Adaptive Shielding via Parametric Safety Proofs

YAO FENG[*], Tsinghua University, China
JUN ZHU, Tsinghua University, China
ANDRÉ PLATZER, Karlsruhe Institute of Technology (KIT), Germany
JONATHAN LAURENT[†], KIT, Germany and Carnegie Mellon University, USA

A major challenge to deploying cyber-physical systems with learning-enabled controllers is to ensure their safety, especially in the face of changing environments that necessitate runtime knowledge acquisition. Model-checking and automated reasoning have been successfully used for shielding, i.e., to monitor untrusted controllers and override potentially unsafe decisions, but only at the cost of hard tradeoffs in terms of expressivity, safety, adaptivity, precision and runtime efficiency. We propose a programming-language framework that allows experts to statically specify *adaptive shields* for learning-enabled agents, which enforce a safe control envelope that gets more permissive as knowledge is gathered at runtime. A shield specification provides a safety model that is parametric in the current agent's knowledge. In addition, a nondeterministic inference strategy can be specified using a dedicated domain-specific language, enforcing that such knowledge parameters are inferred at runtime in a statistically-sound way. By leveraging language design and theorem proving, our proposed framework empowers experts to design adaptive shields with an unprecedented level of modeling flexibility, while providing rigorous, end-to-end probabilistic safety guarantees.

CCS Concepts: • **Computer systems organization** → **Embedded and cyber-physical systems**; • **Software and its engineering** → **Formal methods**; • **Computing methodologies** → *Reinforcement learning*.

Additional Key Words and Phrases: Safe Reinforcement Learning, Programming Languages, Differential Dynamic Logic, Statistical Inference, Hybrid Systems

ACM Reference Format:
Yao Feng, Jun Zhu, André Platzer, and Jonathan Laurent. 2025. Adaptive Shielding via Parametric Safety Proofs. *Proc. ACM Program. Lang.* 9, OOPSLA1, Article 109 (April 2025), 28 pages. https://doi.org/10.1145/3720450

1 Introduction

Learning-based methods such as reinforcement learning have shown great promise in the fields of autonomous driving [28, 38] and robot control [19, 37, 39]. However, their deployment in the real-world has been held back by reliability and safety concerns. The use of formal methods has been suggested to guarantee the safety of learning-enabled systems, both *after* and *during* training [10, 15, 16, 20]. Many different approaches have been proposed to do so, which share as a foundation the idea of *sandboxing* or *shielding* [2, 15, 29, 43]. In this framework, the intended actions of a learning-enabled agent are monitored at runtime and overridden by appropriate fallbacks whenever they cannot be proved safe with respect to a model of the environment.

[*]Yao Feng contributed a majority of the case studies and experiments.
[†]Jonathan Laurent contributed a majority of the theoretical framework and writing.

Authors' Contact Information: Yao Feng, Tsinghua University, Beijing, China, y-feng23@mails.tsinghua.edu.cn; Jun Zhu, Tsinghua University, Beijing, China, dcszj@mail.tsinghua.edu.cn; André Platzer, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, platzer@kit.edu; Jonathan Laurent, KIT, Karlsruhe, Germany and Carnegie Mellon University, Pittsburgh, USA, jonathan.laurent@kit.edu.

ACM 2475-1421/2025/4-ART109
https://doi.org/10.1145/3720450

Most existing approaches leverage automated techniques for proving safety such as reachability analysis [26, 43], LTL model-checking [2, 27] or Hamilton-Jacobi solving [13]. Such techniques can be applied *offline* to precompute numerical, table-based control envelopes that indicate what actions are safe in every possible state of the system. Alternatively, they can be used at runtime to analyze the safety of different actions with respect to the current state. Such *online* methods mostly offer only bounded-horizon guarantees and must typically make aggressive tradeoffs to regain efficiency at the cost of precision or generality. However, an advantage of online methods is that they allow the model to be updated at runtime as the agent gathers information about its environment. Indeed, in many situations, a fully-specified model of the environment is not known at design-time and using a necessarily conservative static model may lead to overly cautious behavior. In principle, adaptivity can also be offered by offline methods, by precomputing control envelopes for *every* possible model that may be considered at runtime. However, doing so further compounds the scalability challenges of model-checking. Finally, existing approaches to adaptive shielding, whether *online* or *offline*, are either offering limited expressivity to encode model uncertainty (e.g. *bounded disturbance terms* [3] or *finite model families* [16]) or failing to provide rigorous end-to-end guarantees under assumptions that can be easily validated (e.g. methods based on learning Gaussian processes [4, 5, 9, 13]).

This work introduces a programming-language framework for designing *adaptive* safety shields, leveraging a minimal yet crucial amount of human insight to target a very general class of model families, while shifting the full burden of safety analysis offline. In this framework, control envelopes are described by nondeterministic controllers, whose safety is established using differential dynamic logic [31, 32]. Environment models can be parametrized by unknown function symbols. Those function symbols are subject to parametric bounds, whose parameters are accessible to the controller and can be *instantiated and refined at runtime*. Updating these parameters in a way that is sound and efficient is a great challenge in itself. We address this challenge by offering experts a domain-specific language to express nondeterministic inference strategies that are *sound by construction*, and whose nondeterminism is resolved by an *inference policy* that can be programmed or learned in the same way that the *control policy* is.

We illustrate our framework on four case studies, demonstrating its safety and its ability to handle advanced uncertainty models that are beyond the reach of existing methods. We also showcase the possibility of having agents learn how to manage their own safety budget by learning inference policies, which – to the best of our knowledge – has no equivalent in the literature.

2 Overview

In this section, we motivate and illustrate our framework on a series of closely-related examples.

2.1 Extracting Shields from Verified Nondeterministic Controllers

Our framework builds on the idea of extracting runtime controller monitors from provably safe, nondeterministic controllers [15, 29]. The safety of such nondeterministic controllers is established using *differential dynamic logic* (dL) [31, 32], which is designed specifically to verify hybrid systems with both discrete and continuous dynamics. In dL, we use hybrid programs (HPs) to model both controllers and the differential equations of the physical environments they interact with. A summary of the syntax and semantics of hybrid programs can be found in Table 1. Proving the safety of a cyber-physical system using dL typically comes down to proving a modal formula of the form $\mathsf{Init} \rightarrow [(\mathsf{Ctrl}\,;\mathsf{Plant})^*]\mathsf{Safe}$, where Init and Safe are logical formulas while Ctrl and Plant are hybrid programs. This means that under a certain initial condition (Init), no matter how often we execute the discrete-time controller (Ctrl) and let the continuous-time system evolve to the next control cycle (Plant), the safety condition (Safe) is satisfied.

Table 1. Semantics of Hybrid Programs in dL

Syntax	Semantics
$x := e$	Assign the value of e to variable x, leaving all other variables unchanged.
$x := *$	Assign variable x nondeterministically to some real value.
$?Q$	If Q is true, continue running; else abort.
$x' = f(x) \;\&\; Q$	Follow the system of differential equations $x' = f(x)$ for a certain (nondeterministic) amount of time while Q holds true.
$\alpha \cup \beta$	Nondeterministically run either HP α or β.
$\alpha\,;\beta$	Sequentially run β after α.
α^*	Run α repeatedly for any ≥ 0 amount of iterations.

$$\begin{aligned}
\mathsf{Model} &\equiv \mathsf{Init} \rightarrow [(\mathsf{Ctrl}\,;\mathsf{Plant})^*]\ \mathsf{Safe} && (1)\\
\mathsf{Init} &\equiv (A > 0 \wedge B > 0 \wedge T > 0) \wedge (x + v^2/2B \leq e) \\
\mathsf{Ctrl} &\equiv (a := -B) \cup (?(x + vT + AT^2/2 + (v + AT)^2/2B \leq e)\ ;\ a := A) && (2)\\
\mathsf{Plant} &\equiv t := 0\ ;\ \{x' = v,\ v' = a,\ t' = 1\ \&\ t \leq T \wedge v \geq 0\} \\
\mathsf{Safe} &\equiv x \leq e
\end{aligned}$$

Fig. 1. A Simple dL Model of a Train Braking-Control System [33]

For example, the dL model described in Figure 1 designates a nondeterministic train controller that must stop by the *end of movement authority e* located somewhere ahead of the train, as assigned by the train network scheduler [33]. Variable x models the position of the train on its tracks. At every control cycle, the driver can choose between braking with acceleration $-B < 0$ or accelerating with acceleration $A > 0$. It then gives control to the environment until the next control cycle, which happens in at most T seconds. However, the option to accelerate is only acceptable when a sufficient distance remains to accelerate and then brake safely. The safety of this nondeterministic train controller can be established by proving the validity of Formula 1, using the rules and axioms of dL. This can be done interactively in a proof assistant such as KeYmaera X [14]. In particular, doing so requires finding a loop invariant Inv that holds initially, implies the safety property and is preserved by any run of hybrid program Ctrl ; Plant. Here, one can take Inv ≡ Init.

Crucially, one can extract a runtime monitor from the nondeterministic controller shown in Equation 2 and use it as a shield for a learning-enabled agent [15]. For example, one could use reinforcement learning to learn a deterministic train controller that optimizes for speed, energy efficiency and passenger comfort, while guaranteeing safety by overriding acceleration whenever the distance to the end-of-movement authority is insufficient (i.e. the guard in Equation 2 is violated).

2.2 Adding Adaptivity via Parametric Bounds

As illustrated in the previous section, previous work [15, 29] has shown how to extract shields from nondeterministic controllers whose safety is established using differential dynamic logic. However, doing so requires an accurate model of every safety-relevant aspect of the system. In cases where only a conservative model is available, the shield may force the system into an overly cautious behavior. One contribution of this work is to allow the design of *adaptive* shields, whose behavior is refined at runtime as the agent gathers more information about its environment. To do so, we allow environment models to be parameterized by unknown quantities and control envelopes to be parameterized by bounds on these same quantities.

An example of such an adaptive shield is described in Figure 2. It is specified using our proposed *shield specification language*, which we define rigorously in Sections 4 and 5. This example is a variation of the train control system from Figure 1 with a continuous action space: instead of making a binary choice between *accelerating* and *braking*, the agent must select a *commanded acceleration* $u \in [-B, A]$, which is expressed in dL using a nondeterministic assignment $u := *$ followed by a test $?(-B \leq u \leq A)$. In addition, the relationship between the commanded acceleration u and the *actual acceleration* a of the train is governed by an unknown linear function: $a = \theta u + \varphi$ where $\theta > 0$ and φ are unknown real parameters. At runtime, the actual acceleration of the train is measured repeatedly, allowing the agent to compute increasingly precise estimates of the systematic disturbance θ and φ and thus allowing the shield to be increasingly permissive.

The shield specification in Figure 2 consists of eleven sections, each of them introduced by a distinct keyword. CONSTANT introduces real quantities that are known by the agent at runtime such as the maximum commanded acceleration A. UNKNOWN introduces quantities that are *not* known and must be estimated at runtime, namely θ and φ. ASSUME gathers global assumptions about constant and unknown symbols such as $A > 0$ and $\theta > 0$. Such assumptions could be encoded as system invariants instead (see INVARIANT) but separating them from state-dependent invariants leads to greater conceptual clarity and more concise proof obligations (global assumptions are preserved by definition). The CONTROLLER, PLANT, SAFE and INVARIANT sections define a dL model similar in shape to the one already studied in Figure 1. However, the plant model can feature unknown symbols, which it does here via the $v' = \theta u + \varphi$ differential equation. Also the controller can depend on *bound parameters* that constrain these unknowns and that are introduced in the BOUND section. Here, we introduce a lower bound $\underline{\theta}$ on θ, an upper bound $\bar{\theta}$ on θ, and an upper bound $\bar{\varphi}$ on φ. These parameters are instantiated and refined at runtime as the agent is gathering knowledge, using statistical inference. The controller is similar to Equation 2, except that it offers a continuous action space and conservatively assumes a maximum achievable braking rate of $\underline{\theta}B - \bar{\varphi}$. Thus, the control envelope defined by the nondeterministic controller gets increasingly permissive as tighter bounds are obtained on θ and φ.

The invariant can depend on bound parameters but only monotonically, in the sense that it can only be made more permissive by a tightening of the bounds. This is an intuitive requirement: acquiring knowledge must never transition an agent from a state considered safe to a state considered unsafe. Here, this requirement clearly holds since the maximum guaranteed braking rate of $\underline{\theta}B - \bar{\varphi}$ can only get larger as $\underline{\theta}$ and $\bar{\varphi}$ tighten. Finally, NOISE and OBSERVE specify the kind of information that may become available at runtime to compute and refine such bounds. Here, we are assuming that at every control cycle, an estimate of the *actual* acceleration of the system *may* be measured, with some Gaussian noise of standard deviation σ. No guarantee is offered on *when* and *under what conditions* such an observation becomes available. INFER defines a family of provably-sound strategies for updating $\underline{\theta}, \bar{\theta}$ and $\bar{\varphi}$ based on such observations, as we discuss shortly in Section 2.4. Before we do so though, we introduce a last variant of our train control shield that illustrates how *functional* unknowns can be handled, using the concept of a *local bound.*

2.3 Handling Functional Unknowns with Local Bounds

In our previous example from Figure 2, two real-valued unknowns are estimated at runtime. However, our framework can handle a more powerful and flexible form of model-uncertainty in the form of *functional unknowns.* Figure 3 provides an example, considering a train that evolves on tracks of unknown, varying slope. Variable x denotes the train position on an arc-length parametrization

```
CONSTANT $A, B, T, \sigma$
UNKNOWN $\theta, \varphi$
ASSUME $A > 0,\ B > 0,\ T > 0,\ \sigma > 0,\ \theta > 0$
BOUND $\underline{\theta} : \underline{\theta} \leq \theta,\ \bar{\theta} : \bar{\theta} \geq \theta,\ \bar{\varphi} : \bar{\varphi} \geq \varphi$
CONTROLLER
    $u := * \ ;\ ?(-B \leq u \leq A)\ ;\ ?(x + vT + (\bar{\theta}u + \bar{\varphi})T^2/2 + (v + (\bar{\theta}u + \bar{\varphi})T)^2/2(\underline{\theta}B - \bar{\varphi}) \leq e)$
PLANT $t := 0\ ;\{x' = v, v' = \theta u + \varphi, t' = 1\ \&\ t \leq T \land v \geq 0\}$
SAFE $x \leq e$
INVARIANT $(\underline{\theta}B - \bar{\varphi} > 0) \land (x + v^2/2(\underline{\theta}B - \bar{\varphi}) \leq e)$
NOISE $\eta \sim \mathcal{N}(0, \sigma^2)$
OBSERVE $\omega = \theta u + \varphi - \eta$
INFER
    $\underline{\theta}, \bar{\theta}$ := AGGREGATE $i, j : (\omega_j - \omega_i)/(u_j - u_i)$ AND $(\eta_j - \eta_i)/(u_j - u_i)$ WHEN $u_j > u_i$ ;
    $\bar{\varphi}$ := AGGREGATE $i : \omega_i - \bar{\theta}u_i$ AND $\eta_i$ WHEN $u_i \leq 0$
```

Fig. 2. An adaptive shield for a train control system, where the relationship between the commanded and actual train acceleration is governed by an unknown linear function.

of the tracks. A train at position x is subject to an additional acceleration term of $-g \cdot \sin(\theta_x)$ where $g \approx 9.81\,\mathrm{m\,s^{-2}}$ is Earth's gravitational acceleration and θ_x the track angle at coordinate x. We model this influence by defining the train's kinematics as $v' = a + f(x)$, with a the acceleration commanded by the controller and f an unknown function of x. In addition, f is assumed to be k-Lipschitz, globally lower-bounded by $-A$ (ensuring that the train will always move forward when instructed to accelerate at rate A) and upper-bounded by a known constant F. One could simply use this global bound to implement a conservative shield. Our challenge is to do better by estimating superior, local estimates of f based on runtime observations.

We do so by introducing the concept of a *local bound*. Figure 3 defines such a bound, namely $\bar{f} \geq f(x)$. As opposed to the *global* bounds used in the previous section, this bound involves quantity $f(x)$ that is state-dependent. The guarantee that comes with such a definition is that before every control cycle, the inference module must provide a value $\bar{f}$ that is an upper bound on the value of $f(x)$, as evaluated in the current state. The value of $\bar{f}$ can be used in the controller. However, $\bar{f}$ *cannot* be mentioned in the invariant since $\bar{f} \geq f(x)$ may not hold anymore as the plant executes and the train moves further on the tracks. A local bound is only guaranteed to be valid at discrete points in time, after each run of the inference module and right before the controller executes. Thus, we also maintain an upper bound y on $f(x)$ that holds *throughout* the plant and can therefore be used in the invariant. Variable y is updated before the controller executes with the value of $\bar{f}$ whenever the latter provides a tighter bound. Then, it is evolved via the differential equation $y' = kv$, degrading the precision of our bound at a rate proportional to the Lipschitz constant of f to ensure its preservation by the plant ($\mathrm{d}f/\mathrm{d}t \leq \mathrm{d}f/\mathrm{d}x \cdot \mathrm{d}x/\mathrm{d}t \leq k \cdot v$).

We can now derive our invariant, from which the controller's acceleration guard follows. Given a particular state and provided a bound $y \geq f(x)$, we wonder whether the train can be kept safe indefinitely from the current state by fully engaging the brakes. Given a constant, effective braking rate of b and starting with speed v, it can be shown that the distance needed for the train to brake to a full stop is $\mathrm{Bdist}_v(b) \equiv v^2/2b$. Thus, a sufficient condition for the train to be safe indefinitely is

```
CONSTANT $A, B, F, k, \sigma$
UNKNOWN $f(*)$
ASSUME
    $A > 0,\ B > 0,\ T > 0,\ k > 0,\ \sigma > 0,\ F < B,\ A + F > 0,$
    $(\forall x\ -A \leq f(x) \leq F),\ (\forall x\, \forall y\ |f(x) - f(y)| \leq k|x - y|)$
BOUND $\bar{f} : f(x) \leq \bar{f}$
CONTROLLER
    $y := \min(y, \bar{f})$ ;
    $((a := -B) \cup$
     $(?(x + vT + \frac{1}{2}(A + F)T^2 + \mathsf{Bdist}_{v+(A+F)T}(B - \min(F,\ y + k(vT + \frac{1}{2}(A + F)T^2) +$
        $k \cdot \mathsf{Bdist}_{v+(A+F)T}(B - F))) \leq e)$ ; $a := A)$
PLANT $t := 0$ ; $\{x' = v, v' = a + f(x), y' = kv, t' = 1\ \&\ t \leq T \land v \geq 0\}$
SAFE $x \leq e$
INVARIANT $(v \geq 0) \land (y \geq f(x)) \land (x + \mathsf{Bdist}_v(B - \min(F,\ y + k \cdot \mathsf{Bdist}_v(B - F))) \leq e)$
NOISE $\eta \sim \mathcal{N}(0, \sigma^2)$
OBSERVE $\omega = f(x) - \eta$
INFER $\bar{f} := F$ ; $\bar{f} :=$ BEST $i : \bar{f}_i + k|x - x_i|$ ; $\bar{f} :=$ AGGREGATE $i : \omega_i + k|x - x_i|$ AND $\eta_i$
```

Fig. 3. An adaptive shield for a train control system where the railway tracks are assumed to follow an unkown, space-varying slope function. We write $\mathsf{Bdist}_v(b) \equiv v^2/2b$. For simplicity, the train faces a *binary* choice between *acceleration* and *braking* (as in Figure 1 and unlike in Figure 2).

$x + \mathsf{Bdist}_v(B - F) \leq e$, since we can guarantee an effective braking rate of at least $B - F$ using the global bound F on f. However, a stronger guarantee may result from combining the local bound $y \geq f(x)$ and the fact that f is k-Lipschitz. Indeed, since we already know from our naive estimate that the train can stop within distance $\mathsf{Bdist}_v(B - F)$, we also know that the value of f along this trajectory must be upper-bounded by $y + k \cdot \mathsf{Bdist}_v(B - F)$. This gives us an effective braking rate along the stopping trajectory of $B - (y + k \cdot \mathsf{Bdist}_v(B - F))$. In turn, a new stopping distance can be computed from this estimate, yielding the invariant shown in Figure 3. The controller's acceleration guard follows from the invariant, as it simply ensures that the invariant will still hold after executing the plant for time T.

So far, we have seen how adaptive shields can be extracted from nondeterministic, parametric controllers. Our next step is to show how inference modules can also be synthesized that are guaranteed to instantiate such parameters from runtime observations in a statistically sound way.

2.4 Inferring Statistically-Sound Bound Parameters

In this section, we delve into our proposed *inference strategy language*, which is used to specify nondeterministic inference strategies for instantiating bound parameters at runtime. Inference strategies are introduced by the INFER keyword. We consider the strategy from Figure 3 as an example (the strategy from Figure 2 is analyzed in the extended version of this paper [11, Appendix E.1]). An inference strategy consists of a sequence of *inference assignments*, where each assignment computes a value for a particular bound parameter and updates this parameter whenever the new value is tighter than the old one. Our language supports three forms of assignments, which are all represented in the strategy from Figure 3. Each assignment yields a proof obligation that establishes its soundness.

The first assignment $\bar{f} := F$ indicates that the global bound F can always be used as a local bound on $f(x)$. Its soundness is justified by the validity of the following proof obligation, which is automatically generated by our framework and to be proved by the user:

$$\cdots \land (\forall x\, (-A \leq f(x) \leq F)) \land \cdots \;\rightarrow\; f(x) \leq F.$$

Here, the right-hand-side of the implication is simply the definition of $\bar{f}$, where $\bar{f}$ has been substituted into the assignment's right-hand-side. The left-hand-side consists of the global assumptions, from which only the relevant ones are shown here. The second assignment $\bar{f} :=$ BEST $i : \bar{f}_i + k|x - x_i|$ indicates that any previously established local bound $\bar{f}_i$ induces a new bound $\bar{f}_i + k|x - x_i|$ for the current state. The associated proof obligation is:

$$(\forall x\, \forall y\, |f(x) - f(y)| \leq k|x - y|) \land (f(x_i) \leq \bar{f}_i) \;\rightarrow\; f(x) \leq \bar{f}_i + k|x - x_i|,$$

where all irrelevant assumptions have been removed for clarity. To a first approximation, the semantics of this assignment is to compute one such bound for every element in the agent's history and assign the tightest one to $\bar{f}$ if it beats its current value. The third assignment $\bar{f} :=$ AGGREGATE $i : \omega_i + k|x - x_i|$ AND η_i is most interesting and the one that performs statistical inference from observations. To understand it better, let us start from the associated proof obligation:

$$(\forall x\, \forall y\, |f(x) - f(y)| \leq k|x - y|) \land (\omega_i = f(x_i) - \eta_i) \;\rightarrow\; (f(x) \leq \omega_i + k|x - x_i| + \eta_i). \tag{3}$$

The validity of this obligation establishes the inequality $f(x) \leq \omega_i + k|x - x_i| + \eta_i$ for every triple (x_i, ω_i, η_i) in the agent's history. This does not directly give us a usable bound since ω_i and η_i are random variables, only the first one of which is observed. However, given a tolerance-level of ε, we can use the standard tail bound on Gaussians to establish a concrete upper-bound on $f(x)$ that holds with probability at least $1 - \varepsilon$. Here, we get the upper-bound $b_i \equiv \omega_i + k|x - x_i| + \sigma \cdot z_\varepsilon$, where $z_\varepsilon \equiv \sqrt{2} \cdot \mathrm{erf}^{-1}(1 - 2\varepsilon)$ and erf^{-1} is the inverse of the *error function* [6]. Indeed:

$$\mathbb{P}\{f(x) > b_i\} \;\leq\; \mathbb{P}\{\omega_i + k|x - x_i| + \eta_i > b_i\} \;=\; \mathbb{P}\{\eta_i > \sigma \cdot z_\varepsilon\} \;=\; \varepsilon.$$

The first inequality above is a consequence of Equation 3, the middle equality holds by substituting the definition of b_i, and the last equality holds by definition of the error function.

Such a probabilistic bound may be acceptable in cases where measurement noise is small. However, when σ is large, it may be unacceptably conservative. Fortunately, we can get tighter bounds by aggregating multiple independent observations together. More precisely, consider some nonnegative coefficients λ_i that sum up to one. Then, one can show that the following is a bound on $f(x)$ with probability at least $1 - \varepsilon$:

$$b_{\lambda,\varepsilon} \;\equiv\; \sum_i \lambda_i \cdot (\omega_i + k|x - x_i|) \;+\; \sqrt{\sum_i \lambda_i^2} \cdot \sigma \cdot z_\varepsilon. \tag{4}$$

The proof is similar to the one for Equation 3. We have $f(x) > b \rightarrow \omega_i + k|x - x_i| + \eta_i > b$ for all i and b. Thus, taking a convex combination, we have $f(x) > b \rightarrow \sum_i \lambda_i(\omega_i + k|x - x_i| + \eta_i) > b$ for all b. As a consequence:

$$\mathbb{P}\{f(x) > b_{\lambda,\varepsilon}\} \;\leq\; \mathbb{P}\left\{\sum_i \lambda_i(\omega_i + k|x - x_i| + \eta_i) > b_{\lambda,\varepsilon}\right\} \;=\; \mathbb{P}\left\{\sum_i \lambda_i \eta_i > \sigma' z_\varepsilon\right\} \;=\; \varepsilon, \tag{5}$$

where $\sigma' \equiv \sigma\sqrt{\sum_i \lambda_i^2}$ is the standard deviation of $\sum_i \lambda_i \eta_i$, which is a zero-mean Gaussian provided that the choice of λ is uninformed by – and thus *independent* from – the values of η_i and ω_i.

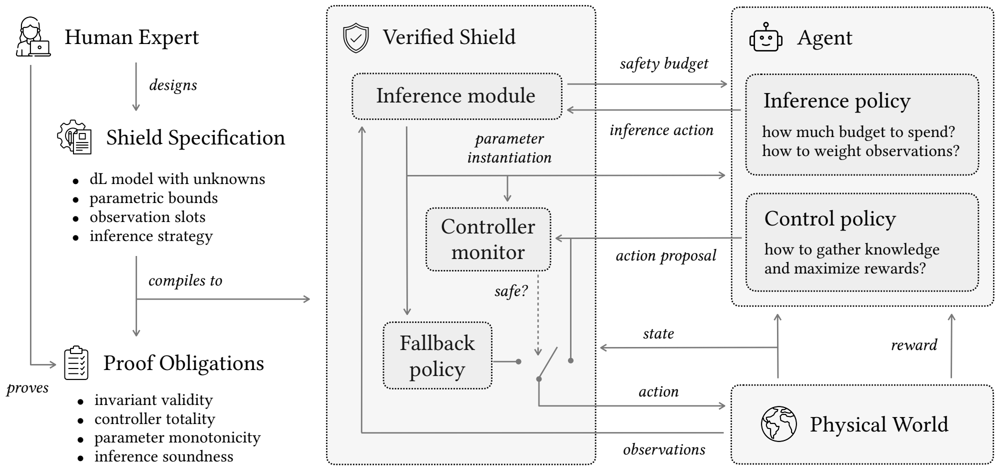

Fig. 4. Adaptive Shielding Overview Diagram

The bound obtained in Equation 4 is the sum of two terms. Minimizing the first term requires putting all the weight on the *closest* measurement, where $k|x - x_i|$ is smallest. On the other hand, minimizing the second term requires spreading the weights uniformly between measurements, resulting in a value of $(\sigma \cdot z_\varepsilon)/\sqrt{n}$, where n is the number of considered measurements. Finding the right tradeoff can be subtle and situation-dependent. Similarly, the value chosen for ε at every control cycle must be substracted from a global *safety budget*, whose proper management constitutes an important challenge. Fortunately, our framework does not force such choices on the shield designers. Rather, it enables the specification of nondeterministic inference strategies, where the choice of ε and λ is determined at each control cycle by a non-soundness-critical *inference policy* that can be learned similarly to the *control policy*.

The AGGREGATE construct from our inference strategy language allows generalizing the reasoning above to a large class of bounds and distributions. Provided a symbolic bound that can be expressed as the sum of an *observable* component and of a *noise* component (syntactically separated using the AND keyword), AGGREGATE computes a weighted average of multiples instances of such bounds, using probabilistic tail inequalities to handle the aggregated noise term. More generally, a strategy written in our proposed inference languages can be compiled into one proof obligation per inference assignment, along with an inference module that integrates with an inference policy at runtime.

2.5 System Overview

Figure 4 provides a diagrammatic summary of our proposed framework for designing adaptive shields. In this framework, human experts are offered a domain-specific language for specifying shields in the form of parametric dL models coupled with nondeterministic inference strategies. A specification written in this language can be automatically compiled into a shield and into a series of proof obligations that are sufficient in establishing its soundness. All obligations are dL formulas, which can be discharged automatically in many cases but also proved interactively in a proof assistant such as KeYmaera X [14]. Obligations include the following:

- The proposed invariant must be preserved when running the controller and the plant in sequence, assuming that all assumptions and bounds hold initially.
- The proposed invariant and the global bounds must imply the postcondition.

$$
\begin{aligned}
\text{Hybrid Program:}\quad & \alpha, \beta \ ::= \ x := e \mid x := * \mid ?P \mid x' = f(x) \ \&\ Q \mid \alpha \cup \beta \mid \alpha ; \beta \mid \alpha^* \\
\text{Formula:}\quad & P, Q \ ::= \ \theta_1 \sim \theta_2 \mid \forall x\, P \mid \exists x\, P \mid [\alpha]P \mid \langle\alpha\rangle P \mid \neg P \mid P \vee Q \mid P \wedge Q \mid P \rightarrow Q \\
\text{Term:}\quad & \theta \ ::= \ r \mid x \mid f(\theta_1, \ldots, \theta_n) \mid |\theta| \mid \theta_1 \odot \theta_2
\end{aligned}
$$

$$
\text{Number literal: } r \in \mathbb{R} \qquad \text{Variable: } x \in \mathsf{Var} \qquad \text{Function symbol: } f \in \mathsf{Symb}
$$

$$
\text{Term comparison: } \sim \in \{=, <, \leq, \geq, >\} \qquad \text{Arithmetic operator: } \odot \in \{+, \times, \min, \max\}
$$

$$
\mathsf{Fun} \equiv \bigcup_{n \in \mathbb{N}} (\mathbb{R}^n \rightarrow \mathbb{R}) \quad \mathsf{Interp} \equiv \mathsf{Symb} \rightarrow \mathsf{Fun} \quad \mathsf{VState} \equiv \mathsf{Var} \rightarrow \mathbb{R}
$$

$$
[\![\alpha]\!] : \mathsf{Interp} \rightarrow \mathcal{P}(\mathsf{VState} \times \mathsf{VState}) \quad [\![P]\!] : \mathcal{P}(\mathsf{Interp} \times \mathsf{VState}) \quad [\![\theta]\!] : \mathsf{Interp} \times \mathsf{VState} \rightarrow \mathbb{R}
$$

Fig. 5. Syntax and semantics of dL. $\mathcal{P}(\cdot)$ denotes the powerset operator.

- The nondeterministic controller must be *total*, in the sense that at least one action should be available in any state where the invariant and bounds are true.
- All inference assignments must be sound.

In the particular case where no functional unknowns are used and all differential equations admit an analytical solution expressible as a multivariate polynomial, all generated proof obligations are decidable. In general, manual proof assistance may be required for certain obligations. Notably, dL has proved effective at reasoning about *unsolvable* differential equations [23, 35].

The generated shield consists of three components. The first one is an *inference module* that is extracted from the expert-defined, nondeterministic inference strategy. The other two are a *controller monitor* and a *fallback policy*, both of which are extracted from the nondeterministic dL controller. At runtime, an untrusted agent interacts with the physical world under the protection of the shield, which overrides any proposed action that is not validated by the controller monitor using its fallback policy. The inference module is tasked to instantiate the model's bound parameters, which the controller monitor and the fallback policy depend on. The inference module is guided by the agent's inference policy, which provides hints on how the safety budget should be spent and how observations should be weighted when building aggregates. The safety budget is initialized with the system's tolerated probability of failure, which is typically a small value such as 10^{-6}. In cases where not enough budget remains to honor the inference policy's recommendation, the inference module skips the associated inference assignment.

Importantly, the inference policy must *not* depend on the value of the observations processed by the inference module and must only base its decisions on the *availability* of these observations and on features of their associated states. Intuitively, this restriction prevents the agent from p-hacking its way to unsafety [30]. Indeed, it would be unsound for the agent to make several measurements of the same quantity and then discard all measurements but the most favorable one. Another requirement enforced by the inference module is that the same observation cannot be reused across control cycles. Indeed, allowing the reuse of observations across control cycles enables an indirect form of cherry-picking since the next state after each control cycle may be influenced by the observations made during this cycle and thus carry some amount of information about it.

3 Background

3.1 Differential Dynamic Logic

Differential dynamic logic (dL) [31, 32] is designed specifically to verify hybrid systems with both discrete and continuous dynamics. The syntax and semantics of dL is summarized in Figure 5. A *state* is defined as a mapping from variable names to real values. An *interpretation* is defined as a mapping from function symbols to real functions of proper arity (possibly zero). The semantics of a *hybrid program* α can be defined by induction on α. For I an interpretation, $[\![\alpha]\!](I)$ denotes the set of all pairs of states (s_1, s_2) such that s_2 is reachable from s_1 by executing α. We provide an intuitive summary of such semantics in Table 1 but formal definitions can be found in the literature [32]. The semantics $[\![P]\!]$ of a *formula* P is defined as the set of all (I, s) pairs such that P is true in state s and under interpretation I. It follows naturally from first-order logic. In addition, $[\alpha]P$ is true if and only if P is true after all runs of the hybrid program α and $\langle\alpha\rangle P$ is true if and only if there exists a run of the hybrid program α after which P is true. Atomic formulas consist in comparisons of *terms*. The semantics $[\![\theta]\!]$ of a term θ maps a pair of an interpretation and a state to a real value. A formula P is said to be *valid* (written $\vDash P$) if it is true in all states and interpretations.

3.2 Notations for Valuations

Given a set of variables V, a *valuation* of V is defined as a function $\rho : V \to \mathbb{R}$ that maps each variable in V to a value. Moreover, a *partial valuation* of V is defined as a partial function $\rho' : V \rightharpoonup \mathbb{R}$ that attributes values to a subset $\operatorname{dom} \rho'$ of variables from V (note the special arrow symbol for partial functions). We denote $\{\}$ an empty valuation. Moreover, if v and v' are valuations of two disjoint subsets of variables V and V', we write $v \cup v'$ the valuation of $V \cup V'$ induced by v and v'.

3.3 Reinforcement Learning

Reinforcement learning (RL) is about learning to make decisions in an environment in such a way to maximize rewards. Environments are typically represented using Markov Decision Processes (MDPs) [41]. An MDP is defined by a tuple $\langle S, A, P, R, \gamma\rangle$ where S is the state space, A is the action space, and $P : S \times A \to \mathcal{D}(S)$ is the transition function, which maps any state-action pair to a probability distribution over new states. $R : S \times A \times S \to \mathbb{R}$ is the reward function, which describes the reward associated with a specific state-action pair and a specific next state. Finally, $\gamma \in (0, 1)$ is the factor by which future rewards are discounted at each time step. The goal of RL is to find an optimal policy $\pi : S \to \mathcal{D}(A)$ for the agent to maximize the expected discounted sum of future rewards: $\mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i R(s_i, a_i)\right]$. In the most common *model-free* RL setting, P and R are unknown and the agent accesses samples of these by directly interacting with the environment.

3.4 Safe Reinforcement Learning via Shielding

This work builds on the Justified Speculative Control framework (JSC) [15], which leverages models written in dL to shield reinforcement learning agents by only allowing provably-safe control actions. JSC crucially relies on the ability to extract monitors [29] from nondeterministic dL controllers. To define those monitors [29], we need to bridge dL models with reinforcement learning environments. Thus, consider a dL model of the kind defined in Equation 1. Such a model induces a state space $\mathsf{State} \equiv \mathrm{FV}(\mathsf{Model}) \to \mathbb{R}$, where a state maps each free variable occuring in the model to a real value. In addition, the controller Ctrl induces an action space that can be defined inductively:

$$\mathcal{A}[\![\alpha \cup \beta]\!] \equiv \mathcal{A}[\![\alpha]\!] \uplus \mathcal{A}[\![\beta]\!], \quad \mathcal{A}[\![\alpha\,;\beta]\!] \equiv \mathcal{A}[\![\alpha]\!] \times \mathcal{A}[\![\beta]\!], \quad \mathcal{A}[\![x := *]\!] \equiv \mathbb{R}, \quad \mathcal{A}[\![x := e]\!] \equiv \mathcal{A}[\![?Q]\!] \equiv \mathbf{1}.$$

In this definition, $X \uplus Y \equiv \{(\mathsf{left}, x) : x \in X\} \cup \{(\mathsf{right}, y) : y \in Y\}$ denotes the *disjoint union* of sets X and Y and $\mathbf{1} \equiv \{\bullet\}$ denotes a set with a single element. Intuitively, an action records a sequence of choices that uniquely characterizes a run of the controller. For example, the discrete controller defined in Equation 2 has an action space $\mathcal{A}[\![\mathsf{Ctrl}]\!] = \mathbf{1} \uplus (\mathbf{1} \times \mathbf{1})$, which is isomorphic to a 2-element set containing a *braking* and an *accelerating* action. The continuous controller used in Figure 2 has an action space $\mathcal{A}[\![\mathsf{Ctrl}]\!] = \mathbb{R} \times \mathbf{1}$. For any state $s \in \mathsf{State}$ and action $a \in \mathcal{A}[\![\mathsf{Ctrl}]\!]$, we write $\mathcal{E}[\![\mathsf{Ctrl}]\!](s, a)$ for the state that results from executing Ctrl while performing the sequence of choices encoded in a. A formal definition is provided in the extended version of this paper [11, Appendix D.1]. From the structure of a nondeterministic controller, one can extract a *controller monitor* that takes as an input a state and an action and returns a Boolean value indicating whether or not the action is permissible.

Lemma 3.1 (Existence of a controller monitor and fallback policy). *Let* Ctrl *be a nondeterministic controller in the form of a* dL *hybrid program that is free of loops, modalities, differential equations, quantifiers and function symbols of nonzero arity. Then, there exists a* computable *function* $\mathcal{M}[\![\mathsf{Ctrl}]\!] : (\mathrm{FV}(\mathsf{Ctrl}) \to \mathbb{R}) \times \mathcal{A}[\![\mathsf{Ctrl}]\!] \to \{\mathsf{true}, \mathsf{false}\}$ *that we call* controller monitor *and such that for all state s and action a, $\mathcal{M}[\![\mathsf{Ctrl}]\!](s, a) = \mathsf{true}$ if and only if $(s,\ \mathcal{E}[\![\mathsf{Ctrl}]\!](s, a)) \in [\![\mathsf{Ctrl}]\!](\{\})$. In addition, given any formula* Inv *such that* $\vDash \mathsf{Inv} \to \langle \mathsf{Ctrl} \rangle\, \mathsf{true}$ *(i.e. there exists an outgoing controller transition for all states in* Inv*), there exists a computable function* $\mathcal{F}[\![\mathsf{Ctrl}]\!] : (\mathrm{FV}(\mathsf{Ctrl}) \to \mathbb{R}) \to \mathcal{A}[\![\mathsf{Ctrl}]\!]$ *that we call* fallback policy *and such that for all state s such that* $(\{\}, s) \in [\![\mathsf{Inv}]\!]$, $(s,\ \mathcal{E}[\![\mathsf{Ctrl}]\!](s,\ \mathcal{F}[\![\mathsf{Ctrl}]\!](s)) \in [\![\mathsf{Ctrl}]\!](\{\})$.

In our train example from Figure 2 (with a *continuous* control input), the action space $\mathcal{A}[\![\mathsf{Ctrl}]\!]$ is isomorphic to $\mathbb{R}$. An action is a commanded acceleration u and the runtime monitor $\mathcal{M}[\![\mathsf{Ctrl}]\!]$ simply evaluates the two tests featured in the controller definition for the selected value of u: it checks that both formula $(-B \le u \le A)$ and formula $(x + vT + (\bar{\theta}u + \bar{\varphi})T^2/2 + (v + (\bar{\theta}u + \bar{\varphi})T)^2/2(\underline{\theta}B - \bar{\varphi}) \le e)$ evaluate to *true* in the current state and for the current values of bound parameters $\underline{\theta}$, $\bar{\theta}$ and $\bar{\varphi}$. In our train example from Figure 3 (with a *binary* control input), the action space is isomorphic to $\{\mathsf{brake}, \mathsf{accelerate}\}$. The braking action is always allowed. To determine whether or not to allow acceleration, the monitor evaluates the test from the associated branch of the nondeterministic controller from Figure 3. In general, controller monitors can be derived from shield specifications via a simple syntactic transformation. We formalize this transformation and provide a proof of Lemma 3.1 in the extended version of this paper [11, Appendix D]. The KeYmaera X toolchain features a tool called ModelPlex [29] for extracting executable controller monitors from dL models.

The existence of a fallback policy is guaranteed by the assumption $\vDash \mathsf{Inv} \to \langle \mathsf{Ctrl} \rangle\, \mathsf{true}$, which indicates that there exists *at least* one way to resolve the nondeterministic choices in Ctrl in such a way to pass all tests (i.e. not trigger any monitor alert). The corresponding action can be computed systematically by solving a decidable SMT problem, hence the existence of a decidable fallback policy (see the extended version of this paper for details [11, Appendix D.3]). However, for the sake of monitoring efficiency, tools like ModelPlex require the user to specify an *explicit* fallback policy on a case-by-case basis. Doing so is never a problem in practice since proving $\vDash \mathsf{Inv} \to \langle \mathsf{Ctrl} \rangle\, \mathsf{true}$ requires characterizing such a policy anyway. In particular, an explicit fallback policy can be automatically extracted from a constructive proof of this formula [7]. In our train examples, a possible fallback policy is to brake with maximal force, which corresponds to action $-B$ in the continuous case and action “brake” in the discrete case.

4 Adaptive Shielding

4.1 Shield Specifications

We define a language to specify adaptive shields via parametric safety models. Models can assume a set of parametric bounds on a number of unknown quantities. The derived runtime controller is parametric in these same bounds, which are maintained and improved at runtime by a derived inference module. Examples of *shield specifications* are available in Figures 2 and 3. Formally, a *shield specification* is defined by a tuple of the following elements:

Const: a set of *constants*, which are dL symbols of arity zero whose value is known at runtime.

Unknown: a set of *unknowns*, which are dL symbols of fixed arity, whose value is *not* known at runtime and can be *estimated* by the inference module using observations.

Assum: a set of *assumptions*, which are dL formulas representing global assumptions about the constants and unknowns. Assumptions can feature symbols but no free variables. When clear from the context, we also write Assum for the *conjunction* of all assumptions.

StateVar: a set of *state variables*, which are dL variables used to represent model state. In our concrete syntax, the set of state variables is inferred as the set of all mentioned free variables that are not parameters, observation variables or noise variables (see below).

Param, Dir, Bound: a set of *parametric bounds*. Param is a set of dL variables called *bound parameters* (or *parameters* for short). Each parameter p is mapped to a dL formula Bound_p that is either *monotone* or *anti-monotone* with respect to p, depending on *bound direction* $\mathsf{Dir}_p \in \{\mathsf{up}, \mathsf{lo}\}$. In our concrete syntax, the bound direction can be omitted whenever easily inferrable. For θ a dL term, we write $\mathsf{Bound}_p[\theta]$ for the result of substituting θ for p in Bound_p. If $\mathsf{Dir}_p = \mathsf{up}$, we call Bound_p an *upper-bound* and we mandate that it is monotone, meaning that $\vDash \forall xy\ (x \leq y \rightarrow \mathsf{Bound}_p[x] \rightarrow \mathsf{Bound}_p[y])$. If $\mathsf{Dir}_p = \mathsf{lo}$, we call Bound_p a *lower-bound* and we mandate that it is anti-monotone: $\vDash \forall xy\ (x \leq y \rightarrow \mathsf{Bound}_p[y] \rightarrow \mathsf{Bound}_p[x])$. Bound_p can feature constants, unknowns and state variables. It cannot feature any parameter other than p. It is said to be *local* if it contains a free occurence of a state variable. Otherwise, it is said to be *global*. The associated parameter p is also called *local* or *global* accordingly. We write LParam the set of *local* parameters and GParam the set of *global* parameters. We write $\mathsf{GBound} \equiv \bigwedge_{p \in \mathsf{GParam}} \mathsf{Bound}_p$ for the conjunction of all global bounds. Finally, when clear from the context, we use the notational shortcut $\mathsf{Bound} \equiv \bigwedge_{p \in \mathsf{Param}} \mathsf{Bound}_p$ for the conjunction of all bounds. For $b \in \mathsf{Param} \rightharpoonup \mathbb{R}$, we also write $\mathsf{Bound}[b] \equiv \bigwedge_{p \in \mathrm{dom}\, b} \mathsf{Bound}_p[b(p)]$.

Ctrl: a *controller*, in the form of a (nondeterministic) dL hybrid program that is free of loops, modalities, differential equations and quantifiers. The controller can involve constants, parameters and state variables but *no unknowns*. Only state variables can be modified. For $c \in \mathsf{Const} \rightarrow \mathbb{R}$ and $b \in \mathsf{Param} \rightarrow \mathbb{R}$, we write $\mathsf{Ctrl}[c, b]$ for the result of substituting all constants and parameters in Ctrl by their value according to c and b. Thus, $\mathsf{Ctrl}[c, b]$ has no symbols and its free variables are all in StateVar. For any (c, b) pair, Lemma 3.1 guarantees that a controller monitor and a fallback policy can be extracted from $\mathsf{Ctrl}[c, b]$.

Plant: a dL hybrid program that represents the *physical environment*. As opposed to the controller, it can feature unknowns but no parameters. Only state variables can be modified.

Safe: a dL formula that represents the *safety constraint* under which the system must operate. It can involve constants, unknowns and state variables but no parameters, since the safety condition cannot change over time as knowledge is gathered.

Inv: an *invariant*, in the form of a dL formula. The invariant can feature constants, state variables and unknowns but only *global* parameters. Local parameters are not allowed in the invariant. The following proof obligations are generated that involve the invariant:

(1) The invariant must imply the postcondition: $\vDash \mathsf{Assum} \land \mathsf{GBound} \land \mathsf{Inv} \rightarrow \mathsf{Safe}$.
(2) The invariant must be preserved: $\vDash \mathsf{Assum} \land \mathsf{Bound} \land \mathsf{Inv} \rightarrow [\mathsf{Ctrl}\,;\mathsf{Plant}]\ \mathsf{Inv}$.
(3) There always exists a safe controller action: $\vDash \mathsf{Assum} \land \mathsf{Bound} \land \mathsf{Inv} \rightarrow \langle \mathsf{Ctrl} \rangle\ \mathsf{true}$.
(4) The invariant must be *monotone* with respect to upper-bound parameters and *anti-monotone* with respect to lower-bound parameters.

NoiseVar, Noise: a set of *noise variable declarations*. NoiseVar is a set of dL variables that can be used in the definition of observations and that model samples from *mutually independent* random variables. Each noise variable $\eta \in \mathsf{NoiseVar}$ is mapped to an expression Noise_η that denotes a probability distribution. $\mathsf{NoiseVar}_\eta$ can be either $\mathcal{N}(\theta_1, \theta_2)$ for a normal distribution, $\mathcal{U}(\theta_1, \theta_2)$ for a uniform distribution or $\mathcal{B}(\theta_1)$ for a Bernouilli distribution. In all cases, θ_1 and

θ_2 are dL terms that can feature constants and state variables. We write $[\![\mathsf{Noise}]\!] : (\mathsf{Const} \to \mathbb{R}) \to \mathsf{NoiseVar} \to \mathsf{Distr}(\mathbb{R})$ for the semantic denotation of Noise.

ObsVar, Obs: a set of *observations*. ObsVar is a set of dL variables called *observation variables*. Each observation variable ω is mapped to a defining dL term Obs_ω that may feature constants, unknowns, state variables and noise variables (but no bound parameter).

Infer: an *inference strategy*, which defines an inference module that issues concrete values for bound parameters at runtime, using an external *inference policy* for guidance.

The inference strategy can be defined in a custom, domain-specific language that is explained in details in Section 5. Before we dive into this language though, we characterize the abstract requirements that an inference module must fulfill in Section 4.2. The soundness of our proposed inference strategy language can then be established against these requirements.

4.2 Symbolic Bound Instantiations

An inference strategy Infer induces an action space $\mathcal{A}[\![\mathsf{Infer}]\!]$ for the agent's inference policy, which is defined rigorously in Section 5. In particular, inference actions associated with a single AGGREGATE assignment are (ε, λ) pairs, as seen in Section 2.4. An inference strategy denotes a function $[\![\mathsf{Infer}]\!] : \mathcal{A}[\![\mathsf{Infer}]\!] \to \mathsf{List}(\mathsf{Param} \times \mathsf{SBInst} \times [0, 1])$ that maps an inference action to a list of symbolic inference assignments. A *symbolic inference assignment* consists of a triple (p, e, ε) of a bound parameter p, a *symbolic bound instantiation* e and a failure probability ε.

To illustrate the concept of a *symbolic bound instantiation*, consider the following example statement from Section 2.4: "$\bar{f}$:= AGGREGATE $i : \omega_i + k|x - x_i|$ AND η_i". Provided a (ε, λ) pair and a history of states and observations, the right-hand-side of this statement can be evaluated into a concrete, numerical value proposal for $\bar{f}$. This is done in two stages, the first one of which being the computation of a *symbolic bound instantiation* e for $\bar{f}$. For example, if $\varepsilon = 10^{-8}$, $\lambda_2 = 0.3$, $\lambda_5 = 0.7$ and $\lambda_i = 0$ for all $i \notin \{2, 4\}$, we obtain the following symbolic bound instantiation:

$$e \;\equiv\; 0.3 \times (\omega_2 + k|x - x_2|) + 0.7 \times (\omega_5 + k|x - x_5|) + \overline{\mathrm{CDF}}^{-1}_{\eta_2, \eta_5 \sim \mathcal{N}(0,\, \sigma^2)}(0.3\eta_2 + 0.7\eta_5, 10^{-8}). \quad (6)$$

In the expression above, the $\overline{\mathrm{CDF}}^{-1}$ construct provides a symbolic representation of the *inverse tail function* of a probability distribution, also called its *inverse complementary cumulative distribution function* – hence the notation. Given a random expression and a tolerance ε, $\overline{\mathrm{CDF}}^{-1}$ returns the best upper-bound obtainable on this expression that is true with probability $1 - \varepsilon$ at least. For example, we have $\overline{\mathrm{CDF}}^{-1}_{X \sim \mathcal{U}(0,1)}(X + 1, \varepsilon) = 2 - \varepsilon$ for all $\varepsilon \in [0, 1]$ since $\mathbb{P}_{X \sim \mathcal{U}(0,1)}\{X + 1 > 2 - \varepsilon\} = \varepsilon$. Details on how to compute or approximate the $\overline{\mathrm{CDF}}^{-1}$ operator numerically are available in the extended version of this paper [11, Appendix E.2].

Provided some values for the constant symbols, for the current state and for past states and observations from the agent's history, the symbolic bound instantiation from Equation 6 can be in turn evaluated into a real number. There are two reasons for performing such staging. First, it makes it very clear that $[\![\mathsf{Infer}]\!]$ has no access to the actual observation values when building e, which is critical for soundness (as previously discussed in Section 2.4). Second, this pushes the orthogonal challenge of evaluating $\overline{\mathrm{CDF}}^{-1}$ outside of the core inference machinery.

4.2.1 Formal Syntax and Semantics. A sketch of a formal syntax and semantics for symbolic bound instantiations is provided in Figure 6. A symbolic bound instantiation e can be a dL term θ, the sum of two other symbolic bound instantiations, an application of the $\overline{\mathrm{CDF}}^{-1}$ operator or a *guarded expression* (e when P), where P is a quantifier-free, modality-free dL formula. Importantly, the evaluation of a symbolic bound instantiation can fail and return a special value $\perp$. It does so when attempting to access the value of an undefined variable or symbol, or when evaluating (e when P) while P evaluates to false. Support for $\perp$ is important since our framework does not mandate

$$\delta ::= \mathcal{N}(\theta_1,\theta_2) \mid \mathcal{U}(\theta_1,\theta_2) \mid \mathcal{B}(\theta) \qquad e ::= \theta \mid e_1 + e_2 \mid e \text{ when } P \mid \overline{\text{CDF}}^{-1}_{\eta_1\sim\delta_1\cdots\eta_n\sim\delta_n}(\theta_1,\theta_2)$$

$$[\![e]\!] \;:\; (\mathsf{Symb} \to \mathsf{Fun}) \times (\mathsf{Var} \rightharpoonup \mathbb{R}) \to \mathbb{R} \cup \{\bot\}$$

$$[\![\overline{\text{CDF}}^{-1}_{\eta_1\sim\delta_1\cdots\eta_n\sim\delta_n}(\theta_1,\theta_2)]\!](I,v) \in \left\{ b : \mathbb{P}_{\eta_i\sim[\![\delta_i]\!](I,v)}\big([\![\theta_1]\!](I, v\cup(\cup_i\eta_i)) > b\big) \le [\![\theta_2]\!](I,v) \right\} \cup \{\bot\}$$

Fig. 6. Syntax and semantics of symbolic bound instantiations. We only show the contract that $\overline{\text{CDF}}^{-1}$ must fulfill for soundness. Concrete implementations must optimize for small, real return values.

observations to be available at each cycle, making some observation variables possibly unassigned. In addition, “ when ” is necessary for our inference strategy language to support conditionally sound bounds, which we use in the example from Figure 2 and discuss further in Section 5.

4.2.2 Indexed Variables. Finally, symbolic bound instantiations can contain special dL variables, whose name contains an integer index (i.e. ω_2). Semantically, there is nothing special about such variables and we can just regard them as normal variables following a naming convention. The only thing that matters is that for any $x \in V$ and $i \in \mathbb{N}$, $x_i \notin V$ where $V \equiv \mathsf{StateVar} \cup \mathsf{Param} \cup \mathsf{ObsVar} \cup \mathsf{NoiseVar}$. We also introduce some notations to ease the manipulation of indexed variables. For v a partial valuation over V and $i \in \mathbb{N}$, we write $v^{(i)} \equiv \{x_i \mapsto r : v(x) = r\}$ the same valuation in which all domain variables are tagged with index i. For example, if $x, y \in V$ and $v = \{x \mapsto 0, y \mapsto 1\}$, then $v^{(2)} = \{x_2 \mapsto 0, y_2 \mapsto 1\}$. In addition, for P a formula with variables in V, we write $P^{(i)}$ the formula that results from P after tagging every free variable with index i. Finally, we also allow the use of symbolic names as indices. For example, in the inference assignment “AGGREGATE $i : \omega_i + k|x - x_i|$ AND η_i”, ω_i is a normal dL variable whose name follows a particular convention. We introduce a special syntactic transformation for “instantiating” symbolic indices in formulas. For example, if $P \equiv x_i + x_j > 0$, we write $P[i,j\backslash 2,3] \equiv x_2 + x_3 > 0$.

4.3 Soundness of Symbolic Inference Assignments

As mentioned in the previous section, an inference strategy denotes a function $[\![\mathsf{Infer}]\!] : \mathcal{A}[\![\mathsf{Infer}]\!] \to \mathsf{List}(\mathsf{Param} \times \mathsf{SBInst} \times [0,1])$ that maps an inference action to a list of *symbolic inference assignments.* Our proposed inference strategy language is *sound* in the sense that any strategy expressed with it produces *sound* assignments by construction. In turn, a symbolic inference assignment (p, e, ε) is *sound* if e evaluates to a “correct” instantiation of p with probability at least $1 - \varepsilon$. To make this definition more precise, we need a couple of preliminary definitions:

- A *state* is defined as a valuation of the state variables: $\mathsf{State} \equiv \mathsf{StateVar} \to \mathbb{R}$.
- A *bound instantiation* is a partial valuation of the parameters: $\mathsf{Inst} \equiv \mathsf{Param} \rightharpoonup \mathbb{R}$.
- An *observation availability set* is a subset of observation variables indicating what observation were made at one point in time: $\mathsf{ObsAvail} \equiv \mathcal{P}(\mathsf{ObsVar})$.

We can now define the notion of *soundness* for symbolic inference assignments. Intuitively, (p, e, ε) is *sound* if and only if for any possible history of states, observations and bound instantiations and for any intepretation of the model unknowns, updating an instantiation b by assigning the value of e to p preserves the truth of $\mathsf{Bound}[b]$ with probability at least $1 - \varepsilon$. The challenge in formalizing this intuition is to precisely characterize what counts as a *possible* history. We do so in Definition 4.1, building a valuation v that stands for an arbitrary, consistent history.

Definition 4.1 (Sound inference assignment). A symbolic inference assignment (p, e, ε) is said to be *sound* if and only if for any $c \in \mathsf{Const} \to \mathbb{R}$, $u \in \mathsf{Unknown} \to \mathsf{Fun}$, $n \in \mathbb{N}$, $(s_1 \ldots s_n) \in \mathsf{State}^n$, $(z_1 \ldots z_n) \in \mathsf{ObsAvail}^n$ and $(b_1 \ldots b_n) \in \mathsf{Inst}^n$ such that $(c \cup u, s_i) \in [\![\mathsf{Assum} \wedge \mathsf{Inv} \wedge \mathsf{Bound}[b_i]]\!]$

for all $i \leq n$, we have:

$$\mathbb{P}\left\{(c \cup u, s_n) \notin [\![\mathsf{Bound}[b_n[p \leftarrow r]]]\!]\right\} \leq \varepsilon$$

where $r = [\![e]\!](c, v)$, $v = \left(s_n \cup b_n\right) \cup \left(\bigcup_{1 \leq i \leq n} {s_i}^{(i)} \cup {b_i}^{(i)} \cup {o_i}^{(i)}\right)$, $o_i = \{\omega \mapsto [\![\mathsf{Obs}_\omega]\!](c \cup u, s_i \cup \eta_i) : \omega \in \mathsf{Param} \cap z_i\}$ for all $1 \leq i \leq n$, and $\eta_1 \cdots \eta_n \sim [\![\mathsf{Noise}]\!](c)$ i.i.d.

4.4 Compatible Environments

A shield acts as a buffer between an untrusted agent and a *compatible* environment. By *compatible*, we mean that the environment must conform to the assumptions made by the shield specification, without which no safety guarantee can be provided. The following definition formalizes this notion.

Definition 4.2 (Compatible environments). Consider a shield specification $\langle \mathsf{Const}, \ldots, \mathsf{Infer} \rangle$. A *compatible environment* is a tuple $\left\langle \langle S, A, P, R, \gamma \rangle, \varphi_\mathrm{s}, \varphi_\mathrm{a}, \varphi_\mathrm{a}^{-1}, c, u, \alpha, \mu \right\rangle$ where:

- $\langle S, A, P, R, \gamma \rangle$ is a Markov Decision Process.
- $\varphi_\mathrm{s} : S \longrightarrow \mathsf{State}$ is a *state mapping* function that maps environment states to shield states.
- $\varphi_\mathrm{a} : A \longrightarrow \mathcal{A}[\![\mathsf{Ctrl}]\!]$ is a surjective *action mapping*.
- $\varphi_\mathrm{a}^{-1} : \mathcal{A}[\![\mathsf{Ctrl}]\!] \longrightarrow A$ is a reverse action mapping such that $\varphi_\mathrm{a} \circ \varphi_\mathrm{a}^{-1} = \mathrm{id}$.
- $c : \mathsf{Const} \longrightarrow \mathbb{R}$ is an interpretation of the shield's constant symbols.
- $u : \mathsf{Unknown} \longrightarrow \mathsf{Fun}$ is an interpretation of the shield's unknown symbols.
- $\alpha : S \longrightarrow \mathcal{P}(\mathsf{ObsVar})$ is an *observation availability function.*
- $\mu : S \longrightarrow \mathcal{D}(\mathsf{ObsVar} \longrightarrow \mathbb{R})$ is an *observation measurement function.*

In addition, the following should hold:

(1) The shield's plant correctly models the environment's transition function: for all $s, s' \in S$, $a \in A$ and $b \in \mathsf{Inst}$, assuming that *(1)* $s' \in \mathrm{supp}(P(s, a))$ (i.e (s, a, s') is a valid transition), *(2)* $(c \cup u, \varphi_\mathrm{s}(s) \cup b) \in [\![\mathsf{Assum} \wedge \mathsf{Bound} \wedge \mathsf{Inv}]\!]$, and *(3)* $(\varphi_\mathrm{s}(s) \cup b, s'' \cup b) \in [\![\mathsf{Ctrl}]\!](c)$ where $s'' \equiv \mathcal{E}[\![\mathsf{Ctrl}[c, b]]\!](\varphi_\mathrm{s}(s), \varphi_\mathrm{a}(a))$, then we have $(s'', \varphi_\mathrm{s}(s')) \in [\![\mathsf{Plant}]\!](c \cup u)$.

(2) The observation measurement function behaves consistently with Noise and Obs: for all $s \in S$ and $o \in \mathsf{ObsVar} \longrightarrow \mathbb{R}$, then $\mathbb{P}_{\eta \sim [\![\mathsf{Noise}]\!](c)}\{[\![\mathsf{Obs}]\!](c \cup u, \varphi_\mathrm{s}(s) \cup \eta) = o\} = \mu(s)(o)$.

A compatible environment extends a standard RL environment with a formal mapping between this environment and a shield specification's underlying model. It also defines some ground-truth values for all unknown symbols, along with two observation functions for acquiring knowledge about those. We choose to have two separate functions, where α indicates what observations are available in any given state and μ performs an actual measurement of those observations. Such a separation is convenient since the inference policy is allowed to access information about observation availability but *cannot* access actual observation values.

4.5 Shielded Environments and Main Safety Theorem

We define our main shielding algorithm by describing how a shield acts as a *wrapper* around compatible environments, resulting into *shielded environments* that are amenable to RL themselves and in which no unsafe state is reachable by construction.

Definition 4.3 (Shielded environment). Consider a shield specification $\langle \mathsf{Const}, \ldots, \mathsf{Infer} \rangle$ and a compatible environment $\langle \langle S, A, P, R, \gamma \rangle, \ldots, \mu \rangle$ (see Definition 4.2). We define the *induced shielded environment* as an MDP $\left\langle \hat{S}, \hat{A}, \hat{P}, \hat{R}, \gamma \right\rangle$ where:

- A *state* $\hat{s} = (s, h, b_\mathrm{G}, \varepsilon_\mathrm{rem})$ consists of a state s from the original environment, a history h, an instantiation of global parameters b_G and a remaining safety budget $\varepsilon_\mathrm{rem} \geq 0$. In turn, a history is a list of (s', z, b_L) triples where s' is a state, z is an observation availability set and

Algorithm 1 Sampling a transition in a *shielded MDP*

```
procedure SHIELDED_TRANSITION ($\hat{s}$, $\hat{a}$)
    $s, h, b_G, \varepsilon_{rem} \leftarrow \hat{s}$    ▷ Decompose $\hat{s} \in \hat{S}$
    $a_{ctrl}, a_{inf} \leftarrow \hat{a}$    ▷ Decompose $\hat{a} \in \hat{A}$
    $B \leftarrow [\![\mathsf{Infer}]\!](a_{inf})$    ▷ Get list of symbolic bound assignments $B$
    $\Omega = \{\omega_i \in \mathrm{FV}(e) : (p, e, \varepsilon) \in B,\ \omega \in \mathsf{ObsVar}\}$    ▷ Determine observations to make
    $n \leftarrow |h| + 1$    ▷ Get index of history element being built
    $v \leftarrow \varphi_s(s) \cup \varphi_s(s)^{(n)} \cup b_G \cup b_G{}^{(n)}$    ▷ Initialize a valuation for $B$
    for $i \in \{i : \omega_i \in \Omega\}$ do    ▷ For every relevant step $i$ in history
        $s', z', b'_L \leftarrow h_i$
        $o \leftarrow$ sample $\mu(s')$    ▷ Measure observations for step $i$
        $v \leftarrow v \cup \{\omega_i \mapsto o(\omega) : \omega_i \in \Omega,\ \omega \in z'\}$    ▷ Update $v$ with accessible measurements
        $v \leftarrow v \cup \varphi_s(s')^{(i)} \cup b'_L{}^{(i)}$    ▷ Update $v$ with history element $i$
        $h \leftarrow h[i \leftarrow (s', \{\}, b'_L)]$    ▷ Ensure observations are not reused
    for $(p, e, \varepsilon) \in B$ do    ▷ For every symbolic bound assignment in $B$
        if $\varepsilon > \varepsilon_{rem}$ then    ▷ If not enough budget remains
            continue    ▷ We skip the assignment
        $\varepsilon_{rem} \leftarrow \varepsilon_{rem} - \varepsilon$    ▷ Update the remaining safety budget
        $r \leftarrow [\![e]\!](c, v)$    ▷ Evaluate symbolic instantiation with $v$
        if $r \neq \bot$ and ($p \notin \mathrm{dom}\, v$ or $r < v(p)$) then    ▷ Whenever a tighter value is obtained
            $v \leftarrow v[p \leftarrow r][p^{(n)} \leftarrow r]$    ▷ Update $v$ with the new tighter bound
    $b_L \leftarrow \{p \mapsto v(p) : p \in \mathsf{LParam}\}$
    $b_G \leftarrow \{p \mapsto v(p) : p \in \mathsf{GParam}\}$
    assert $\mathrm{dom}\, b_L = \mathsf{LParam}$    ▷ Ensure that all local parameters are set
    $h \leftarrow h \cdot (s, \alpha(s), b_L)$    ▷ Add current state to the inference history
    safe $\leftarrow \mathcal{M}[\![\mathsf{Ctrl}[c,\ b_G \cup b_L]]\!]$    ▷ Instantiate the control monitor
    fallback $\leftarrow \mathcal{F}[\![\mathsf{Ctrl}[c,\ b_G \cup b_L]]\!]$    ▷ Instantiate the fallback policy
    if not safe($\varphi_s(s)$, $\varphi_a(a_{ctrl})$) then    ▷ When the control action is deemed unsafe
        $a_{ctrl} \leftarrow \varphi_a^{-1}(\mathrm{fallback}(\varphi_s(s)))$    ▷ Override it with a fallback action
    $s \leftarrow$ sample $P(s, a_{ctrl})$    ▷ Perform the action in the underlying MDP
    return $(s, h, b_G, \varepsilon_{rem})$    ▷ Return a new value of $\hat{s}$
```

b_L is a local parameter instantiation. Formally, $\hat{S} \equiv S \times \mathsf{Hist} \times (\mathsf{GParam} \to \mathbb{R}) \times [0, 1]$ and $\mathsf{Hist} \equiv \mathsf{List}(S \times \mathcal{P}(\mathsf{ObsVar}) \times (\mathsf{LParam} \to \mathbb{R}))$.

- An *action* $\hat{a} = (a_{ctrl}, a_{inf})$ is defined as a pair of a *control action* and of an *inference action*. Formally, $\hat{A} \equiv \mathcal{A}[\![\mathsf{Ctrl}]\!] \times \mathcal{A}[\![\mathsf{Infer}]\!]$.
- The *transition function* $P : \hat{S} \times \hat{A} \to \mathcal{D}(\hat{S})$ is defined as in Algorithm 1.
- The *reward function* is lifted from the original MDP: if $\hat{s} = (s, h, b_G, \varepsilon_{rem})$, $\hat{a} = (a_{ctrl}, a_{inf})$ and $\hat{s}' = (s', h', b'_G, \varepsilon'_{rem})$, then $\hat{R}(\hat{s}, \hat{a}, \hat{s}') = R(s, a_{ctrl}, s')$.

Note that in our definition, a state of the shielded MDP does not contain any information about past observation values. Keeping such knowledge from the agent is crucial for soundness since the agent could otherwise engage in cherry-picking. Instead, the history component of a state only indicates which past observations are available and not used already.

Algorithm 1 rigorously defines the process of sampling a transition from a shielded environment, indirectly defining the behavior of our proposed adaptive shield (see Figure 4 for a synthetic view).

Lines 4 to 24 describe the execution of the inference module, which computes valuations b_{G} and b_{L} for global and local bound parameters respectively while updating the history h and the remaining safety budget ε_{rem}. Lines 25 to 28 instantiate the controller monitor and fallback policy using b_{G} and b_{L}. If the action proposed by the agent is rejected by the controller monitor, it is overriden via the fallback policy. Finally, line 29, executes the resulting action in the real environment.

Note that we rely on a formalization trick where Algorithm 1 performs observation measurements lazily (Line 10), sometimes on past states, rather than performing them eagerly and storing them. Although physically unrealistic, this formalization enables us to stay within the standard MDP framework that is dominant in RL. Concrete implementations would of course perform observations eagerly and "secretely" cache them, offering the same interface. An alternative choice would have been to define our shielding algorithm using framework of Partially-Observed Markov Decision Processes (POMDPs) [24]. However, doing so would complicate our proofs and definitions for little gain. Finally, instantiating the controller monitor (Line 25) requires *every* global and local parameter to be mapped to a real value. This trivially holds for global parameters, by definition of a shielded environment state (Definition 4.3). However, this does not obviously hold for *local* parameters, hence the assertion on Line 23. It is the inference strategy itself that must guarantee the validity of this assertion, which it does by statically enforcing that every local parameter is provided a *default* value that is independent from observations or other local parameters. In the example from Figure 3, the first assignment $\bar{f} := F$ serves this role. We can now state our main safety theorem (a proof sketch is provided in the extended version of this paper [11, Appendix F.1]).

Theorem 4.4 (Safety of shielded environments). *Consider a shield specification* $\langle \mathsf{Const}, \dots \rangle$ *and a compatible environment* $\langle \langle S, \dots \rangle, \dots, \mu \rangle$ *(see Definition 4.2). Let* $\hat{E}$ *be the resulting shielded MDP (see Definition 4.3). Then, given a safety budget* $\varepsilon \in [0, 1]$, *an initial global bound instantiation* $b \in \mathsf{GParam} \to \mathbb{R}$ *and an initial state* $s \in S$ *such that* $(c \cup u, \varphi_{\mathrm{s}}(s) \cup b) \in [\![\mathsf{Assum} \wedge \mathsf{Bound} \wedge \mathsf{Inv}]\!]$, *any trace in* $\hat{E}$ *starting with state* $\hat{s}_0 = (s, \emptyset, b, \varepsilon)$ *is* safe *with probability at least* $1 - \varepsilon$, *meaning that* $(c \cup u, \varphi_{\mathrm{s}}(s_t)) \in [\![\mathsf{Safe}]\!]$ *for all state* $\hat{s}_t = (s_t, \dots)$ *in the trace.*

4.6 Learning in a Shielded Environment

Adaptive shields such as defined in Section 4.5 can be used to protect *any* agent from acting unsafely in its environment, whether or not it is capable of learning and regardless of how it is implemented. This is emphasized by our choice of formalizing shields as *environment wrappers*, which are agent-agnostic by construction. Still, for the sake of concreteness, it is useful to illustrate the use of Algorithm 1 and Theorem 4.4 in two typical reinforcement-learning scenarios:

Learning in a fixed environment. A shielded agent is placed in a fixed, *compatible* environment. It ignores the value of *unknowns* but those are kept constant across training. A total safety budget is defined for the inference module, which must be small enough to make crashes unlikely (as per Theorem 4.4). Initially, the parameter bounds provided by the inference module are very loose and so the shield's action monitor is equally conservative. However, at each control cycle, the inference module spends some safety budget to improve those bounds, thereby allowing the agent to take increasingly aggressive exploratory moves. When the safety budget is fully spent, the inference module is not allowed to update parameter bounds anymore. However, the agent is still guaranteed to remain safe indefinitely, and in particular as it finishes to optimize its control policy. Section 6 demonstrates this setting by showcasing a train that learns to navigate on a *fixed* circuit.

Meta-learning across a family of environments. In many practical scenarios, autonomous cyber-physical systems must be capable of *quickly* adapting to changing environments without training a dedicated policy from scratch. It is thus useful to train an agent across a *family* of environments so that it can generalize across them. Such a *meta-learning* setting integrates particularly well with our shielding framework. Indeed, we can train a shielded agent through a series of episodes, each of which takes place in a possibly different environment that is nonetheless compatible with the agent's shield (for possibly different values of the unknowns). At the start of each episode, the inference module is reinitialized and a fixed, *per-episode* safety budget is granted. In addition to the state information, the agent is given access to the current bound parameters and the remaining safety budget at all times. It must learn to gather suitable information about its environment and adapt its actions accordingly *within each episode*. In particular, this allows a form of *active sensing*, where the agent learns to act in such a way to receive useful information from the inference module. This also allows optimizing the behavior of the inference module itself by learning *inference policies*, as we discuss in Section 5. In the meta-learning setting, Theorem 4.4 is applied at *each* episode and so a global safety bound can be obtained by multiplying the *per-episode* safety budget by the number of training episodes. Section 6 explores this setting in three different case studies, including one with a train controller that must learn to quickly adapt to different circuits with different slope profiles.

5 The Inference Strategy Language

We define the syntax and semantics of our *inference strategy language* in Figure 7. An inference strategy is defined as a sequence of inference assignments. Each kind of assignment defines its own action space and the action space for a whole strategy is the product of the action spaces of individual assignments. Three kinds of assignments are supported. Any assignment can be attached a WHEN-clause that restricts its applicability, as illustrated in Figure 2 and in the extended version of this paper [11, Appendix E.1]. A direct assignment "$p := \theta$" does not require any input from the inference policy and executing it costs no budget. It yields a single symbolic bound instantiation, which is θ itself. An assignment of the form "$p := \textsc{best}\ i_1, \cdots, i_n : \theta$" expects as an input a list of n-tuples of history indices and yields one bound instantiation $\theta[i_1, \cdots, i_n \setminus j_1, \cdots, j_n]$ for each tuple $(j_1, \cdots, j_n)$. An alternate design would expect no input and consider *every* possible combination of indices through the current history. However, the number of such combinations may grow intractably large, which explains our current pragmatic choice.

Semantics of AGGREGATE. An assignment of the form "$p := \textsc{aggregate}\ i_1, \cdots, i_n : \theta_1\ \textsc{and}\ \theta_2$" takes as an input an (ε, D) pair, where ε indicates how much safety budget should be spent and D is a finite distribution over all n-tuples of history indices. More precisely, D is a sequence of (λ, j) pairs where j indexes a combination of measurements from history and λ is a positive weight attached to it ($\sum_{\lambda,j\in D} \lambda = 1$). It yields a single bound instantiation that performs a weighted average over all instances of the *observable bound component* θ_1, while using the $\overline{\text{CDF}}^{-1}$ operator to handle the *noise component* θ_2. As a requirement, θ_1 can contain observation variables but no noise variables, while θ_2 can contain noise variables but no observation variables. The semantics of AGGREGATE generalizes our reasoning from Section 2.4 (see Equation 5). Indeed, let us assume that $\text{Bound}_p[(\theta_1 + \theta_2)[i \setminus j]]$ is true for any instantiation $j \equiv (j_1, \ldots, j_n)$ of $i \equiv (i_1, \ldots, i_n)$, as guaranteed by the generated proof obligation. Also, let us assume without loss of generality that p is an *upper*-bound. Since Bound_p is monotone in p and thus convex, we know that $\text{Bound}_p[b^*]$ is true where $b^* \equiv \sum_{\lambda,j\in D} \lambda(\theta_1 + \theta_2)[i \setminus j]$. However, b^* is not known at runtime since θ_2 features noise variables that are not observed directly. Instead, our semantics yields instantiation $b \equiv \sum_{\lambda,j\in D} \lambda\theta_1[i \setminus j] + \delta$, where $\delta \equiv \overline{\text{CDF}}^{-1}(\sum_{\lambda,j\in D} \lambda\theta_2[i \setminus j],\ \varepsilon)$. Since Bound_p is monotone, we have $b \geq b^* \rightarrow \text{Bound}_p[b]$. Contraposing, we get $\neg\text{Bound}_p[b] \rightarrow b^* > b$. We can then establish the soundness of (p, b, ε) with

$$\iota \ ::= \ p_1 := e_1 \ ; \ \ldots \ ; \ p_n := e_n \qquad (p_1, \ldots, p_n := e) \ \equiv \ (p_1 := e \ ; \ \ldots \ ; \ p_n := e)$$
$$e \ ::= \ \theta \ \textsc{when} \ G \ \mid \ \textsc{best} \ i_1, \ldots, i_n : \theta \ \textsc{when} \ G \ \mid \ \textsc{aggregate} \ i_1, \ldots, i_n : \theta_1 \ \textsc{and} \ \theta_2 \ \textsc{when} \ G$$

$$\begin{aligned}
\mathcal{A}[\![p_1 := e_1 \ ; \ \ldots \ ; \ p_n := e_n]\!] &= \mathcal{A}[\![p_1 := e_1]\!] \times \cdots \times \mathcal{A}[\![p_n := e_n]\!] \\
\mathcal{A}[\![p := e]\!] &= \mathcal{A}[\![e]\!] \\
\mathcal{A}[\![\theta]\!] &= \mathbf{1} \\
\mathcal{A}[\![\textsc{best} \ i_1, \ldots, i_n : \theta]\!] &= \mathsf{List}(\mathbb{N}^n) \\
\mathcal{A}[\![\textsc{aggregate} \ i_1, \ldots, i_n : \theta_1 \ \textsc{and} \ \theta_2]\!] &= [0,1] \times \mathsf{FinDistr}(\mathbb{N}^n)
\end{aligned}$$

$$\begin{aligned}
[\![\iota]\!] &: \mathcal{A}[\![\iota]\!] \longrightarrow \mathsf{List}(\mathsf{Param} \times \mathsf{SBInst} \times [0,1]) \\
[\![p_1 := e_1 \ ; \ \ldots \ ; \ p_n := e_n]\!]((a_1, \ldots, a_n)) &= [\![p_1 := e_1]\!](a_1) \oplus \cdots \oplus [\![p_n := e_n]\!](a_n) \\
[\![p := e]\!](a) &= [(p, e, \varepsilon) : (e, \varepsilon) \in [\![e]\!](a)] \\
[\![\theta \ \textsc{when} \ G]\!](\bullet) &= [(\theta \text{ when } G, \ 0)] \\
[\![\textsc{best} \ i_1, \ldots, i_n : \theta \ \textsc{when} \ G]\!](J) &= [(\theta[i \backslash j] \text{ when } G[i \backslash j], \ 0) : j \in J]
\end{aligned}$$

$$[\![\textsc{aggregate} \ i_1, \ldots, i_n : \theta_1 \ \textsc{and} \ \theta_2 \ \textsc{when} \ G]\!]((\varepsilon, D))$$
$$= \left[\left(\left(\sum_{\lambda, j \in D} \lambda \cdot \theta_1[i \backslash j] \ + \ \overline{\mathrm{CDF}}^{-1}\left(\sum_{\lambda, j \in D} \lambda \cdot \theta_2[i \backslash j] \, , \ \varepsilon \right) \right) \text{ when } \bigwedge_{\lambda, j \in D} G[i \backslash j] \, , \ \varepsilon \right) \right]$$

Fig. 7. Syntax and semantics of the inference strategy language. Without loss of generality, all bounds are assumed to be upper-bounds (otherwise, substitute some parameter p by $-p$). All WHEN-clauses can be omitted when $G = \text{true}$. We use the *list concatenation* operator $\oplus$ and the *list comprehension* notation. $\mathsf{FinDistr}(X)$ denotes the set of all distributions over set X with finite support: $\mathsf{FinDistr}(X) \equiv \{[(\lambda_1, x_1), \ldots, (\lambda_n, x_n)] : n \in \mathbb{N}, \ x_1, \ldots x_n \in X, \ \lambda_1, \ldots, \lambda_n > 0, \ \sum_i \lambda_i = 1\}$.

the exact same reasoning that was demonstrated in Equation 5:

$$\mathbb{P}(\neg \mathsf{Bound}_p[b]) \ \leq \ \mathbb{P}(b^* > b) \ = \ \mathbb{P}\left(\sum_{i, \lambda \in D} \lambda \cdot \theta_2[i \backslash j] \ > \ \delta \right) \leq \ \varepsilon.$$

Details on how to evaluate or approximate the $\overline{\mathrm{CDF}}^{-1}$ operator are available in the extended version of this paper [11, Appendix E.2]. In the particular case where θ_2 is a linear combinations of Gaussian variables (with possibly state-dependent coefficients), closed-form solutions exist that involve the erf^{-1} function. Concentration inequalities (e.g. Chebyshev, Hoeffding, or Chernoff bounds) can be used to approximate it conservatively in the general case [6].

Proof obligations and soundness. An inference strategy Infer induces a proof obligation in the form of a dL formula, $\mathcal{P}[\![\mathsf{Infer}]\!]$, which is defined inductively in Figure 8. Provided that this obligation is valid, our language guarantees the soundness of all resulting inferences. This is expressed in Theorem 5.1 (a proof is provided in the extended version of this paper [11, Appendix F.2].

Theorem 5.1 (Soundness of the inference strategy language). *Let* $\langle \mathsf{Const}, \ldots, \mathsf{Infer} \rangle$ *a shield specification. If the formula* $\mathcal{P}[\![\mathsf{Infer}]\!]$ *is valid, then* $[\![\mathsf{Infer}]\!](a)$ *is a list of* sound *inference assignments for any inference action* $a \in \mathcal{A}[\![\mathsf{Infer}]\!]$.

$$\begin{aligned}
\mathcal{P}[\![p_1 := e_1 \;;\; \dots \;;\; p_n := e_n]\!] &= \mathcal{P}[\![p_1 := e_1]\!] \land \cdots \land \mathcal{P}[\![p_n := e_n]\!] \\
\mathcal{P}[\![p := e]\!] &= \mathsf{Assum} \land \Big(\bigwedge \mathcal{D}(\mathrm{FV}(e))\Big) \rightarrow \mathcal{P}[\![e]\!](p) \\
\mathcal{P}[\![\theta \text{ WHEN } G]\!](p) &= G \longrightarrow \mathsf{Bound}_p[\theta] \\
\mathcal{P}[\![\text{BEST } i_1, \dots, i_n : \theta \text{ WHEN } G]\!](p) &= G \longrightarrow \mathsf{Bound}_p[\theta] \\
\mathcal{P}[\![\text{AGGREGATE } i_1, \dots, i_n : \theta_1 \text{ AND } \theta_2 \text{ WHEN } G]\!](p) &= G \longrightarrow \mathsf{Bound}_p[\theta_1 + \theta_2]
\end{aligned}$$

$$\mathcal{D}(x_k) = \mathcal{D}(x)^{(k)} \quad \mathcal{D}(p) = \mathsf{Bound}_p \quad \mathcal{D}(\omega) = (\omega = \mathsf{Obs}_\omega) \quad \mathcal{D}(v) = \mathsf{Inv}$$

Fig. 8. Obligation induced by an inference strategy. Per convention, $p \in \mathsf{Param}$, $\omega \in \mathsf{ObsVar}$ and $v \in \mathsf{StateVar}$.

Learning inference policies. A crucial feature of our inference strategy language is its nondeterminism: an inference strategy does not fully specify the behavior of the inference module and key decisions about what measurements to aggregate, how much budget to spend and when are left to a separate *inference policy*. As illustrated in Section 2.4, such a policy must make subtle tradeoffs. However, it is not soundness-critical and amenable to learning. Learning an inference policy is possible within the *meta-learning* setting described in Section 4.6. A *per-episode* safety-budget is fixed and the agent must learn a way to best manage this budget within an episode, in a way that generalizes across a family of environments and allows maximizing returns. Section 6 illustrates this idea in three different case studies.

6 Experimental Evaluation

We evaluate our framework on a series of case studies and demonstrate the claims:

C1 Our framework allows expressing diverse types of model uncertainty in a unified way.
C2 Shielded agents always remain safe.
C3 Adaptive shields allow more aggressive control strategies than non-adaptive shields.
C4 Inference policies can be learned that generalize across settings.
C5 Control policies can be learned that actively seek information as a prerequisite for success.

All case studies are summarized in Table 2. Each case study trains a shielded reinforcement learning agent, either in a fixed environment or using the meta-learning setting described in Section 4.6. We demonstrate the qualitative claim C1 by having each case study illustrate distinct modelling concepts, all of which are listed in Table 2. Claims C2 and C3 are evaluated on all case studies. As shown in Table 3 and validating Theorem 4.4, shielded agents never encounter crashes. Unshielded agents reach unsafe states during training, but also *after* training in all but one environment. Furthermore, disabling the inference module leads to inferior policies in all examples. Claims C4 and C5 are relevant in the *meta-learning* setting, which is explored in three case studies whose results we summarize below. Full details on our experimental protocol, results, and on the shield used for each case study are available in the extended version of this paper [11, Appendix A].

Versatile Train. This case study uses the shield from Figure 3, which is inspired by published models of train control systems [23, 34], with added support for runtime track slope estimation. Our key results are summarized in Figures 9 and 10. We show how different inference policies work best for different kinds of train tracks and noise models. A policy that aggregates observations in *small* batches works better in a setting with irregular tracks (large k) and low sensor noise (small σ), while a policy that aggregates observations in *large* batches works better in settings with regular tracks and high sensor noise. Importantly, inference policies can be learned that are at least competitive with the best hardcoded solution in both cases.

Table 2. Summary of our Experimental Evaluation. Each case study is annotated with references to the main experimental claims C_i that it provides evidence for. A "FIXED" annotation means that learning is conducted in a fixed environment, while a"META" annotation means that the case study involves meta-learning (as defined in Section 4.6). No shielded agent encountered an unsafe state in our experiments (Table 3).

Case Study	**Property**	
SISYPHEAN TRAIN C1, C2, C3, FIXED	Concepts:	*Fixed environment, Concentration inequalities*
	Description:	A train must go up-and-down a hill repeatedly to transport merchandise, protected by a variant of the shield from Figure 3 that assumes uniform measurement noise.
	Results:	Enabling the inference module yields a more efficient controller. Hoeffding bounds beat Chebyshev bounds for inference.
VERSATILE TRAIN C1, C2, C3, C4, META	Concepts:	*Meta-learning, Learned inference policy*
	Description:	A train is exposed to a variety of track topologies and must learn to spend its safety budget wisely in unknown terrains.
	Results:	Different types of tracks call for different inference policies. Appropriate inference policies can be learned automatically.
CROSSING THE RIVER C1, C2, C3, C5, META	Concepts:	*Mapping, Active sensing*
	Description:	A robot equipped with a lamp must cross a bridge located at an unknown position along a river, at night. The lamp sheds light within a fixed radius.
	Results:	The robot successfully learns to approach the river, switch on its lamp and go along the river until it locates the bridge. Disabling the inference module prevents learning. A non-shielded agent frequently falls into the river, even *after* training.
REVISITING ACAS X C1, C2, C3, C4, META	Concepts:	*Discrete model switching, Boolean inference*
	Description:	We consider a variant of the next-generation Airborne Collision Avoidance System (ACAS X) [22]. The agent controls a plane and must learn to react to an intruder aircraft entering its airspace.
	Results:	A provably-safe policy can be learned that attempts to infer whether or not the intruding aircraft flies in ACAS-compliant ways in order to avoid drastic maneuvers whenever possible.

Crossing the River. This case study illustrates the concept of *active sensing* and demonstrates claim C5 in a minimal setting. The goal is for a robot equipped with a lamp to cross a bridge at night. The robot must find the position of the bridge, which translates into a model where the unknown is in the safety property instead of the plant. The robot only observes the bridge when it gets close enough and its lamp is on. In addition to an inference policy, it must learn a control policy that seeks knowledge *explicitly*. We illustrate this learned control policy in Figure 11.

Revisiting ACASX. This case study revisits a classic of cyberphysical-system verification: the next-generation Airborne Collision Avoidance System (ACAS X) [22]. However, rather than assuming a *known* intruder trajectory or a *random-walk* intruder model [25], we propose a new uncertainty model where intruders are either ACAS-compliant (in which case they are also actively trying to avoid collision) or not (in which case they must be treated as adversarial). The agent must attempt to infer the compliance of the intruder so as to avoid drastic maneuvers whenever possible. We illustrate the effectiveness of our learned inference policy in Figure 12.

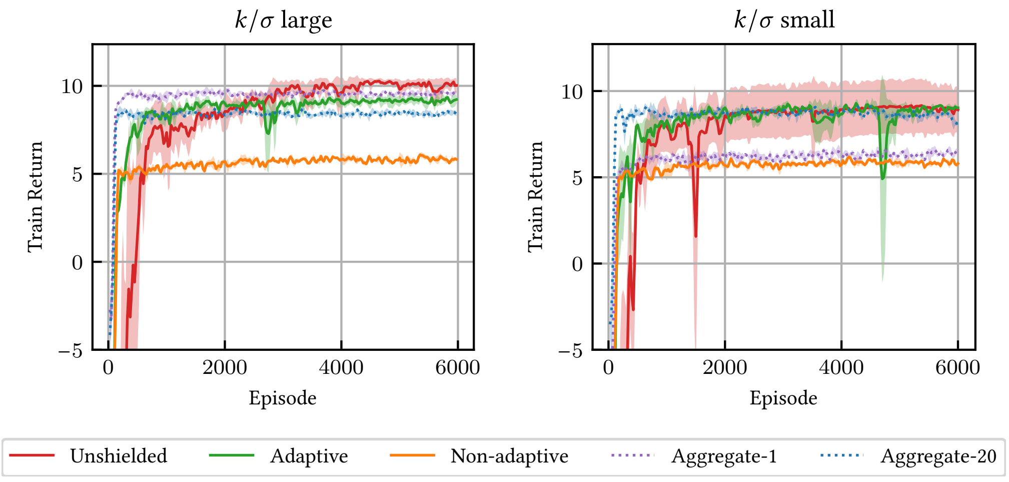

Fig. 9. Returns for the VERSATILE TRAIN case study, for two different combinations of k and σ (and averaged across three random seeds, with standard deviations represented as shaded areas around the mean). The train gets rewarded for reaching the station as quickly as possible (a fixed negative reward is issued at every step before the train reaches destination) and penalized for (unsafely) overshooting its target. Training returns are shown for an unshielded agent (red), an agent using our proposed adaptive shield (green), and a nonadaptive variant where the inference module is deactivated (orange). In addition, the *Aggregate-i* agent is a variant of the *Adaptive* agent that uses a hardcoded inference policy that spends a fixed budget to aggregate all available observations every i steps. Different hardcoded inference policies perform best in different settings but the learned inference policy manages to match the best one in both cases.

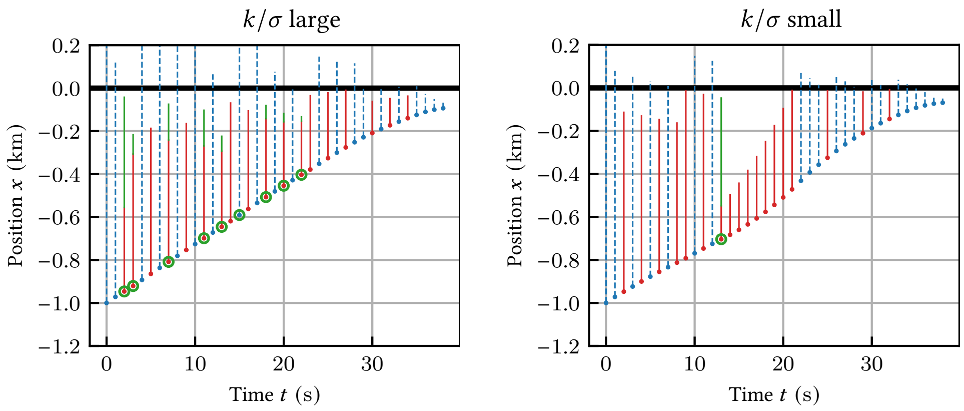

Fig. 10. A visualization of the learned control and inference policies for the VERSATILE TRAIN case study, on two random tracks and for two different combinations of k and σ. We show the position of the train over time using dots. The train needs to stop before $x = 0$ (bold black line). At each time step, a vertical segment indicates the estimated braking distance assuming that the train decides to accelerate for this time step. The segment is drawn in red if the train effectively decides to accelerate and in blue if it decides to brake. A green circle is shown whenever the inference module aggregates existing measurements to improve the local slope estimate. The resulting reduction in the shield's estimated braking distance is plotted as a green segment. In particular, we observe as expected that the learned inference policy aggregates measurements in bigger batches when k/σ is small.

Table 3. Return at testing time (average value of the last 100 episodes during 10,000 evaluation steps) and number of crashes during training and testing time, for different case studies and under different settings. The number of training steps and the maximum episode length for each setting are 80,000/100; 400,000/100; 400,000/100; 400,000/50; and 400,000/40 respectively. Agents are given a total safety budget of 10^{-3} in *fixed-environment* settings (Sisyphean Train) and 10^{-7} per episode in *meta-learning* settings (all others).

Setting	Method	Return	Crashes during training	Crashes during testing
Sisyphean Train	Unshielded	8.3 ± 0.1	110.7 ± 21.5	1.3 ± 0.5
	Adaptive	7.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
	Non-adaptive	5.5 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Versatile Train (k/σ large)	Unshielded	10.0 ± 0.7	102.7 ± 45.0	0.0 ± 0.0
	Adaptive	9.1 ± 0.3	0.0 ± 0.0	0.0 ± 0.0
	Non-adaptive	5.8 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Versatile Train (k/σ small)	Unshielded	9.5 ± 1.2	94.3 ± 6.2	0.7 ± 0.5
	Adaptive	9.1 ± 0.2	0.0 ± 0.0	0.0 ± 0.0
	Non-adaptive	5.8 ± 0.1	0.0 ± 0.0	0.0 ± 0.0
Crossing the River	Unshielded	3.9 ± 0.4	325.7 ± 81.8	0.7 ± 0.5
	Adaptive	4.2 ± 0.7	0.0 ± 0.0	0.0 ± 0.0
	Non-adaptive	−5.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Revisiting ACAS X	Unshielded	6.7 ± 0.5	987.0 ± 66.4	25.0 ± 6.7
	Adaptive	6.6 ± 0.2	0.0 ± 0.0	0.0 ± 0.0
	Non-adaptive	4.5 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

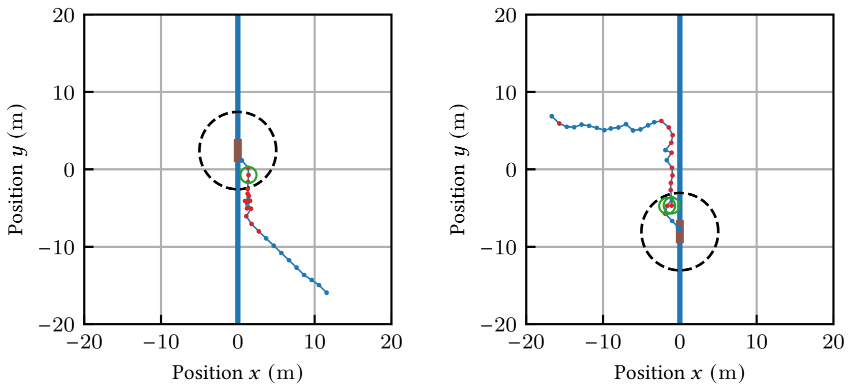

Fig. 11. Visualization of the learned policy for the crossing the river case study. The river is drawn as a vertical blue line at $x = 0$ and the bridge is drawn in brown. The range within which the agent can observe the bridge is plotted as a bold, black dashed circle. The agent's position is marked by a point for each control cycle, which is blue if the lamp is off and red if it is on. A green circle around the current position indicates some safety budget being spent. Regardless of its starting position, the agent learns to go near the river, activate the lamp when it gets close enough and then move along the river until it observes the bridge.

7 Discussion

Runtime overhead. Our framework pushes most of the burden of safety analysis offline via static proof obligations. Thus, it benefits from a small and predictable runtime overhead. This overhead results from running the inference module, monitoring the proposed control actions and executing

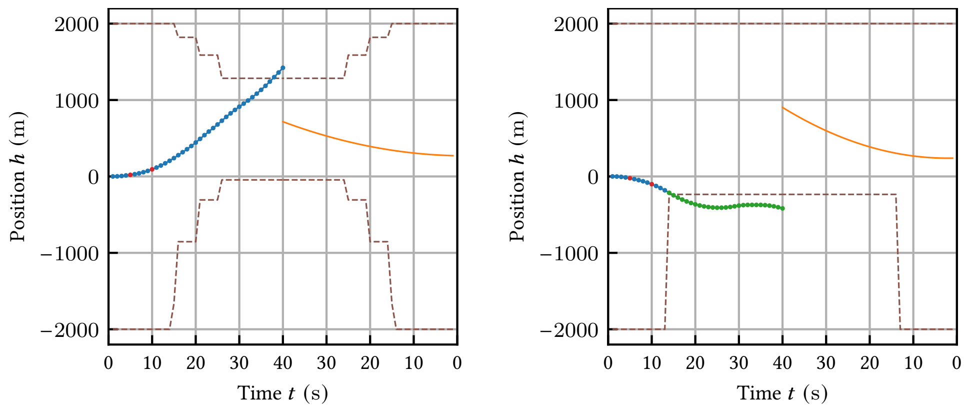

Fig. 12. Learned policy for the Revisiting ACAS X case study, in two different scenarios. The scenario on the right features an ACAS-compliant intruder while the scenario on the left involves a non-compliant intruder. In both cases, the ownship's trajectory is shown on the left of the plot while the intruder's trajectory is shown on the right. The estimated upper and lower bounds for the intruder's final altitude at meeting time is shown for each time step as a dashed line. Red dots on the ownship's trajectory indicate the observation of information that is relevant to estimating the intruder's compliance. Green dots indicate that the intruder has been identified as compliant. When this happens, a less aggressive avoidance maneuver can be executed.

the fallback policy when needed. The complexity of running a cycle of the inference module is linear in the size of the specification and in the size of the inference action being considered (e.g. the number of measurements being aggregated), assuming that the $\overline{\text{CDF}}^{-1}$ operator can be evaluated in linear time (which is true in particular in the Gaussian case or when using concentration inequalities, see details in the extended version of this paper [11, Appendix E.2]). Running the controller monitor is inexpensive since it consists in evaluating a formula whose size does not exceed the size of the shield itself. Finally, assuming that an explicit fallback policy is provided (see discussion in Section 3.4), the cost of computing fallback actions is small and predictable. In our experiments, the total overhead of shielding during training never exceeds 15% (see details in the extended version of this paper [11, Appendix A, Table 4]).

On the complexity of designing adaptive shields. There is an inherent complexity to the engineering task of designing adaptive monitors for rich, realistic model families, which typically necessitates the exploitation of domain-specific insights. Our framework allows designers to focus on this essential complexity, while eliminating the accidental complexity of enforcing end-to-end soundness guarantees that encompass statistical inference. For example, while our paper shows how tricky and error-prone statistical reasoning can be (e.g. reusing measurements is sound within an inference cycle but not across cycles), our framework exposes abstractions that fully protect users from such considerations and generates dL proof obligations that require no probabilistic reasoning. Still, specifying formal models of hybrid systems and proving properties about them using interactive theorem proving is a nontrivial skill to acquire. Doing so is eased by the increasing availability of automation in tools such as KeYmaera X [14, 40]. We show the proof obligations generated by our tool for all case studies in the extended version of this paper [11, Appendix B]. Out of 32 obligations, 27 can be discharged fully automatically or with trivial human guidance (such as providing a single term for instantiating an existential variable). The others require more effort but can be proved in under an hour by an experienced KeYmaera X user [11, Appendix C].

8 Related Work

Safe reinforcement learning is widely studied [8, 17, 18], including several approaches without formal verification. Several policy-search algorithms have been proposed that take into account safety constraints specified separately from the reward signal, but do not offer any guarantee that those constraints will not be violated during training or deployment [1, 42, 44]. In contrast, this work is part of a family of approaches that are often called *sandboxing* or *shielding-based* [2, 15, 43], and where the intended actions of an RL agent are monitored at runtime and overridden by appropriate fallbacks whenever they cannot be proved safe with respect to a model of the environment.

Virtually all existing approaches to shielding aim for full-automation of the shield computation, imposing hard tradeoffs in terms of safety, adaptivity, precision, expressivity and scalability. In circumstances where they run fast enough to be used online, methods based on reachability analysis [21, 26, 43] are naturally amenable to a form of adaptivity. However, they only offer bounded-horizon guarantees and precision comes at the expense of runtime efficiency. Methods based on LTL model-checking [2, 27] or finite-MDP solving [36] can handle infinite time-horizons but require discretization of hybrid dynamics, often at a significant cost in terms of precision and safety. They also tend to scale poorly with increasing dimensionality. Methods based on Hamilton-Jacobi solving [13] face similar challenges. In contrast, our proposed framework allows leveraging human ingenuity by extracting shields from nondeterministic, symbolic controllers that are proved safe using interactive theorem proving. It builds on the *Justified Speculative Control* (JSC) framework [15], which it generalizes in a crucial way to support adaptivity.

Our framework offers a uniquely expressive language for modelling environment uncertainty via arbitrarily constrained function symbols. For example, reachability analysis typically uses bounded disturbance terms [3] to model environment uncertainty, which is sufficient to model the example from Figure 2 but not those from Figure 3 and from the ACASX case study. To the best of our knowledge, these last two examples cannot be accommodated by any pre-existing approach. Another standard way of representing model uncertainty for the purpose of adaptive shielding is to use *Gaussian processes* to model an unknown, state-dependent term added to the system's dynamics [4, 5, 9, 13]. This approach offers a different form of modelling flexibility, where assumptions about functional unknowns are implicitly encoded into prior kernels rather than hard logical constraints. However, this also makes the resulting safety guarantees harder to interpret and fundamentally dependent on assumptions that are nearly impossible to validate experimentally.

Another approach has been proposed to extend the JSC framework with a form of adaptivity, in which an agent starts with a finite set of plausible models and then progressively discards those that are found inconsistent with observations [16]. Our framework handles a more general form of parametric model uncertainty and additionally supports noisy observations and statistical reasoning. Finally, our idea of having experts define nondeterministic inference strategies that are sound by construction and refined via learning – thereby making shielded agents in charge of their own safety budget – has, to the best of our knowledge, no equivalent in the literature.

9 Conclusion

Our framework gives experts full access to the power of differential dynamic logic to build adaptive shields for hybrid systems. Its unique flexibility raises the equally unique challenge of performing statistical inference soundly and efficiently. We tackle this challenge with a mix of language design and learning, introducing the concepts of an *inference strategy* and of an *inference policy* respectively. Future work may explore the integration of reachability analysis and model-checking within our framework, allowing hybrid combinations of symbolic and numerical, offline and online proving.

Data-Availability Statement

An artifact that contains our framework implementation, instructions to replicate all experiments, and formal proofs for all case studies is available at https://doi.org/10.5281/zenodo.14916164 [12].

Acknowledgments

This work was supported by the NSFC (Nos. 62276149, 92370124, 62350080, 92248303, U2341228, 62061136001), BNRist (BNR2022RC01006), Tsinghua Institute for Guo Qiang, and the High Performance Computing Center, Tsinghua University. J. Zhu was also supported by the XPlorer Prize. Finally, this work was supported by the Alexander von Humboldt Professorship program. We thank the anonymous reviewers for their helpful feedback.

References

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. 2017. Constrained Policy Optimization. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 22–31. http://proceedings.mlr.press/v70/achiam17a.html

[2] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe Reinforcement Learning via Shielding. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 2669–2678. doi:10.1609/AAAI.V32I1.11797

[3] Matthias Althoff and John M Dolan. 2014. Online verification of automated road vehicles using reachability analysis. *IEEE Transactions on Robotics* 30, 4 (2014), 903–918. doi:10.1109/TRO.2014.2312453

[4] Felix Berkenkamp and Angela P. Schoellig. 2015. Safe and robust learning control with Gaussian processes. In *14th European Control Conference, ECC 2015, Linz, Austria, July 15-17, 2015*. IEEE, 2496–2501. doi:10.1109/ECC.2015.7330913

[5] Felix Berkenkamp, Matteo Turchetta, Angela P. Schoellig, and Andreas Krause. 2017. Safe Model-based Reinforcement Learning with Stability Guarantees. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 908–918. https://proceedings.neurips.cc/paper/2017/hash/766ebcd59621e305170616ba3d3dac32-Abstract.html

[6] Dimitri Bertsekas and John N Tsitsiklis. 2008. *Introduction to probability*. Vol. 1. Athena Scientific.

[7] Brandon Bohrer and André Platzer. 2020. Constructive game logic. In *Programming Languages and Systems: 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings 29*. Springer International Publishing, 84–111. doi:10.1007/978-3-030-44914-8_4

[8] Lukas Brunke, Melissa Greeff, Adam W Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P Schoellig. 2022. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems* 5 (2022), 411–444. doi:10.1146/ANNUREV-CONTROL-042920-020211

[9] Richard Cheng, Gábor Orosz, Richard M. Murray, and Joel W. Burdick. 2019. End-to-End Safe Reinforcement Learning through Barrier Functions for Safety-Critical Continuous Control Tasks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 3387–3395. doi:10.1609/aaai.v33i01.33013387

[10] Ingy Elsayed-Aly, Suda Bharadwaj, Christopher Amato, Rüdiger Ehlers, Ufuk Topcu, and Lu Feng. 2021. Safe Multi-Agent Reinforcement Learning via Shielding. In *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, Frank Dignum, Alessio Lomuscio, Ulle Endriss, and Ann Nowé (Eds.). ACM, 483–491. doi:10.5555/3463952.3464013

[11] Yao Feng, Jun Zhu, André Platzer, and Jonathan Laurent. 2025. Adaptive Shielding via Parametric Safety Proofs. arXiv:2502.18879 [cs.PL] https://arxiv.org/abs/2502.18879

[12] Yao Feng, Jun Zhu, André Platzer, and Jonathan Laurent. 2025. *Adaptive Shielding via Parametric Safety Proofs*. doi:10.5281/zenodo.14916164

[13] Jaime F. Fisac, Anayo K. Akametalu, Melanie N. Zeilinger, Shahab Kaynama, Jeremy H. Gillula, and Claire J. Tomlin. 2019. A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems. *IEEE Trans. Autom. Control.* 64, 7 (2019), 2737–2752. doi:10.1109/TAC.2018.2876389

[14] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. 2015. KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems. In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings (LNCS, Vol. 9195)*, Amy P. Felty and Aart Middeldorp (Eds.). Springer, 527–538. doi:10.1007/978-3-319-21401-6_36

[15] Nathan Fulton and André Platzer. 2018. Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. In *8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 6485–6492. doi:10.1609/AAAI.V32I1.12107

[16] Nathan Fulton and André Platzer. 2019. Verifiably Safe Off-Model Reinforcement Learning. In *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I (LNCS, Vol. 11427)*, Tomás Vojnar and Lijun Zhang (Eds.). Springer, 413–430. doi:10.1007/978-3-030-17462-0_28

[17] Javier García and Fernando Fernández. 2015. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.* 16 (2015), 1437–1480. doi:10.5555/2789272.2886795

[18] Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, Yaodong Yang, and Alois C. Knoll. 2022. A Review of Safe Reinforcement Learning: Methods, Theory and Applications. *CoRR* abs/2205.10330 (2022). doi:10.48550/arXiv.2205.10330 arXiv:2205.10330

[19] Todd Hester, Michael J. Quinlan, and Peter Stone. 2012. RTMBA: A Real-Time Model-Based Reinforcement Learning Architecture for robot control. In *IEEE International Conference on Robotics and Automation, ICRA 2012, 14-18 May, 2012, St. Paul, Minnesota, USA*. IEEE, 85–90. doi:10.1109/ICRA.2012.6225072

[20] Nathan Hunt, Nathan Fulton, Sara Magliacane, Trong Nghia Hoang, Subhro Das, and Armando Solar-Lezama. 2021. Verifiably safe exploration for end-to-end reinforcement learning. In *HSCC '21: 24th ACM International Conference on Hybrid Systems: Computation and Control, Nashville, Tennessee, May 19-21, 2021*, Sergiy Bogomolov and Raphaël M. Jungers (Eds.). ACM, 14:1–14:11. doi:10.1145/3447928.3456653

[21] Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. 2019. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 169–178. doi:10.1145/3302504.3311806

[22] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Ryan Gardner, Aurora Schmidt, Erik Zawadzki, and André Platzer. 2015. A formally verified hybrid system for the next-generation airborne collision avoidance system. In *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21*. Springer, 21–36. doi:10.1007/978-3-662-46681-0_2

[23] Aditi Kabra, Stefan Mitsch, and André Platzer. 2022. Verified train controllers for the federal railroad administration train kinematics model: Balancing competing brake and track forces. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4409–4420. doi:10.1109/TCAD.2022.3197690

[24] Mykel J Kochenderfer. 2015. *Decision making under uncertainty: theory and application.* MIT press.

[25] Mykel J Kochenderfer, Jessica E Holland, and James P Chryssanthacopoulos. 2012. Next generation airborne collision avoidance system. *Lincoln Laboratory Journal* 19, 1 (2012), 17–33.

[26] Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. 2018. Learning-Based Model Predictive Control for Safe Exploration. In *57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018*. IEEE, 6059–6066. doi:10.1109/CDC.2018.8619572

[27] Bettina Könighofer, Florian Lorber, Nils Jansen, and Roderick Bloem. 2020. Shield Synthesis for Reinforcement Learning. In *Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12476)*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer, 290–306. doi:10.1007/978-3-030-61362-4_16

[28] Xiaodan Liang, Tairui Wang, Luona Yang, and Eric P. Xing. 2018. CIRL: Controllable Imitative Reinforcement Learning for Vision-Based Self-driving. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII (LNCS, Vol. 11211)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer, 604–620. doi:10.1007/978-3-030-01234-2_36

[29] Stefan Mitsch and André Platzer. 2016. ModelPlex: verified runtime validation of verified cyber-physical system models. *Formal Methods Syst. Des.* 49, 1-2 (2016), 33–74. doi:10.1007/s10703-016-0241-z

[30] Randall Munroe. 2011. Significant. http://xkcd.com/882/.

[31] André Platzer. 2008. Differential Dynamic Logic for Hybrid Systems. *J. Autom. Reason.* 41, 2 (2008), 143–189. doi:10.1007/s10817-008-9103-8

[32] André Platzer. 2017. A Complete Uniform Substitution Calculus for Differential Dynamic Logic. *J. Autom. Reason.* 59, 2 (2017), 219–265. doi:10.1007/s10817-016-9385-1

[33] André Platzer and Jan-David Quesel. 2008. Logical Verification and Systematic Parametric Analysis in Train Control.. In *HSCC (LNCS, Vol. 4981)*, Magnus Egerstedt and Bud Mishra (Eds.). Springer, 646–649. doi:10.1007/978-3-540-78929-1_55

[34] André Platzer and Jan-David Quesel. 2009. European Train Control System: A case study in formal verification. In *International Conference on Formal Engineering Methods*. Springer, 246–265. doi:10.1007/978-3-642-10373-5_13

[35] André Platzer and Yong Kiam Tan. 2018. Differential equation axiomatization: The impressive power of differential ghosts. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 819–828. doi:10.1145/3209108.3209147

[36] Stefan Pranger, Bettina Könighofer, Martin Tappler, Martin Deixelberger, Nils Jansen, and Roderick Bloem. 2021. Adaptive shielding under uncertainty. In *2021 American Control Conference (ACC)*. IEEE, 3467–3474. doi:10.23919/ACC50511.2021.9482889

[37] Loris Roveda, Jeyhoon Maskani, Paolo Franceschi, Arash Abdi, Francesco Braghin, Lorenzo Molinari Tosatti, and Nicola Pedrocchi. 2020. Model-Based Reinforcement Learning Variable Impedance Control for Human-Robot Collaboration. *J. Intell. Robotic Syst.* 100, 2 (2020), 417–433. doi:10.1007/s10846-020-01183-3

[38] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. 2016. Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. *CoRR* abs/1610.03295 (2016). arXiv:1610.03295 http://arxiv.org/abs/1610.03295

[39] William D. Smart and Leslie Pack Kaelbling. 2002. Effective Reinforcement Learning for Mobile Robots. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation, ICRA 2002, May 11-15, 2002, Washington, DC, USA*. IEEE, 3404–3410. doi:10.1109/ROBOT.2002.1014237

[40] Andrew Sogokon, Stefan Mitsch, Yong Kiam Tan, Katherine Cordwell, and André Platzer. 2022. Pegasus: Sound Continuous Invariant Generation. *Form. Methods Syst. Des.* 58, 1 (2022), 5–41. doi:10.1007/s10703-020-00355-z Special issue for selected papers from FM'19.

[41] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[42] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. 2019. Reward Constrained Policy Optimization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=SkfrvsA9FX

[43] Jakob Thumm and Matthias Althoff. 2022. Provably Safe Deep Reinforcement Learning for Robotic Manipulation in Human Environments. In *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022*. IEEE, 6344–6350. doi:10.1109/ICRA46639.2022.9811698

[44] Qisong Yang, Thiago D. Simão, Simon H. Tindemans, and Matthijs T. J. Spaan. 2021. WCSAC: Worst-Case Soft Actor Critic for Safety-Constrained Reinforcement Learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 10639–10646. doi:10.1609/AAAI.V35I12.17272

Received 2024-10-16; accepted 2025-02-18