In many instances, programs are concerned only with processing or manipulating data and displaying them to a user, who becomes the agent that ends up taking physical action. However, in some instances, we create software to control other analog devices or machin-

tems. However, a hybrid system contains the interface between these two domains and requires new logics.

F6 – – 6

To guarantee that the control algorithm predictably influences the machinery with which it interacts, we need to develop hybrid logics that are tailored to include both models of discrete programs as well as the continuous equations that govern the analog components.

Engineers can write code to model a hybrid system and then run these models with particular inputs to see how the system behaves. However, at best, this approach affords only the equivalent of evaluating individual test cases, which cannot guarantee the exploration of important corner cases, nor always eliminate unexpected behavior.

article, even though more advanced controls turn out to be necessary.[7]

To create a model of a cyber-physical system by using d$\mathcal{L}$, we write a HP in d$\mathcal{L}$ modeling language. The language in which we write HPs to model cyber-physical systems is not designed to be executed but to be formally reasoned about; its syntactic constructs are simpler than what we find in executable languages today and are shown in Table 1.

The HP language used in d$\mathcal{L}$ contains arithmetic operators, assignments, and sequential composition of statements and assertions—operations that are found in many imperative languages. State variables in the program can have types that are discrete, finite sets, or R, the reals.

To write a conditional statement in this language, we

the statements describe discrete computations and state changes inside a computer and have no real time associated with them. It is as if when discrete statements are encountered in sequence, the continuous dynamics of the system are "frozen," and when the system encounters a dynamics statement, discrete computation ceases and the continuous dynamics evolve.

Despite the apparent stop-and-go nature of the dynamics in this model, it is effective at modeling a system that is doing discrete computations as the continuous dynamics evolve. In the real world, discrete computations and continuous dynamics operate concurrently, but the HP forces us to linearize them, putting them into a sequential order. We conceptually associate the period of time in which the physics evolved to be concurrent with the discrete computation that preceded it in the program. Within the HP, the discrete program and the continuous physics can share state at the edges of these linearized transitions, as shown in Fig. 2.

ment that ϕ holds for all values of x). We also can write modal operators (e.g., [α]ϕ means that after running the HP α, the predicate ϕ always holds). The modal operators are very powerful because they allow a sort of quantification over different executions of a given program, and they are at the core of how we describe properties of hybrid models of cyber-physical systems.

We write the safety property we desire to prove for our algorithm as

$$(p >= 0) \rightarrow [\text{ctrl}](p >= 0). \quad (6)$$

This predicate is called our goal, or the theorem that we wish to prove. It says: if the tool starts in a safe place (i.e., $p >= 0$) and you run the control program, then the tool will remain in a safe place at every instant of time, no matter what input you provide for it.

To do the proof, we apply sound inference rules to our goal, decomposing it into simpler goals and eventually statements in real arithmetic. This real arithmetic can be solved by a computer program called a decision procedure. This proving process is the heart of d$\mathcal{L}$; the logic is constructed so that completing such proofs translates into a guarantee about the system's behavior, for all possible system inputs.

The proof will be structured the same way as the model, first decomposing the outer loop into three different subgoals: the base case, the inductive step, and the postcondition. Within the proof of the inductive step, there are three cases, one for each possible nondeterministic choice represented by the dynamics statements. The differential equation must be solved for each one, and the proof about the inductive step can be completed.

KeYmaera[8] is a tool that takes a HP and allows an engineer to create a machine-checked proof by using d$\mathcal{L}$. This is called a mechanization of the logic. This system is simple enough that once the HP model and safety property are entered, a loop invariant can be provided, and safety for this simplified 1-D system can be proved automatically at the press of a button. A loop invariant is a logical statement that describes an important, unchanging attribute that will hold at the beginning and end of a loop; it is necessary to resolve the behavior of complicated loops automatically.

The simplified model represents input as a constant during a time step and does not accurately represent a lag in the system. It assumes that the program that implements the controller runs all the time. It also fixes the position of the virtual boundary and provides control in only one dimension. Although these simplifying assumptions were a useful starting point, we need to replace them with more realistic assumptions to ensure that our conclusions are true. This section refines the previous model to relax these assumptions and create a more realistic controller in two dimensions.

The first enhancement to the model will be a more accurate representation of user input. The previous model represented user input for each "step" in the system as a constant, by setting the force components to a nondeterministic value by writing $f_x := *$; $f_y := *$. A more accurate model of the force input to the system would be to create a piecewise linear representation of the force. To do this, we assign nondeterministic values to some state variables, $f_{xp} := *$; $f_{yp} := *$, and then logically associate these with the derivative of the force by requiring $f_x' = f_{xp}$ and $f_y' = f_{yp}$ in the set of differential equations that we specify during continuous evolution. When we make this change, it ensures that at each step, we have a constant acceleration. This makes our velocity piecewise linear and our path quadratic, given the simple relationship between these state variables. Consequently, there are many additional possible types of force curves that can be exerted on the system during a time step, depending on the acceleration and the initial direction of the force during the step. We can distinguish between these different types of curves and design our controller to recognize and behave differently in each situation. The different movement scenarios are shown in Fig. 3. Each subfigure represents a movement scenario in which the SBS robot must enforce safety. Each case must start above the x axis because the system starts in a safe loca

rithm. In d$\mathcal{L}$, this is a relatively simple matter: we add a state variable, t, to the program to hold the value of the clock at any instant of time; before each dynamics statement, we reset our time counter $t := 0$; within each dynamics statement, we require that $t' = 1$ (i.e., time progresses continuously at a rate of 1); and we put a constraint on the evolution in each dynamics statement to ensure that a time step remains below a certain threshold, $t \leq \epsilon$, to represent a step in the control algorithm. In this case, ϵ represents the largest time stop the controller can take before responding to its environment.

This is a common idiom for representing time in a d$\mathcal{L}$ HP. The left program is the original, containing discrete

tive factor less than 1), but instead the virtual boundary will feel somewhat slippery because of the selective removal of the movement component in the direction of the boundary.

We implemented the realistic model of our controller and proved its safety using KeYmaera. We can trust KeYmaera because it faithfully

tance controller, which should not change regardless of the mode the system is in or the damping decision made by the control algorithm.

A more realistic model of our control algorithm that includes these enhancements is shown in Table 3. This algorithm enforces a single virtual boundary based on the subtractive control law given in Eq. 2. The subtractive control law works by modifying the overall velocity by subtracting the part of the vector out of the movement that is normal to the virtual boundary. This boundary should feel abrupt; it is encountered with no warning, so the surgeon needs another way of visualizing how close to it he or she is. When pressed against it, the tool will tend to bounce back slightly, giving the boundary a bouncy feel. This comes from a combination of the subtractive control law and the system's delay.

The user should not experience any of the stickiness we expect from the control algorithms produced by multiplicative damping strategies (i.e., those strategies that modify the overall velocity by applying a multiplica-

the boundary. This work is detailed in Ref. 7 and was proved for arbitrarily many boundaries.

[illegible]

We have powerful tools available today to reason about cyber-physical systems. With them, we can create an accurate model of a cyber-physical system component and make guarantees about the system's behavior under all possible input conditions.

These tools are new, and it is sometimes difficult to apply them to larger, more complicated system components because the proof becomes commensurately more complex.

Along with investigating many other advances in proof automation in KeYmaera, we are in the process of exploring how to effectively scale these capabilities to larger systems and how to compose proofs about small system components together to make guarantees about larger subsets of the system.

REFERENCES

[1]Fränzle, M., and Herde, C., "HySAT: An Efficient Proof Engine for Bounded Model Checking of Hybrid Systems," *Formal Methods in System Design* 30(2), 179–198 (2007).

[2]Platzer, A., "Differential-Algebraic Dynamic Logic for Differential-Algebraic Programs," *J. Log. Comput.* 20(1), 309–352 (2010).

[3]Platzer, A., "A Complete Axiomatization of Quantified Differential Dynamic Logic for Distributed Hybrid Systems," *Logical Methods in Computer Science* 8(4), 1–44 (2012).

[4]Platzer, A., "Differential Dynamic Logic for Hybrid Systems," *J. Autom. Reasoning* 41(2), 143–189 (2008).

[5]Xia, T., Baird, C., Jallo, G., Hayes, K., Nakajima, N., et al. "An Integrated System for Planning, Navigation and Robotic Assistance for Skull Base Surgery," *Int. J. of Med. Robot.* 4(4), 321–330 (2008).

[6]Kazanzides, P., "Virtual Fixture Computation. Note on Combining the Effects of Multiple Virtual Fixtures" (2011). [Please contact author to obtain a copy of document.]

[7]Kouskoulas, Y., Renshaw, D., Platzer, A., and Kazanzides, P., "Certifying the Safe Design of a Virtual Fixture Control Algorithm for a Surgical Robot," in *Proc. 16th International Conf. on Hybrid Systems: Computation and Control*, pp. 263–272 (2013).

[8]Platzer, A., and Quesel, J.-D., "KeYmaera: A Hybrid Theorem Prover for Hybrid Systems," in *Automated Reasoning: 4th Int. Joint Conf.*, Vol. 5195 of LNCS, A. Armando, P. Baumgartner, and G. Dowek (eds.), SpringerVerlag, Berlin Heidelberg, pp. 171–178 (2008).

Yanni Kouskoulas is a group chief scientist [illegible]