SYNTHESIS OF SHIELDS FOR SAFE REINFORCEMENT LEARNING IN BOTH DISCRETE AND CONTINUOUS STATE AND ACTION SPACES WITH LINEAR DYNAMICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement Learning agents are increasingly deployed in domains where unsafe actions can have serious real-world consequences. These domains impose rules or constraints that the agent must observe, of the forms "always/never do X", "do X within k timesteps", "repeatedly do X next", etc. Currently, agents 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 for actions accepted by the shield. Existing shield construction methods are largely limited to finite, discrete state and action spaces. 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 whenever synthesis reaches a greatest fixpoint, and is agnostic to the choice of RL algorithm.

1 INTRODUCTION

Reinforcement Learning (RL) agents are increasingly deployed in domains where unsafe actions can have serious real-world consequences, including automated driving Shalev-Shwartz et al. (2016), games Giacobbe et al. (2021), healthcare Jia et al. (2020), real world mechanics Banerjee et al. (2025), and legal Abe et al. (2010) settings. These domains impose rules or constraints that the agent must observe, of the form "always do X", "eventually do X", "do X within k timesteps", "infinitely often do X", and logical combinations of these. Standard RL agents have to learn these constraints the hard way, through trial and error over many training episodes, with no guarantee that they were actually learned. Beyond ensuring that the agent does not enter bad states during deployment, such states may also be unacceptable *during training itself*, for agents that learn directly in the real world rather than in simulation.

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.

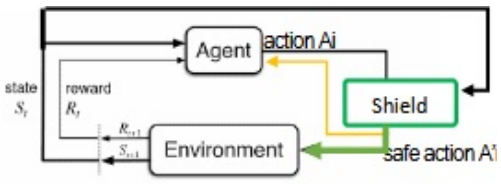


Figure 1: Role of the shield: accepting or rejecting actions choices by the agent

Our approach is *neuro-symbolic*: a symbolic, SMT-verified shield is combined with a neural RL policy, intervening only when needed to guarantee safety while otherwise deferring to the policy, although more loosely coupled than in current taxonomies Kautz (2022) – ours most closely matches

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

"Neuro;Symbolic". In contrast to many current approaches (see Related Work), which rely on finite-state abstractions or a fixed, bounded lookahead horizon, to our best knowledge ours is the first shield synthesis approach that simultaneously supports all the following features

- safety guarantee that holds over an unbounded horizon (useful for long-lived agents)
- uniform handling of both continuous state and action spaces (useful in hybrid systems) *without move enumeration or discretization*
- temporal safety constraints expressed in decidable theories with quantifier elimination, such as linear real and integer arithmetic (useful for real world quantitative constraints)

2 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 probabilistic (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. Similar to Alshiekh et al. (2018), a Markov Decision Process (MDP) is a tuple $\mathcal{M} = (S, s_I, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where S is a set of states, $s_I \in S$ is the initial state, $\mathcal{A}$ is a set of Boolean actions, $\mathcal{P} : S \times \mathcal{A} \to \text{Dist}(S)$ is the probabilistic transition function, and $\mathcal{R} : S \times \mathcal{A} \times S \to \mathbb{R}$ is a reward function. As in Alshiekh et al. (2018), shield synthesis does not use numerical transition probabilities or rewards, but conservatively reasons over possible behaviors, but unlike Alshiekh et al. (2018), we do not require a finite state set or action set.

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). We use a first order linear temporal logic (FOLTL) Baier & Katoen (2008) over infinite traces of states, restricted to safety properties expressible with the "next" ($\bigcirc$) and "always" ($\Box$) operators. A *safety property* is an LTL formula prefixed by $\Box$, stating that some condition holds at every point of a trace. The full grammar and inductive trace semantics ($\sigma, n \models \phi$) are standard and are given in Appendix A for completeness.

LABELED NODE TRANSITION SYSTEMS

Our approach to shield synthesis is based on an extension to the model refinement approach (MR) of Smith & Nedunuri (2024) which operates on *labeled node transition systems* (LNTS) (referred to as a labeled control flow graph in Smith & Nedunuri (2024)). 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* or *labels* 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. A $legal$ state is a state satisfying p. *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 possibly infinite set of state transitions over a node and labels an arc over a node. $[\![a]\!]$ denotes a predicate over a pair of states, and $a(s,t)$ denotes the truth value $[\![a]\!](s,t)$ for states s and t. Any infinite sequence of states produced by an LNTS starting from a given legal state and following any allowable action from a state is called a *trace* of the LNTS.

```
// initialize the node and arc invariants to the
// state and step safety properties, resp.
if stateP(φ)
then: L := L ∧ φ
else L_a ← L_a ∧ φ

// iterate over the constraints until no further
//refinement is possible
do
    for each arc a
        G_a := G_a ∧ wcp(L)
        L := L ∧ ∃u. G_a
until for all a, L → G_a ∧ L.
```

Figure 2: Model Refinement Algorithm

A legal state s of M is *non-blocking* if there is a an arc $a = \langle m, m \rangle$ satisfying $[\![a]\!]$. In game-theoretic terms, if all reachable 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. Further details are in Appendix A.

MODEL REFINEMENT (MR)

Space permits us to provide only a high level overview of MR Smith & Nedunuri (2024), further details are in Appendix B. MR begins with an LNTS that overapproximates the permitted behaviors and iteratively strengthens its invariants and guards until the resulting model admits only behaviors satisfying the required safety properties.

A straightforward algorithm for solving the constraint system is presented in Fig. 2. Here ϕ is a safety property being enforced; $stateP(\phi)$ determines whether ϕ constrains a node (updating L) or a transition (updating the arc invariant L_a). Let *wcp* be the *weakest controllable predecessor* defined as:

$$\forall e.\, e \in E(s) \rightarrow L(f_a(s, e, u))$$

where E is the predicate defining the adversary's possible choices and f_a provides the semantics of the LNTS. The loop then repeatedly strengthens each arc's guard G_a to $G_a \wedge wcp(L)$ — the actions from which every adversary response stays within L — and correspondingly narrows L to states from which some permitted action still exists ($L := L \wedge \exists u\, G_a$), until guards and invariant do not change in an iteration. Assuming Algorithm 2 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. It is important to note that unlike many reactive synthesis approaches Bloem et al. (2012) or invariant synthesis approaches Bradley (2011), MR operates over the Lindenbaum lattice of *constraints* not over the *state space*. Several key problems arise in solving the constraint system; these are also outlined in Appendix B.

3 SHIELD SYNTHESIS

Definition 1 (Shield)**.** *A* shield *for a model M and a required safety/liveness property Φ is a pair (L, G), consisting of an invariant L over the states of M and, for every arc a, a guard G_a strengthening a's enabling condition, such that every trace generated by the resulting refined model satisfies Φ and every legal state remains non-blocking.* Shield synthesis *is the problem of computing such an (L, G); we solve it via model refinement (MR).*

We extend MR in two independent ways to support the kind of problems we encounter when constructing shields for RL In Smith & Nedunuri (2024) the *wcp* is defined so as to commit the control input before the environment's move, giving it the quantifier structure $\exists u\, \forall e$. We support the dual quantifier order, $\forall e\, \exists u$ -- i.e.,control observes the environment's move before responding -- by defining

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

and modifying the refinement algorithm correspondingly.

Second, we extend MR to to support the *next* ($\bigcirc$) operator. For plain (unnested) bounded reachability, as in Example 1, Predator-Prey ("reach the target within k steps"), the target predicate is unrolled backward through wcp for a fixed k iterations, yielding a bounded reachability guard without adding any state. For properties with a nested *next* operator, as in Example 2, Water Tank ("stay open, then closed, for the next m steps"), we instead compile the property into an ordinary safety property by adding one or more fresh protected integer timer variables to the model's own state, incremented while waiting on each sub-goal and reset once it is met; the augmented system is then solved by the same fixpoint algorithm (Fig. 2). Full constructions and a correctness proof for the general case are in Appendix B.

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/houseofwealth/Shielded-RL`.

4 Example 1: Predator and Prey

The first example we consider is one in which a predator must catch a prey before it reaches the safety of the herd Chance et al. (2021). This example was partly chosen to illustrate the novelty of our approach. It cannot be handled by Alshiekh et al. (2018) on account of the continuous state and action space. And while Rodríguez et al. (2025) and Anand et al. (2021) can handle continuous space problems, they both require a fixed enumeration of the agent's possible actions beforehand, which again is not possible in this problem.

The problem setup is 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.

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
$\square\ L \leq x \leq R$
$\square\ B \leq y \leq T$
End Module

Figure 3: Initial Specification of Shield for Geofencing Constraint

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. 3. 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. 4.

The starting invariant is simply the given safety property (Fig. 4(a)). The *wcp* (Fig. 4(b)) universally quantifies over the next state reached by substituting the dynamics into the invariant. (In our experiments, the timestep t is 1). This formula is in the language of linear arithmetic, either integer

(a) starting invariant:

$$x \geq -10 \wedge x \leq 10 \wedge y \geq 0 \wedge y \leq 10$$

(b) wcp:

$$\forall x', y', v_x', v_y'.\ (x' = x + v_x t + \tfrac{1}{2} a_x t^2 \wedge y' = y + v_y t + \tfrac{1}{2} a_y t^2) \rightarrow (x' \geq -10 \wedge x' \leq 10 \wedge y' \geq 0 \wedge y' \leq 10)$$

(c) simplified wcp ($t = 1$):

$$x + v_x + \tfrac{1}{2} a_x \geq -10 \wedge x + v_x + \tfrac{1}{2} a_x \leq 10 \wedge y + v_y + \tfrac{1}{2} a_y \geq 0 \wedge y + v_y + \tfrac{1}{2} a_y \leq 10$$

(d) new guard:

$$a_x \geq -10 \wedge a_x \leq 10 \wedge a_y \geq -10 \wedge a_y \leq 10 \wedge x + v_x + \tfrac{1}{2} a_x \geq -10 \wedge x + v_x + \tfrac{1}{2} a_x \leq 10 \wedge y + v_y + \tfrac{1}{2} a_y \geq 0 \wedge y + v_y + \tfrac{1}{2} a_y \leq 10$$

(e) guard disjunction (after QE):

$$(2y + 2v_y \leq 30) \vee (2x + 2v_x \leq 30 \wedge -x - v_x \leq 15 \wedge 2y + 2v_y \leq 30 \wedge -y - v_y \leq 5)$$

(f) simplified guard disjunction:

$$2y + 2v_y \leq 30 \wedge 2x + 2v_x \leq 30 \wedge -x - v_x \leq 15 \wedge -y - v_y \leq 5$$

(g) updated invariant:

$$x \geq -10 \wedge x \leq 10 \wedge y \geq 0 \wedge y \leq 10 \wedge 2y + 2v_y \leq 30 \wedge 2x + 2v_x \leq 30 \wedge -x - v_x \leq 15 \wedge -y - v_y \leq 5$$

Figure 4: Compact summary of the seven key quantities computed during the first refinement iteration of the Geofencing example; the full Z3 console trace is shown in Fig. 10 in Appendix C.

or real (LIA or LRA) De Moura & Bjørner (2011) which admits quantifier elimination (QE), so our Z3-based prototype simplifies it to the form shown in Fig. 4(c).

Next, the new guard, the guard disjunction, and the result of quantifier elimination and simplification on the guard disjunction are automatically computed (Fig. 4(d)–(f)); Finally, the invariant itself is updated (Fig. 4(g)), and as it has changed, the solving process continues to the next iteration. the full Z3 solver trace for this iteration is given in Fig. 10 in Appendix C. This process continues for three further iterations until there are no more changes to the invariant or any of the guards, the solver converges to final refined model shown in Fig. 5. 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. 6 (first row) shows the results of training 16 RL agents to intercept prey within a geofence. Both variants attain the maximum mean episode return, but shielding accelerates learning and substantially reduces constraint violations.

(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 C.

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**
    - $x \leq 10 \wedge x \geq -10 \wedge$
    - $2y + 2v_y \leq 30 \wedge y \geq 0 \wedge y \leq 10 \wedge$
    - $x + v_x \geq -15 \wedge 2x + 2v_x \leq 30 \wedge$
    - $y + v_y \geq -5 \wedge x + 2v_x \geq -30 \wedge$
    - $2x + 4v_x \leq 60$
  - **Theorems**
    - $\Box\ -10 \leq x \leq 10$
    - $\Box\ 0 \leq y \leq 10$
- **End Module**

Figure 5: Final Refined Specification of Shield for Geofencing Constraint

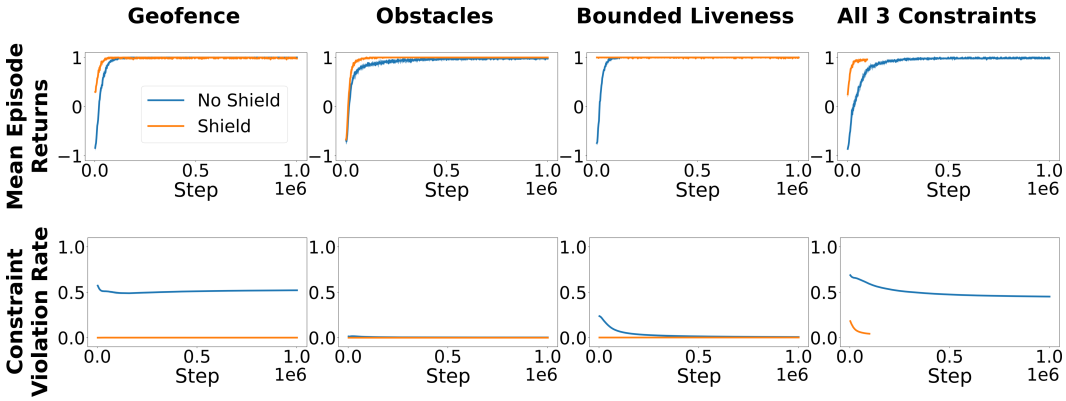


Figure 6: Mean episode returns (possible range: [-1 1]) and constraint violation rate (possible range: [0 1]) for each constraint type for the Predator-Prey task. Lines and shaded regions (barely visible) represent the median and interquartile ranges across 16 trained agents. For the All 3 Constraints shielded case, early stopping was employed.

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

In addition to handling standard invariance safety properties such as "always/never do X," the shield can handle bounded liveness properties such as "do X within t steps." An example of this would be if the predator needs to be sure it can reach the prey within a certain number of steps before attempting pursuit. The initial specifications is in Fig. 13 in Appendix C. The shield synthesis converges after four iterations with the final spec shown in Fig. 14 in Appendix C.

Summary of Predator-Prey Experiments

Figure 6 shows the learning curves, and Table 1 summarizes the results. For geofencing, shielding accelerated convergence while greatly reducing violations. For obstacle avoidance, both variants attained the maximum return, but shielding converged faster and produced substantially fewer violations. For bounded liveness, shielded agents attained the maximum return without training and violated the constraint less frequently. A fourth experiment shows the results of combining all the 3 constraints; shielding again converged faster and reduced violations, although unshielded agents attained a slightly higher median return. All shielded violations occurred after the agent failed to propose an accepted action in 100 attempts, at which point the implementation executed an unshielded random action. In every experiment, synthesis reached the greatest fixpoint without widening or the 50-iteration cutoff, retaining the weakest-refinement and minimal-interference guarantees.

Table 1: Predator--Prey results (shielded/unshielded).

Constraint	Steps	Return	Violations
Geofencing	68K/152K	1.00/1.00	749/522K
Obstacle	102K/612K	1.00/1.00	14/3,035
Bounded	Untrained/96K	1.00/1.00	2,348/7,483
All three	66K/432K	0.96/1.00	107K/453K

5 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."

Additionally, unlike Alshiekh et al. (2018), we can handle the continuous version of this problem without needing to construct a fixed finite abstraction.

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. For the purposes of illustration, we consider a simpler version of the problem in which the valve is only required to open for two steps after a change (Fig. 7). (Fig 15 in Appendix C shows the complete initial specification which is implemented in the prototype and available in the github repo). Fig. 17 shows how the specification is transformed into a standard form and 18 the resulting final specification after MR.

...
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 7: Simplified Water Tank problem

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.

Table 5 compares the violation and intervention rates of our tool vs. Alshiekh et al. (2018). Several issues arose in attempting to compare our tool against theirs. The first obstacle we encountered was that their published GitHub repo did not contain a DFA (abstraction automaton) for this example. However we were (with the help of AI) able to construct it from looking at the format of the other DFAs that were available and the description in the paper. Secondly, their specification only covers the non overflow part (we call ϕ_1) of the stated problem, not the entire requirement regarding remaining open/closed a fixed number of timesteps ($\phi_1 + \phi_2$). Thirdly, we noticed that their shield was written in such a way that once an action was overridden, it entered a permanent OFF_MODEL state (they refer to as a "paradise" state) in which no further bad actions are blocked (this is why the "always_open" case has 300 out of 300 violations). In contrast ,our shield continues to monitor bad actions. Finally, they use DQN learning whereas we use PPO so a direct runtime performance comparison was omitted.

Regarding shield computation time, theirs took 0.4s to compute. Ours was nearly 10x that time. However, it must be borne in mind that their time do not include the human effort to construct the abstraction which, while not difficult, involves some effort (e.g. consider the case of an inflow rate between 0.9 and 2.2 and outflow between 0 and 1.25, say, which requires interval analysis).

6 RELATED WORK

Related work on safe reinforcement learning spans shielding, invariant-based methods, reward shaping, and safe exploration; see García & Fernández (2015) for a broad survey. Our work is is inspired

[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, where the word "plant" corresponds to the RL term "environment".

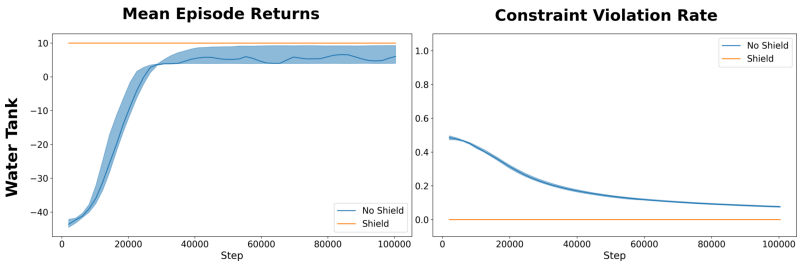


Figure 8: Mean episode returns (maximum possible: 10) and constraint violation rate (possible range: [0 1]) for each constraint type for the water tank task. Lines and shaded regions represent the median and interquartile ranges across 16 trained agents.

Shield	Override%	Viol.
Alshiekh	0.00%	0/300
OFF_MODEL	–	1/300
Ours ϕ_1	0.00%	0/300
Ours ϕ_1, ϕ_2	24.50%	0/300

random

Shield	Override%	Viol.
Alshiekh	1.57%	300/300
OFF_MODEL	–	300/300
Ours ϕ_1	34.57%	0/300
Ours ϕ_1, ϕ_2	36.27%	0/300

always_open

Shield	Override%	Viol.
Alshiekh	0.63%	91/300
OFF_MODEL	–	170/300
Ours ϕ_1	1.23%	0/300
Ours ϕ_1, ϕ_2	1.71%	0/300

always_close

by the pioneering work of Alshiekh et al. (2018), who introduced shielding for reinforcement learning with formal safety guarantees and minimal interference, Their construction is automata-based and, for continuous systems, relies on a finite abstraction of the state space. Like Alshiekh et al. (2018) our LNTS abstracts the MDP for shield synthesis by retaining possible behaviors while discarding transition probabilities and rewards, but the two abstractions differ structurally. Theirs is a deterministic finite automaton over action-observation symbols, where actions are uninterpreted alphabet symbols whose effects are encoded extensionally by the automaton's transitions, and alternative stochastic outcomes appear as adversarially chosen observations; And while never formally specified as a construction procedure, building this automaton does appear to require manual effort: "Our experiments show, however, that the effort to construct an abstraction is well-spent" Alshiekh et al. (2018). Our LNTS, by contrast, permits an infinite state space, associates each action with an explicit symbolic state-transition relation, and represents adversarial non-determinism as disjunction within that relation; Our is obtained by a straightforward projection of the MDP.

Where Alshiekh et al. (2018) must first discretize the environment into a finite automaton, our method synthesizes shields directly via constraint propagation and SMT solving, avoiding explicit state-space abstraction and representing the set of admissible actions symbolically. The same algorithm specializes to bounded integer domains simply by declaring the variables as integers, with Z3 selecting linear integer arithmetic; correctness then follows from the existing MR result in Smith & Nedunuri (2021).

Table 2 situates our approach against the closest prior work on synthesizing shields/controllers for continuous-space systems. Anand et al. (2021) synthesizes a maximal permissive controller offline via a fixpoint computed with quantifier elimination, but requires the adversary's and agent's move choices to be enumerated in advance as a fixed disjunction. Rodríguez et al. (2025) extends reactive LTL synthesis to handle theories over continuous domains, but does so by partitioning the continuous input space into a finite set of symbolic "reactions" before synthesis, i.e. an automated case-split rather than a fully symbolic guard. Kim et al. (2025) synthesizes shields that provably return a safe action for continuous-state systems, but does so by calling an SMT solver or optimizer online at every timestep the current action is found unsafe. No prior approach combines a continuous state space, a fully offline/static shield (no runtime solver calls), and the avoidance of move enumeration or discretization; To the best of our knowledge, our approach is the only one to satisfy all three simultaneously. Dalal et al. (2018) learns approximate dynamics and minimally corrects unsafe actions via a quadratic program, but does not support infinite lookahead and assumes at most one active constraint. Anderson et al. (2020) iteratively refines a neurosymbolic policy with a neural controller and a discrete shield, but relies on invariant construction at deployment time. Zhu et al. (2019) synthesizes inductive invariants from user-provided sketches, but does not 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

7 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.

Table 2: Comparison with prior shield/controller synthesis approaches for continuous-space systems.

Method	Continuous state space	Offline/ static	No move enum.
Anand et al. (2021)	✓	✓	×
Alshiekh et al. (2018)	×	✓	×
Rodríguez et al. (2025)	✓	✓	×
Kim et al. (2025)	✓	×	✓
Ours	✓	✓	✓

8 FUTURE WORK

Future work follows directly from the current formulation. First, a shield that computes a richer description of the safe action set (a polyhedron, in the linear case) than the binary admissibility signal currently exposed to the policy; incorporating that structure into learning, e.g. via constrained policy optimization Achiam et al. (2017), could give more informative guidance when the admissible region is narrow. Second, we plan to extend the framework from the current adversarial (environment-vs.-agent) setting to genuinely multi-agent settings with several cooperating or competing controllable agents, by generalizing the constraint structure to alternate quantificationv over each agent's actions in turn. Third, our current action selector finds a safe action by rejection sampling from the policy's proposed actions; when the admissible region is narrow this can occasionally fail within the sampling budget (100 attempts), forcing a fallback to an unshielded action. We plan to instead directly project onto the synthesized admissible set (e.g. the polyhedron above), guaranteeing a safe action is found without relying on sampling.

Reproducibility Statement. Our shield synthesis procedure is fully specified as a deterministic sequence of Z3/SMT queries with correctness assumptions and proofs in Appendix B. The water-tank comparison experiments use fixed random seeds per policy (specified in compare_shields_watertank.py), and the Alshiekh et al. (2018) baseline uses the original authors' unmodified C++ shield-synthesis tool together with a finite-abstraction generator we wrote to translate the continuous water-tank spec into their required DFA input format. Anonymized source code for the shield synthesizer, and the examples has been made available at the GitHub repo stated in the main body of the paper.

LLM Usage. An LLM-based coding assistant was used during the preparation of this work for: (1) implementation and debugging of experiment/comparison scripts (2) LaTeX formatting and compilation debugging, and (3) drafting and refining prose in the Related Work section, which the authors reviewed, verified against primary sources, and edited. The LLM was not involved in formulating the core shield-synthesis method, designing the theoretical contributions, designing the experiments, or running them. All experimental results, proofs, and technical claims were independently verified by the authors.

REFERENCES

Naoki Abe, Prem Melville, Cezar Pendus, Chandan K. Reddy, David L. Jensen, Vince P. Thomas, James J. Bennett, Gary F. Anderson, Brent R. Cooley, Melissa Kowalczyk, Mark Domick, and Timothy Gardinier. Optimizing debt collections using constrained reinforcement learning. In *Proc.the 16th ACM SIGKDD Intl Conference on Knowledge Discovery and Data Mining*, KDD '10, pp. 75—-84, 2010. ISBN 9781450300551.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proc. of the 34th International Conference on Machine Learning*, volume 70, pp. 22–31, Aug 2017.

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. AAAI'18/IAAI'18/EAAI'18, 2018.

Ashwani Anand, Kaushik Mallik, and Anne-Kathrin Schmuck. Gensys: A scalable fixed-point engine for maximal controller synthesis over infinite state spaces. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE)*, 2021.

Greg Anderson, Abhinav Verma, Isil Dillig, and Swarat Chaudhuri. Neurosymbolic reinforcement learning with formally verified exploration. In *Proc. of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, 2020.

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

Chayan Banerjee, Kien Nguyen, Clinton Fookes, and Maziar Raissi. A survey on physics informed reinforcement learning: Review and open problems. *Expert Systems with Applications*, 287: 128166, 2025.

Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Saar. Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012. URL https://www.sciencedirect.com/science/article/pii/S0022000011000869. In Commemoration of Amir Pnueli.

Aaron R. Bradley. SAT-based model checking without unrolling. In *Verification, Model Checking, and Abstract Interpretation*, volume 6538 of *LNCS*, pp. 70–87. Springer, 2011. doi: 10.1007/978-3-642-18275-4_7.

Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In *Computer Aided Verification (CAV)*, volume 8559 of *LNCS*, pp. 334–342, 2014.

F. S. Chance, C. Little, M. M. McKenzie, R. A. Dellana, D. E. Small, T. R. Gayle, and D. K. Novick. Biologically inspired interception on an unmanned system. Technical Report SAND2021-13549R, Sandia National Laboratories, October 2021.

Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pp. 238–252. ACM, 1977.

Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces, 2018. URL https://arxiv.org/abs/1801.08757.

L. De Moura and N. Bjørner. Satisfiability modulo theories: introduction and applications. *Commun. ACM*, 54(9):69–77, September 2011. ISSN 0001-0782.

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

J. García and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16:1437–1480, 2015. URL http://jmlr.org/papers/v16/garcia15a.html.

Mirco Giacobbe, Mohammadhosein Hasanbeig, Daniel Kroening, and Hjalmar Wijk. Shielding atari games with bounded prescience. In *Proc. of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1507—1509, 2021.

Yan Jia, John Burden, Tom Lawton, and Ibrahim Habli. Safe reinforcement learning for sepsis treatment. In *2020 IEEE Intl Conf. on Healthcare Informatics (ICHI)*, pp. 1–7, 2020.

Henry A. Kautz. The third AI summer: AAAI Robert S. Engelmore memorial lecture. *AI Magazine*, 43(1):105–125, 2022. doi: 10.1002/aaai.12036.

Kyungmin Kim, Davide Corsi, Andoni Rodríguez, JB Lanier, Benjami Parellada, Pierre Baldi, César Sánchez, and Roy Fox. Realizable continuous-space shields for safe reinforcement learning. *arXiv preprint arXiv:2410.02038*, 2025. L4DC 2025.

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

Andoni Rodríguez, Guy Amir, Davide Corsi, César Sánchez, and Guy Katz. Shield synthesis for LTL modulo theories. *arXiv preprint arXiv:2406.04184*, 2025. AAAI 2025.

Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving, 2016. URL `https://arxiv.org/abs/1610.03295`.

D.R. Smith and S. Nedunuri. Model refinement. Tech. Rep. 21.0, Kestrel Institute, 2021. URL `https://www.kestrel.edu/people/smith/pub/mr-tr.pdf`.

D.R. Smith and S. Nedunuri. Deductive model refinement. In *NASA Formal Methods: 16th International Symposium, NFM 2024, Proceedings*, pp. 147–165, 2024.

He Zhu, Zikang Xiong, Stephen Magill, and Suresh Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pp. 686–701, 2019.

A TEMPORAL LOGIC AND LNTS

TEMPORAL LOGIC: SYNTAX AND SEMANTICS

A *basic linear temporal logic (LTL) formula* is defined by the following grammar:

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

where ϕ 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. That a formula ϕ holds 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:

- $\sigma, n \models p$, for p a state formula, if p holds at state $\sigma[n]$, i.e. $[\![p]\!](\sigma[n])$;
- $\sigma, n \models a$, 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 $i \geq n$

A *Labeled Node Transition System* LNTS differ from the usual LTS Baier & Katoen (2008) because arcs are viewed as operating over *nodes* rather than individual states. 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. Similar to a node, the semantics of an arc, $[\![a]\!]$, is given by the transitions or steps that satisfy the arc label:

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

More specifically, an arc label L_a is a predicate that specifies a non-deterministic action and is derived from a combination of the adversary's action, the guard on the arc, and the semantics of the environment (MDP), that is :

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

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.

a state s is a non-blocking state if there is a 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.

Traces. A *trace* is an infinite 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 LNTS M is the language of M and is denoted $\mathscr{L}(M)$.

B MODEL REFINEMENT (MR)

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 & Nedunuri (2021) for the proof).

Given specification $\mathcal{S} = \langle \mathcal{M}, \Phi \rangle$, the goal of *property enforcing model refinement* (MR) 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 MR is the elimination of blocking states in the model.

SYNTHESIS AS CONSTRAINT SOLVING

MR 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:

$$wcp \;\equiv\; \forall e.\, e \in E(s) \rightarrow L(f_a(s, e, u)) \tag{1}$$

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:

$$U(s, u) \rightarrow wcp \tag{2}$$

[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. We have reformulated it to allow for the agent to respond to the environment input.

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

$$L(s) \rightarrow \bigvee_a \exists u \cdot U(s,u) \tag{3}$$

Thus MR 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**: (2) and (3) 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 (2) serve to eliminate any blocking states at a node with respect to an outgoing arc. Constraints (3) serve to eliminate blocking states at a node with respect to all of its outgoing arcs. Given a specification $\mathcal{S} = \langle M, \Phi\rangle$, MR 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.

Problems Arising in Solving the Constraint System

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 & 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 & 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.

Bounded Liveness

(a) Unnested operator

In the Prey Reachability example considered earlier, in addition to the invariant L in the earlier formula there is now a variant V which is initialized to the bounded reachability 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

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

(b) General Case

Unlike in the Prey Reachability example, the Water Tank problem has a nested "next" operator. We now introduce a transformation for the nested next operator. First we provide some commutative and distributive laws. Their proof is straightforward so it is omitted.

$$\begin{aligned} \neg \bigcirc \phi &\equiv \bigcirc \neg\phi \\ \bigcirc(\phi_1 \wedge \phi_2) &\equiv \bigcirc\phi_1 \wedge \bigcirc\phi_2 \\ \bigcirc(\phi_1 \vee \phi_2) &\equiv \bigcirc\phi_1 \vee \bigcirc\phi_2 \end{aligned} \tag{4}$$

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 & Katoen (2008) by applying the equivalences in B 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 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 $\Box$ 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. $\Box\,(t = 0\ \wedge \phi_0) \rightarrow t' = 1$
      iii. $\Box\,(t = 0\ \wedge \neg\phi_0) \rightarrow t' = 0$
      iv. For every $i, 1 \leq i < n$, add $\Box\,(t = i\ \wedge \phi_i) \rightarrow t' = t$
      v. For every $i, 1 \leq i < n$, add $\Box\,(t = i\ \wedge \neg\phi_i) \rightarrow t' = t + 1$
      vi. $\Box\, t = n \rightarrow \phi_n \wedge t' = 0$

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

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 B, 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. $\square$

C EXPERIMENTS AND RESULTS

(A) GEOFENCING

Figs. 9, 10, and 11 show the initial specification, the first iteration of the refinement process, and the final 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
$\square\ L \leq x \leq R$
$\square\ B \leq y \leq T$
End Module

Figure 9: Initial Specification of Shield for Geofencing Constraint

(B) OBSTACLE AVOIDANCE

Fig. 12 shows the final refined specification.

(C) 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 reachability formulas using a *variant* (see Appendix B 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 "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. Figs. 13 and 14show the initial spec and the final spec resp.

```
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
$x' = x + v_x + a_x/2 \wedge$
$v_x' = v_x + a_x \wedge$
$y' = y + v_y + a_y/2 \wedge$
$v_y' = v_y + a_y$
Control Pred
$(t = 1 \rightarrow \ y + v_y + (1/2)a_y = y_{prey} \wedge$
$x + v_x + (1/2)a_x = x_{prey}) \wedge$
$(t = 2 \rightarrow -x + -2v_x + -(3/2)a_x + x_{prey} \leq 5 \wedge$
$x + 2v_x + (3/2)a_x + -x_{prey} \leq 5 \wedge$
$y + 2v_y + (3/2)a_y + -y_{prey} \leq 5 \wedge$
$-y + -2v_y + -(3/2)a_y + y_{prey} \leq 5) \wedge$
$(t = 3 \rightarrow (2y + 6v_y + 5a_y - 2y_{prey} \leq 40 \wedge$
$2x + 6v_x + 5a_x - 2x_{prey} \leq 40 \wedge$
$-y + -3v_y + -(5/2)a_y + y_{prey} \leq 20 \wedge$
$-x + -3v_x + -(5/2)a_x + x_{prey} \leq 20)$
Variant
$(x = x_{prey} \wedge y = y_{prey})$
$\vee \ (-x + -v_x + x_{prey} \leq 5 \wedge$
$y + v_y + -y_{prey} \leq 5 \wedge$
$x + v_x + -x_{prey} \leq 5 \wedge$
$-y + -v_y + y_{prey} \leq 5)$
$\vee \ (2x + 4v_x + -2x_{prey} \leq 40 \wedge$
$2y + 4v_y + -2y_{prey} \leq 40 \wedge$
$-y + -2v_y + y_{prey} \leq 20 \wedge$
$-x + -2v_x + x_{prey} \leq 20)$
$\vee \ ((2/5)x + (6/5)v_x + -(2/5)x_{prey} \leq 18 \wedge$
$-y + -3v_y + y_{prey} \leq 45 \wedge$
$(2/5)y + (6/5)v_y + -(2/5)y_{prey} \leq 18 \wedge$
$-x + -3v_x + x_{prey} \leq 45)$
Theorems
$|x - x_{prey}| \leq d \wedge |y - y_{prey}| \leq d$
End Module

Figure 14: Final Specification for Bounded Liveness Requirement

EXAMPLE: WATER TANK PROBLEM

The initial specification is in Fig. 15

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 15: Initial specification for Water Tank problem

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:

The transformed specification is as follows (where NUM_NEXTS is 2):

This can now be passed to the refinement engine which produces the following refined specification.

...
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 16: Simplified Water Tank problem

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

Figure 18: Final refined specification of Water Tank problem

...

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

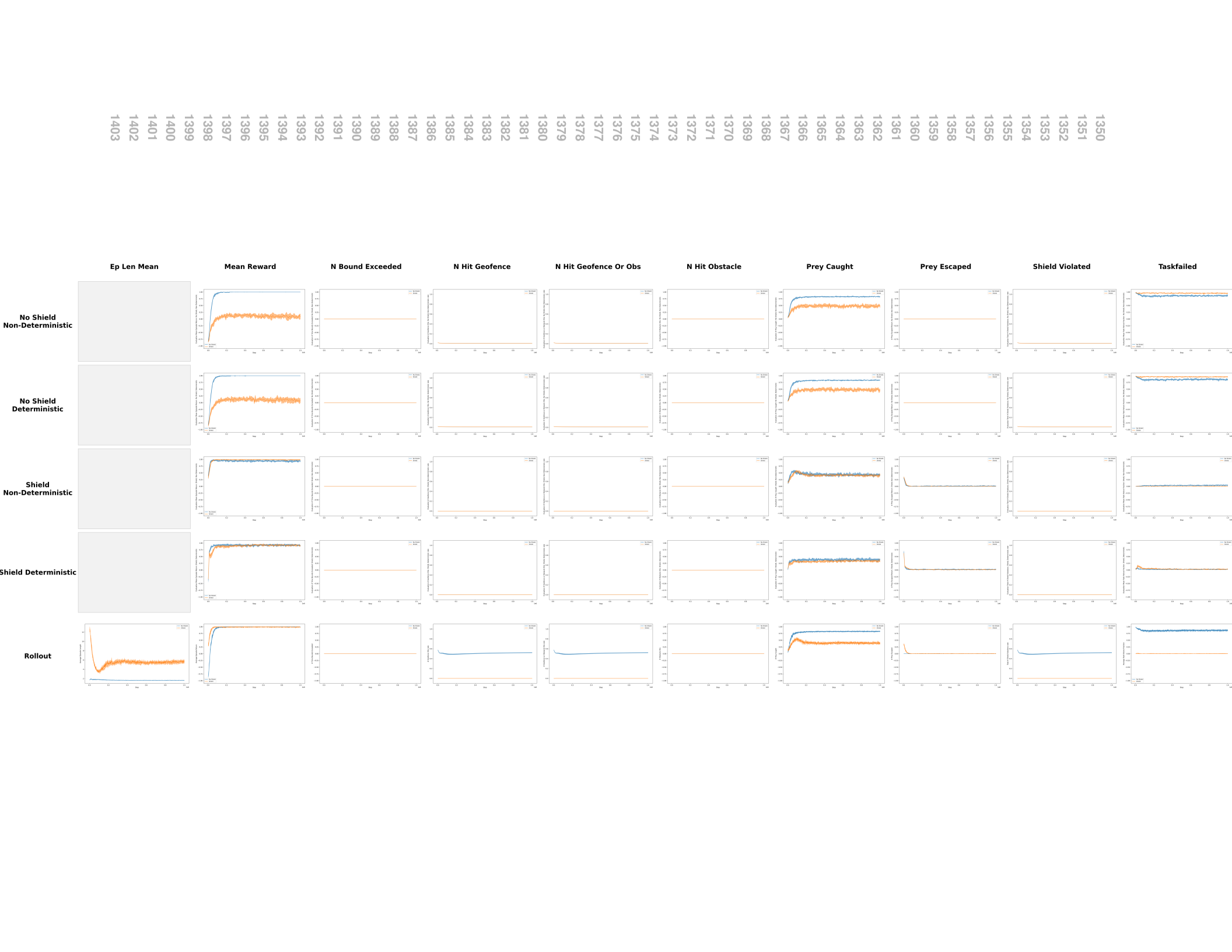
$0 \leq tank \leq 100 \wedge$
$(t = 0 \wedge s = 2 \wedge tank \leq 98) \vee$
$(s = 1 \wedge t = 1 \wedge tank \leq 98) \vee$
$(t = 1 \wedge s = 0 \wedge tank \leq 98) \vee$
$(s = 0 \wedge t = s \wedge tank \leq 98) \vee$
$(t = 0 \wedge s = 1 \wedge tank \leq 98) \vee$
$(s = 1 \wedge t = 2 \wedge 1 \leq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

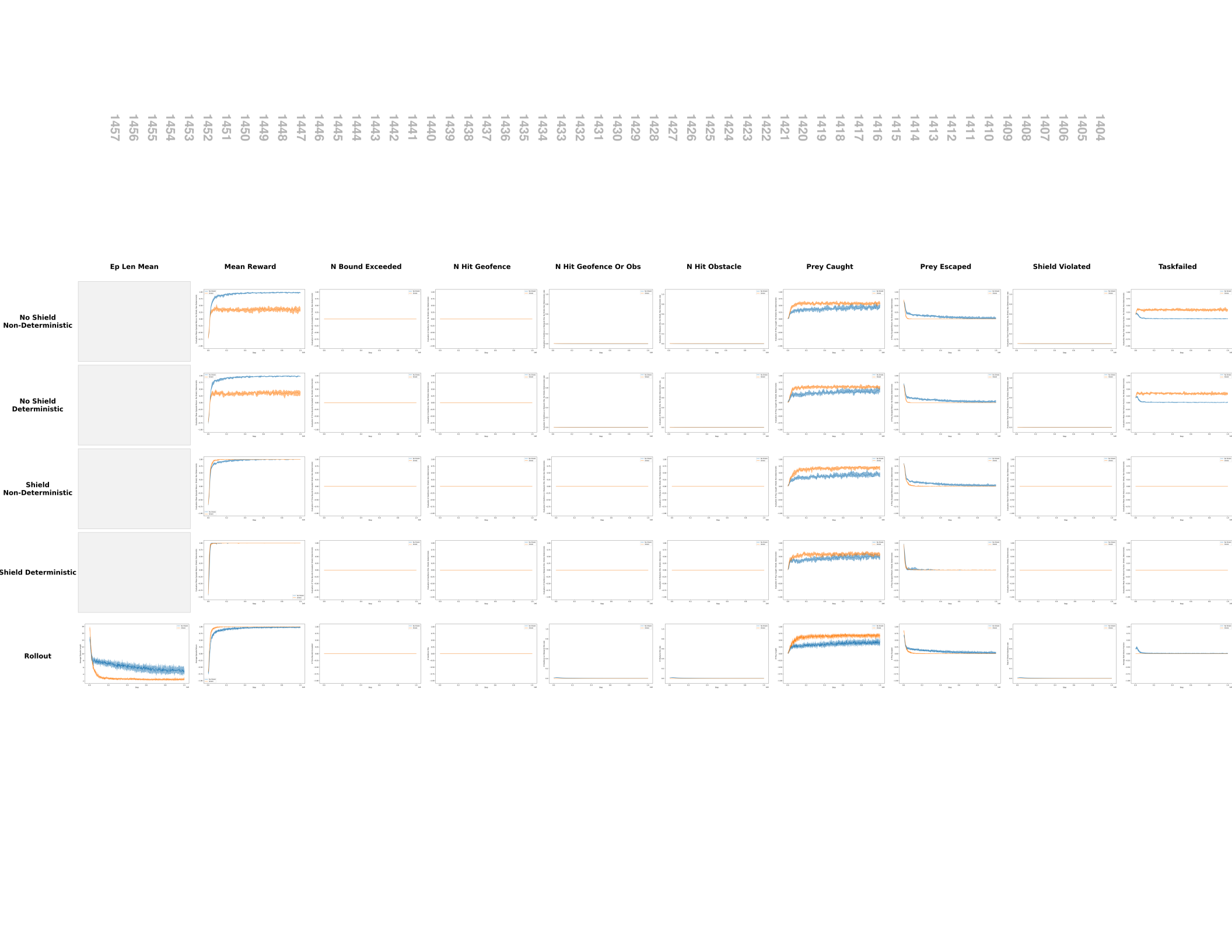
Theorems

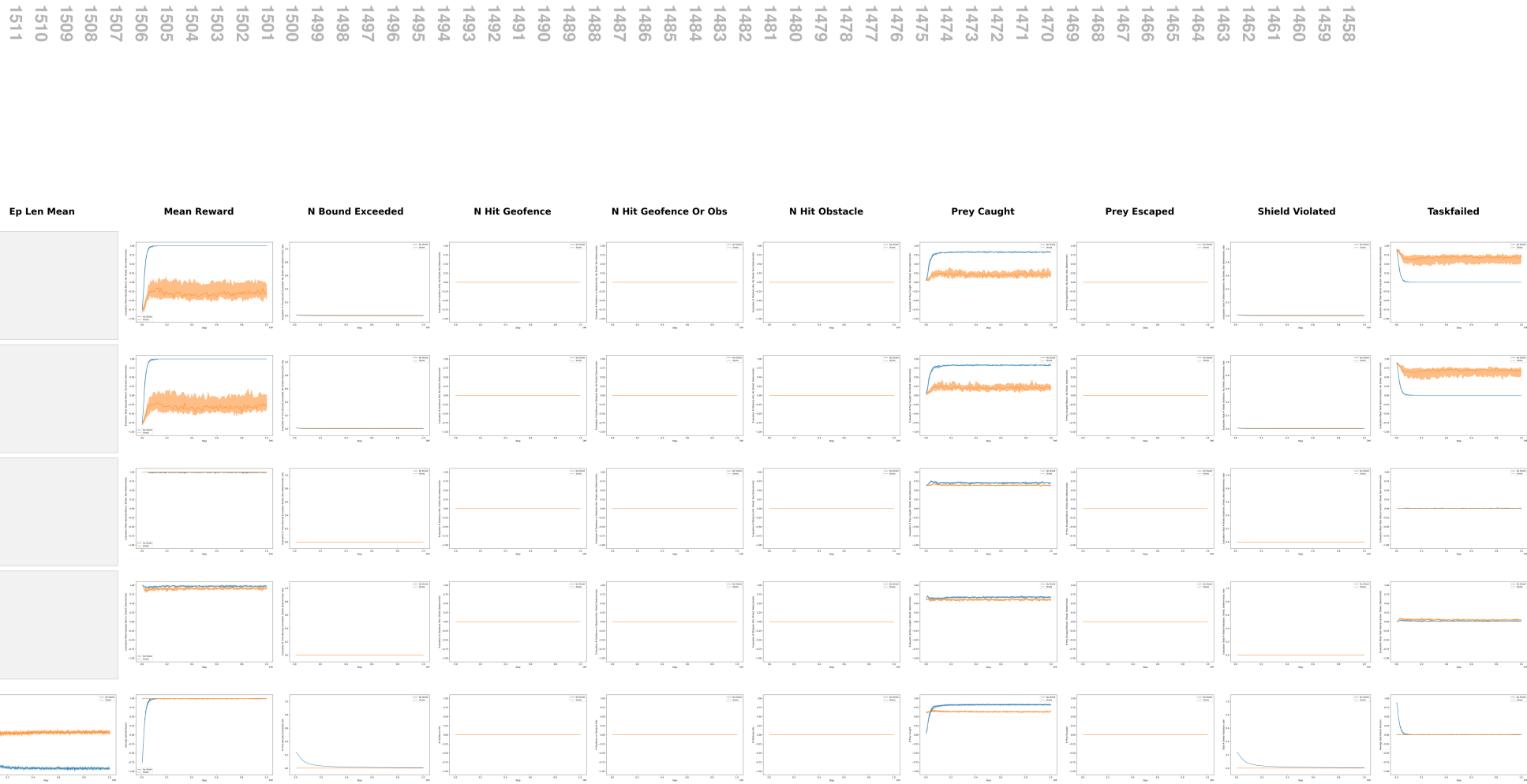
$\Box\ 0 \leq tank, tank \leq 100$
$\Box\ 0 \leq t, t \leq$ NUM_NEXTS
$\Box\ open \wedge \bigcirc \neg open \rightarrow \bigcirc\bigcirc\ \neg open$
$\Box\ \neg open \wedge \bigcirc open \rightarrow \bigcirc\bigcirc\ open$

End Module

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


1403 1402 1401 1400 1399 1398 1397 1396 1395 1394 1393 1392 1391 1390 1389 1388 1387 1386 1385 1384 1383 1382 1381 1380 1379 1378 1377 1376 1375 1374 1373 1372 1371 1370 1369 1368 1367 1366 1365 1364 1363 1362 1361 1360 1359 1358 1357 1356 1355 1354 1353 1352 1351 1350
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


1457
1456
1455
1454
1453
1452
1451
1450
1449
1448
1447
1446
1445
1444
1443
1442
1441
1440
1439
1438
1437
1436
1435
1434
1433
1432
1431
1430
1429
1428
1427
1426
1425
1424
1423
1422
1421
1420
1419
1418
1417
1416
1415
1414
1413
1412
1411
1410
1409
1408
1407
1406
1405
1404
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


1511 1510 1509 1508 1507 1506 1505 1504 1503 1502 1501 1500 1499 1498 1497 1496 1495 1494 1493 1492 1491 1490 1489 1488 1487 1486 1485 1484 1483 1482 1481 1480 1479 1478 1477 1476 1475 1474 1473 1472 1471 1470 1469 1468 1467 1466 1465 1464 1463 1462 1461 1460 1459 1458
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

1565 1564 1563 1562 1561 1560 1559 1558 1557 1556 1555 1554 1553 1552 1551 1550 1549 1548 1547 1546 1545 1544 1543 1542 1541 1540 1539 1538 1537 1536 1535 1534 1533 1532 1531 1530 1529 1528 1527 1526 1525 1524 1523 1522 1521 1520 1519 1518 1517 1516 1515 1514 1513 1512

