ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models[*]

Stefan Mitsch and André Platzer

Computer Science Department
Carnegie Mellon University, Pittsburgh PA 15213, USA,
{smitsch,aplatzer}@cs.cmu.edu

Abstract. Formal verification and validation play a crucial role in making cyber-physical systems (CPS) safe. Formal methods make strong guarantees about the system behavior *if* accurate models of the system can be obtained, including models of the controller and of the physical dynamics. In CPS, models are essential; but any model we could possibly build necessarily deviates from the real world. If the real system fits to the model, its behavior is guaranteed to satisfy the correctness properties verified w.r.t. the model. Otherwise, all bets are off. This paper introduces ModelPlex, a method ensuring that verification results about models apply to CPS implementations. ModelPlex provides correctness guarantees for CPS executions at runtime: it combines offline verification of CPS models with runtime validation of system executions for compliance with the model. ModelPlex ensures that the verification results obtained for the model apply to the actual system runs by monitoring the behavior of the world for compliance with the model, assuming the system dynamics deviation is bounded. If, at some point, the observed behavior no longer complies with the model so that offline verification results no longer apply, ModelPlex initiates provably safe fallback actions. This paper, furthermore, develops a systematic technique to synthesize provably correct monitors automatically from CPS proofs in differential dynamic logic.

1 Introduction

Cyber-physical systems (CPS) span controllers and the relevant dynamics of the environment. Since safety is crucial for CPS, their models (e. g., hybrid system models [29]) need to be verified formally. Formal *verification* guarantees that a model is safe w.r.t. a safety property. The remaining task is to *validate* whether those models are adequate, so that the verification results transfer to the system implementation [16,38]. This paper introduces ModelPlex, a method to *synthesize monitors by theorem proving*: it uses sound proof rules to formally verify that a model is safe and to synthesize provably correct monitors that validate compliance of system executions with that model.

System execution, however, provides many opportunities for surprising deviations from the model: faults may cause the system to function improperly [39], sensors may deliver uncertain values, actuators suffer from disturbance, or the formal verification

[*] This material is based on research sponsored by DARPA under agreement number DARPA FA8750-12-2-0291. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

may have assumed simpler ideal-world dynamics for tractability reasons or made unrealistically strong assumptions about the behavior of other agents in the environment. Simpler models are often better for real-time decisions and optimizations, because they make predictions feasible to compute at the required rate. The same phenomenon of simplicity for predictability is often exploited for the models in formal verification and validation. As a consequence, the *verification results obtained about models of a CPS only apply to the actual CPS at runtime to the extent that the system fits to the model*.

Validation, i. e., checking whether a CPS implementation fits to a model, is an interesting but difficult problem. Even more so, since CPS models are more difficult to analyze than ordinary (discrete) programs because of the physical plant, the environment, sensor inaccuracies, and actuator disturbance. In CPS, models are essential; but any model we could possibly build necessarily deviates from the real world. Still, good models are approximately right, i. e., within certain error margins.

In this paper, we settle for the question of *runtime model validation*, i. e. validating whether the model assumed for verification purposes is adequate for a *particular system execution* to ensure that the verification results apply *to the current execution*.[1] But we focus on *verifiably correct runtime validation* to ensure that verified properties of models provably apply, which is important for safety and certification [5].

If the observed system execution fits to the verified model, then this execution is safe according to the offline verification result about the model. If it does not fit, then the system is potentially unsafe because it no longer has an applicable safety proof, so we initiate a verified fail-safe action to avoid safety risks. Checking whether a system execution fits to a verified model includes checking that the actions chosen by the (unverified) controller implementation fit to *one of* the choices and requirements of the verified controller model. It also includes checking that the observed states can be explained by the plant model. The crucial questions are: How can a compliance monitor be synthesized that provably represents the verified model? How much safety margin does a system need to ensure that fail-safe actions are initiated early enough for the system to remain safe even if its behavior ceases to comply with the model?

The second question is related to feedback control and can only be answered when assuming constraints on the deviation of the real system dynamics from the plant model [33]. Otherwise, i. e., if the real system can be infinitely far off from the model, safety guarantees are impossible. By the sampling theorem in signal processing [37], such constraints further enable compliance monitoring solely on the basis of sample points instead of the unobservable intermediate states about which no sensor data exists.[2] This paper presents ModelPlex, a method to synthesize verifiably correct runtime validation monitors automatically. ModelPlex uses theorem proving with sound proof rules [29] to turn hybrid system models into monitors in a verifiably correct way. Upon

[1] ModelPlex checks system execution w.r.t. a monitor specification, and thus, belongs to the field of runtime verification [16]. In this paper we use the term *runtime validation* in order to clearly convey the purpose of monitoring (i. e., runtime verification: monitor properties without offline verification; ModelPlex: monitor model adequacy to transfer offline verification results).

[2] When such constraints are not available, our method still generates verifiably correct *runtime tests*, which detect deviation from the model at the sampling points, just not between them. A fail-safe action will then lead to best-effort mitigation of safety risks (rather than guaranteed safety).

noncompliance, ModelPlex initiates provably safe fail-safe actions. System-level challenges w.r.t. monitor implementation and violation cause diagnosis are discussed elsewhere [8,19,41].

2 Preliminaries: Differential Dynamic Logic

For hybrid systems verification we use *differential dynamic logic* $\mathsf{d}\mathcal{L}$ [27,29,31], which has a notation for hybrid systems as *hybrid programs*. $\mathsf{d}\mathcal{L}$ allows us to make statements that we want to be true for all runs of a hybrid program ($[\alpha]\phi$) or for at least one run ($\langle\alpha\rangle\phi$). Both constructs are necessary to derive safe monitors: we need $[\alpha]\phi$ proofs so that we can be sure all behavior of a model (including controllers) are safe; we need $\langle\alpha\rangle\phi$ proofs to find monitor specifications that detect whether or not system execution fits to the verified model. Table 1 summarizes the relevant syntax fragment of hybrid programs together with an informal semantics. The semantics $\rho(\alpha)$ of hybrid program α is a relation on initial and final states of running α (defined in [27,32]). The set of $\mathsf{d}\mathcal{L}$ formulas is generated by the following grammar ($\sim \in \{<, \leq, =, \geq, >\}$ and θ_1, θ_2 are arithmetic expressions in $+, -, \cdot, /$ over the reals):

$$\phi ::= \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x \phi \mid \exists x \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

Differential dynamic logic comes with a verification technique to prove correctness properties of hybrid programs (cf. [31] for an overview of $\mathsf{d}\mathcal{L}$ and KeYmaera).

3 ModelPlex Approach for Verified Runtime Validation

CPS are almost impossible to get right without sufficient attention to prior analysis, for instance by formal verification and formal validation techniques. We assume to be given a verified model of a CPS, i. e. formula (1) is proved valid,[3] for example using [27,31].

$$\phi \rightarrow [\alpha^*]\psi \qquad \text{with invariant } \varphi \rightarrow [\alpha]\varphi \text{ s.t. } \phi \rightarrow \varphi \text{ and } \varphi \rightarrow \psi \tag{1}$$

[3] We use differential dynamic logic ($\mathsf{d}\mathcal{L}$) and KeYmaera as a theorem prover to illustrate our concepts throughout this paper. The concept of ModelPlex is not predicated on the use of KeYmaera to prove (1). Other verification techniques could be used to establish validity of this formula. The flexibility of the underlying logic $\mathsf{d}\mathcal{L}$, its support for both $[\alpha]\phi$ and $\langle\alpha\rangle\phi$, and its proof calculus, however, are exploited for systematically constructing monitors from proofs in the sequel.

Table 1: Hybrid program representations of hybrid systems.

Statement	Effect
$\alpha;\ \beta$	sequential composition, first run hybrid program α, then hybrid program β
$\alpha \cup \beta$	nondeterministic choice, following either hybrid program α or β
α^*	nondeterministic repetition, repeats hybrid program α $n \geq 0$ times
$x := \theta$	assign value of term θ to variable x (discrete jump)
$x := *$	assign arbitrary real number to variable x
$?F$	check that a particular condition F holds, and abort if it does not
$\big(x_1' = \theta_1, \ldots,$ $x_n' = \theta_n \ \&\ F\big)$	evolve x_i along differential equation system $x_i' = \theta_i$ restricted to maximum evolution domain F

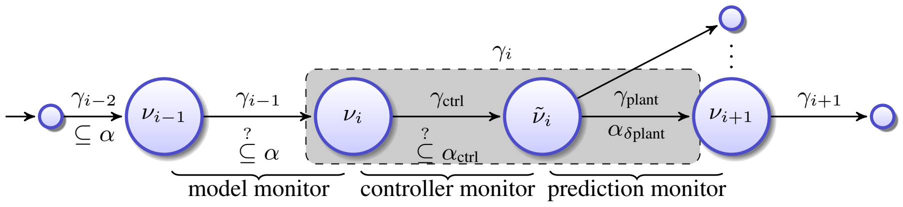

Fig. 1: Use of ModelPlex monitors along a system execution

Formula (1) expresses that all runs of the hybrid system α^*, which start in states that satisfy the precondition ϕ and repeat the model α arbitrarily many times, must end in states that satisfy the postcondition ψ. Formula (1) is proved using some form of induction, which shows that a loop invariant φ holds after every run of α if it was true before. The model α is a *hybrid system model* of a CPS, which means that it describes both the discrete control actions of the controllers in the system and the continuous physics of the plant and the system's environment.

The safety guarantees that we obtain by proving formula (1) about the model α^* transfer to the real system, *if* the actual CPS execution fits to α^*. Since we want to preserve safety properties, a CPS γ fits to a model α^*, *if* the CPS reaches at most those states that are reachable by the model, i. e., $\rho(\gamma) \subseteq \rho(\alpha^*)$. However, we do not know γ and therefore need to find a condition based on α^* that we can check at runtime to see if concrete runs of γ behave like α^*. Checking the postcondition ψ is not sufficient because, if ψ does not hold, the system is already unsafe. Checking the invariant φ is insufficient as well, because if φ does not hold the controller can no longer guarantee safety, even though the system may not yet be unsafe. But if we detect when a CPS is about to deviate from α^* *before* leaving φ, we can still switch to a fail-safe controller to avoid $\neg\psi$ from happening.

ModelPlex derives three kinds of monitors (model monitor, controller monitor, and prediction monitor, cf. Fig. 1). We check reachability between consecutive states in α, α_{ctrl}, and $\alpha_{\delta\text{plant}}$ by verifying states during execution against the corresponding monitor.

Model monitor In each state ν_i we test the sample point ν_{i-1} from the previous execution γ_{i-1} for deviation from the single α, not α^* i. e., test $(\nu_{i-1}, \nu_i) \in \rho(\alpha)$. If violated, other verified properties may no longer hold for the system; the system, however, is still safe if a prediction monitor was satisfied on ν_{i-1}. Frequent violations indicate an inadequate model that should be revised to better reflect reality.

Controller monitor In intermediate state $\tilde{\nu}_i$ we test the current controller decisions of the implementation γ_{ctrl} for compliance with the model, i. e., test $(\nu_i, \tilde{\nu}_i) \in \rho(\alpha_{\text{ctrl}})$. Controller monitors are designed for switching between controllers similar to Simplex [36]. If violated, the commands from a fail-safe controller replace the current controller's decisions to ensure that no unsafe commands are ever actuated.

Prediction monitor In intermediate state $\tilde{\nu}_i$ we test the worst-case safety impact of the current controller decisions w.r.t. the predictions of a bounded deviation plant model $\alpha_{\delta\text{plant}}$, which has a tolerance around the model plant α_{plant}, i. e., check $\nu_{i+1} \models \varphi$ for all ν_{i+1} such that $(\tilde{\nu}_i, \nu_{i+1}) \in \rho(\alpha_{\delta\text{plant}})$. Note, that we simultaneously check all ν_{i+1} by checking $\tilde{\nu}_i$ for a characterizing condition of $\alpha_{\delta\text{plant}}$. If violated, the current control choice is not guaranteed to keep the system safe until the next control cycle and, thus, a fail-safe controller takes over.

The assumption for the prediction monitor is that the real execution is not arbitrarily far off the plant models used for safety verification, because otherwise guarantees can be neither made on unobservable intermediate states nor on safety of the future system evolution [33]. We propose separation of disturbance causes in the models: ideal plant models α_{plant} for correctness verification purposes, implementation deviation plant models $\alpha_{\delta\text{plant}}$ for monitoring purposes. We support any deviation model (e. g., piecewise constant disturbance, differential inclusion models of disturbance), as long as the deviation is bounded and differential invariants can be found. We further assume that monitor evaluations are at most some ε time units apart (e. g., along with a recurring controller execution). Note that disturbance in $\alpha_{\delta\text{plant}}$ is more manageable compared to α^*, because we can focus on single runs α instead of repetitions for monitoring.

3.1 Relation between States

We systematically derive a check that inspects states of the actual CPS to detect deviation from the model α^*. We first establish a notion of state recall and show that, when all previous state pairs complied with the model, compliance of the entire execution can be checked by checking the latest two states (ν_{i-1}, ν_i) (see [25, App. A] for proofs).

Definition 1 (State recall). *We use V to denote the set of variables whose state we want to recall. We use $\Upsilon_V^- \equiv \bigwedge_{x\in V} x = x^-$ to express a characterization of the values of variables in a state prior to a run of α, where we always assume the fresh variables x^- to occur solely in Υ_V^-. The variables in x^- can be used to recall this state. Likewise, we use $\Upsilon_V^+ \equiv \bigwedge_{x\in V} x = x^+$ to characterize the posterior states and expect fresh x^+.*

With this notation the following lemma states that an interconnected sequence of α transitions forms a transition of α^*.

Lemma 1 (Loop prior and posterior state). *Let α be a hybrid program and α^* be the program that repeats α arbitrarily many times. Assume that all consecutive pairs of states $(\nu_{i-1}, \nu_i) \in \rho(\alpha)$ of $n \in \mathbb{N}^+$ executions, whose valuations are recalled with $\Upsilon_V^i \equiv \bigwedge_{x\in V} x = x^i$ and Υ_V^{i-1} are plausible w.r.t. the model α, i. e., $\models \bigwedge_{1\leq i\leq n} \left(\Upsilon_V^{i-1} \rightarrow \langle\alpha\rangle\Upsilon_V^i\right)$ with $\Upsilon_V^- = \Upsilon_V^0$ and $\Upsilon_V^+ = \Upsilon_V^n$. Then, the sequence of states originates from an α^* execution from Υ_V^0 to Υ_V^n, i. e., $\models \Upsilon_V^- \rightarrow \langle\alpha^*\rangle\Upsilon_V^+$.*

Lemma 1 enables us to check compliance with the model α^* up to the current state by checking reachability of a posterior state from a prior state on each execution of α (i. e., online monitoring [16], which is easier because the loop was eliminated). To find compliance checks systematically, we construct formula (2), which relates a prior state of a CPS to its posterior state through at least one path through the model α. [4]

$$\Upsilon_V^- \rightarrow \langle\alpha\rangle\Upsilon_V^+ \tag{2}$$

[4] Consecutive states for α^* mean before and after executions of α (i. e., $\alpha\overset{\downarrow}{;}\alpha\overset{\downarrow}{;}\alpha$, not within α).

This formula is satisfied in a state ν, if there is at least one run of the model α starting in the state ν recalled by Υ_V^- and results in a state ω recalled using Υ_V^+. In other words, at least one path through α explains how the prior state ν got transformed into the posterior state ω. The d$\mathcal{L}$ formula (2) characterizes the state transition relation of the model α directly. Its violation witnesses compliance violation. Compliance at all intermediate states cannot be observed by real-world sensors, see Section 3.5.

In principle, formula (2) would be a monitor, because it relates a prior state to a posterior state through the model of a CPS; but the formula is hard if not impossible to evaluate at runtime, because it refers to a hybrid system α, which includes nondeterminism and differential equations. The basic observation is that any formula that is equivalent to (2) but conceptually easier to evaluate in a state would be a correct monitor. We use theorem proving for simplifying formula (2) into quantifier-free first-order real arithmetic form so that it can be evaluated efficiently at runtime. The resulting first-order real arithmetic formula can be easily implemented in a runtime monitor and executed along with the actual controller. A monitor is executable code that only returns true if the transition from the prior system state to the posterior state is compliant with the model. Thus, deviations from the model can be detected at runtime, so that appropriate fallback and mitigation strategies can be initiated.

Remark 1. The complexity for evaluating an arithmetic formula over the reals for concrete numbers is linear in the formula size, as opposed to deciding the validity of such formulas, which is doubly exponential. Evaluating the same formula on floating point numbers is inexpensive, but may yield wrong results due to rounding errors; on exact rationals the bit-complexity can be non-negligible. We use interval arithmetic to obtain reliable results efficiently (cf. [25, App. C]).

Example 1. We will use a simple water tank as a running example to illustrate the concepts throughout this section. The water tank has a current level x and a maximum level m. The water tank controller, which runs at least every ε time units, nondeterministically chooses any flow f between a maximum outflow -1 and a maximum inflow $\frac{m-x}{\varepsilon}$. This water tank never overflows, as witnessed by a proof for the following d$\mathcal{L}$ formula.

$$\underbrace{0 \le x \le m \wedge \varepsilon > 0}_{\phi} \rightarrow \Big[\big(f := *;\ ?\big(-1 \le f \le \tfrac{m-x}{\varepsilon}\big);\ t := 0;\ (x' = f,\ t' = 1 \,\&\, x \ge 0 \wedge t \le \varepsilon)\big)^* \Big] \overbrace{(0 \le x \le m)}^{\psi}$$

3.2 ModelPlex Monitor Synthesis

This section introduces the nature of ModelPlex monitor specifications, our approach to generate such specifications from hybrid system models, and how to turn those specifications into monitor code that can be executed at runtime along with the controller.

A ModelPlex specification corresponds to the d$\mathcal{L}$ formula (2). If the current state of a system does not satisfy a ModelPlex specification, some behavior that is not reflected in the model occurred (e. g., the wrong control action was taken, unanticipated dynamics in the environment occurred, sensor uncertainty led to unexpected values, or the system was applied outside the specified operating environment).

A *model monitor* χ_{m} checks that two consecutive states ν and ω can be explained by an execution of the model α, i. e., $(\nu, \omega) \in \rho(\alpha)$. In the sequel, $BV(\alpha)$ are bound variables in α, $FV(\psi)$ are free variables in ψ, Σ is the set of all variables, and $A \backslash B$ denotes the set of variables being in some set A but not in some other set B. Furthermore, we use $\nu|_A$ to denote ν projected onto the variables in A.

Theorem 1 (Model monitor correctness). *Let α^* be provably safe, so $\models \phi \rightarrow [\alpha^*]\psi$. Let $V_m = BV(\alpha) \cup FV(\psi)$. Let $\nu_0, \nu_1, \nu_2, \nu_3 \ldots \in \mathbb{R}^n$ be a sequence of states, with $\nu_0 \models \phi$ and that agree on $\Sigma \backslash V_m$, i. e., $\nu_0|_{\Sigma \backslash V_m} = \nu_k|_{\Sigma \backslash V_m}$ for all k. We define $(\nu, \nu_{i+1}) \models \chi_m$ as χ_m evaluated in the state resulting from ν by interpreting x^+ as $\nu_{i+1}(x)$ for all $x \in V_m$, i. e., $\nu_{x^+}^{\nu_{i+1}(x)} \models \chi_m$. If $(\nu_i, \nu_{i+1}) \models \chi_m$ for all $i < n$ then we have $\nu_n \models \psi$ where*

$$\chi_m \equiv \left(\phi|_{const} \rightarrow \langle\alpha\rangle \Upsilon^+_{V_m}\right) \tag{3}$$

and $\phi|_{const}$ denotes the conditions of ϕ that involve only constants that do not change in α, i. e., $FV(\phi|_{const}) \cap BV(\alpha) = \emptyset$.

Our approach to generate monitor specifications from hybrid system models takes a verified d$\mathcal{L}$ formula (1) as input and produces a monitor χ_{m} in quantifier-free first-order form as output. The algorithm, listed in [25, App. D], involves the following steps:

1. A d$\mathcal{L}$ formula (1) about a model α of the form $\phi \rightarrow [\alpha^*]\psi$ is turned into a specification conjecture (3) of the form $\phi|_{\mathrm{const}} \rightarrow \langle\alpha\rangle \Upsilon^+_{V_m}$.
2. Theorem proving on the specification conjecture (3) is applied until no further proof rules are applicable and only first-order real arithmetic formulas remain open.
3. The monitor specification χ_{m} is the conjunction of the unprovable first-order real arithmetic formulas from open sub-goals.

Generate the monitor conjecture. We map d$\mathcal{L}$ formula (1) syntactically to a specification conjecture of the form (3). By design, this conjecture will not be provable. But the unprovable branches of a proof attempt will reveal information that, had it been in the premises, would make (3) provable. Through $\Upsilon^+_{V_m}$, those unprovable conditions collect the relations of the posterior state of model α characterized by x^+ to the prior state x, i. e., the conditions are a representation of (2) in quantifier-free first-order real arithmetic.

Example 2. The specification conjecture for the water tank model is given below. It is constructed from the model by removing the loop, flipping the modality, and formulating the specification requirement as a property, since we are interested in a relation between two consecutive states ν and ω (recalled by x^+, f^+ and t^+). Using theorem proving [34], we analyze the conjecture to reveal the actual monitor specification.

$$\underbrace{\varepsilon > 0}_{\phi|_{\mathrm{const}}} \rightarrow \Big\langle f := *; ?\left(-1 \le f \le \tfrac{m-x}{\varepsilon}\right); \; t := 0; \; (x' = f, \, t' = 1 \,\&\, x \ge 0 \wedge t \le \varepsilon)\Big\rangle \overbrace{\left(x = x^+ \wedge f = f^+ \wedge t = t^+\right)}^{\Upsilon^+_{V_m}}$$

Use theorem proving to analyze the specification conjecture. We use the proof rules of d$\mathcal{L}$ [27,31] to analyze the specification conjecture χ_{m}. These proof rules syntactically decompose a hybrid model into easier-to-handle parts, which leads to sequents with first-order real arithmetic formulas towards the leaves of a proof. Using real arithmetic quantifier elimination we close sequents with logical tautologies, which do not need to be checked at runtime since they always evaluate to $true$ for any input. The conjunction of the remaining open sequents is the monitor specification; it implies (2).

A complete sequence of proof rules applied to the monitor conjecture of the water tank is described in [25, App. B]. Most steps are simple when analyzing specification conjectures: sequential composition ($\langle ; \rangle$), nondeterministic choice ($\langle \cup \rangle$), deterministic assignment ($\langle := \rangle$) and logical connectives ($\wedge$r etc.) replace current facts with simpler ones or branch the proof (cf. rules in [27,32]). Challenge arise from handling nondeterministic assignment and differential equations in hybrid programs.

Let us first consider nondeterministic assignment $x := *$. The proof rule for nondeterministic assignment ($\langle * \rangle$) results in a new existentially quantified variable. By sequent proof rule $\exists$r, this existentially quantified variable is instantiated with an arbitrary term θ, which is often a new logical variable that is implicitly existentially quantified [27]. Weakening (Wr) removes facts that are no longer necessary.

$$(\langle * \rangle)\ \frac{\exists X \langle x := X \rangle \phi}{\langle x := * \rangle \phi}\ ^{1} \qquad (\exists \mathrm{r})\ \frac{\Gamma \vdash \phi(\theta), \exists x\, \phi(x), \Delta}{\Gamma \vdash \exists x\, \phi(x), \Delta}\ ^{2} \qquad (\mathrm{Wr})\ \frac{\Gamma \vdash \Delta}{\Gamma \vdash \phi, \Delta}$$

1 X is a new logical variable
2 θ is an arbitrary term, often a new (existential) logical variable X.

Optimization 1 (Instantiation Trigger). *If the variable is not changed in the remaining α, $x_i = x_i^+$ is in $\Upsilon^+_{V_m}$ and X is not bound in $\Upsilon^+_{V_m}$, then instantiate the existential quantifier by rule $\exists$r with the corresponding x_i^+ that is part of the specification conjecture (i. e., $\theta = x_i^+$), since subsequent proof steps are going to reveal $\theta = x_i^+$ anyway.*

Otherwise, we introduce a new logical variable, which may result in an existential quantifier in the monitor specification if no further constraints can be found later in the proof.

Example 3. The corresponding steps in the water tank proof use $\langle * \rangle$ for the nondeterministic flow assignment ($f := *$) and $\exists$r to instantiate the resulting existential quantifier $\exists F$ with a new logical variable F (*plant* is an abbreviation for $x' = f, t' = 1 \,\&\, 0 \le x \wedge t \le \varepsilon$). We show the proof without and with application of Opt. 1.

$$\langle * \rangle\ \dfrac{\exists \mathrm{r,Wr}\ \dfrac{\phi \vdash \langle f := F \rangle \langle ?{-1} \le f \le \frac{m-x}{\varepsilon} \rangle \langle plant \rangle \Upsilon^+}{\phi \vdash \exists F \langle f := F \rangle \langle ?{-1} \le f \le \frac{m-x}{\varepsilon} \rangle \langle plant \rangle \Upsilon^+}}{\phi \vdash \langle f := *; ?{-1} \le f \le \frac{m-x}{\varepsilon} \rangle \langle plant \rangle \Upsilon^+}$$

w/o Opt. 1

$$\exists \mathrm{r,Wr}\ \dfrac{\phi \vdash \langle f := f^+ \rangle \langle ?{-1} \le f \le \frac{m-x}{\varepsilon} \rangle \langle plant \rangle \Upsilon^+}{\dots}$$

with Opt. 1 (anticipate $f = f^+$ from Υ^+)

Next, we handle differential equations. Even when we can solve the differential equation, existentially and universally quantified variables remain. Let us inspect the corresponding proof rule from the d$\mathcal{L}$ calculus [31]. For differential equations we have to prove that there exists a duration t, such that the differential equation stays within the evolution domain H throughout all intermediate times $\tilde{t}$ and the result satisfies ϕ at the end. At this point we have three options:

- we can instantiate the existential quantifier, if we know that the duration will be t^+;
- we can introduce a new logical variable, which is the generic case that always yields correct results, but may discover monitor specifications that are harder to evaluate;

$$(\langle'\rangle)\ \frac{\exists T{\geq}0\,\big((\forall 0{\leq}\tilde{t}{\leq}T\,\langle x := y(\tilde{t})\rangle H) \wedge \langle x := y(T)\rangle\phi\big)}{\langle x' = \theta \,\&\, H\rangle\phi}\ ^{1} \qquad (\text{QE})\ \frac{\text{QE}(\phi)}{\phi}\ ^{2}$$

1 T and $\tilde{t}$ are fresh logical variables and $\langle x := y(T)\rangle$ is the discrete assignment belonging to the solution y of the differential equation with constant symbol x as symbolic initial value

2 iff $\phi \equiv \text{QE}(\phi)$, ϕ is a first-order real arithmetic formula, $\text{QE}(\phi)$ is an equivalent quantifier-free formula computable by [7]

– we can use quantifier elimination (QE) to obtain an equivalent quantifier-free result (a possible optimization could inspect the size of the resulting formula).

Example 4. In the analysis of the water tank example, we solve the differential equation (see $\langle'\rangle$) and apply the substitutions $f := F$ and $t := 0$. In the next step (see $\exists$r,Wr), we instantiate the existential quantifier $\exists T$ with t^+ (i. e., we choose $T = t^+$ using Opt. 1 with the last conjunct) and use weakening right (Wr) to systematically get rid of the existential quantifier that would otherwise still be left around by rule $\exists$r. Finally, we use quantifier elimination (QE) to reveal an equivalent quantifier-free formula.

$$\cfrac{\cfrac{\cfrac{\phi \vdash F = f^+ \wedge x^+ = x + Ft^+ \wedge t^+ \geq 0 \wedge x \geq 0 \wedge \varepsilon \geq t^+ \geq 0 \wedge Ft^+ + x \geq 0}{\phi \vdash \forall 0{\leq}\tilde{t}{\leq}T\,(x + f^+\tilde{t} \geq 0 \wedge \tilde{t} \leq \varepsilon) \wedge F = f^+ \wedge x^+ = x + Ft^+ \wedge t^+ = t^+}\,\text{QE}}{\phi \vdash \exists T{\geq}0((\forall 0{\leq}\tilde{t}{\leq}T\,(x + f^+\tilde{t} \geq 0 \wedge \tilde{t} \leq \varepsilon)) \wedge F = f^+ \wedge (x^+ = x + FT \wedge t^+ = T))}\,\exists\text{r,Wr}}{\phi \vdash \langle f := F; t := 0\rangle\langle\{x' = f, t' = 1 \,\&\, x \geq 0 \wedge t \leq \varepsilon\}\rangle\Upsilon^+}\,\langle'\rangle$$

The analysis of the specification conjecture finishes with collecting the open sequents from the proof to create the monitor specification $\chi_{\text{m}} \overset{\text{def}}{\equiv} \bigwedge(\textit{open sequent})$. The collected open sequents may include new logical variables and new (Skolem) function symbols that were introduced for nondeterministic assignments and differential equations when handling existential or universal quantifiers. We use the invertible quantifier rule i$\exists$ to re-introduce existential quantifiers for the new logical variables (universal quantifiers for function symbols, see [27] for calculus details). Often, the now quantified logical variables are discovered to be equal to one of the post-state variables later in the proof, because those variables did not change in the model after the assignment. If this is the case, we can use proof rule $\exists\sigma$ to further simplify the monitor specification by substituting the corresponding logical variable x with its equal term θ.

$$(\text{i}\exists)\ \frac{\Gamma \vdash \exists X\,\big(\bigwedge_i(\Phi_i \vdash \Psi_i)\big)\,, \Delta}{\Gamma, \Phi_1 \vdash \Psi_1, \Delta \quad \cdots \quad \Gamma, \Phi_n \vdash \Psi_n, \Delta}\ ^{1} \qquad (\exists\sigma)\ \frac{\phi(\theta)}{\exists x\,(x = \theta \wedge \phi(x))}\ ^{2}$$

1 Among all open branches, free logical variable X only occurs in the branches $\Gamma, \Phi_i \vdash \Psi_i, \Delta$

2 Logical variable x does not appear in term θ

Example 5. The two open sequents of Examples 3 and 4 use a new logical variable F for the nondeterministic flow assignment $f := *$. After further steps in the proof, the assumptions reveal additional information $F = f^+$. Thus, we re-introduce the existential quantifier over all the open branches (i$\exists$) and substitute f^+ for F ($\exists\sigma$). The sole open sequent of this proof attempt is the monitor specification χ_m of the water tank model.

$$\begin{array}{ll}
 & \phi \vdash -1 \leq f^+ \leq \frac{m-x}{\varepsilon} \wedge x^+ = x + f^+t^+ \wedge t^+ \geq 0 \wedge x \geq 0 \dots \\
\hline
{\scriptstyle \exists\sigma} & \phi \vdash \exists F(-1 \leq F \leq \frac{m-x}{\varepsilon} \wedge F = f^+ \wedge x^+ = x + Ft^+ \wedge t^+ \geq 0 \wedge x \geq 0 \dots) \\
\hline
{\scriptstyle \mathrm{i}\exists} & \phi \vdash -1 \leq F \leq \frac{m-x}{\varepsilon} \qquad \phi \vdash F = f^+ \wedge x^+ = x + Ft^+ \wedge t^+ \geq 0 \wedge x \geq 0 \dots
\end{array}$$

3.3 Controller Monitor Synthesis

A *controller monitor* χ_c checks that two consecutive states ν and ω are reachable with one controller execution α_{ctrl}, i. e., $(\nu, \omega) \in \rho(\alpha_{\text{ctrl}})$ with $V_c = BV(\alpha_{\text{ctrl}}) \cup FV(\psi)$. We systematically derive controller monitors from formulas $\phi|_{\text{const}} \rightarrow \langle \alpha_{\text{ctrl}} \rangle \Upsilon^+_{V_c}$. A controller monitor can be used to initiate controller switching similar to Simplex [36].

Theorem 2 (Controller monitor correctness). *Let α of the canonical form $\alpha_{ctrl}; \alpha_{plant}$. Assume $\models \phi \rightarrow [\alpha^*]\psi$ has been proven with invariant φ as in (1). Let $\nu \models \phi|_{const} \wedge \varphi$, as checked by χ_m (Theorem 1). Furthermore, let $\tilde{\nu}$ be a post-controller state. If $(\nu, \tilde{\nu}) \models \chi_c$ with $\chi_c \equiv \phi|_{const} \rightarrow \langle \alpha_{ctrl} \rangle \Upsilon^+_{V_c}$ then we have that $(\nu, \tilde{\nu}) \in \rho(\alpha_{ctrl})$ and $\tilde{\nu} \models \varphi$.*

3.4 Monitoring in the Presence of Expected Uncertainty and Disturbance

Up to now we considered exact ideal-world models. But real-world clocks drift, sensors measure with some uncertainty, and actuators are subject to disturbance. This makes the exact models safe but too conservative, which means that monitors for exact models are likely to fall back to a fail-safe controller rather often. In this section we discuss how we find ModelPlex specifications so that the safety property (1) and the monitor specification become more robust to expected uncertainty and disturbance. That way, only unexpected deviations beyond those captured in the normal operational uncertainty and disturbance of α^* cause the monitor to initiate fail-safe actions.

In d$\mathcal{L}$, we can, for example, use nondeterministic assignment from an interval to model sensor uncertainty and piecewise constant actuator disturbance (e. g., as in [22]), or differential inequalities for actuator disturbance (e. g., as in [35]). Such models include nondeterminism about sensed values in the controller model and often need more complex physics models than differential equations with polynomial solutions.

Example 6. We incorporate clock drift, sensor uncertainty and actuator disturbance into the water tank model to express expected deviation. The measured level x_s is within a known sensor uncertainty u of the real level x (i.e. $x_s \in [x - u, x + u]$). We use differential inequalities to model clock drift and actuator disturbance. The clock, which wakes the controller, is slower than the real time by at most a time drift of c; it can be arbitrarily fast. The water flow disturbance is at most d, but the water tank is allowed to drain arbitrarily fast (even leaks when the pump is on). To illustrate different modeling possibilities, we use additive clock drift and multiplicative actuator disturbance.

$$\begin{aligned}
&0 \leq x \leq m \wedge \varepsilon > 0 \wedge c < 1 \wedge 0 \leq u \wedge 0 < d \\
&\quad \rightarrow \Big[\big(x_s := *; ?\,(x - u \leq x_s \leq x + u)\,;\; f := *; ?\,\big(-1 \leq f \leq \tfrac{m - x_s - u}{d\varepsilon}(1 - c)\big); \\
&\qquad t := 0;\; \{x' \leq fd,\; 1 - c \leq t' \;\&\; x \geq 0 \wedge t \leq \varepsilon\}\big)^* \Big](0 \leq x \leq m)
\end{aligned}$$

We analyze Example 6 in the same way as the previous examples, with the crucial exception of the differential inequalities. We cannot use the proof rule $\langle ' \rangle$ to analyze this model, because differential inequalities do not have polynomial solutions. Instead, we use the DR and DE proof rules of $\mathsf{d}\mathcal{L}$ [28,29] to turn differential inequalities into a differential-algebraic constraint form that lets us proceed with the proof. Rule DE turns a differential inequality $x' \leq \theta$ into a quantified differential equation $\exists \tilde{d}(x' = \tilde{d} \,\&\, \tilde{d} \leq \theta)$ with an equivalent differential-algebraic constraint. Rule DR turns a differential-algebraic constraint $\mathscr{E}$ into another differential-algebraic constraint $\mathscr{D}$, which implies $\mathscr{E}$, written $\mathscr{D} \rightarrow \mathscr{E}$, as defined in [28] (cf. [25, App. B] for an example).

$$(\text{DR})\ \frac{\mathscr{D} \rightarrow \mathscr{E} \quad \langle \mathscr{D} \rangle \phi}{\langle \mathscr{E} \rangle \phi}\ ^{1} \qquad (\text{DE})\ \frac{\forall X (\exists \tilde{d} (X = \tilde{d} \wedge \tilde{d} \leq \theta \wedge H) \rightarrow X \leq \theta \wedge H) \quad \langle \exists \tilde{d} (x' = \tilde{d} \,\&\, \tilde{d} \leq \theta \wedge H) \rangle \phi}{\langle x' \leq \theta \,\&\, H \rangle \phi}\ ^{2}$$

[1] differential refinement: differential-algebraic constraints $\mathscr{D}, \mathscr{E}$ have the same changed variables
[2] differential inequality elimination: special case of DR, which rephrases the differential inequalities $\leq$ as differential-algebraic constraints (accordingly for other or mixed inequalities systems).

Currently, for finding model monitors our prototype tool solves differential equations by the proof rule $\langle ' \rangle$. Thus, it finds model monitor specifications for differential algebraic equations with polynomial solutions and for differential algebraic inequalities, which can be refined into solvable differential algebraic equations as in Example 6. For prediction monitors (discussed in Section 3.5) we use $\mathsf{d}\mathcal{L}$ techniques for finding differential variants and invariants, differential cuts [28], and differential auxiliaries [30] to handle differential equations and inequalities *without polynomial solutions*.

3.5 Monitoring Compliance Guarantees for Unobservable Intermediate States

With controller monitors, non-compliance of a controller implementation w.r.t. the modeled controller can be detected right away. With model monitors, non-compliance of the actual system dynamics w.r.t. the modeled dynamics can be detected when they first occur. We switch to a fail-safe action, which is verified using standard techniques, in both non-compliance cases. The crucial question is: can such a method always guarantee safety? The answer is linked to the image computation problem in model checking (i. e., approximation of states reachable from a current state), which is known to be not semi-decidable by numerical evaluation at points; approximation with uniform error is only possible if a bound is known for the continuous derivatives [33]. This implies that we need additional assumptions about the deviation between the actual and the modeled continuous dynamics to guarantee compliance for unobservable intermediate states. Unbounded deviation from the model between sample points just is unsafe, no matter how hard a controller tries. Hence, worst-case bounds capture how well reality is reflected in the model.

We derive a prediction monitor to check whether a current control decision will be able to keep the system safe for time ε even if the actual continuous dynamics deviate from the model. A prediction monitor checks the current state, because all previous states are ensured by a model monitor and subsequent states are then safe by (1).

Definition 2 **(ε-bounded plant with disturbance δ).** *Let α_{plant} be a model of the form $x' = \theta \,\&\, H$. An ε-*bounded plant with disturbance δ*, written $\alpha_{\delta plant}$, is a plant model of the form $x_0 := 0;\ \ (f(\theta,\delta) \leq x' \leq g(\theta,\delta) \,\&\, H \land x_0 \leq \varepsilon)$ for some f, g with fresh variable $\varepsilon > 0$ and assuming $x_0' = 1$. We say that disturbance δ is* constant *if $x \notin \delta$; it is* additive *if $f(\theta,\delta) = \theta - \delta$ and $g(\theta,\delta) = \theta + \delta$.*

Theorem 3 **(Prediction monitor correctness).** *Let α^* be provably safe, i. e., $\models \phi \to [\alpha^*]\psi$ has been proved using invariant φ as in (1). Let $V_p = BV(\alpha) \cup FV([\alpha]\varphi)$. Let $\nu \models \phi|_{const} \land \varphi$, as checked by χ_m from Theorem 1. Further assume $\tilde{\nu}$ such that $(\nu,\tilde{\nu}) \in \rho(\alpha_{ctrl})$, as checked by χ_c from Theorem 2. If $(\nu,\tilde{\nu}) \models \chi_p$ with $\chi_p \equiv (\phi|_{const} \land \varphi) \to \langle\alpha_{ctrl}\rangle(\Upsilon^+_{V_p} \land [\alpha_{\delta plant}]\varphi)$, then we have for all $(\tilde{\nu},\omega) \in \rho(\alpha_{\delta plant})$ that $\omega \models \varphi$.*

Remark 2. By adding a controller execution $\langle\alpha_{\text{ctrl}}\rangle$ prior to the disturbed plant model, we synthesize prediction monitors that take the actual controller decisions into account. For safety purposes, we could just as well use a monitor definition without controller $\chi_{\text{p}} \equiv (\phi|_{\text{const}} \land \varphi) \to [\alpha_{\delta\text{plant}}]\varphi$. But doing so results in a conservative monitor, which has to keep the CPS safe without knowledge of the actual controller decision.

3.6 Decidability and Computability

One useful characteristic of ModelPlex beyond soundness is that monitor synthesis is computable, which yields a synthesis algorithm, and that the correctness of those synthesized monitors w.r.t. their specification is decidable, cf. Theorem 4.

Theorem 4 **(Monitor correctness is decidable and monitor synthesis computable).** *We assume canonical models of the form $\alpha \equiv \alpha_{ctrl};\alpha_{plant}$ without nested loops, with solvable differential equations in α_{plant} and disturbed plants $\alpha_{\delta plant}$ with constant additive disturbance δ (see Def. 2). Then, monitor correctness is decidable, i. e., the formulas $\chi_m \to \langle\alpha\rangle\Upsilon^+_V$, $\chi_c \to \langle\alpha_{ctrl}\rangle\Upsilon^+_V$, and $\chi_p \to \langle\alpha\rangle(\Upsilon^+_V \land [\alpha_{\delta plant}]\phi)$ are decidable. Also, monitor synthesis is computable, i. e., the functions $synth_m : \langle\alpha\rangle\Upsilon^+_V \mapsto \chi_m$, $synth_c : \langle\alpha_{ctrl}\rangle\Upsilon^+_V \mapsto \chi_c$, and $synth_p : \langle\alpha\rangle(\Upsilon^+_V \land [\alpha_{\delta plant}]\phi) \mapsto \chi_p$ are computable.*

4 Evaluation

We developed a software prototype, integrated into our modeling tool Sphinx [24], to automate many of the described steps. The prototype generates χ_{m}, χ_{c}, and χ_{p} conjectures from hybrid programs, collects open sequents, and interacts with KeYmaera [34].

To evaluate our method, we created monitors for prior case studies of non-deterministic hybrid models of autonomous cars, train control systems, and robots (adaptive cruise control [18], intelligent speed adaptation [23], the European train control system [35], and ground robot collision avoidance [22]). Table 2 summarizes the evaluation. For the model, we list the dimension in terms of the number of function symbols and state variables, and the size of the safety proof (i. e., number of proof steps and branches). For the monitor, we list the dimension of the monitor conjecture in terms of the number of variables, compare the number of steps and open sequents when deriving

Table 2: Monitor complexity case studies

	Case Study	Model		Monitor				
		dim.	proof size (branches)	dim.	steps (open seq.) w/ Opt. 1	steps (open seq.) auto	proof steps (branches)	size
χ_m	Water tank	5	38 (4)	3	16 (2)	20 (2)	64 (5)	32
χ_m	Cruise control [18]	11	969 (124)	7	127 (13)	597 (21)	19514 (1058)	1111
χ_m	Speed limit [23]	9	410 (30)	6	487 (32)	5016 (126)	64311 (2294)	19850
χ_c	Water tank	5	38 (4)	1	12 (2)	14 (2)	40 (3)	20
χ_c	Cruise control [18]	11	969 (124)	7	83 (13)	518 (106)	5840 (676)	84
χ_c	Robot [22]	14	3350 (225)	11	94 (10)	1210 (196)	26166 (2854)	121
χ_c	ETCS safety [35]	16	193 (10)	13	162 (13)	359 (37)	16770 (869)	153
χ_p	Water tank	8	80 (6)	1	135 (4)	N/A	307 (12)	43

http://www.cs.cmu.edu/~smitsch/resource/modelplex_study.zip

the monitor using manual proof steps to apply Opt. 1 and fully automated w/o Opt. 1, and the number of steps in the monitor correctness proof. Finally, we list the monitor size in terms of arithmetic, comparison, and logical operators in the monitor formula. Although the number of steps and open sequents differ significantly between manual interaction for Opt. 1 and fully automated synthesis, the synthesized monitors are logically equivalent. But applying Opt. 1 usually results in structurally simpler monitors, because the conjunction over a smaller number of open sequents (cf. Table 2) can still be simplified automatically. The model monitors for cruise control and speed limit control are significantly larger than the other monitors, because their size already prevents automated simplification by Mathematica. As future work, KeYmaera will be adapted to allow user-defined tactics in order to apply Opt. 1 automatically.

5 Related Work

Runtime verification and monitoring for finite state discrete systems has received significant attention (e. g., [9,14,20]). Other approaches monitor continuous-time signals (e. g., [10,26]). We focus on hybrid systems models of CPS to combine both.

Several tools for formal verification of hybrid systems are actively developed (e. g., SpaceEx [12], dReal [13], extended NuSMV/MathSat [6]). For monitor synthesis, however, ModelPlex crucially needs the rewriting capabilities and flexibility of (nested) $[\alpha]$ and $\langle\alpha\rangle$ modalities in $\mathsf{d}\mathcal{L}$ [29] and KeYmaera [34]; it is thus an interesting question for future work if other tools could be adapted to ModelPlex.

Runtime verification is the problem of checking whether or not a trace produced by a program satisfies a particular formula (cf. [16]). In [40], a method for runtime verification of LTL formulas on abstractions of concrete traces of a flight data recorder is presented. The RV system for Java programs [21] predicts execution traces from actual traces to find concurrency errors offline (e. g., race conditions) even if the actual trace did not exhibit the error. We, instead, use prediction on the basis of disturbed plant models for *hybrid systems* at runtime to ensure safety for future behavior of the system and switch to a fail-safe fallback controller if necessary. Adaptive runtime verification [4] uses state estimation to reduce monitoring overhead by sampling while still maintaining accuracy with Hidden Markov Models, or more recently, particle filtering [15] to fill the sampling gaps. The authors present interesting ideas for managing the overhead of runtime monitoring, which could be beneficial to transfer into the hybrid systems world. The approach, however, focuses purely on the discrete part of CPS.

The Simplex architecture [36] (and related approaches, e. g., [1,3,17]) is a control system principle to switch between a highly reliable and an experimental controller at runtime. Highly reliable control modules are assumed to be verified with some other approach. Simplex focuses on switching when timing faults or violation of controller specification occur. Our method complements Simplex in that *(i)* it checks whether or not the current system execution fits the entire model, not just the controller; *(ii)* it systematically derives provably correct monitors for hybrid systems; *(iii)* it uses prediction to guarantee safety for future behavior of the system.

Further approaches with interesting insights on combined verification and monitor/controller synthesis for discrete systems include, for instance, [2,11].

Although the related approaches based on offline verification derive monitors and switching conditions from models, none of them validates whether or not the model is adequate for the current execution. Thus, they are vulnerable to deviation between the real world and the model. In summary, this paper addresses safety at runtime as follows:

- Unlike [36], who focus on timing faults and specification violations, we propose a systematic principle to derive monitors that react to any deviation from the model.
- Unlike [4,15,17,21], who focus on the discrete aspects of CPS, we use hybrid system models with differential equations to address controller and plant.
- Unlike [17,36], who assume that fail-safe controllers have been verified with some other approach and do not synthesize code, we can use the same technical approach ($\mathsf{d}\mathcal{L}$) for verifying controllers and synthesizing provably correct monitors.
- ModelPlex combines the leight-weight monitors and runtime compliance of online runtime verification with the design time analysis of offline verification.
- ModelPlex synthesizes provably correct monitors, certified by a theorem prover
- To the best of our knowledge, our approach is the first to guarantee that verification results about a hybrid systems model transfer to a particular execution of the system by verified runtime validation. We detect deviation from the verified model when it first occurs and, given bounds, can guarantee safety with fail-safe fallback. Other approaches (e. g., [3,17,36]) assume the system perfectly complies with the model.

6 Conclusion

ModelPlex is a principle to build and verify high-assurance controllers for safety-critical computerized systems that interact physically with their environment. It guarantees that verification results about CPS models transfer to the real system by safeguarding against deviations from the verified model. Monitors created by ModelPlex are provably correct and check at runtime whether or not the actual behavior of a CPS complies with the verified model and its assumptions. Upon noncompliance, ModelPlex initiates fail-safe fallback strategies. In order to initiate those strategies early enough, ModelPlex uses prediction on the basis of disturbed plant models to check safety for the next control cycle. This way, ModelPlex ensures that verification results about a model of a CPS transfer to the actual system behavior at runtime.

Future research directions include extending ModelPlex with advanced $\mathsf{d}\mathcal{L}$ proof rules for differential equations [31], so that differential equations without polynomial solutions, as we currently handle for prediction monitor synthesis, can be handled for model monitor synthesis as well. An interesting question for certification purposes is end-to-end verification from the model to the final machine code.

References

1. Aiello, A.M., Berryman, J.F., Grohs, J.R., Schierman, J.D.: Run-time assurance for advanced flight-critical control systems. In: AIAA Guidance, Nav. and Control Conf. AIAA (2010)
2. Alur, R., Bodík, R., Juniwal, G., Martin, M.M.K., Raghothaman, M., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.: Syntax-guided synthesis. In: FMCAD. pp. 1–17. IEEE (2013)
3. Bak, S., Greer, A., Mitra, S.: Hybrid cyberphysical system verification with Simplex using discrete abstractions. In: Caccamo, M. (ed.) IEEE Real-Time and Embedded Technology and Applications Symposium. pp. 143–152. IEEE Computer Society (2010)
4. Bartocci, E., Grosu, R., Karmarkar, A., Smolka, S.A., Stoller, S.D., Zadok, E., Seyster, J.: Adaptive runtime verification. In: Qadeer, S., Tasiran, S. (eds.) RV. LNCS, vol. 7687, pp. 168–182. Springer (2012)
5. Blech, J.O., Falcone, Y., Becker, K.: Towards certified runtime verification. In: Aoki, T., Taguchi, K. (eds.) ICFEM. LNCS, vol. 7635, pp. 494–509. Springer (2012)
6. Cimatti, A., Mover, S., Tonetta, S.: SMT-based scenario verification for hybrid systems. Formal Methods in System Design 42(1), 46–66 (2013)
7. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. 12(3), 299–328 (1991)
8. Daigle, M.J., Roychoudhury, I., Biswas, G., Koutsoukos, X.D., Patterson-Hine, A., Poll, S.: A comprehensive diagnosis methodology for complex hybrid systems: A case study on spacecraft power distribution systems. IEEE Transactions on Systems, Man, and Cybernetics, Part A 40(5), 917–931 (2010)
9. D'Angelo, B., Sankaranarayanan, S., Sánchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: LOLA: Runtime monitoring of synchronous systems. In: TIME. pp. 166–174. IEEE Computer Society (2005)
10. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV. LNCS, vol. 8044, pp. 264–279. Springer (2013)
11. Ehlers, R., Finkbeiner, B.: Monitoring realizability. In: Khurshid, S., Sen, K. (eds.) RV. LNCS, vol. 7186, pp. 427–441. Springer (2011)
12. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV. LNCS, vol. 6806, pp. 379–395. Springer (2011)
13. Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE. LNCS, vol. 7898, pp. 208–214. Springer (2013)
14. Havelund, K., Rosu, G.: Efficient monitoring of safety properties. STTT 6(2), 158–173 (2004)
15. Kalajdzic, K., Bartocci, E., Smolka, S.A., Stoller, S.D., Grosu, R.: Runtime verification with particle filtering. In: Legay, A., Bensalem, S. (eds.) RV. LNCS, vol. 8174. Springer (2013)

16. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebr. Program. 78(5), 293–303 (2009)
17. Liu, X., Wang, Q., Gopalakrishnan, S., He, W., Sha, L., Ding, H., Lee, K.: ORTEGA: An efficient and flexible online fault tolerance architecture for real-time control systems. IEEE Trans. Industrial Informatics 4(4), 213–224 (2008)
18. Loos, S.M., Platzer, A., Nistor, L.: Adaptive cruise control: Hybrid, distributed, and now formally verified. In: Butler, M., Schulte, W. (eds.) FM. LNCS, vol. 6664. Springer (2011)
19. McIlraith, S.A., Biswas, G., Clancy, D., Gupta, V.: Hybrid systems diagnosis. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC. LNCS, vol. 1790, pp. 282–295. Springer (2000)
20. Meredith, P.O., Jin, D., Griffith, D., Chen, F., Rosu, G.: An overview of the MOP runtime verification framework. STTT 14(3), 249–289 (2012)
21. Meredith, P.O., Rosu, G.: Runtime verification with the RV system. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G.J., Rosu, G., Sokolsky, O., Tillmann, N. (eds.) RV. LNCS, vol. 6418, pp. 136–152. Springer (2010)
22. Mitsch, S., Ghorbal, K., Platzer, A.: On provably safe obstacle avoidance for autonomous robotic ground vehicles. In: Robotics: Science and Systems (2013)
23. Mitsch, S., Loos, S.M., Platzer, A.: Towards formal verification of freeway traffic control. In: Lu, C. (ed.) ICCPS. pp. 171–180. IEEE (2012)
24. Mitsch, S., Passmore, G.O., Platzer, A.: Collaborative verification-driven engineering of hybrid systems. J. Math. in Computer Science (2014)
25. Mitsch, S., Platzer, A.: ModelPlex: Verified runtime validation of verified cyber-physical system models. Tech. Rep. CMU-CS-14-121, Carnegie Mellon (2014)
26. Nickovic, D., Maler, O.: AMT: A property-based monitoring tool for analog systems. In: Raskin, J.F., Thiagarajan, P.S. (eds.) FORMATS. pp. 304–319. LNCS, Springer (2007)
27. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. 41(2), 143–189 (2008)
28. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. 20(1), 309–352 (2010), advance Access published on November 18, 2008
29. Platzer, A.: Logical Analysis of Hybrid Systems. Springer (2010)
30. Platzer, A.: The structure of differential invariants and differential cut elimination. Logical Methods in Computer Science 8(4) (2011)
31. Platzer, A.: The complete proof theory of hybrid systems. In: LICS. IEEE (2012)
32. Platzer, A.: Logics of dynamical systems. In: LICS. pp. 13–24. IEEE (2012)
33. Platzer, A., Clarke, E.M.: The image computation problem in hybrid systems model checking. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC. LNCS, Springer (2007)
34. Platzer, A., Quesel, J.D.: KeYmaera: A hybrid theorem prover for hybrid systems. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR. LNCS, vol. 5195. Springer (2008)
35. Platzer, A., Quesel, J.D.: European Train Control System: A case study in formal verification. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM. LNCS, vol. 5885. Springer (2009)
36. Seto, D., Krogh, B., Sha, L., Chutinan, A.: The Simplex architecture for safe online control system upgrades. In: American Control Conference. pp. 3504–3508 (1998)
37. Shannon, C.: Communication in the presence of noise. Proc. of the IRE 37(1), 10–21 (1949)
38. Srivastava, A.N., Schumann, J.: Software health management: a necessity for safety critical systems. ISSE 9(4), 219–233 (2013)
39. Wang, D., Yu, M., Low, C.B., Arogeti, S.: Model-based Health Monitoring of Hybrid Systems. Springer (2013)
40. Wang, S., Ayoub, A., Sokolsky, O., Lee, I.: Runtime verification of traces under recording uncertainty. In: Khurshid, S., Sen, K. (eds.) RV. pp. 442–456. LNCS, Springer (2011)
41. Zhao, F., Koutsoukos, X.D., Haussecker, H.W., Reich, J., Cheung, P.: Monitoring and fault diagnosis of hybrid systems. IEEE Transactions on Systems, Man, and Cybernetics, Part B 35(6), 1225–1240 (2005)