; Melville, P.; Pendus, C.; Reddy, C. K.; Jensen, D. L.; Thomas, V. P.; Bennett, J. J.; Anderson, G. F.; Cooley, B. R.; Kowalczyk, M.; Domick, M.; and Gardinier, T. 2010. Optimizing debt collections using constrained reinforcement learning. In *Proc.the 16th ACM SIGKDD Intl Conference on Knowledge Discovery and Data Mining*, KDD '10, 75—-84. ISBN 9781450300551.

Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained Policy Optimization. In *Proc. of the 34th International Conference on Machine Learning*, volume 70, 22–31.

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe reinforcement learning via shielding. AAAI'18/IAAI'18/EAAI'18.

Ames, A. D.; Coogan, S.; Egerstedt, M.; Notomista, G.; Sreenath, K.; and Tabuada, P. 2019. Control Barrier Functions: Theory and Applications. In *2019 18th European Control Conference (ECC)*, 3420–3431.

Ames, A. D.; Xu, X.; Grizzle, J. W.; and Tabuada, P. 2017. Control Barrier Function Based Quadratic Programs for Safety Critical Systems. *IEEE Transactions on Automatic Control*, 62(8): 3861–3876.

Anderson, G.; Verma, A.; Dillig, I.; and Chaudhuri, S. 2020. Neurosymbolic reinforcement learning with formally verified exploration. In *Proc. of the 34th International Conference on Neural Information Processing Systems*, NIPS '20.

Baier, C.; and Katoen, J. 2008. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press.

Banerjee, C.; Nguyen, K.; Fookes, C.; and Raissi, M. 2025. A survey on physics informed reinforcement learning: Review and open problems. *Expert Systems with Applications*, 287: 128166.

Cavada, R.; Cimatti, A.; Dorigatti, M.; Griggio, A.; Mariotti, A.; Micheli, A.; Mover, S.; Roveri, M.; and Tonetta, S. 2014. The nuXmv Symbolic Model Checker. In *Computer Aided Verification (CAV)*, volume 8559 of *LNCS*, 334–342.

Chance, F. S.; Little, C.; McKenzie, M. M.; Dellana, R. A.; Small, D. E.; Gayle, T. R.; and Novick, D. K. 2021. Biologically Inspired Interception on an Unmanned System. Technical Report SAND2021-13549R, Sandia National Laboratories.

Cousot, P.; and Cousot, R. 1977. Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, 238–252. ACM.

Dalal, G.; Dvijotham, K.; Vecerik, M.; Hester, T.; Paduraru, C.; and Tassa, Y. 2018. Safe Exploration in Continuous Action Spaces.

De Moura, L.; and Bjørner, N. 2008. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, 337–340. Springer.

De Moura, L.; and Bjørner, N. 2011. Satisfiability modulo theories: introduction and applications. *Commun. ACM*, 54(9): 69–77.

García, J.; and Fernández, F. 2015. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research*, 16: 1437–1480.

Giacobbe, M.; Hasanbeig, M.; Kroening, D.; and Wijk, H. 2021. Shielding Atari Games with Bounded Prescience. In *Proc. of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 1507—1509.

Jia, Y.; Burden, J.; Lawton, T.; and Habli, I. 2020. Safe Reinforcement Learning for Sepsis Treatment. In *2020 IEEE Intl Conf. on Healthcare Informatics (ICHI)*, 1–7.

Lamport, L. 2002. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley.

Ng, A. Y.; Harada, D.; and Russell, S. J. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML 99, 278 – 287.

Rehof, J.; and Mogensen, T. 1999. Tractable Constraints in Finite Semilattices. *Science of Computer Programming*, 35: 191–221.

Shalev-Shwartz, S.; Shammah, S.; and Shashua, A. 2016. Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving.

Shperberg, S. S.; Liu, B.; Allievi, A.; and Stone, P. 2022. A Rule-based Shield: Accumulating Safety Rules from Catastrophic Action Effects. In *Proceedings of The 1st Conference on Lifelong Learning Agents*, volume 199 of *Proc. of Machine Learning Research*, 231–242.

Shperberg, S. S.; Liu, B.; and Stone, P. 2023. Relaxed Exploration Constrained Reinforcement Learning. In *Proc. of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, 2821–2823.

Smith, D.; and Nedunuri, S. 2021. Model Refinement. Tech. Rep. 21.0, Kestrel Institute.

Smith, D.; and Nedunuri, S. 2024. Deductive Model Refinement. In *16th Symp.*, 147–165.

Zhu, H.; Xiong, Z.; Magill, S.; and Jagannathan, S. 2019. An inductive synthesis framework for verifiable reinforcement learning. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, 686–701.

Appendix 1

(a) Geofencing

Figs. 9 and 11 show the initial and refined specifications resp.

Module Predator-Prey-Geofence-Initial
State Vars
$x, y, v_x, v_y : Real$
Control Vars
$a_x, a_y : Real$
Transition
Pred
$x' = x + v_x + a_x/2 \wedge$
$v_x' = v_x + a_x \wedge$
$y' = y + v_y + a_y/2 \wedge$
$v_y' = v_y + a_y$
Control Pred
$A_{MIN} \leq a_x \leq A_{MAX} \wedge A_{MIN} \leq a_y \leq A_{MAX}$
Invariant
$True$
Required Properties
$\Box\ L \leq x \leq R$
$\Box\ B \leq y \leq T$
End Module

Figure 9: Initial Specification of Shield for Geofencing Constraint

(b) Obstacle avoidance

Fig. 12 shows the final refined specification for obstacle avoidance.

(c) Bounded Liveness

The figures below show the initial spec and the final spec after the refinement.

```
Starting invariant on node main:
    And(x \geq -10, x \leq 10, y \geq 0, y \leq 10)
-----------
iteration 0
-- Refining the state invariant for node main
wcp:
 ForAll([xX, yX, v_xX, v_yX],
       Implies(And(xX == x + v_x*1 + ((a_x*1)/2)*1,
                   v_xX == v_x + a_x*1,
                   yX == y + v_y*1 + ((a_y*1)/2)*1,
                   v_yX == v_y + a_y*1),
                And(xX \geq -10,
                    xX \leq 10,
                    yX \geq 0,
                    yX \leq 10)))
wcp after QE (wcp_qe):
And(x + v_x + 1/2*a_x \geq -10,
    x + v_x + 1/2*a_x \leq 10,
    y + v_y + 1/2*a_y \geq 0,
    y + v_y + 1/2*a_y \leq 10)
new guard:
And(a_x \geq -10,
    a_x \leq 10,
    a_y \geq -10,
    a_y \leq 10,
    x + v_x + 1/2*a_x \geq -10,
    x + v_x + 1/2*a_x \leq 10,
    y + v_y + 1/2*a_y \geq 0,
    y + v_y + 1/2*a_y \leq 10)
guardDisjunction:
Or(False,
   Exists([a_x, a_y],
          And(a_x \geq -10,
              a_x \leq 10,
              a_y \geq -10,
              a_y \leq 10,
              x + v_x + 1/2*a_x \geq -10,
              x + v_x + 1/2*a_x \leq 10,
              y + v_y + 1/2*a_y \geq 0,
              y + v_y + 1/2*a_y \leq 10)))
guardDisjunction after QE:
Or(False,
    And(2*y + 2*v_y \leq 30,
        2*x + 2*v_x \leq 30,
        And(2*x + 2*v_x \leq 30,
            -1*x + -1*v_x \leq 15,
            2*y + 2*v_y \leq 30,
            -1*y + -1*v_y \leq 5)))
simplified guardDisjunction:
 And(2*y + 2*v_y \leq 30,
    2*x + 2*v_x \leq 30,
    -1*x + -1*v_x \leq 15,
    -1*y + -1*v_y \leq 5)
Updated invariant:
 And(x \geq -10,
    x \leq 10,
    y \geq 0,
    y \leq 10,
    2*y + 2*v_y \leq 30,
    2*x + 2*v_x \leq 30,
    -1*x + -1*v_x \leq 15,
    -1*y + -1*v_y \leq 5)
----------------
iteration 1
...
```

Figure 10: First iteration of the refinement algorithm

Module Predator-Prey-Geofence-Final
State Vars
$x, y, v_x, v_y : Real$
Control Vars
$a_x, a_y : Real$
Transition
Pred
$x' = x + v_x + a_x/2 \wedge$
$v_x' = v_x + a_x \wedge$
$y' = y + v_y + a_y/2 \wedge$
$v_y' = v_y + a_y$
Control Pred
$2y + 4v_y + 3a_y \leq 30 \wedge$
$2x + 4v_x + 3a_x \leq 30 \wedge$
$x + 2v_x + (3/2)a_x \geq -15 \wedge$
$y + 2 * v_y + (3/2)a_y \geq -5$
Invariant
$x \geq -10 \wedge$
$x \leq 10 \wedge$
$y \geq 0 \wedge$
$y \leq 10 \wedge$
$2y + 2v_y \leq 30 \wedge$
$2x + 2v_x \leq 30 \wedge$
$x + v_x \geq -15 \wedge$
$y + v_y \geq -5 \wedge$
$x + 2v_x \geq -30 \wedge$
$2x + 4v_x \leq 60$
Theorems
$\Box\ -10 \leq x \leq 10$
$\Box\ 0 \leq y \leq 10$
End Module

Figure 11: Final Refined Specification of Shield for Geofencing Constraint

Module Predator-Prey-Obstacle-Final
State Vars
$x, y, v_x, v_y : Real$
Control Vars
$a_x, a_y : Real$
Transition
Pred
$x' = x + v_x + a_x/2 \wedge$
$v_x' = v_x + a_x \wedge$
$y' = y + v_y + a_y/2 \wedge$
$v_y' = v_y + a_y$
Control Pred
$x + v_x + a_x/2 > 4 \vee$
$y + v_y + a_y/2 > 8 \vee$
$x + v_x + a_x/2 < -4 \vee$
$y + v_y + a_y/2 < 3$
Invariant
$x < -4 \vee$
$x > 4 \vee$
$y > 3 \vee$
$y > 8$
End Module

Figure 12: Final Refined Specification for Obstacle Avoidance Constraint

Module Predator-Prey-Bounded-Liveness-Initial
State Vars
$x, y, v_x, v_y : Real$
Control Vars
$a_x, a_y : Real$
Transition
Pred
$x' = x + v_x + a_x/2 \wedge$
$v_x' = v_x + a_x \wedge$
$y' = y + v_y + a_y/2 \wedge$
$v_y' = v_y + a_y$
Control Pred
$A_{MIN} \leq a_x \leq A_{MAX} \wedge A_{MIN} \leq a_y \leq A_{MAX}$
Variant
$True$
Required Properties
$|x - x_{prey}| \leq d \wedge |y - y_{prey}| \leq d$
End Module

Figure 13: Initial Specification for Bounded Liveness Requirement

Module Predator-Prey-Bounded-Liveness-Final

State Vars

$x, y, v_x, v_y : Real$

Control Vars

$a_x, a_y : Real$

Transition

Pred

$$
\begin{aligned}
& x' = x + v_x + a_x/2 \wedge \\
& v_x' = v_x + a_x \wedge \\
& y' = y + v_y + a_y/2 \wedge \\
& v_y' = v_y + a_y
\end{aligned}
$$

Control Pred

$$
\begin{aligned}
& (t = 1 \rightarrow y + v_y + (1/2)a_y = y_{prey} \wedge \\
& \quad x + v_x + (1/2)a_x = x_{prey}) \wedge \\
& (t = 2 \rightarrow -x + -2v_x + -(3/2)a_x + x_{prey} \leq 5 \wedge \\
& \quad x + 2v_x + (3/2)a_x + -x_{prey} \leq 5 \wedge \\
& \quad y + 2v_y + (3/2)a_y + -y_{prey} \leq 5 \wedge \\
& \quad -y + -2v_y + -(3/2)a_y + y_{prey} \leq 5) \wedge \\
& (t = 3 \rightarrow (2y + 6v_y + 5a_y - 2y_{prey} \leq 40 \wedge \\
& \quad 2x + 6v_x + 5a_x - 2x_{prey} \leq 40 \wedge \\
& \quad -y + -3v_y + -(5/2)a_y + y_{prey} \leq 20 \wedge \\
& \quad -x + -3v_x + -(5/2)a_x + x_{prey} \leq 20)
\end{aligned}
$$

Variant

$$
\begin{aligned}
& (x = x_{prey} \wedge y = y_{prey}) \\
\vee \quad & (-x + -v_x + x_{prey} \leq 5 \wedge \\
& \quad y + v_y + -y_{prey} \leq 5 \wedge \\
& \quad x + v_x + -x_{prey} \leq 5 \wedge \\
& \quad -y + -v_y + y_{prey} \leq 5) \\
\vee \quad & (2x + 4v_x + -2x_{prey} \leq 40 \wedge \\
& \quad 2y + 4v_y + -2y_{prey} \leq 40 \wedge \\
& \quad -y + -2v_y + y_{prey} \leq 20 \wedge \\
& \quad -x + -2v_x + x_{prey} \leq 20) \\
\vee \quad & ((2/5)x + (6/5)v_x + -(2/5)x_{prey} \leq 18 \wedge \\
& \quad -y + -3v_y + y_{prey} \leq 45 \wedge \\
& \quad (2/5)y + (6/5)v_y + -(2/5)y_{prey} \leq 18 \wedge \\
& \quad -x + -3v_x + x_{prey} \leq 45)
\end{aligned}
$$

Theorems

$$|x - x_{prey}| \leq d \wedge |y - y_{prey}| \leq d$$

End Module

Figure 14: Final Specification for Bounded Liveness Requirement

Appendix 2

Specifications and Refinement.

A *specification* $\mathcal{S} = \langle \mathcal{M}, \Phi \rangle$ is comprised of a labeled transition system (LTS) $\mathcal{M}$ and a set of required properties Φ. A system specification denotes the set of traces generable by $\mathcal{M}$ that also satisfy all properties in Φ. That is, the meaning of S, denoted $[\![S]\!]$is:

$$[\![\mathcal{S}]\!] = \{t \mid t \in \tau(\mathcal{M}) \wedge t \models \Phi\}.$$

There is a preorder relation on specifications called *refinement*: $\mathcal{S} \sqsubseteq \mathcal{S}'$ if $[\![\mathcal{S}]\!] \supseteq [\![\mathcal{S}']\!]$. Also, $\mathcal{S}$ refines to $\mathcal{S}'$ or $\mathcal{S}'$ is a refinement of $\mathcal{S}$.

Given nodes $\mathcal{M}, \mathcal{M}'$, $\mathcal{M}' \sqsubseteq \mathcal{M}$ if every trace of $\mathcal{M}'$ is a trace of $\mathcal{M}$, i.e. $\tau(\mathcal{M}') \subseteq \tau(\mathcal{M})$

Trace inclusion between nodes is characterized using simulation relations, represented as simulation maps.

Given specifications $\mathcal{S} = \langle \mathcal{M}, \Phi_1 \rangle$ and $\mathcal{S}' = \langle \mathcal{M}', \Phi_2 \rangle$, if $\mathcal{M} \sqsubseteq \mathcal{M}'$ and $\Phi_2 \implies \Phi_1$ then $\mathcal{S} \sqsubseteq \mathcal{S}'$ (See (Smith and Nedunuri 2021) for the proof).

Given specification $\mathcal{S} = \langle \mathcal{M}, \Phi \rangle$, the goal of *property enforcing model refinement* (PEMR) is to transform $\mathcal{S}$ to a new specification $\mathcal{S}' = \langle \mathcal{M}', \{\} \rangle$ that refines $\mathcal{S}$. Operationally speaking, the refinement is carried out by strengthening the guards on arcs in $\mathcal{M}$. The intent is that $\mathcal{M}'$ is the least refinement of $\mathcal{M}$ that satisfies all properties in Φ. A key part of PEMR is the elimination of blocking states in the model.

Synthesis as Constraint Solving

PEMR transforms a model $\mathcal{M}$ and required properties Φ into a model $\mathcal{M}'$ such that $\mathcal{M} \sqsubseteq \mathcal{M}' \wedge \mathcal{M}' \models \Phi$. A constraint system whose solutions correspond to refinements of $\mathcal{M}$ that satisfy Φ can now be defined. The intent is to find the greatest solution of the constraint system, which corresponds to the minimal refinement of $\mathcal{M}$ that satisfies Φ. In some cases, it may be necessary to settle for a near-greatest solution instead. In formulating model refinement as a constraint satisfaction problem, we treat the node labels L_m and arc labels L_a as variables, whose assigned values are state and action predicates, respectively. We can view the constraint system as taking place in the Boolean lattice of formulas with implication as the partial order (i.e. a Tarski-Lindenbaum algebra). Each constraint provides an upper bound on feasible values of one variable. A feasible solution to the constraint system is an assignment of formulas to each variable that satisfies all the constraints of the system. We discuss below how to assure finite convergence of the constraint solving process as the lattice may be of infinite height. To simplify the presentation, in this paper we consider only a single node (but multiple state) systems and assume linear arithmetic (either real or integer) as the underlying theory. The work generalizes to the multi-node case.

Constraint Generation

Given a *transition system* as defined earlier in the main section of this paper, assume a system dynamics given by a function f_a on an arc a, which takes the current state s and control value u and returns a new state. A state label is a state invariant L (to be found) characterizing the safe states. Define a predicate, the *weakest controllable predecessor* (wcp) as follows:

$$wcp \;\equiv\; \forall e.\, e \in E(s) \rightarrow L(f_a(s, e, u)) \tag{2}$$

That is, wcp is the weakest formula over $V \bigcup \{u\}$ characterizing those states from which the controller for a given control value u can force the system into a safe state (one that satisfies the invariant) regardless of the adversary choice e (constrained by some predicate E).[2]). Then, the control predicate $U(s, u)$ on any given arc must ensure the wcp:

$$U(s, u) \rightarrow wcp \tag{3}$$

Finally, to close the loop, the invariant L characterizes those states from which for some arc a there is a control value that satisfies the control predicate on the arc

$$L(s) \rightarrow \bigvee_a \exists u \cdot U(s, u) \tag{4}$$

Thus PEMR can be characterized by a two-stage process. The first stage creates non-temporal constraints and the second stage creates general behavioral constraints.

1. Stage 0. Let S be a solution to the constraint-satisfaction problem posed by the conjunction of static (non-temporal) properties. For each variable $v \in dom(S)$, set the initial value of v to $S(v)$.

2. Stage 1. Generate the following constraints for each required temporal property $\Box\phi$:

1. **Node Localization:** $L \rightarrow \phi$ if ϕ is a state predicate expressed over the variables at m;
2. **Arc Localization:** $L_a \rightarrow \phi$ for the arc $a = \langle m, m \rangle \in A$ if ϕ is an action expressed over the variables at m;
3. **Behavioral**: (3) and (4) above

The Node and Arc Localization constraints provide upper bounds on the invariants. The Behavioral constraints are the essentially synthetic aspect of model refinement. Constraints (3) serve to eliminate any blocking states at a node with respect to an outgoing arc. Constraints (4) serve to eliminate blocking states at a node with respect to all of its outgoing arcs. Given a specification $\mathcal{S} = \langle M, \Phi \rangle$, PEMR first refines $\mathcal{S}$ by solving the stage 0 constraint problem to initialize state variables, then further refines the specification by solving the stage 1 constraints.

[2] In (Alshiekh et al. 2018), the environment, which is an abstraction of the MDP, has chosen the current state (which is either the starting state or the result of the previous action), and the system chooses the action. Both players transition simultaneously to a new pair (game) state, with the new environment state determined by the MDP and the new system state driven by the specification automaton. The constraint as stated here achieves a similar effect. In contrast with the automata based formulation, it can be easily reformulated to allow for the agent to respond to the environment input.

Bounded Liveness (Un-nested Next Operator) These are properties that take the form of "do X within t steps." Now, in addition to the invariant L above there is a variant V which is initialized to the target property just as L is initialized to the safety property: Similar to earlier,

$$wcp \equiv \forall e.\, e \in E(s) \rightarrow V(f_a(s,e,u))$$

where U is bounded from above as before and

$$V(s) \rightarrow \bigvee_a \exists u \cdot U(s,u)$$

Bounded Liveness (Nested Next Operator) Theorem 1. *If $\mathcal{S}'$is the result of applying the above transformation to spec $\mathcal{S}$, then $\mathcal{S} \equiv \mathcal{S}'$*

Proof. Since the transformation adds no new nodes or arcs, a simulation map will be the obvious bijection between the old and new nodes and arcs. First we show that $\mathcal{S} \sqsubseteq S'$. To apply Theorem , it remains to show that $\Phi' \Rightarrow \Phi$. Let σ be a trace of $\mathcal{S}'$. First consider what happens at at the start of the trace. At least one of the ϕ_i must hold (as enforced by constraint 3c(v)). Since i is only incremented (by 1) if ϕ_i does not hold, this means that when ϕ_i holds it holds after i timesteps. This is equivalent to $\bigcirc^i \phi_i$. Therefore the trace is also in $\mathcal{S}$. In order that the shield does not unnecessarily constrain the agent, we would also like that $\mathcal{S}' \sqsubseteq S$, namely every trace in $\mathcal{S}$ is also in $\mathcal{S}'$. This follows from the fact that t is reset whenever a literal holds, therefore $t = n$ is only reached if all $\phi_{j<i}$ were false. Thus only one literal needs to hold. Finally, the $\Box$ operator requires that this proof holds for every timestep. However, we do not need an infinite number of timers as every valid trace can be checked in at most n steps, a timer is reset within n steps, and can be reused, ensuring that no more than n timers are needed. The only requirement is that the timers be staggered, that is the$(j+1)$th timer is started in the step after the jth timer has started. To support this, fresh protected variables $t^1, t^2, \ldots, t^n : Int$ are added to V. $\Box$

Example: Water Tank Problem The initial specification is shown in Fig. 15.

For the purposes of illustration we consider a slightly simpler version of the problem in which the valve is only required to open for two steps after a change (the full worked example is available in the repo). The relevant fragment of the simplified form is in Fig. 16.

The transformed specification is in Fig. 17 (where NUM_NEXTS is 2):

This can now be passed to the refinement engine which produces the refined specification split over Figs. 18 and 19.

Module Water-Tank-Initial
- **State Vars**
  - $tank : Real$
- **Env Vars**
  - $in, out : Real$
- **Env Pred**
  - $1 \leq in \leq 2 \wedge 0 \leq out \leq 1$
- **Control Vars**
  - $open : Bool$
- **Transitions**
  - $OpenAction$
    - **Pred** $tank' = tank + in - out$
    - **Control Pred** $open$
  - $CloseAction$
    - **Pred** $tank' = tank - out$
    - **Control Pred** $\neg open$
- **Invariant**
  - $True$
- **Required Properties**
  - $\Box\ 0 \leq tank, tank \leq 100$
  - $\Box\ open \wedge \bigcirc\neg open \rightarrow \bigcirc\bigcirc\ \neg open \wedge \bigcirc\bigcirc\bigcirc\neg open$
  - $\Box\ \neg open \wedge \bigcirc open \rightarrow \bigcirc\bigcirc\ open \wedge \bigcirc\bigcirc\bigcirc open$

End Module

Figure 15: Initial specification for Water Tank problem

...
Required Properties
- $\Box\ 0 \leq tank, tank \leq 100$
- $\Box\ open \wedge \bigcirc\neg open \rightarrow \bigcirc\bigcirc\ \neg open$
- $\Box\ \neg open \wedge \bigcirc open \rightarrow \bigcirc\bigcirc\ open$

...

Figure 16: Simplified Water Tank problem

Module Water Tank Transformed
State Vars
$tank : Real$
$s, t : Int$
Env Vars
$in, out : Real$
Env Pred
$1 \leq in \leq 2 \wedge 0 \leq out \leq 1$
Control Vars
$open : Bool$
Transitions
$OpenAction$
Pred $tank' = tank + in - out$
Control Pred $open$
$CloseAction$
Pred $tank' = tank - out$
Control Pred $\neg open$
Invariant
$True$
Required Properties
$\square\ 0 \leq tank, tank \leq 100$

//$open \wedge \bigcirc \neg open \rightarrow \bigcirc \bigcirc \neg open$
$\square\ 0 \leq t, t \leq$ NUM_NEXTS
$\square\ t = 0 \wedge open \rightarrow t' = 1$
$\square\ t = 0 \wedge \neg open \rightarrow t' = 0$
$\square\ t = 1 \wedge \neg open \rightarrow t' = 2$
$\square\ t = 1 \wedge open \rightarrow t' = 1$
$\square\ t = 2 \rightarrow \neg open \wedge t' = 0$

//$\neg open \wedge \bigcirc open \rightarrow \bigcirc \bigcirc open$
$\square\ 0 \leq s, s \leq$ NUM_NEXTS
$\square\ s = 0 \wedge \neg open \rightarrow s' = 1$
$\square\ s = 0 \wedge open \rightarrow s' = 0$
$\square\ s = 1 \wedge open \rightarrow s' = 2$
$\square\ s = 1 \wedge \neg open \rightarrow s' = 1$
$\square\ s = 2 \rightarrow open \wedge s' = 0$
End Module

Figure 17: Transformed specification for Water Tank problem

Module Water Tank Final
State Vars
$tank : Real$
$s, t : Int$
Env Vars
$in, out : Real$
Env Pred
$1 \leq in \leq 2 \wedge 0 \leq out \leq 1$
Control Vars
$open : Bool$
Transitions
$OpenAction$
Pred
$tank' = tank + in - out \wedge$
$t = 0 \rightarrow t' = 1 \wedge$
$t = 1 \rightarrow t' = 1 \wedge$
$s = 0 \rightarrow s' = 0 \wedge$
$s = 1 \rightarrow s' = 2 \wedge$
$s = 2 \rightarrow s' = 0$
Control Pred
$open \wedge$
$t \neq 2 \wedge$
$\big((t = 1 \wedge s = 1 \wedge \neg(1 \leq tank)) \vee$
$(t = 1 \wedge s = 1)\ \vee (s = 1 \wedge t = 0) \vee$
$(s = 0 \wedge t = 1 \wedge tank \leq 98) \vee$
$(t = s \wedge s = 0 \wedge tank \leq 98) \vee$
$(s = 2 \wedge t = 0)\ \vee (s = 2 \wedge t = 1)\big) \wedge$
$\big((t = 0 \wedge s = 1 \wedge tank \leq 96) \vee$
$(t = 0 \wedge s = 2) \vee$
$(t = 1 \wedge s = 2) \vee$
$(t = 0 \wedge s = 0) \vee$
$(t = s \wedge s = 1 \wedge tank \leq 96) \vee$
$(t = 1 \wedge s = 0)\big) \wedge$
$\big((t = 1 \wedge s = 0 \wedge tank \leq 98) \vee$
$(s = 0 \wedge t = s \wedge tank \leq 98) \vee$
$(t = 0 \wedge s = 2)\ \vee (t = 1 \wedge s = 2) \vee$
$(s = 1 \wedge t = 1 \wedge \neg(1 \leq tank)) \vee$
$(s = 1 \wedge t = 1)\ \vee (t = 0 \wedge s = 1)\big) \wedge$
$\big((s = 0 \wedge t = 0) \vee$
$(s = 1 \wedge t = s \wedge tank \leq 96) \vee$
$(s = 1 \wedge t = 0 \wedge tank \leq 96) \vee$
$(s = 2 \wedge t = 0) \vee$
$(s = 0 \wedge t = 1) \vee$
$(s = 2 \wedge t = 1)\big)$
$CloseAction$
Pred
$tank' = tank - out \wedge$
$s = 0 \rightarrow s' = 1 \wedge$
$s = 1 \rightarrow s' = 1 \wedge$
$t = 0 \rightarrow t' = 0 \wedge$
$t = 1 \rightarrow t' = 2 \wedge$
$t = 2 \rightarrow t' = 0$
...

Figure 18: Final refined specification of Water Tank problem (1st part)

... **Control Pred**

$\neg open \wedge$
$s \neq 2 \wedge$
$\big((t = 1 \wedge s = 1) \vee$
$(t = 2 \wedge s = 1) \vee$
$(t = s \wedge s = 0 \wedge 1 \leq tank) \vee$
$(s = 1 \wedge t = 0 \wedge 1 \leq tank) \vee$
$(t = 2 \wedge s = 0) \vee$
$(s = 0 \wedge t = 1)\big) \wedge$
$\big((t = s \wedge s = 0) \vee$
$(t = 1 \wedge s = 1 \wedge 2 \leq tank) \vee$
$(s = 1 \wedge t = 0) \vee$
$(t = 2 \wedge s = 1) \vee$
$(t = 1 \wedge s = 0 \wedge 2 \leq tank) \vee$
$(t = 2 \wedge s = 0)\big) \wedge$
$\big((s = 1 \wedge t = 2) \vee$
$(t = 0 \wedge s = 1 \wedge 1 \leq tank) \vee$
$(s = 0 \wedge t = 2) \vee$
$(s = 0 \wedge t = s \wedge 1 \leq tank) \vee$
$(s = 1 \wedge t = 1) \vee$
$(t = 1 \wedge s = 0)\big) \wedge$
$\big((s = 1 \wedge t = 2) \vee$
$(s = 0 \wedge t = 2) \vee$
$(s = 0 \wedge t = s) \vee$
$(t = 0 \wedge s = 1) \vee$
$(s = 0 \wedge t = 1 \wedge 2 \leq tank) \vee$
$(s = 1 \wedge t = 1 \wedge 2 \leq tank)\big)$

Invariant

$0 \leq tank \leq 100 \wedge$
$\big((t = 0 \wedge s = 2 \wedge tank \leq 98) \vee$
$(s = 1 \wedge t = 1 \wedge tank \leq 98) \vee$
$(t = 1 \wedge s = 0 \wedge tank \leq 98) \vee$
$(s = 0 \wedge t = s \wedge tank \leq 98) \vee$
$(t = 0 \wedge s = 1 \wedge tank \leq 98) \vee$
$(s = 1 \wedge t = 2 \wedge 1 \leq tank) \vee$
$(t = 1 \wedge s = 0 \wedge 1 \leq tank) \vee$
$(s = 1 \wedge t = 1 \wedge \neg(1 \leq tank)) \vee$
$(s = 0 \wedge t = 2 \wedge 1 \leq tank) \vee$
$(t = 1 \wedge s = 2 \wedge tank \leq 98) \vee$
$(t = 0 \wedge s = 1 \wedge 1 \leq tank) \vee$
$(s = 1 \wedge t = 1 \wedge 1 \leq tank) \vee$
$(s = 0 \wedge t = s \wedge 1 \leq tank)\big)$

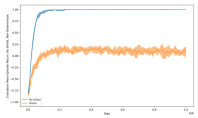
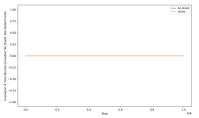
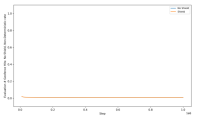
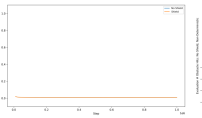
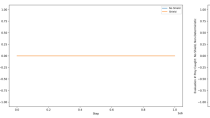
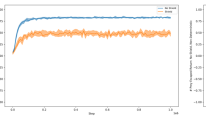
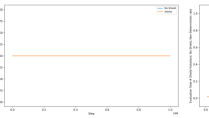
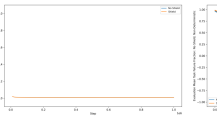
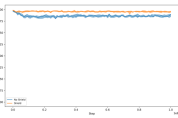
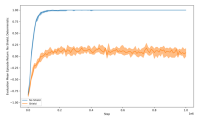
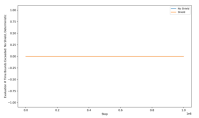
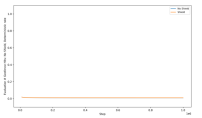
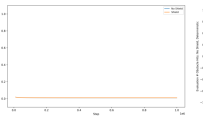
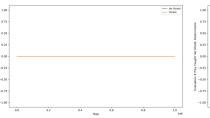
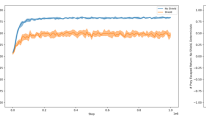
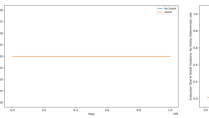
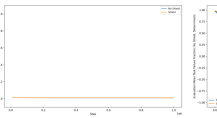
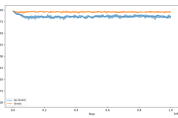
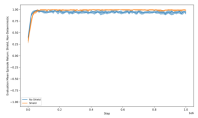
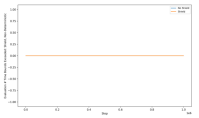
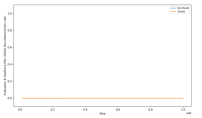
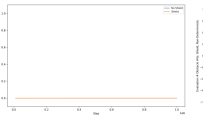
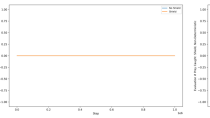
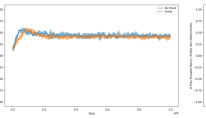
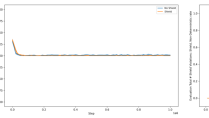
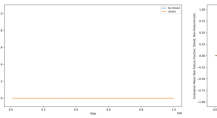
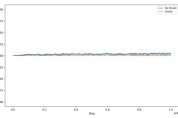
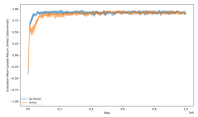
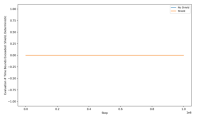
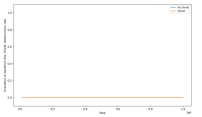
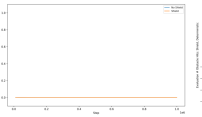
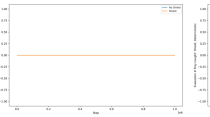
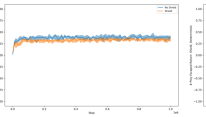
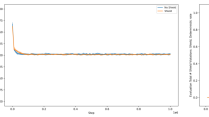
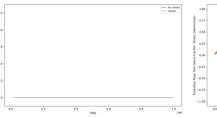
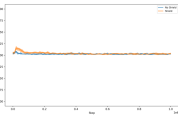
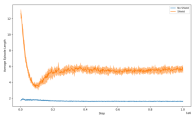
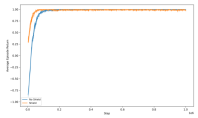
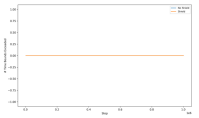
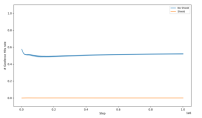
Theorems

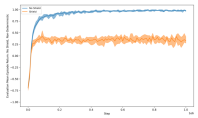
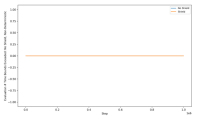
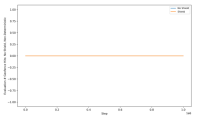
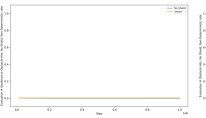
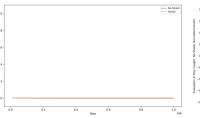
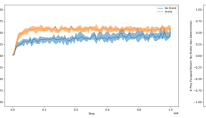
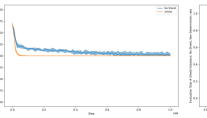
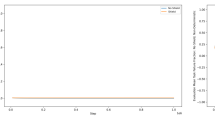
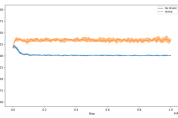
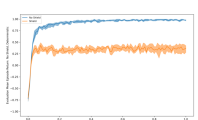
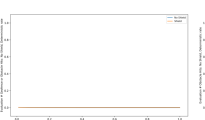
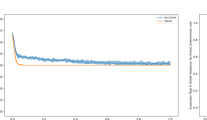
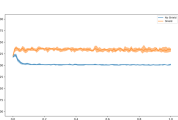
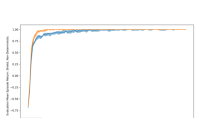
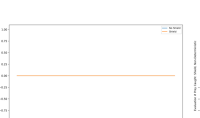
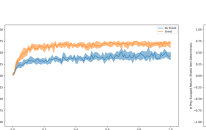
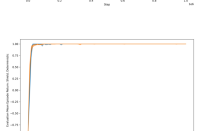
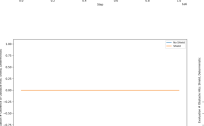
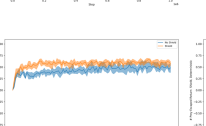
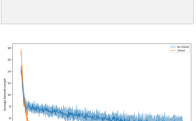
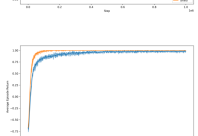
$\Box\ 0 \leq tank, tank \leq 100$
$\Box\ 0 \leq t, t \leq$ NUM_NEXTS
$\Box\ open \wedge \bigcirc \neg open \rightarrow \bigcirc\bigcirc\ \neg open$
$\Box\ \neg open \wedge \bigcirc open \rightarrow \bigcirc\bigcirc\ open$

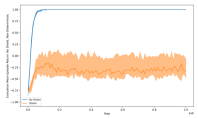
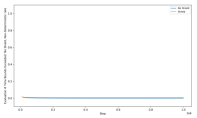
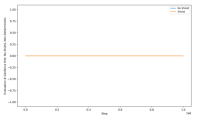
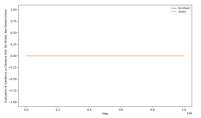
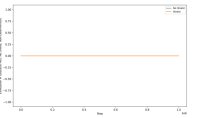
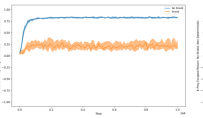
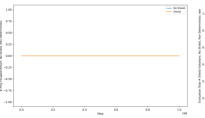
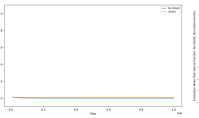
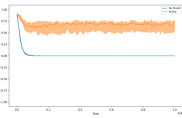
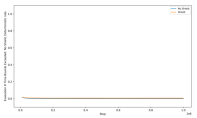
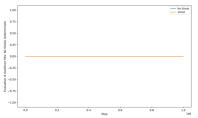
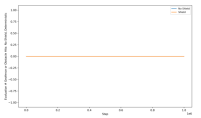
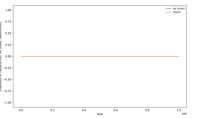
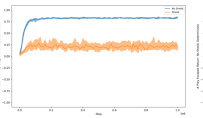
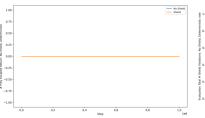
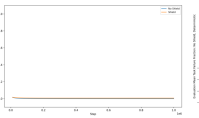
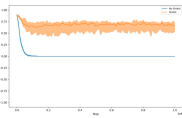
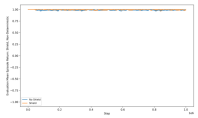
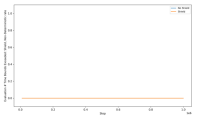
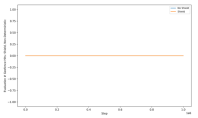
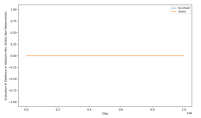
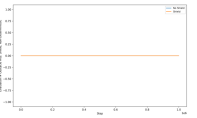
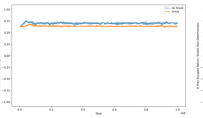
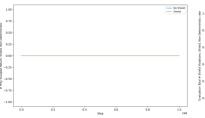
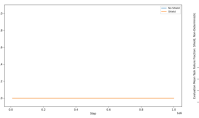
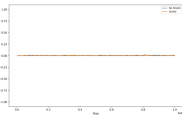
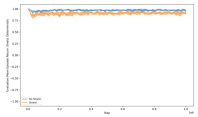
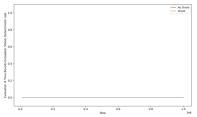
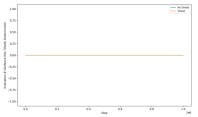
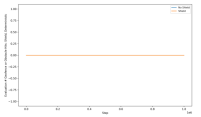
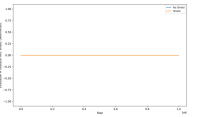
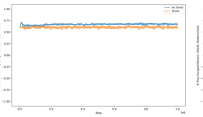
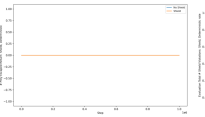
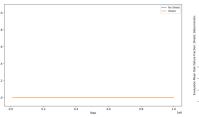
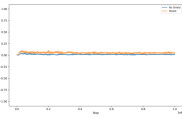
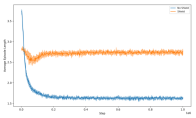
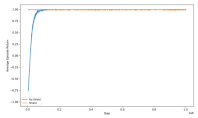
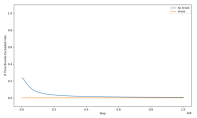
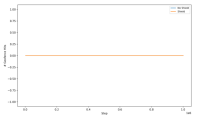
End Module

Figure 19: Final refined specification of Water Tank problem (contd)

Appendix 3 Summary Figures

	Ep Len Mean	Mean Reward	N Bound Exceeded	N Hit Geofence	N Hit Geofence Or Obs	N Hit Obstacle	Prey Caught	Prey Escaped	Shield Violated	Taskfailed
No Shield Non-Deterministic										
No Shield Deterministic										
Shield Non-Deterministic										
Shield Deterministic										
Rollout										

	Ep Len Mean	Mean Reward	N Bound Exceeded	N Hit Geofence	N Hit Geofence Or Obs	N Hit Obstacle	Prey Caught	Prey Escaped	Shield Violated	Taskfailed
No Shield Non-Deterministic										
No Shield Deterministic										
Shield Non-Deterministic										
Shield Deterministic										
Rollout										

	Ep Len Mean	Mean Reward	N Bound Exceeded	N Hit Geofence	N Hit Geofence Or Obs	N Hit Obstacle	Prey Caught	Prey Escaped	Shield Violated	Taskfailed
No Shield Non-Deterministic										
No Shield Deterministic										
Shield Non-Deterministic										
Shield Deterministic										
Rollout										


Ep Len Mean
Mean Reward
N Bound Exceeded
N Hit Geofence
N Hit Geofence Or Obs
N Hit Obstacle
Prey Caught
Prey Escaped
Shield Violated
Taskfailed
No Shield Non-Deterministic
No Shield Deterministic
Shield Non-Deterministic
Shield Deterministic
Rollout
No Shield
Shield
Step