Modular Deductive Verification of Sampled-Data Systems

Daniel Ricketts
UC San Diego
La Jolla, California 92037
daricket@cs.ucsd.edu

Gregory Malecha
UC San Diego
La Jolla, California 92037
gmalecha@gmail.com

Sorin Lerner
UC San Diego
La Jolla, California 92037
lerner@cs.ucsd.edu

ABSTRACT

Unsafe behavior of cyber-physical systems can have disastrous consequences, motivating the need for formal verification of these kinds of systems. Deductive verification in a proof assistant such as Coq is a promising technique for this verification because it (1) justifies all verification from first principles, (2) is not limited to classes of systems for which full automation is possible, and (3) provides a platform for proving powerful, higher-order modularity theorems that are crucial for scaling verification to complex systems. In this paper, we demonstrate the practicality, utility, and scalability of this approach by developing in Coq sound and powerful rules for modular construction and verification of sampled-data cyber-physical systems. We evaluate these rules by using them to verify a number of non-trivial controllers enforcing safety properties of a quadcopter, e.g. a geo-fence. We show that our controllers are realistic by running them on a real, flying quadcopter.

1. INTRODUCTION

Errors in cyber-physical software can lead to disastrous consequences. These consequences mean that such systems demand the most rigorous verification techniques. There has been a variety of work on developing fully-automated tools for verification of cyber-physical systems [11, 16], but due to the complexity of the domain, these tools are only able to verify particular classes of systems and properties. On the other hand, all cyber-physical systems are in range for deductive verification in a proof assistant, at least in theory. However, one of the typically-stated drawbacks of verification in proof assistants is the extremely high manual labor cost required to produce these proofs.

In this paper, we demonstrate how to mitigate this burden using modular verification techniques. These techniques allow us to verify systems that are outside the scope of fully automated techniques, such as hybrid system model checkers, without a massive amount of manual labor.

We focus on the modular construction and verification of *sampled-data systems* [7]. In this important class of systems, there is a single discrete controller that runs periodically. In between executions of the controller, the system evolves according to continuous physical dynamics.

For this class of systems, one of the challenges is in ensuring that the specification of the discrete controller is always enabled, i.e. it always specifies at least one successor state. While this might seem trivial, consider the following scenario. Suppose we have built a module that prevents a quadcopter from exceeding some maximum altitude. Furthermore, suppose we have also built another module that prevents the quadcopter from violating some *minimum* altitude. If we have separately verified that these two modules enforce their respective properties, we would like to compose them in parallel to guarantee both properties simultaneously. That is, we would like the composed system to guarantee that the system never goes too high or too low. However, this is not always possible; a module could enforce the upper bound on altitude by always accelerating downwards. Likewise, a module could enforce the lower bound on altitude by always accelerating upwards. Clearly, naïvely composing the controllers of these modules in parallel would result in a system that gets stuck – there is never an acceleration that both controllers can agree on.

In this paper, we present sound techniques for resolving this and other potential pitfalls for reusing and composing modules for sampled-data systems. We observed that modularity is facilitated by separating verification into two parts: *preservation* and *progress*. Preservation ensures that the model guarantees the safety property inductively, while progress ensures that the system model is always enabled. This separation facilitates the application of several simple, general, and powerful operators, namely substitution, conjunction, and disjunction. More precisely, we state sufficient conditions for applying these operators to individual modules to produce a new sampled-data system with the desired properties (e.g. the conjunction of the safety properties of conjoined modules). Crucially, these sufficient conditions are in terms of preservation and progress.

To validate the expressiveness of our theorems, we apply them in the context of *quadcopters*, by showing how to compose several simple verified controllers together in different ways to produce many different verified composed controllers. To ensure that our controllers are practical, we run them on an actual quadcopter. In summary:

- We implement in the Coq proof assistant a general approach for modular verification of sampled-data systems by separately exposing proofs of preservation and

EMSOFT'16, October 01-07 2016, Pittsburgh, PA, USA

ISBN 978-1-4503-4485-2/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2968478.2968495

progress. The development is available from the project webpage: http://veridrone.ucsd.edu.

- We apply this approach to build and verify arbitrary 3D geofences for a quadcopter, including walls, boxes, and rectangular donuts, starting with two simple verified 1D controllers. We show that our modular verification techniques keep the proof burden manageable.
- We evaluate our geo-fences by running them on an actual quadcopter, and show that they work in practice.
- We discuss the capabilities of three state-of-the-art fully-automated tools (SpaceEx, Flow*, and dReach) in verifying our geo-fence controllers.

2. OVERVIEW

We start with an informal description of the operators that our theory covers: substitution, conjunction, and disjunction. We then give an overview of verifying controllers using these operators. Finally, we describe how we applied this to build a verified family of geo-fences for a quadcopter.

Operators. *Substitution* of expressions for variables represents a form of reuse, allowing us to transform systems and their properties into a different coordinate system. For example, given a model of a system defined in the x-y plane, the substitution $\{\boldsymbol{x} \mapsto \boldsymbol{r}\cos\boldsymbol{\phi},\ \boldsymbol{y} \mapsto \boldsymbol{r}\sin\boldsymbol{\phi}\}$ transforms the model to polar coordinates, the substitution $\{\boldsymbol{x} \mapsto \boldsymbol{y},\ \boldsymbol{y} \mapsto \boldsymbol{x}\}$ rotates the system, and the substitution $\{\boldsymbol{x} \mapsto \boldsymbol{x} + 5\}$ translates the system by 5 units in the $\boldsymbol{x}$ dimension. However, not all substitutions soundly transport both systems *and* their properties; our theory (Section 5.1) gives formal conditions under which substitutions are permitted.

Disjunction of two systems represents nondeterministic choice between the controllers of the system. For example, if we have a controller that prevents a quadcopter from flying too far to the west and another controller that prevents a quadcopter from flying too far north, then their disjunction enforces a north-west no-fly zone – the quadcopter must stay to the north *or* to the west of the no-fly zone. Unlike conjunction, there is no risk of the composed system getting stuck. Instead, the challenge with disjunction is in constraining the nondeterministic choice between the controllers. Our theorems and definitions in Section 5.2 make this formal by including the inductive invariants of each system within the composed controller.

Conjunction of two systems represents parallel composition of these systems. For example, if we have a system that enforces an upper bound on velocity and a system that enforces a lower bound on velocity, then their conjunction enforces both an upper and a lower bound on velocity. We can also conjoin systems that control or restrict different variables, such as a system that enforces a bound on velocity and a system that enforces a bound on position. However, as discussed in the introduction, the challenge of applying this operator is in ensuring that the conjoined systems never get stuck, e.g. when the controller of one system requires positive acceleration while the other requires negative acceleration. Again, our theory (Section 5.3) gives formal conditions under which conjunctive composition is possible.

Note that disjunctive and conjunctive composition are related to alternative and parallel composition.

Controller Verification. To illustrate how the operators work, we explain the construction and modular verification

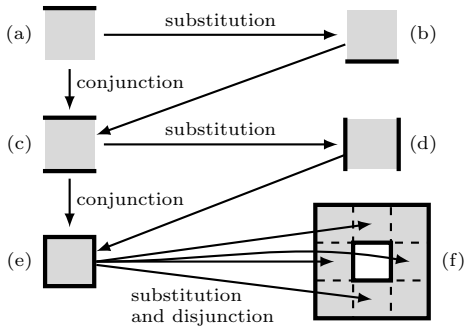

Figure 1: Overview of construction and verification of position bounding controllers.

of several general purpose controllers for enforcing state constraints, depicted in Figure 1. We begin with a simple verified module: a controller that enforces an upper bound on position in one spatial dimension by controlling acceleration (depicted by (a) in Figure 1). We use substitution to "mirror" this module and its correctness property, thus obtaining a module (b) that enforces a lower bound on position, again in one dimension. We conjoin these two modules to form a controller (c) enforcing upper and lower bounds on position, still in one dimension. We use substitution to rotate this interval controller into a second, orthogonal dimension (d), then conjoin (c) and (d) to form a controller (e) enforcing a 2 dimensional rectangle, i.e. upper and lower bounds on position in two dimensions. Finally, we use substitution to build and verify four translated copies of (e) and disjunction of these four copies to enforce a rectangular donut (f). We use disjunction to enforce that the system must, at all times, be in the first copy of (e), the second, the third, *or* the fourth. Moreover, since the rectangles are overlapping, the system can transition from one rectangle to another.

Quadcopter. Although the above approach can enforce state constraints for a variety of applications (e.g. trains, intelligent cruise control), we evaluate our approach in the context of *quadcopter* controllers that enforce position and velocity bounds. Connecting the verification methodology above to quadcopters required application of the three operators under the complex, coupled dynamics of a quadcopter, thus showcasing the applicability of our rules to solve complex problems. Crucially, our Coq theorems for each of these operators give formal conditions under which this is sound.

Ultimately, we were able to use the verification techniques in Figure 1 to build a three dimensional bounding box of both position and velocity for the quadcopter. This bounding cube provides a powerful building block for constructing "pixelated shapes" (analogous to (f) in Figure 1), which can be used to enforce interesting shapes such as a flying around and over but not near the pilot. The results of our verification along with actual flight tests are in Section 6.

3. SAMPLED-DATA SYSTEMS

Before presenting our approach, we describe our model of cyber-physical systems and in particular, sampled-data systems. All of our work is formalized inside the Coq proof assistant, but for exposition purposes, we focus on the mathematical concepts rather than concrete Coq syntax. In the remainder of this section we cover relevant background on

our temporal logic, the way we use it to describe sampled-data systems, and give a simple example illustrating the difficulty of modularity in this domain.

Linear Temporal Logic. Based on work by Ricketts *et al* [26], we encode sampled-data systems and their properties within discrete-time linear temporal logic (LTL). An LTL formula specifies the possible traces of a system. In our model, a trace is an infinite sequence of states representing observations of a system at discrete points in time. Formally, a state is a mapping from variables to real numbers. Inspired by Lamport's Temporal Logic of Actions (TLA) [18], our logic consists of *state formulas* (predicates over a single state), *action formulas* (state relations specifying system transitions), and *trace formulas* (predicates over traces). In action formulas, the values of variables in the current state are notated using bold script, e.g. $\boldsymbol{x}$, while the values of variables in the next state use bold script with a prime, e.g. $\boldsymbol{x}'$. Variables not mentioned in a formula are unconstrained.

For example, the following formula describes a system where the initial value of $\boldsymbol{x}$ is 0 and the value of $\boldsymbol{x}$ is incremented during each transition.

$$\boldsymbol{x} = 0 \ \wedge \ \Box\left(\boldsymbol{x}' = \boldsymbol{x} + 1\right)$$

The initial condition ($\boldsymbol{x} = 0$) is a state formula. The transition relation ($\boldsymbol{x}' = \boldsymbol{x} + 1$) is an action formula and refers to values in the next state using a prime, e.g. $\boldsymbol{x}'$. Both the transition relation and the property are lifted to trace formulas using the always modality ($\Box$). When always is applied to an action formula, it means that all pairs of temporally adjacent states are related by the action formula. When always is applied to a state formula, it means that all states satisfy the property.

Finally, we use $tr \models P$ to denote that formula P holds on trace tr and $P \vdash Q$ to denote that Q holds on all traces that P holds on ($\vdash P$ denotes $\mathsf{True} \vdash P$). For example, the following states that all traces of the above system have the property that $\boldsymbol{x}$ is always at least 0.

$$\vdash \boldsymbol{x} = 0 \wedge \Box\left(\boldsymbol{x}' = \boldsymbol{x} + 1\right) \rightarrow \Box\left(\boldsymbol{x} \geq 0\right)$$

The implication means that the traces of the system are a subset of the traces for which $\boldsymbol{x}$ is at least 0 in all states.

Sampled-data systems in LTL. In a sampled-data system, the state repeatedly transitions either continuously according to some differential (in)equations or discretely according to the (possibly nondeterministic) controller. In addition, the elapsed time between discrete transitions of the controller is bounded by some constant. In LTL, we can model such systems using a formula of the form

$$I \wedge \Box(\mathsf{Sys}_\Delta \ D \ \mathcal{W})$$

Here, we borrow from [26] the action formula $\mathsf{Sys}_\Delta \ D \ \mathcal{W}$, specifying transitions of a sampled-data system, where: D is an action formula specifying the discrete controller, and $\mathcal{W}$ is a system of differential (in)equalities specifying the continuous transition. Formally,

$$\begin{aligned}\mathsf{Sys}_\Delta \ D \ \mathcal{W} \triangleq \\ \quad & D \wedge \boldsymbol{\tau} = 0 \wedge 0 < \boldsymbol{\tau}' \leq \Delta \\ \vee \quad & \mathsf{Continuous}\ (\mathcal{W} \wedge \dot{\boldsymbol{\tau}} = -1) \wedge \boldsymbol{\tau}' \geq 0\end{aligned}$$

In this action formula, the disjunction captures the fact that the system transitions either continuously according to the physical world or discretely according to the controller. The definition encapsulates both the semantics of the continuous transition and the timing characteristics of the system.

Informally, $\mathsf{Continuous}(\mathcal{W})$ means that the state evolves for *some* amount of time according to a continuous function satisfying the differential (in)equalities in $\mathcal{W}$. Formally, $\mathsf{Continuous}(\mathcal{W})$ is an LTL action formula, defined as follows:

$$\begin{aligned}&\mathsf{Continuous}(\dot{x_1} \sim_1 e_1, \ldots, \dot{x_n} \sim_n e_n) \equiv \\ &\quad \exists(r : \mathbb{R})\,(f : \mathbb{R} \rightarrow \mathsf{Var} \rightarrow \mathbb{R}), 0 < r \\ &\qquad \wedge \mathsf{Solves}(f, \dot{x_1} \sim_1 e_1, \ldots, \dot{x_n} \sim_n e_n, r) \\ &\qquad \wedge x_1 = f(0, x_1) \wedge \ldots \wedge x_n = f(0, x_n) \\ &\qquad \wedge {x_1}' = f(r, x_1) \wedge \ldots \wedge {x_n}' = f(r, x_n)\end{aligned}$$

Here, r is the amount of time that the system evolves for, and f is a solution to the differential (in)equations, expressed by the Solves predicate (defined using Coq's real analysis library) with $\sim_i \in \{=, <, \leq, >, \geq\}$. The final two lines relate the current state to the value of the solution f at 0 and the next state to the value of f at time r.

At first glance, this definition of continuous transitions may look strange since it seems to allow the trace to "skip" states. However, while a single trace may skip a certain state during a continuous transition, another trace does include that state because the definition of $\mathsf{Continuous}$ captures *all* possible continuous transitions of any duration. Thus, a formula of the form $I \wedge \Box(\mathsf{Sys}_\Delta \ D \ \mathcal{W})$ captures all possible sequences of discrete observations of a system. The soundness of this encoding is argued by Lamport in [19].

The timing constraint is captured using the variable $\boldsymbol{\tau}$ (not mentioned in D or $\mathcal{W}$), which tracks the time that elapses between successive transitions of the discrete controller. During the continuous evolution of the system, $\boldsymbol{\tau}$ decreases at the same rate as time, i.e. $\dot{\boldsymbol{\tau}} = -1$, and $\boldsymbol{\tau}' \geq 0$ ensures that no more than Δ time elapses between successive discrete transitions of the controller. The discrete transition occurs when the timer has expired ($\boldsymbol{\tau} = 0$); this transition resets the timer to a positive value that is at most Δ.

Notations. Throughout this paper when X is a system, we use $\mathcal{D}_X$ and $\mathcal{W}_X$ to denote the discrete and continuous transitions of X, respectively. Also, we use the inductive invariant of a system as its initial condition; thus we use I to denote an inductive invariant and I_X to denote the inductive invariant of system X. In practice, one must prove that the initial condition of a system implies the inductive invariant.

Stuck Specifications. The physical world always evolves because time always evolves. Cyber-physical system specifications should adhere to this property – the specification should never reach a state in which it is stuck, i.e. in which a transition is impossible. For example, consider the system $\mathsf{Sys}_\Delta \ \mathsf{False} \ \mathcal{W}$. In this system, there is never a discrete transition (expressed using the unsatisfiable action formula False). Since a discrete transition never occurs, a continuous transition is not possible once time reaches Δ. Readers familiar with Zeno specifications [1] will note that Sys specifications that are never stuck are non-Zeno.

We rule out such specifications using a new abstraction called System, defined as follows:

$$\mathsf{System}_\Delta \ D \ \mathcal{W} \triangleq \mathsf{Sys}_\Delta \ D \ \mathcal{W} \vee \neg\mathsf{Enabled}\left(\mathsf{Sys}_\Delta \ D \ \mathcal{W}\right)$$

In the above, $\mathsf{Enabled}$ takes an action formula and returns a state formula. In particular, $\mathsf{Enabled}(A)$ holds on a given state st iff there exists a next state st' such that $(st, st') \in A$, i.e. the system can take an A transition. A specification

whose transition is built using System can never become stuck; if the underlying Sys becomes stuck (not Enabled), then the clause $\neg\mathsf{Enabled}\left(\mathsf{Sys}_\Delta\ D\ \mathcal{W}\right)$ conservatively expresses that anything can happen. Informally, we will not be able to prove any interesting global properties of a System when the underlying Sys can reach a state in which it is not Enabled since we will know nothing about the next state.

It may seem trivial to avoid writing stuck specifications for sampled-data systems, and thus the distinction between Sys and System appears to be only theoretical. However, Section 4 will show that avoiding stuck specifications is a core challenge of building sampled-data systems modularly.

4. A MODULAR BASIS FOR REASONING

In this section we present the foundation of our approach to modular reasoning about sampled-data systems: separating proofs into preservation and progress. This foundation will allow us to build the theory for applying substitution, conjunction, and disjunction (presented in Section 5).

In general, our end goal is to prove properties of the form:

$$\vdash I \wedge \Box(\mathsf{System}_\Delta\ D\ \mathcal{W}) \rightarrow \Box S$$

This property states that, starting with initial condition I, condition S always holds if at each point in the trace, the transition relation is described by $(\mathsf{System}_\Delta\ D\ \mathcal{W})$. Unfortunately, properties like the one above are not modular. For example, suppose we have two discrete transitions D_1 and D_2 which independently ensure S_1 and S_2, i.e.

$$\vdash I \wedge \Box(\mathsf{System}_\Delta\ D_1\ \mathcal{W}) \rightarrow \Box S_1$$
$$\vdash I \wedge \Box(\mathsf{System}_\Delta\ D_2\ \mathcal{W}) \rightarrow \Box S_2$$

We would like to combine these proofs to show that $S_1 \wedge S_2$ is an invariant of the conjoined system $\mathsf{System}_\Delta\ (D_1 \wedge D_2)\ \mathcal{W}$. Unfolding the definition of System reveals that this is not, in general, true. The problem is that $\mathsf{Enabled}\ D_1 \wedge \mathsf{Enabled}\ D_2$ does not necessarily imply $\mathsf{Enabled}\ (D_1 \wedge D_2)$.[1] This formalizes the challenge described in the introduction – naïve parallel composition (conjunction) of controllers can result in a controller that gets stuck.

Crucially, Enabled is inherently non-modular, so global invariant proofs of systems specified using System are inherently non-modular. By ruling out stuck specifications, we also rule out the modularity of *global* invariant proofs.

Regaining Modularity. The key to regaining modularity is a shift from global proofs to local ones. In particular, we will make the inductive invariant of the system explicit and use it to prove two properties independently: preservation of the invariant, and progress of the system under the invariant. As we will see in Section 5, this decomposition of the global property into local ones makes it much easier to combine and re-use systems and their proofs.

Preservation. Preservation of property (I) under an action formula states if I holds in the current state then it holds in the next state. Formally,

$$\mathsf{SysPreserves}\ I\ \left(\mathsf{Sys}_\Delta\ D\ \mathcal{W}\right)\ \triangleq\ I \wedge \mathsf{Sys}_{\mathsf{inv}} \wedge \mathsf{Sys}_\Delta\ D\ \mathcal{W} \rightarrow I'$$

where I' represents the state formula I with all variables primed. The $\mathsf{Sys}_{\mathsf{inv}}$ premise expresses the invariants guaranteed by the Sys abstraction, namely that no more than Δ time elapses between discrete transitions.

[1] Consider $\mathsf{Enabled}\ (\boldsymbol{x}' = 1\ \wedge\ \boldsymbol{x}' = 0)$

Progress. Progress under an invariant justifies that the system is Enabled assuming the invariant. Formally,

$$\mathsf{SysProgress}\ I\ \ (\mathsf{Sys}_\Delta\ D\ \mathcal{W}) \triangleq$$
$$I \wedge \mathsf{Sys}_{\mathsf{inv}} \rightarrow \mathsf{Enabled}\ (\mathsf{Sys}_\Delta\ D\ \mathcal{W})$$

This condition allows us to prove that a Sys and a System describe exactly the same system.

Note that, here, progress is a safety property that is closely related to the notion of progress in programming languages. It is different than progress properties in distributed systems, and it is different than convergence to an equilibrium in control theory.

Combining Preservation & Progress. The combination of preservation of and progress under the *same* inductive invariant is sufficient to prove that the invariant is a global invariant of the corresponding System, which is ultimately our goal. This is captured by the following theorem:

THEOREM 1. LOCALTOGLOBAL

$$\begin{array}{ll} & \mathsf{SysPreserves}\ I\ (\mathit{Sys}_\Delta\ D\ \mathcal{W}) \\ \wedge & \mathsf{SysProgress}\ I\ \ (\mathit{Sys}_\Delta\ D\ \mathcal{W}) \\ \wedge & I \rightarrow S \\ \vdash & I \wedge \Box\ (\mathsf{System}_\Delta\ D\ \mathcal{W}) \rightarrow \Box S \end{array}$$

5. MODULAR SAMPLED-DATA SYSTEMS

In this section, we show how to use preservation and progress to reason modularly about sampled-data systems. In particular, for each of our three operators (substitution, disjunction, and conjunction), we present theorems that state formal conditions under which application of the operator guarantees preservation and progress. We illustrate each of the operators and corresponding theorems by building verified state-constraining controllers for quadcopters. This allows us to construct and verify controllers enforcing policies such as "do not fly above 400 feet" (FAA regulation for recreational drones), "do not fly within 5 miles of an airport", and "do not fly within 5 feet of the pilot."

It is important to note that all of the state-constraining controllers that we verify are *non-deterministic*. This means that the discrete transitions do not compute a single value for each control variable (e.g. acceleration) but instead describe a set of allowed values that ensure the desired state-constraint. As we will see, this non-determinism is crucial for conjunctive composition (Section 5.3). In Section 6, we discuss how the actual implementation of these controllers resolves this non-determinism.

Building blocks. As the basic building blocks of our development, we start with two sampled-data systems developed and verified by Ricketts *et al.* [26]. Both operate in one spatial dimension, i.e.

$$\mathcal{W}_{1D}\ \ \triangleq\ \ \dot{\boldsymbol{y}} = \boldsymbol{v} \wedge \dot{\boldsymbol{v}} = \boldsymbol{a} \wedge \dot{\boldsymbol{a}} = 0$$

The two controllers each enforce constant bounds on a state variable by controlling acceleration $(\boldsymbol{a})$. The first-derivative controller (M_∂) bounds velocity using acceleration (the first-derivative of velocity). The second-derivative controller $(\mathsf{M}_{\partial^2})$ bounds position using acceleration (the second-derivative of position). To ensure that M_{∂^2} can stop before violating the boundary, the controller is parameterized by a_{min} which represents the braking acceleration and smallest possible acceleration (and is negative). Figure 2 gives the discrete transitions and inductive invariants for the two systems. Each

First-Derivative Controller ($\mathsf{M}_\partial = \mathsf{Sys}_\Delta\ D_\partial\ W_{1D}$)

$$\begin{aligned} D_\partial &\triangleq \ \mathcal{C}_a{}' \wedge (\boldsymbol{a}' \cdot \Delta + \boldsymbol{v} \leq ub \vee \boldsymbol{a}' \leq 0) \\ I_\partial &\triangleq \ (\boldsymbol{a} < 0 \rightarrow \boldsymbol{v} \leq ub) \wedge (\boldsymbol{a} \geq 0 \rightarrow \boldsymbol{a} \cdot \boldsymbol{\tau} + \boldsymbol{v} \leq ub) \end{aligned}$$

Second-Derivative Controller ($\mathsf{M}_{\partial^2} = \mathsf{Sys}_\Delta\ D_{\partial^2}\ W_{1D}$)

$$\begin{aligned} D_{\partial^2} \triangleq\ & (0 \leq \boldsymbol{v} + \boldsymbol{a}' \cdot \Delta \rightarrow \\ & \quad \mathsf{td}(\boldsymbol{v}, \boldsymbol{a}', \Delta) + \mathsf{sd}(\boldsymbol{v} + \boldsymbol{a}' \cdot \Delta) + \boldsymbol{y} \leq \mathsf{y_{ub}}) \\ & (\boldsymbol{v} + \boldsymbol{a}' \cdot \Delta \leq 0 \wedge 0 < \boldsymbol{v} \rightarrow \\ & \quad \mathsf{td}(\boldsymbol{v}, \boldsymbol{a}', \tfrac{-\boldsymbol{v}}{\boldsymbol{a}'}) + \boldsymbol{y} \leq \mathsf{y_{ub}}) \wedge \mathcal{C}_a{}' \\ I_{\partial^2} \triangleq\ & \forall t : \mathbb{R}, 0 \leq t \leq \boldsymbol{\tau} \rightarrow \\ & \quad \boldsymbol{y} + \mathsf{td}(\boldsymbol{v}, \boldsymbol{a}, t) + \mathsf{sd}(\max(0, \boldsymbol{v} + \boldsymbol{a} \cdot t)) \leq \mathsf{y_{ub}} \end{aligned}$$

where

$$\mathsf{td}(v, a, \Delta) \triangleq v \cdot \Delta + \tfrac{a \cdot \Delta^2}{2} \qquad \mathsf{sd}(v) \triangleq -\tfrac{v^2}{2 \cdot a_{\mathsf{min}}}$$
$$\mathcal{C}_a \triangleq a_{\mathsf{min}} \leq \boldsymbol{a} \qquad a_{\mathsf{min}} < 0$$

Figure 2: Discrete transitions and inductive invariants from [26].

inductive invariant states that, given the time until the next discrete transition, the system can stop before the boundary.

Ricketts *et al.* verified both of these controllers in a global style but did *not* compose them. We ported each of the global proofs to our local, modular specification by extracting the inductive invariant (which was stated explicitly in the proof) and the preservation proof (which formed the inductive case). Beyond extracting the safety proofs, we also had to verify progress, which was not addressed by Ricketts *et al.*, but is trivial for such basic modules. In the remainder of this section, we denote the preservation and progress proofs of the two controllers by: ∂-Preserves, ∂-Progress, ∂^2-Preserves, and ∂^2-Progress.

Quadcopter Controller. To build and verify controllers for quadcopters, we need a model of the physical dynamics of a quadcopter, called $\mathcal{W}_{QC}$:

$$\mathcal{W}_{QC} \triangleq \begin{pmatrix} & \dot{\boldsymbol{x}} = \boldsymbol{v_x} \wedge \dot{\boldsymbol{y}} = \boldsymbol{v_y} \wedge \dot{\boldsymbol{z}} = \boldsymbol{v_z} \\ \wedge & \dot{\boldsymbol{v_x}} = \boldsymbol{T} \cos \boldsymbol{\phi} \sin \boldsymbol{\theta} \\ \wedge & \dot{\boldsymbol{v_y}} = -\boldsymbol{T} \sin \boldsymbol{\phi} \\ \wedge & \dot{\boldsymbol{v_z}} = \boldsymbol{T} \cos \boldsymbol{\phi} \cos \boldsymbol{\theta} - g \\ \wedge & \dot{\boldsymbol{\phi}} = 0 \wedge \dot{\boldsymbol{\theta}} = 0 \wedge \dot{\boldsymbol{T}} = 0 \end{pmatrix}$$

Here $\boldsymbol{T}$ represents the combined thrust of the motors (normalized with respect to the mass of the quadcopter), $\boldsymbol{\theta}$ represents the pitch (the angle around the y-axis), and $\boldsymbol{\phi}$ represents the roll (the angle around the x-axis). Our model is based on the simplifying assumption (called the "small angle condition") that a trusted attitude controller can instantaneously achieve any pitch and roll within the bounds -30° to 30° with a thrust greater than or equal to 0, while holding yaw constant at 0.

$$\mathcal{C}_{\theta\phi} \triangleq |\boldsymbol{\theta}| \leq 30^\circ \wedge |\boldsymbol{\phi}| \leq 30^\circ \wedge 0 \leq \boldsymbol{T}$$

Prior work has suggested that this is a reasonable approximation under this small-angle condition ($\mathcal{C}_{\theta\phi}$), since the attitude dynamics are significantly faster than the velocity and position dynamics [13]. We capture this condition by requiring that all quadcopter controllers are enabled under $\mathcal{C}_{\theta\phi}$. That is, our goal is to build controllers D such that

$$\begin{aligned} &\mathsf{SysPreserves}\ I\ (\mathsf{Sys}_\Delta\ (D \wedge \mathcal{C}_{\theta\phi}{}')\ \mathcal{W}_{QC}) \wedge \\ &\mathsf{SysProgress}\ I\ (\mathsf{Sys}_\Delta\ (D \wedge \mathcal{C}_{\theta\phi}{}'))\ \mathcal{W}_{QC}) \end{aligned}$$

For some state-constraints and their corresponding controllers, it is only necessary to reason about an abstraction of the quadcopter dynamics $\mathcal{W}_{QC}$. For example, reasoning about a controller that enforces a bound on the vertical position $\boldsymbol{z}$ might only require reasoning about the portion of the dynamics on which $\boldsymbol{z}$ depends. We formalize this with:

$$\begin{aligned} &\mathsf{SysPreserves}\ I\ (\mathsf{Sys}_\Delta\ (D \wedge \mathcal{C}_{\theta\phi}{}'))\ \mathcal{W}) \wedge \\ &\mathsf{SysProgress}\ I\ (\mathsf{Sys}_\Delta\ (D \wedge \mathcal{C}_{\theta\phi}{}')\ \mathcal{W}) \\ &(\mathcal{W}_{QC} \rightarrow \mathcal{W}) \wedge \\ \vdash\ &\mathsf{SysPreserves}\ I\ (\mathsf{Sys}_\Delta\ (D \wedge \mathcal{C}_{\theta\phi}{}')\ \mathcal{W}_{QC}) \wedge \\ &\mathsf{SysProgress}\ I\ (\mathsf{Sys}_\Delta\ (D \wedge \mathcal{C}_{\theta\phi}{}')\ \mathcal{W}_{QC}) \end{aligned}$$

where $\mathcal{W}_{QC} \rightarrow \mathcal{W}$ states that $\mathcal{W}$ is an abstraction of $\mathcal{W}_{QC}$.

5.1 Reuse via Substitution

Substitution of expressions for variables is a simple but powerful operator that allows us to reuse controllers and their properties. For example, substitution allows us to perform familiar geometric transformations such as translations, reflections, scaling, and rotations. In addition, substitution allows us to project simple dynamics onto more complex dynamics; a technique we use to build verified state-constraining controllers for the quadcopter. We will explain the general technique by using it to transport (re-use) the second-derivative controller (M_{∂^2}), and its safety proof, to enforce a maximum altitude for our quadcopter.

For a formula P and substitution σ (map from variables to expressions), the semantic definition of substitution is:

$$tr \models \{\sigma\}P \quad \triangleq \quad \{\sigma\}tr \models P$$

which states that a substituted formula ($\{\sigma\}P$) holds on a trace (tr) if the formula (P) holds on the renamed trace ($\{\sigma\}tr$). Under this definition, application of substitution always guarantees preservation:

Theorem 2. SubstPreserves

$$\frac{\vdash \mathsf{SysPreserves}\ I\ S}{\vdash \mathsf{SysPreserves}\ (\{\sigma\}I)\ (\{\sigma\}S)}$$

This proof rule allows us to easily transport ∂^2-Preserves to the quadcopter. For example, using it we can conclude

$$\vdash \mathsf{SysPreserves}\ (\{\sigma_{\partial^2 \rightarrow QC}\}I_{\partial^2})\ (\{\sigma_{\partial^2 \rightarrow QC}\}\mathsf{M}_{\partial^2})$$
$$\sigma_{\partial^2 \rightarrow QC} \triangleq \{\boldsymbol{a} \mapsto \boldsymbol{T} \cos \boldsymbol{\phi} \cos \boldsymbol{\theta} - g,\ \boldsymbol{y} \mapsto \boldsymbol{z},\ \boldsymbol{v} \mapsto \boldsymbol{v_z}\}$$

Note that the first argument to SysPreserves is the inductive invariant for the new system, and can be read directly from the conclusion of the preservation theorem. This is the case for all of our preservation theorems.

Next, we need to justify the progress of the substituted system. The interaction between substitution and progress is a bit subtle because substitutions can introduce coupling between values that were uncoupled before the substitution. For example, $(\boldsymbol{x}' = 1 \wedge \boldsymbol{y}' = 0)$ is Enabled while $\{\boldsymbol{x} \mapsto \boldsymbol{z},\ \boldsymbol{y} \mapsto \boldsymbol{z}\}(\boldsymbol{x}' = 1 \wedge \boldsymbol{y}' = 0)$, which equals $\boldsymbol{z}' = 1 \wedge \boldsymbol{z}' = 0$, is not.

However, we can prove that *invertible* substitutions preserve progress. This is because the inverse of the substitution is a function for computing the Enabledness witness for the substituted system from the Enabledness witness for the original system. Because of this, we can actually state a stronger progress theorem for substitution, which captures the fact that the inverse substitution preserves known constraints on Enabledness witnesses, such as $\mathcal{C}_a$ for the first and

second derivative controllers. As we will see, this is crucial for proving the small-angle constraint ($\mathcal{C}_{\theta\phi}$). Formally,

THEOREM 3. SUBSTPROGRESS *For all formulas S, state formulas Q and R, and substitutions σ, if there exists a σ^{-1} such that $R \vdash (\sigma \circ \sigma^{-1})\,\boldsymbol{x} = \boldsymbol{x}$ for all variables $\boldsymbol{x}$ that occur primed in S, and if $R \vdash \{\sigma^{-1}\}Q$ then*

$$\frac{\vdash \mathsf{SysProgress}\ I\ (S \wedge R')}{\vdash \mathsf{SysProgress}\ \big(\{\sigma\}I\big)\ \big(\{\sigma\}S \wedge Q'\big)}$$

When we apply this theorem to prove the progress of the quadcopter altitude controller, the following inverse works:

$$\sigma^{-1}_{\partial^2 \to QC} \triangleq \boldsymbol{\phi} \mapsto 0, \boldsymbol{\theta} \mapsto 0, \boldsymbol{T} \mapsto \boldsymbol{a} + g, \boldsymbol{z} \mapsto \boldsymbol{y}, \boldsymbol{v_z} \mapsto \boldsymbol{v}$$

We instantiate Q with $\mathcal{C}_{\theta\phi}$ and R with $\mathcal{C}_a$ to guarantee

$$\mathsf{SysProgress}\ I\ \big(\mathsf{Sys}_\Delta\ (\{\sigma_{\partial^2 \to QC}\}D_{\mathsf{M}_{\partial^2}} \wedge \mathcal{C}_{\theta\phi}{}')\ (\{\sigma_{\partial^2 \to QC}\}\mathcal{W}_{1D})\big)$$

Finally, as noted above, we need to prove that $\mathcal{W}_{QC} \to \{\sigma_{\partial^2 \to QC}\}\mathcal{W}_{1D}$, i.e. the continuous dynamics produced by the substition is an abstraction of the full quadcopter dynamics. This reasoning involves standard substition within differential equations, which we have formalized and proved sound in Coq, and mechanical arithmetic reasoning.

Enforcing Planar Boundaries. Using our two substitution theorems, we can map the first- and second-derivative controllers onto the quadcopter dynamics in many ways, allowing us to verify many properties with relatively little effort. We showed how to use it to implement an upper bound on altitude. In general, we can use substitution on the second derivative controller to enforce that the quadcopter stays on one side of any 2D plane in 3D space. For example, we can enforce a maximum-west boundary to prevent the quadcopter from flying into a building. Similarly, by applying substitution to the first-derivative controller we can place bounds on velocity in any direction. The key is that our two substitution theorems allow us to transfer the correctness of any controller to work on a new dynamics that can be constructed from an invertible substitution.

5.2 Disjunctive Composition

In this section we present rules to compose systems using disjunction. For example, suppose that we wish to enforce a rectangular no-fly zone centered around the origin (depicted in Figure 3). A system can avoid the no-fly zone if at all times it is to the north, the south, the east, *or* to the west of the rectangle. We can build such a system by disjoining subsystems M_N, M_S, M_E, and M_W, which are each built from a substitution applied to M_{∂^2} to enforce the northern, southern, eastern, and western boundaries of the box, respectively. As we will see, separately exposing preservation and progress allows us to define a disjunction operator that is fully compositional and that permits the system to transition from an inductive invariant of one subsystem to another (e.g. north of the no-fly zone to west of the no-fly zone) during a single trace; this would not be possible with global invariant proofs.

The disjunctive composition of two systems is defined by the $\oplus$ operator, which is indexed by the inductive invariants of the two systems. Formally,

$$\begin{aligned}(\mathsf{Sys}_\Delta\ D_1\ \mathcal{W})\ {}^{I_1}\!\oplus^{I_2}\ (\mathsf{Sys}_\Delta\ D_2\ \mathcal{W}) \triangleq \\ \mathsf{Sys}_\Delta\ \big((I_1 \wedge D_1) \vee (I_2 \wedge D_2)\big)\ \mathcal{W}\end{aligned}$$

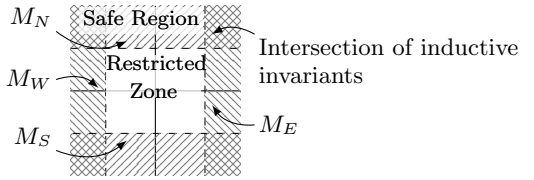

Figure 3: Staying out of restricted airspace using the disjunction of four controllers.

The inclusion of the *inductive* invariants in this definition is essential to enforce the disjunction of the properties. To see why, consider our example and suppose the system is currently along the western edge of the no-fly zone. Since the system is outside of the inductive invariant of the eastern controller, the eastern controller could allow the system to do anything, including moving east into the no-fly zone. Thus, at each discrete transition, the composed controller must consider the discrete transitions of a sub-controller whose inductive invariant currently holds. This guarantees that the inductive invariant of that subsystem holds after the transition. Moreover, it means that the system can transition from one inductive invariant to another, only where the inductive invariants overlap.

This definition of disjunction is fully compositional with both SysPreserves and SysProgress. In particular, the disjunctive composition of two systems that independently preserve I_A and I_B preserves $I_A \vee I_B$:

THEOREM 4. DISJOINPRESERVES

$$\begin{aligned}&\mathsf{SysPreserves}\ I_A\ A\ \wedge\ \mathsf{SysPreserves}\ I_B\ B \\ \vdash\ &\mathsf{SysPreserves}\ \big(I_A \vee I_B\big)\ \big(A\ {}^{I_A}\!\oplus^{I_B}\ B\big)\end{aligned}$$

Using only this theorem we can easily construct a proof that the disjunctive composition of our four controllers enforces the no-fly zone property.

A similar theorem states that $\oplus$ guarantees SysProgress.

THEOREM 5. DISJOINPROGRESS

$$\begin{aligned}&\mathsf{SysProgress}\ I_A\ A\ \wedge\ \mathsf{SysProgress}\ I_B\ B \\ \vdash\ &\mathsf{SysProgress}\ (I_A \vee I_B)\ \big(A\ {}^{I_A}\!\oplus^{I_B}\ B\big)\end{aligned}$$

The proof follows from the fact that I_A ensures the Enabledness of A and I_B ensures the Enabledness of B. Since the new inductive invariant $(I_A \vee I_B)$ ensures that at least one of I_A or I_B holds, at least one of A or B must be Enabled.

Disjunction is a very powerful composition mechanism that is applicable in a wide variety of circumstances. For example, we can use it to guarantee that a train can only have a high velocity when it is not in a curve by composing a maximum velocity controller with a controller that stops the train before curves. A controller such as this one could have prevented the Amtrak derailment in Philadelphia in 2015 that killed 8 people and injured 200.

5.3 Conjunctive Composition

In this section, we present rules to compose systems to ensure the conjunction of their properties. For example, one might want to ensure that a system's upward velocity does not exceed 1 m/s *and* that the system stays below 100m by conjoining the first and second derivative controllers. Unlike disjunctive composition, conjunctive composition of two systems satisfying progress does not guarantee a system satis-

fying progress, due to coupling. However, our ability to separately prove progress allows us to push forward. We show that, even in coupled domains, conjunctive composition can lead to substantial savings in proof effort, and demonstrate some clever tricks that allow us to decouple domains that, on the surface, seem intricately linked.

The conjunction of two systems is defined as follows:

$$\mathsf{Sys}_\Delta\ D_1\ \mathcal{W}_1 \otimes \mathsf{Sys}_\Delta\ D_2\ \mathcal{W}_2 \triangleq \mathsf{Sys}_\Delta\ (D_1 \wedge D_2)\ (\mathcal{W}_1 \wedge W_2)$$

The crucial feature of this definition is how it interacts with SysPreserves.

THEOREM 6. CONJOINPRESERVES

$$\begin{array}{ll} & \mathsf{SysPreserves}\, I_A\ A\ \wedge\ \mathsf{SysPreserves}\, I_B\ B \\ \vdash & \mathsf{SysPreserves}\, \big(I_A \wedge I_B\big)\ \big(A \otimes B\big) \end{array}$$

Intuitively, if we start in a state satisfying both I_A and I_B, A guarantees that we stay in I_A, and B guarantees that we remain in I_B, then if both A and B hold, we must remain in the intersection of I_A and I_B.

The difficulty of conjunctive composition lies in justifying progress. Even though A and B may independently be Enabled under the inductive invariant, there is no guarantee that their conjunction is Enabled. For example, suppose that we wish to compose an overly-conservative upper-bound controller that insists on a negative acceleration and a similarly conservative lower-bound controller that insists on a positive acceleration. Since acceleration can not be simultaneously positive and negative, the conjunction of these controllers does not satisfy progress.

Nevertheless, all is not lost when conjoining two systems. There are a variety of techniques for proving Enabledness of conjunctions. We will illustrate these techniques through a sequence of examples, ultimately culminating in a 3D bounding box for both position and velocity.

Example: Staying within an Interval. Consider constructing a controller that enforces both an upper and a lower bound on both position (y) and velocity (v) in a single spatial dimension. We can build such a controller (which we call Int) using $\otimes$ and an application of our substitution operator to the second- and first-derivative controllers:

$$\mathsf{Int} \triangleq \mathsf{M}_{\partial^2} \otimes \{\sigma_-\}\mathsf{M}_{\partial^2} \otimes \mathsf{M}_{\partial} \otimes \{\sigma_-\}\mathsf{M}_{\partial}$$
$$\sigma_- \triangleq \{\boldsymbol{y} \mapsto -\boldsymbol{y},\ \boldsymbol{v} \mapsto -\boldsymbol{v},\ \boldsymbol{a} \mapsto -\boldsymbol{a}\}$$

Here, the σ_- substitution mirrors the controller's logic so that rather than enforcing an upper bound on $\boldsymbol{y}$ of y_{ub} (resp. v_y of v_{ub}), the substituted controller enforces a lower bound on $\boldsymbol{y}$ of $-\mathsf{y}_{\mathsf{ub}}$ (resp. v_y of $-v_{\mathsf{ub}}$).

The preservation of this composition follows immediately from CONJOINPRESERVES, ∂^2-PRESERVES, ∂-PRESERVES, and SUBSTPRESERVES. However, since conjoined systems are not guaranteed to satisfy progress, we must prove this separately. Formally, we must prove progress of Int under the conjunction of the inductive invariants of the subsystems:

$$\mathsf{SysProgress}\ \big(I_{\partial^2} \wedge \{\sigma_-\}I_{\partial^2} \wedge I_{\partial} \wedge \{\sigma_-\}I_{\partial}\big)\ \mathsf{Int}$$

Informally, this states that, under the inductive invariant, there exists an action that is acceptable to all of the (nondeterministic) controllers. Unfolding the definitions and applying the substitution (σ_-) reveals that the progress of Int reduces to first-order reasoning over real arithmetic. At first glance, it may seem as though modularity was a failure here; however, by separating preservation and progress, the nonmodularity of progress did not prevent us from modularly proving preservation. Moreover, our split crucially allows us to assume the inductive invariant when proving progress.

This is in contrast to other work [4] where conjunctive (parallel) composition is only allowed when the individual modules output to disjoint sets of variables. Furthermore, we found the proof of $\mathsf{SysProgress}\ I_{\mathsf{Int}}\ \mathsf{Int}$ to be *less than one third* the size of the proof of preservation of the individual second-derivative controller M_{∂^2}. This means that $\otimes$ greatly reduces the cost of conjoining systems, even when these systems are tightly coupled.

In certain cases, our proof rules can be used to verify progress compositionally. To demonstrate this, we turn to the task of using our interval controller to modularly build a bounding rectangular prism. We approach the problem in three steps. First, we compose two interval controllers to enforce a bounding box in two spatial dimensions. In the next section, we adapt this controller to the quadcopter by incorporating the small-angle constraint. Finally, we apply the same technique to extend the 2D box into a 3D prism.

Example: Conjunction of Independent Systems. We construct the box by conjoining two instances of Int, using substitution to map them to the x- and z-dimensions respectively.

$$\begin{array}{rcl} \mathsf{Box} & \triangleq & \{\sigma_x\}\mathsf{Int} \otimes \{\sigma_z\}\mathsf{Int} \\ \sigma_x & \triangleq & \{\boldsymbol{y} \mapsto \boldsymbol{x},\ \boldsymbol{v} \mapsto \boldsymbol{v_x},\ \boldsymbol{a} \mapsto \boldsymbol{a_x}\} \\ \sigma_z & \triangleq & \{\boldsymbol{y} \mapsto \boldsymbol{z},\ \boldsymbol{v} \mapsto \boldsymbol{v_z},\ \boldsymbol{a} \mapsto \boldsymbol{a_z} - g\} \end{array}$$

Verifying that Box enforces a bounding box is simply a matter of using CONJOINPRESERVES, SUBSTPRESERVES, and the preservation proof of Int.

As others have noted [4], progress of conjoined systems is compositional, if the two systems output to disjoint sets of variables. In our logic, the output variables of a formula are the next-state (primed) variables, whose disjointness is expressed by $A \perp' B$. Formally,

THEOREM 7. CONJOINPROGRESSDISJOINT *For all systems A and B such that $A \perp' B$, and for all state formulas I_A and I_B,*

$$\begin{array}{ll} & \mathsf{SysProgress}\, I_A\ A\ \wedge\ \mathsf{SysProgress}\, I_B\ B \\ \vdash & \mathsf{SysProgress}\, \big(I_A \wedge I_B\big)\ \big(A \otimes B\big) \end{array}$$

From this theorem and variable disjointness, Box satisfies progress for the dynamics with independent $\boldsymbol{a_x}$ and $\boldsymbol{a_y}$.

Now suppose that instead of rectangular dynamics with independent control inputs $\boldsymbol{a_x}$ and $\boldsymbol{a_y}$, we want a controller for polar coordinates with independent control inputs $\boldsymbol{a}$ and $\boldsymbol{\theta}$. For example, these are the dynamics of a 2D version of the quadcopter (with $\boldsymbol{\phi}$ fixed at 0), in the absence of the small-angle constraint. While the transformation to polar coordinates seems to couple the x and z instantiations of Int, there is always an invertible map from rectangular to polar coordinates. This connection between polar and rectangular coordinates allows us to use the substitution theorems from Section 5.1 to prove both preservation and progress for a version of Box that operates using polar thrust.

The takeaway is that although the system $(\{\sigma_\theta\}\mathsf{Box})$ is not superficially composed of disjoint subsystems, we can still verify both progress and preservation fully modularly.

Example: Incorporating the Small-angle Constraint. The previous example ignored the small-angle constraint,

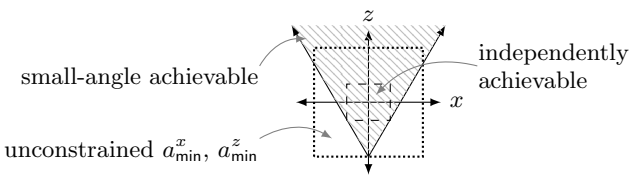

Figure 4: Decoupling of $\boldsymbol{a_x}$ and $\boldsymbol{a_z}$.

which is critical to the accuracy of our model of the quadcopter's dynamics. In this section we address this shortcoming and complete the verification of Box with respect to a 2D version of $\mathcal{W}_{QC}$ (again, with $\boldsymbol{\phi}$ fixed at 0) *in a fully modular way*. Figure 4 shows how the small-angle constraint introduces coupling into the independent box. In particular, the hatched region corresponds to the accelerations achievable under the small-angle constraint while the unconstrained region corresponds to the accelerations necessary for Box developed in the previous section.

Our approach relies on a detail of the verification which we glossed over in the previous presentation. In particular, because the second-derivative controller is *parameterized by* a_{min}, Int is parameterized by an a_{min} and Box is parameterized by a pair of a_{min}'s in the x and z dimensions. The right side of Figure 4 shows how we leverage the parameterization of Box to logically decouple the two dependent dimensions. Our insight is the following. If we can pick values for a^x_{min} and a^z_{min} (note that a^x_{min} and a^z_{min} are negative) such that all accelerations a_x and a_z where $|a_x| \leq -a^x_{\mathsf{min}}$ and $|a_z| \leq -a^z_{\mathsf{min}}$ are achievable by $\boldsymbol{T}$ and $\boldsymbol{\theta}$ under the small-angle constraint then the values of $\boldsymbol{a_x}$ and $\boldsymbol{a_z}$ can be chosen independently *if they fall within the bounds*. Purely trigonometric reasoning reveals that the constraint on $\boldsymbol{\theta}$ is achieved for any values of $\boldsymbol{a_x}$ and $\boldsymbol{a_z}$ satisfying $\mathcal{C}_x \wedge \mathcal{C}_z$ defined as:

$$\begin{array}{rcl} \mathcal{C}_x & \triangleq & -(a_{\mathsf{min}} + g)\tan 30^\circ \leq \boldsymbol{a_x} \leq (a_{\mathsf{min}} + g)\tan 30^\circ \\ \mathcal{C}_z & \triangleq & a_{\mathsf{min}} \leq \boldsymbol{a_z} \leq -a_{\mathsf{min}} \end{array}$$

We can formalize this reasoning by noting that the previous example (Box), instantiated with the above choices of a^x_{min} and a^z_{min}, satisfies progress under the constraint $\mathcal{C}_x \wedge \mathcal{C}_z$. Thus, we can use the substitution proof rule SUBST-PROGRESS with a rectangular-to-polar substitution, instantiating R with $(\mathcal{C}_x \wedge \mathcal{C}_z)$ and Q with the small-angle condition (ignoring $\boldsymbol{\phi}$ since it is assumed to be 0 in this 2D example).

The important point of this verification is that the separation of preservation and progress allowed for a relatively small and modular proof of a complex property of a *highly coupled system*. Without the separation of preservation and progress, the coupling would have forced us to re-prove many of the intermediate properties.

Example: Adding the Third Dimension. We can build a controller enforcing a cube by conjoining Box with another instance of Int substituted to reflect the y dimension. We can then apply this cube to the 3D version of $\mathcal{W}_{QC}$ (including $\boldsymbol{\phi}$ and the coupling small-angle constraint). Again, the modularity of our proofs shields us from much of the complexity. Box enforced the constraints for $\boldsymbol{x}$ and $\boldsymbol{z}$ using $\boldsymbol{a_x}$ and $\boldsymbol{a_z}$. Extending this to handle the third dimension simply requires that we use substitution to view $\boldsymbol{a_x}$ and $\boldsymbol{a_z}$ as a single unit and carry out the same reasoning, independently controlling that composed unit and $\boldsymbol{a_y}$.

id: Name	Built How?	Symbols	Proof
a: 1D vel bound	From Scratch	43	130
b: 1D pos bound	From Scratch	126	484
c: 1D interval	$(\{\}b)^2 \otimes (\{\}a)^2$	323	194
d: 2D square	$\{\}(\{\}c)^2$	805	258
e: 3D cube	$\{\}(d \otimes \{\}c)$	1353	201
f: 3D sqr donut	$\{\}e \oplus ... \oplus \{\}e$	5412	23
g: 3D +, T, $\perp$	$\{\}e \oplus ... \oplus \{\}e$	5412	23
h: 3D pilot box	$\{\}e \oplus ... \oplus \{\}e$	5412	23

Figure 5: Systems implemented and proved correct. The first column is the name of the system. The second column shows how each system was built and proved correct from smaller components. $\{\}i$ indicates a substitution applied to system i (we have omitted the specific substitution); $\otimes$ represents conjunction composition; and $\oplus$ represents disjunction composition. The third and fourth columns show the number of symbols in the discrete controller and the number of lines of proofs, respectively.

6. EVALUATION

Proof Effort. Figure 5 lists all of the geo-fencing controllers that we verified and flew, along with the composition mechanism, proof size, and number of symbols in the discrete transition for each one. The first 2 rows list controllers that we ported from prior work; these are our atomic building blocks. The next 3 rows list controllers that we built and verified using a combination of substitution and conjunction. The remaining rows list controllers that we built using disjunction and substitution.

Note that our 1D position controller (b in Figure 5) involves 3 variables and 126 symbols and is a relatively simple discrete transition requiring only multiplication and addition. However, verifying this position controller required substantial, tedious, manual proof effort because of the difficulty of reasoning about nonlinear real arithmetic. On the other hand, our 3D cube controller (e in Figure 5) contains 9 variables, 1353 symbols, trigonometric functions, and angular constraints. This increase in complexity is actually quite daunting for two reasons: (1) arithmetic decision procedures have very bad worst case complexity, and (2) trigonometric functions make the problem undecidable [14]. Despite the massive increase in complexity, our modular verification approach allowed us to verify the 3D version with only a modest amount of additional manual proof effort.

Moreover, note that all controllers built using disjunction require negligible proof effort on top of the effort required to verify the individual components. This is because disjunction composes proofs of both preservation and progress.

Finally, it is worth noting that we have several admits in our Coq development: (1) basic arithmetic theorems, (2) some theorems bridging the gap between $\mathcal{W}_{QC}$ and its abstractions, (3) theorems stating progress of the continuous transitions in the quadcopter model. Crucially, these admits do not interfere with the core ideas that we are exploring, namely modular reasoning about sampled-data systems.

Expressiveness. Disjunction is extremely powerful when building real-world controllers. For example, we can build up pixelated versions of arbitrary shapes by composing instances of our cube controller. Interesting properties that

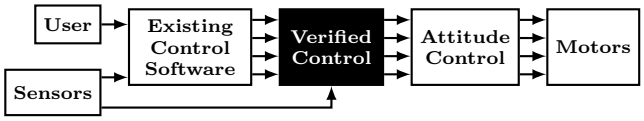

Figure 6: System architecture.

we can build with this include: (1) avoid no-fly zones such as airports, the white house, etc, (2) do not fly within a box surrounding the pilot, (3) stay away from static barriers such as trees and buildings, and (4) avoid skyscrapers in a city.

Conjunctive composition is similarly expressive – the box and the cube both make heavy use of conjunctive composition. Finally, substitution allows us to reuse all controllers by translating and rotating them, and by transforming them into controllers for different physical dynamics.

Behavior in Actual Flight. To make sure that our controllers work well in practice, we manually translated our models to C and ran them on a 3DR Iris+ quadcopter. Figure 6 depicts the architecture of the system with one of our controllers inserted. Our controllers check the outputs of the pre-existing control software, potentially replacing them with default safe values; this resolves the non-determinism of the controller specifications. This architecture means that the behavior of much of the existing control software (left of "Verified Control") cannot cause a violation of the fence. Moreover, if the quadcopter remains sufficiently far from the fence boundaries, the system behaves exactly as it would without our controllers, so any performance properties of the existing control software are preserved.

We flew our quadcopter with all controllers listed in Figure 5. All of the controllers enforced their respective bounds with only minor violations, which can be attributed to unmodeled forces such as wind and to modeling approximations such as instantaneous attitude changes. This final approximation has previously been justified empirically under the small-angle constraint [13].

Comparison with fully automated tools. There are a number of state-of-the-art tools that attempt to automatically verify hybrid systems [11, 8, 17, 16], but due to the complexity of the domain, they are limited to certain classes of systems and properties.

PHAVer [11], which we ran through the SpaceEx tool platform [29], is able to verify only one of our systems, namely the combined upper and lower bound on velocity. However, PHAVer is not able to verify the upper bound on velocity in isolation, probably because there are fewer constraints on acceleration to limit the search space, compared to the controller that bounds velocity from above and below. Finally, PHAVer is not able to run any of our other systems because the discrete transitions involve non-linear arithmetic; analyzing these systems would require manual construction of a linear overapproximation of the discrete transitions.

dReach [17] and Flow* [8] are *bounded* model checkers, which means that they can conclude safety of a system within a user-specified time bound. Our Coq proofs, on the other hand, guarantee safety for infinite runs. It is possible to use dReach and Flow* to guarantee safety for all runs by manually providing the inductive invariant. In this way, our results are complementary, as they provide a decomposition technique to manage scalability issues of these tools. However, these tools are currently unable to handle universally quantified variables with unbounded domains. This prevents them from verifying safety of systems with symbolic parameters, such as a_{min} – they require concrete numerical bounds on these parameters, which weakens the safety theorem.

7. RELATED WORK

Hybrid Automata. Hybrid automata [15, 23] extend traditional automata with continuous transitions. The primary reasoning method for hybrid automata is model checking [11, 16]; however, the complexity of the domain restricts these tools to certain classes of systems and properties. Alur *et al.* [4] present rules for conjoining hybrid automata with syntactically independent interface variables and for renaming variables. Our substitution rules generalize substitution in [4] to allow substituting *expressions* for variables, requiring us to separately justify progress through invertibility of substitution. Also, different modules often need to output to the same actuators, and our work pushes beyond the restriction of syntactically independent variables. Finally, unlike [4], we present rules for disjunctive composition.

Architectures for CPS. Several architectures have been developed to guarantee safety properties of otherwise untrusted controllers [22, 27]. The protectors framework [22] focuses only on conjunctive composition but not on disjunctive composition, substitution, and the combination of all three. Switching control [20] focuses on disjunctive composition. Our inclusion of invariants in the discrete controller corresponds to expressing the switching boundary. However, the focus of switching control theory is optimality, stability, and transitionability, whereas we focus on complementary properties like bounding the state space.

Logics for Hybrid Systems. Hybrid systems have been formalized in proof assistants, including the HHL prover [21] and using $\mathsf{d}\mathcal{L}$ in KeYmaera X [25]. The $\mathsf{d}\mathcal{L}$ logic has compositional proof rules; however, KeYmaera X is not as expressive as Coq, and some of our proof rules are not expressible in KeYmaera X, namely those with side conditions (*e.g.* invertible substitutions, disjointness of primed variables). The only way to add these rules to KeYmaera X would be to extend its core with soundness-critical checking of side conditions or to employ potentially slow and brittle tactics to prove soundness of composition on a case-by-case basis. By working in Coq, we are able to get higher-order modular *theorems* without compromising soundness.

Hybrid systems have been formalized in general-purpose proof assistants [30, 3, 24, 12, 9, 5, 26, 6]. [3] presents deductive reasoning rules and parallel composition, but does not address progress or substitution and disjunction. Similarly, [6] provides a rule for decomposing global safety proofs into local invariant proofs, but does not address the interplay between composition and progress. Finally, [26] verified M_{∂^2} and M_{∂} but did *not* compose them or address progress.

Temporal Logic. There has been a lot of work on composition in temporal logic, most notably by Abadi and Lamport [2, 1]. Their work describes how to reason about the conjunction of LTL specifications, but they do not deal with the interplay between conjunction and progress or substitution and progress. Other work includes techniques for decomposing LTL verification into a search for suitable barrier certificates [31] and deductive rules for synthesizing con-

trollers satisfying ATL* properties [10]. While these approaches apply to more temporal logic formulas than ours, they do not address verification using conjunction, disjunction, and substitution. There has also been work on synthesis using approximate bisimulations [28]. We focus on the complementary task of composing and reusing controllers.

Supervisory Control. Our theory is closely related to the work on modular construction of nonblocking supervisory controllers in discrete-event systems [32]. However, our models explicitly include differential equations rather than requiring a discrete abstraction and do not require a notion of termination in a marked state. Moreover, to the best of our knowledge, none of this work makes the inductive invariant explicit and separates preservation from progress.

8. CONCLUSION

We presented a modular technique for verification of sampled-data systems implemented in the Coq proof assistant. The crucial insight of our technique is the separation of preservation and progress. We demonstrated the utility of this approach by modularly building verified state-constraining controllers for a quadcopter using three simple but powerful operators: substitution, disjunction, and conjunction.

In the future, we are interested in verifying our controllers composed with an attitude controller; translation to C code; controller robustness; and other sampling policies.

Acknowledgments. This research was supported in part by the National Science Foundation through grant 1544757.

9. REFERENCES

[1] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Trans. Program. Lang. Syst.*, 16(5):1543–1571, Sept. 1994.

[2] M. Abadi and L. Lamport. Conjoining specifications. *ACM Trans. Program. Lang. Syst.*, 17(3):507–535, May 1995.

[3] E. Abraham-Mumm, U. Hannemann, and M. Steffen. Verification of hybrid systems: formalization and proof rules in PVS. In *ECCS '01*, pages 48–57, 2001.

[4] R. Alur and T. A. Henzinger. Modularity for timed and hybrid systems. In *CONCUR '97*, volume 1243, pages 74–88. Springer Berlin Heidelberg, 1997.

[5] A. Anand and R. Knepper. ROSCoq: Robots powered by constructive reals. *ITP'15*, 15:2015, 2015.

[6] N. Arechiga, J. Kapinski, J. V. Deshmukh, A. Platzer, and B. H. Krogh. Forward invariant cuts to simplify proofs of safety. In A. Girault and N. Guan, editors, *EMSOFT*, pages 227–236. IEEE, 2015.

[7] T. Chen and B. A. Francis. *Optimal Sampled-Data Control Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1995.

[8] X. Chen, S. Sankaranarayanan, and E. Abraham. Flow* 1.2: More effective to play with hybrid systems. In *ARCH14-15*, volume 34 of *EPiC Series in Computing*, pages 152–159, 2015.

[9] P. Collins, M. Niqui, and N. Revol. A Taylor Function Calculus for Hybrid System Analysis: Validation in Coq (Extended Abstract), 2010.

[10] R. Dimitrova and R. Majumdar. Deductive control synthesis for alternating-time logics. EMSOFT '14, pages 14:1–14:10, New York, NY, USA, 2014. ACM.

[11] G. Frehse. PHAVer: algorithmic verification of hybrid systems past HyTech. *STTT*, 10(3):263–279, 2008.

[12] H. Geuvers, A. Koprowski, D. Synek, and E. van der Weegen. Automated machine-checked hybrid system safety proofs. In *ITP '10*, pages 259–274, 2010.

[13] J. H. Gillula, G. M. Hoffmann, H. Haomiao, M. P. Vitus, and C. J. Tomlin. Applications of hybrid reachability analysis to robotic aerial vehicles. *The International Journal of Robotics Research*, 30(3):335–354, 2011.

[14] J. Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[15] T. A. Henzinger. The theory of hybrid automata. In *LICS '96*, pages 278–292, 1996.

[16] T. A. Henzinger, P. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. In *CAV '97*, 1997.

[17] S. Kong, S. Gao, W. Chen, and E. Clarke. dReach: δ-Reachability Analysis for Hybrid Systems. pages 200–205. Springer Berlin Heidelberg, 2015.

[18] L. Lamport. The temporal logic of actions. *TOPLAS*, 16(3):872–923, 1994.

[19] L. Lamport. Real time is really simple. Technical report, MSR-TR-2005-30, Microsoft Research, 2005.

[20] D. Liberzon. *Switching in systems and control*. Springer Science & Business Media, 2012.

[21] J. Liu, J. Lv, Z. Quan, N. Zhan, H. Zhao, C. Zhou, and L. Zou. A Calculus for Hybrid CSP. APLAS'10, pages 1–15, Berlin, Heidelberg, 2010. Springer-Verlag.

[22] C. Livadas and N. A. Lynch. Formal verification of safety-critical hybrid systems. In *HSCC '98*, 1998.

[23] N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O Automata. *Inf. Comput.*, 185(1):105–157, Aug. 2003.

[24] M. Niqui and O. Tveretina. Modular Development of Hybrid Systems for Verification in Coq. In *HSCC '08*, HSCC '08, pages 638–641. Springer-Verlag, 2008.

[25] A. Platzer. KeYmaera X: A Hybrid Systems Theorem Prover. http://www.ls.cs.cmu.edu/KeYmaeraX/. Accessed: 2015-04-28.

[26] Ricketts, D., Malecha, G., Alvarez, M., Gowda, V., Lerner, S. Towards Verification of Hybrid Systems in a Foundational Proof Assistant. In *MEMOCODE*, 2015.

[27] L. Sha, R. Rajkumar, and M. Gagliardi. Evolving dependable real-time systems. In *Aerospace Applications Conference*, pages 335–346. IEEE, 1996.

[28] P. Tabuada. An approximate simulation approach to symbolic control. *IEEE Transactions on Automatic Control*, 53(6):1406–1418, July 2008.

[29] G. F. *et al.* SpaceEx: Scalable Verification of Hybrid Systems. CAV'11. Springer-Verlag, 2011.

[30] N. VÃűlker. Towards a HOL Framework for the Deductive Analysis of Hybrid Control Systems, 2000.

[31] T. Wongpiromsarn, U. Topcu, and A. Lamperski. Automata theory meets barrier certificates: Temporal logic verification of nonlinear systems. *IEEE Transactions on Automatic Control*, PP(99):1–1, 2015.

[32] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals and Systems*, 1988.