A Vision of Collaborative Verification-Driven Engineering of Hybrid Systems

Stefan Mitsch [1] and **Grant Olney Passmore** [2] and **André Platzer** [3]

Abstract. Hybrid systems with both discrete and continuous dynamics are an important model for real-world physical systems. The key challenge is how to ensure their correct functioning w.r.t. safety requirements. Promising techniques to ensure safety seem to be model-driven engineering to develop hybrid systems in a well-defined and traceable manner, and formal verification to prove their correctness. Their combination forms the vision of verification-driven engineering. Despite the remarkable progress in automating formal verification of hybrid systems, the construction of proofs of complex systems often requires significant human guidance, since hybrid systems verification tools solve undecidable problems. It is thus not uncommon for verification teams to consist of many players with diverse expertise. This paper introduces a verification-driven engineering toolset that extends our previous work on hybrid and arithmetic verification with tools for (i) modeling hybrid systems, (ii) exchanging and comparing models and proofs, and (iii) managing verification tasks. This toolset makes it easier to tackle large-scale verification tasks.

1 Introduction

Motivation Computers that control physical processes, thus forming so-called *cyber-physical systems* (CPS), are today pervasively embedded into our lives. For example, cars equipped with adaptive cruise control form a typical CPS, responsible for controlling acceleration on the basis of distance sensors. Further prominent examples can be found in many safety-critical areas, such as in factory automation, medical equipment, automotive, aviation, and railway industries. From an engineering viewpoint, CPSs can be described in a *hybrid* manner in terms of discrete control decisions (the cyber-part, e. g., setting the acceleration of a car) and in terms of differential equations modeling the entailed physical continuous dynamics (the physical part, e. g., motion) [28]. More advanced models include aspects of distributed hybrid systems [32] or stochasticity [31], but are not addressed in this paper.

Challenge The key challenge in engineering hybrid systems is the question of how to ensure their correct functioning in order to avoid incorrect control decisions w.r.t. safety requirements (e. g., a car with adaptive cruise control will never collide with a car driving ahead). Especially promising techniques to ensure safety seem to be model-driven engineering (MDE) to incrementally develop systems in a well-defined and traceable manner and formal verification to mathematically prove their correctness, together forming the vision of *verification-driven engineering* (VDE) [20]. Despite the remarkable progress in automating formal verification of hybrid systems, still many interesting and complex verification problems remain that are hard to solve in practice with a single tool by a single person.

Because hybrid systems are undecidable, hybrid systems verification tools work over an undecidable theory, and so verifying complicated systems within them often requires significant human guidance. This need for human guidance is true even for *decidable* theories utilized within hybrid systems verification [5], such as the first-order theory of nonlinear real arithmetic (also called the theory of *real closed fields* or **RCF**), a crucial component of real-world verification efforts. Though decidable, **RCF** is fundamentally infeasible (it is worst-case doubly exponential in the number of variables [6]), which poses a problem for the automated verification of hybrid systems. Much expertise is often needed to discharge arithmetical verification conditions in a reasonable amount of time and space, expertise requiring the use of deep results in real algebraic geometry. It is thus not uncommon for serious hybrid systems verification teams to consist of many players, some with expertise in control theory and dynamical systems, some in software engineering, some in mathematical logic, some in real algebraic geometry, and so on. Hence, well-established project management techniques to coordinate team members are crucial to achieve effective collaborative large-scale verification of hybrid systems. Successful examples of team-based large-scale verification of non-hybrid systems include the operating system kernel seL4 [19] in Isabelle/HOL and the Flyspeck project [15], and show, that indeed collaboration is key for proving large systems.

Vision This paper introduces a VDE toolset (including a backend deployment for project management and collaboration support) and sketches our vision on further enhancing this toolset. It applies proof decomposition in-the-large across multiple verification tools, basing on the completeness of differential dynamic logic ($\mathsf{d}\mathcal{L}$ [28, 33]), which is a real-valued first-order dynamic logic for hybrid programs, a program notation for hybrid systems. The VDE toolset $\mathsf{S}\varphi\mathsf{n}\mathsf{x}$ extends our previous work on the deductive verification tool KeYmaera [36] and on the nonlinear real arithmetic verification tools RAHD [26] and MetiTarski [27] with modeling tools for (i) modeling of hybrid systems in $\mathsf{d}\mathcal{L}$, (ii) exchanging and com-

[1]Carnegie Mellon University, Computer Science Department, 5000 Forbes Avenue, Pittsburgh, PA 15213, email: smitsch@cs.cmu.edu
[2]LFCS, Edinburgh and Clare Hall, Cambridge, 10 Crichton Street, Edinburgh, UK, email: grant.passmore@cl.cam.ac.uk [3]Carnegie Mellon University, Computer Science Department, 5000 Forbes Avenue, Pittsburgh, PA 15213, email: aplatzer@cs.cmu.edu

paring models and proofs via a central source repository, and (iii) exchanging knowledge and tasks through a project management backend.

Structure of the paper In the next section, we give an overview on related work. In Sect. 3 we introduce our architecture of a verification-driven engineering toolset, and describe implementation and features of its components, including a vision of further work. Section 4 introduces an autonomous robotic ground vehicle as application example. Finally, in Sect. 5 we conclude the paper with an outlook on real-world application of the toolset.

2 Related Work

Model-driven engineering in a collaborative manner has been successfully applied in the embedded systems community. Efforts, for instance, include transforming between different UML models and SysML [16], modeling in SysML and transforming these models to the simulation tool Orchestra [2], integration of modeling and simulation in Cosmic/Cadena [13], or modeling of reactive systems and integration of various verification tools in Syspect [10].

Recent surveys on verification methods for hybrid systems [1], modeling and analysis of hybrid systems [8], and modeling of cyber-physical systems [9], reveal that indeed many tools are available for modeling and analyzing hybrid systems, but in a rather isolated manner. Supporting collaboration on formal verification by distributing tasks among members of a verification team in a model-driven engineering approach has not yet been the focus. Although current verification tools for hybrid systems (e. g., PHAVer [11], SpaceEx [12]), as well as those for arithmetic (e. g., Z3 [7]) are accompanied by modeling editors of varying sophistication, they are not yet particularly well-prepared for collaboration either. Developments in collaborative verification of source code by multiple complementary static code checkers [4], modular model-checking (e. g., [22]), and extreme verification [17], however, indicate that this is indeed an interesting field. Most notably, usage of online collaboration tools in the Polymath project has led to an elementary proof of a special case of the density Hales-Jewett theorem [14].

3 The VDE Toolset Sφnx

In order to integrate different modeling and verification tools, the verification-driven engineering toolset Sφnx[1] proposed in this paper follows a model-driven architecture: metamodels for different modeling and proof languages form the basis for manipulating, persisting, and transforming models. The notion of a model here denotes an instance of a metamodel, i. e., it comprises models, proofs, and strategies. Following the definition of the OMG[2], a metamodel defines a language to formulate models: one example for a metamodel is the grammar of d$\mathcal{L}$, which, among others, defines language elements for non-deterministic choice, sequential composition, assignment, repetition, and differential equations. An example for a model is given in Sect. 4: it is a set of formulas, differential equations, and other d$\mathcal{L}$ language elements. It conforms to the grammar of d$\mathcal{L}$, and thus is an instance of the d$\mathcal{L}$ metamodel. Figure 1 gives an overview of the toolset architecture. As can be seen, the d$\mathcal{L}$, KeY, arithmetic, and arithmetic proof metamodels represent interfaces between tools and to the backend.

d$\mathcal{L}$ metamodel The hybrid modeling components (textual and graphical editors for d$\mathcal{L}$, as well as model comparison) manipulate models that conform to the d$\mathcal{L}$ metamodel. A transformation runtime transforms between models in d$\mathcal{L}$ and their textual form read by KeYmaera.

KeY proof metamodel The proof comparison component reads proofs that conform to the KeY proof metamodel. These proofs may either be closed ones (completed proofs, nothing else to be done) or partial proofs (to be continued). A transformation runtime transforms between proofs in KeY and their textual form as generated by KeYmaera.

Arithmetic metamodel Arithmetic editors (not yet implemented) manipulate arithmetic models. Again a transformation runtime transforms between models expressed in terms of the arithmetic metamodel and the corresponding textual input (e. g., SMT-LIB syntax [3]) as needed by arithmetic tools, such as RAHD, MetiTarski, or Z3.

Arithmetic proof metamodel Finally, the proof comparison component reads arithmetic proofs expressed in terms of the arithmetic proof metamodel, and transformed to and from the arithmetic tool's (textual) format by a transformation runtime.

Let us exemplify the toolset with a virtual walk-through a collaborative verification scenario. We begin with modeling a hybrid system using textual and graphical d$\mathcal{L}$ editors. As both operate on the same model, changes in either editor are reflected instantly in the other. The resulting model, which conforms to the d$\mathcal{L}$ metamodel, is transformed on-the-fly during editing by the transformation runtime to a textual input file, and loaded into KeYmaera. In KeYmaera, we apply various strategies for proving safety of our hybrid system model, but may get stuck at some difficult arithmetic problem. We mark the corresponding node in the partial proof and save it in KeYmaera's textual output format. The proof collaboration tool transforms the partial proof text file into a model of the partial proof. We persist the hybrid model and the model of the partial proof in the model and proof repository, respectively. Then we create a request for arithmetic verification (ticket) in the project management repository using the task planning component. The assignee of the ticket accesses the linked partial proof, and extracts an arithmetic verification model from the marked proof node. Then a transformation runtime creates the textual input for one of the arithmetic verification tools. In this tool, a proof for the ticket can be created, along with a proof strategy that documents the proof. Such a proof strategy is vital for replaying the proof later, and for detecting whether or not the arithmetic proof still applies if the initial model changed. Both proof and proof strategy, are imported into the proof collaboration tool and persisted to the corresponding repository. The ticket is closed, together with the node on the original proof (if the arithmetic proof is complete; otherwise, the progress made is reported back). We fetch the new proof model version from the repository and inspect it using the proof comparison component. Then we transform the proof model into its textual form, load KeYmaera and continue proving our hybrid system from where we left off, but now with one goal closed. In case the corresponding arithmetic prover is connected to

[1] `http://www.cis.jku.at/sphinx/` [2] `http://www.omg.org`

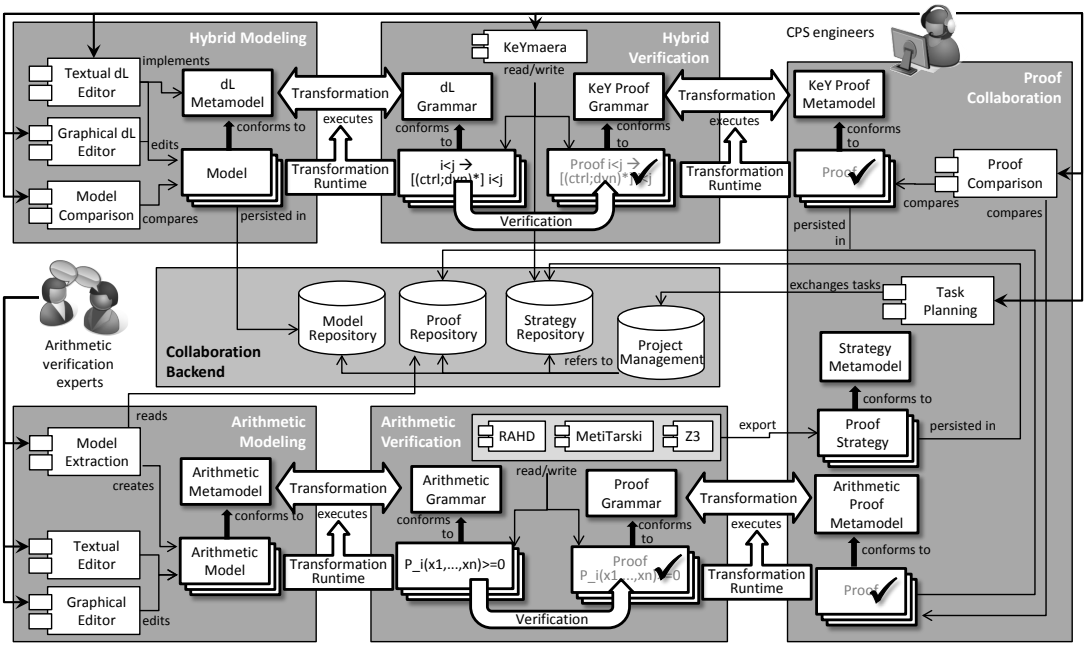

Figure 1: Overview of components in the verification-driven engineering toolset

KeYmaera, we could even load the proof strategy from the strategy repository and repeat it locally.

3.1 KeYmaera: Hybrid System Verification

KeYmaera[3] [36] is a verification tool for hybrid systems that combines deductive, real algebraic, and computer algebraic prover technologies. It is an automated and interactive theorem prover for a natural specification and verification logic for hybrid systems. KeYmaera supports differential dynamic logic (d$\mathcal{L}$) [28, 29, 30, 33], which is a real-valued first-order dynamic logic for hybrid programs, a program notation for hybrid systems. KeYmaera supports hybrid systems with nonlinear discrete jumps, nonlinear differential equations, differential-algebraic equations, differential inequalities, and systems with nondeterministic discrete or continuous input.

For automation, KeYmaera implements a number of automatic proof strategies that decompose hybrid systems symbolically in differential dynamic logic and prove the full system by proving properties of its parts [30]. This compositional verification principle helps scaling up verification, because KeYmaera verifies a big system by verifying properties of subsystems. Strong theoretical properties, including relative completeness results, have been shown about differential dynamic logic [28, 33] indicating how this composition principle can be successful.

KeYmaera implements fixedpoint procedures [34] that try to compute invariants of hybrid systems and differential invariants of their continuous dynamics, but may fail in practice. By completeness [33], this is the only part where KeYmaera's automation can fail in theory. In practice, however, also the decidable parts of dealing with arithmetic may become infeasible at some point, so that interaction with other tools or collaborative verification via Sφnx is crucial.

[3] `http://symbolaris.com/info/KeYmaera.html`

At the same time, it is an interesting challenge to scale to solve larger systems, which is possible according to completeness but highly nontrivial. For systems that are still out of reach for current automation techniques, the fact that completeness proofs are compositional can be exploited by interactively splitting parts of the hybrid systems proof off and investigating them separately within Sφnx. If, for instance, a proof node in arithmetic turns out to be infeasible within KeYmaera, this node could be verified using a different tool connected to Sφnx.

KeYmaera has been used successfully for verifying case studies from train control [37], car control [24], air traffic management [35], and robotic surgery [21]. These verification results illustrate how some systems can be verified automatically while others need more substantial user guidance. The KeYmaera approach is described in detail in a book [30].

In order to guide domain experts in modeling discrete and continuous dynamics of hybrid systems, the case studies, further examples, and their proofs are included in the KeYmaera distribution. When applying proof strategies manually by selection from the context menu in the interactive theorem prover, KeYmaera shows only the applicable ones sorted by expected utility. Preliminary collaboration features include marking and renaming of proof nodes, as well as extraction of proof branches as new subproblems. These collaboration features are used for interaction with the arithmetic verification tools and the collaboration backend described below.

3.2 Arithmetic Verification

Proofs about hybrid systems often require significant reasoning about multivariate polynomial inequalities, i.e., reasoning within the *theory of real closed fields* (**RCF**). Though **RCF** is decidable, it is fundamentally infeasible (hyper-exponential in the number of variables). It is not uncommon for hybrid system models to have tens or even hundreds of real variables,

and **RCF** reasoning is commonly the bottleneck for nontrivial verifications. Automatic **RCF** methods simply do not scale, and manual human expertise is often needed to discharge a proof's arithmetical subproblems.

RCF infeasibility is not just a problem for hybrid systems verification. Real polynomial constraints are pervasive throughout the sciences, and this has motivated a tremendous amount of work on the development of feasible proof techniques for various special classes of polynomial systems. In the context of hybrid systems verification, we wish to take advantage of these new techniques as soon as possible.

Given this fundamental infeasibility, how might one go about deciding large **RCF** conjectures? One approach is to develop a battery of efficient proof techniques for different practically useful fragments of the theory. For example, if an $\exists$ **RCF** formula can be equisatisfiably transformed into an $\wedge\vee$-combination of strict inequalities, then one can eliminate the need to consider any irrational real algebraic solutions when deciding the formula. Tools such as RAHD, Z3 and MetiTarski exemplify this heterogeneous approach to **RCF**, and moreover allow users to define *proof strategies* consisting of heuristic combinations of various specialised proof methods. When faced with a difficult new problem, one works to develop a proof strategy which can solve not only the problem at hand but also other problems sharing similar structure. Such strategies, though usually constructed by domain experts, can then be shared and utilised as automated techniques by the community at large.

3.3 Modeling and Proof Collaboration

In order to interconnect the variety of specialized verification procedures introduced above, Sφnx follows a model-driven engineering approach: it introduces metamodels for the included modeling and proof languages. These metamodels provide a clean basis for model creation, model comparison, and model transformation between the formats of different tools. This approach is feasible, since in principle many of those procedures operate over the theory **RCF**, or at least share a large portion of symbols and their semantics. One could even imagine that very same approach for exchanging proofs between different proof procedures, since proofs in RCF, in theory, can all be expressed in the same formal system. Currently, proofs in Sφnx are exchanged merely for the sake of being repeated in the original tool (although KeYmaera already utilizes many such tools and hence is able to repeat a wide variety of proofs).

In the case of textual languages, Sφnx uses the Eclipse Xtext[4] framework to obtain such metamodels directly from the language grammars (cf. Figure 2, obtained from the d$\mathcal{L}$ grammar [28]), together with other software artifacts, such as a parser, a model serializer, and a textual editor with syntax highlighting, code completion, and cross referencing.

These metamodels are the basis for creating models in d$\mathcal{L}$, as well as for defining transformations between d$\mathcal{L}$ and other modeling languages. The models in d$\mathcal{L}$ make use of mathematical terms, and are embedded in KeY files since KeYmaera uses the KeY [25] format for loading models and saving proofs. In the following sections, we introduce d$\mathcal{L}$ in more detail and describe the support for creating d$\mathcal{L}$ models and working on proofs in Sφnx.

[4] `www.eclipse.org/Xtext`

3.3.1 Differential Dynamic Logic

For specifying and verifying correctness statements about hybrid systems, we use *differential dynamic logic* d$\mathcal{L}$ [28, 30, 33], which supports *hybrid programs* as a program notation for hybrid systems. The syntax of hybrid programs is summarized together with an informal semantics in Table 1; it reflects the metamodel introduced in Figure 2. The sequential composition $\ll\alpha;\ \beta\gg$ expresses that β starts after α finishes (e. g., first let a car choose its acceleration, then drive with that acceleration). The non-deterministic choice $\ll\alpha \cup \beta\gg$ follows either α or β (e. g., let a car decide non-deterministically between accelerating and braking). The non-deterministic repetition operator $\ll\alpha^*\gg$ repeats α zero or more times (e. g., let a car choose a new acceleration arbitrarily often). Discrete assignment $\ll x := \theta\gg$ instantaneously assigns the value of the term θ to the variable x (e. g., let a car choose a particular acceleration), while $\ll x := *\gg$ assigns an arbitrary value to x (e. g., let a car choose any acceleration). $\ll x' = \theta\ \&\ F\gg$ describes a continuous evolution of x within the evolution domain F (e. g., let the velocity of a car change according to its acceleration, but always be greater than zero). The test $\ll?F\gg$ checks that a particular condition expressed by F holds, and aborts if it does not (e. g., test whether or not the distance to a car ahead is large enough). A typical pattern that involves assignment and tests, and which will be used subsequently, is to limit the assignment of arbitrary values to known bounds (e. g., limit an arbitrarily chosen acceleration to the physical limits of a car, as in $x := *; ?x \geq 0$). The deterministic choice $\ll$if(F) then α else $\beta\gg$ executes α if F holds, and β otherwise (e. g., let a car accelerate only when it is safe; brake otherwise). Finally, $\ll$while(F) do α elihw$\gg$ is a deterministic repetition that executes α as long as F holds.

To specify the desired correctness properties of the hybrid programs, differential dynamic logic (d$\mathcal{L}$) provides modal operators $[\alpha]$ and $\langle\alpha\rangle$, one for each hybrid program α. When ϕ is a d$\mathcal{L}$ formula (e.g., a simple arithmetic constraint) describing a safe state and α is a hybrid program, then the d$\mathcal{L}$ formula $[\alpha]\phi$ states that all states reachable by α satisfy ϕ. Dually, d$\mathcal{L}$ formula $\langle\alpha\rangle\phi$ expresses that there is a state reachable by the hybrid program α that satisfies d$\mathcal{L}$ formula ϕ. The set of d$\mathcal{L}$ formulas is generated by the following EBNF grammar (where $\sim \in \{<, \leq, =, \geq, >\}$ and θ_1, θ_2 are arithmetic expressions in $+, -, \cdot, /$ over the reals):

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

Thus, besides comparisons $(<, \leq, =, \geq, >)$, d$\mathcal{L}$ allows one to express negations $(\neg\phi)$, conjunctions $(\phi \wedge \psi)$, universal $(\forall x\phi)$ and existential quantification $(\exists x\phi)$, as well as the already mentioned state reachability expressions $([\alpha]\phi, \langle\alpha\rangle\phi)$.

3.3.2 Creating Models

For creating models of hybrid and cyber-physical systems, Sφnx currently includes d$\mathcal{L}$ as generic modeling language. The concrete textual syntax and d$\mathcal{L}$ editor created from the d$\mathcal{L}$ metamodel is shown in Figure 3, together with a concrete graphical syntax and the KeYmaera prover attached through the console. In order to facilitate creation of textual models in d$\mathcal{L}$, Sφnx includes templates of common model artifacts (e. g., ODEs of linear and circular motion). These templates, when

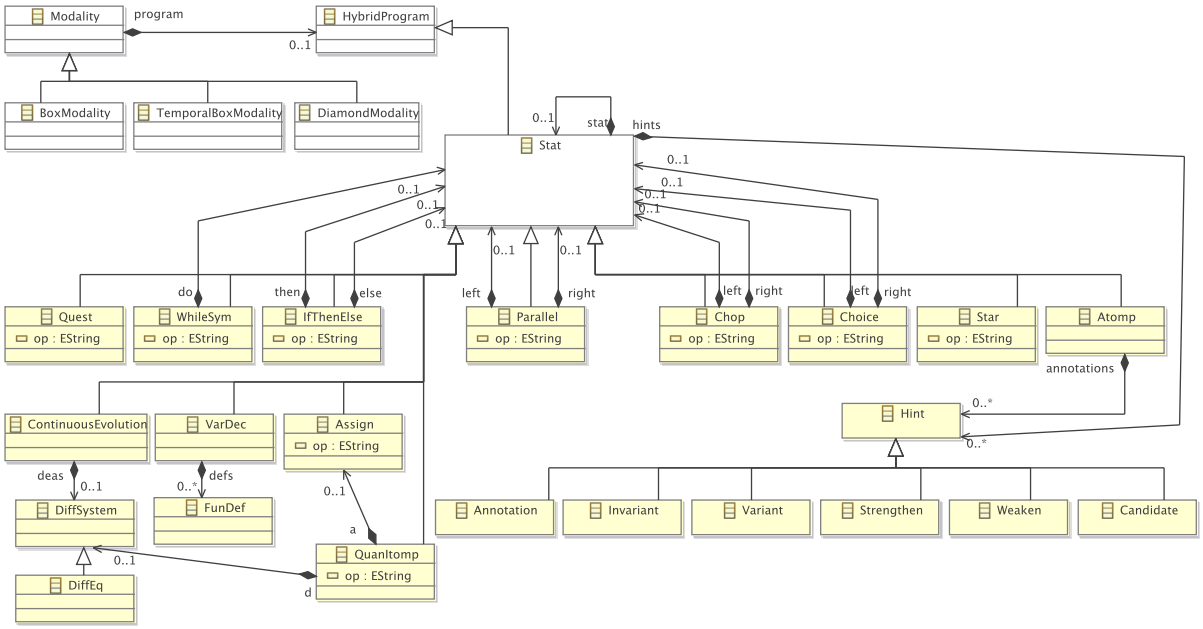

Figure 2: The dℒ metamodel extracted from the input grammar of KeYmaera

Table 1: Statements of hybrid programs

Statement	Metamodel element	Effect
$\alpha;\ \beta$	Chop	sequential composition, first performs α and then β afterwards
$\alpha\ \cup\ \beta$	Choice	nondeterministic choice, following either α or β
α^*	Star	nondeterministic repetition, repeating α $n \geq 0$ times
$x := \theta$	Assign (term)	discrete assignment of the value of term θ to variable x (jump)
$x := *$	Assign (wild card term)	nondeterministic assignment of an arbitrary real number to x
$\big(x_1' = \theta_1, \ldots,$	ContinuousEvolution	continuous evolution of x_i along differential equation system
$\quad x_n' = \theta_n\ \&\ F\big)$	DiffSystem	$x_i' = \theta_i$, restricted to maximum domain or invariant region F
$?F$	Quest	check if formula F holds at current state, abort otherwise
if(F) then α else β	IfThenElse	perform α if F holds, perform β otherwise
while(F) do α end	WhileSym	perform α as long as F holds
$[\alpha]\phi$	BoxModality	dℒ formula ϕ must hold after all executions of hybrid program α
$\langle\alpha\rangle\phi$	DiamondModality	dℒ formula ϕ must hold after at least one execution of hybrid program α

instantiated, allow in-place editing and automated renaming of the template constituents. As usual in the Eclipse platform, such templates can be easily extended and shared between team members.

Since generic modeling languages, such as dℒ for hybrid systems, tend to incur a steep learning curve, the Sφnx platform can be extended with dedicated domain-specific languages (DSL). Such DSLs should be designed to meet the vocabulary of a particular group of domain experts. They can be included into Sφnx in a similar fashion to the generic modeling language dℒ, i. e., in the form of Eclipse plugins that provide the DSL metamodel and the modeling editor. In order to be processable in a verification tool, such as KeYmaera, a model transformation specification (e. g., using the Atlas transformation language ATL [18]) from the DSL to the tool's modeling language (e. g., dℒ) must be provided by these plugins.

For modeling hybrid systems, an interesting opportunity for inspecting the behavior of such a system prior to verification is provided by Mathematica[5] 9, which is able to simulate and plot hybrid system behavior. Specifically, hybrid systems behavior can be plotted using a combination of *NDSolve* and *WhenEvent*. We envision transforming corresponding excerpts of dℒ to Mathematica for visualizing plots of the dynamic behavior and their hybrid programs over time in Sφnx.

[5] `www.wolfram.com/mathematica`

3.3.3 During the Proof

Collaboration support in Sφnx currently comprises model as well as proof comparison, both locally and with the model and proof repositories maintained in a central source code repository. For this, not only textual comparison is implemented, but also structural comparison of models expressed in terms of the dℒ metamodel, as well as of proofs expressed in terms of the KeY metamodel is supported (cf. Figure 4). Especially for collaboration, exchanging proofs and inspecting updates on partial proofs is vital. For example, highlighted changes between different versions of a partial proof lets one easily spot and adopt proof progress made by other team members, go back and forth between versions, and detect conflicts.

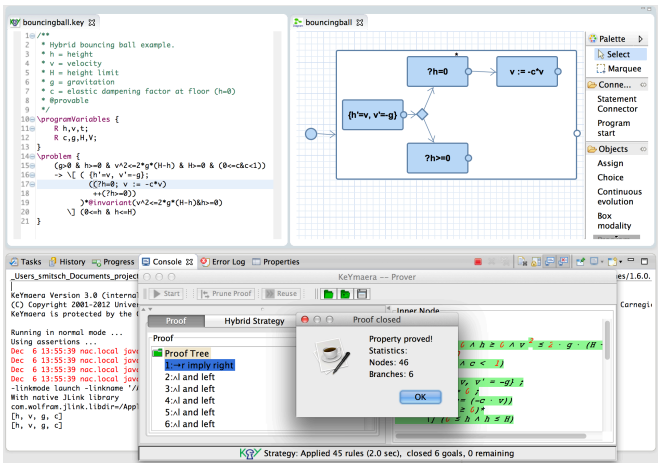

Figure 3: Textual and graphical syntax, proof in KeYmaera

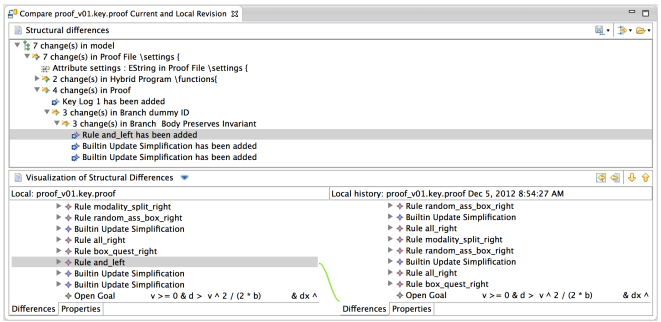

Figure 4: Comparison of the structure of two proof versions

To further facilitate knowledge and expertise exchange, specific unsolved subproblems of a proof (e. g., complex arithmetic problems) can be flagged in KeYmaera and extracted to other tools. Thereby, division of verification work is achieved. An open question, however, concerns the merging of the partial verification results into a coherent proof. In a first step, we plan to extend KeYmaera to let experts close proof branches in the same manner as it currently trusts mathematical solvers. To avoid unintentionally closing a proof goal with a proof that may no longer apply in case the original model changed, Sφnx checks that the statement remained the same (textually) when replaying an arithmetic proof. Later, actual proof certificates and proof strategies will be exchanged to further increase trust, and more sophisticated comparisons of proof goals are envisioned to better support replaying proofs.

3.4 The Collaboration Backend

The Sφnx modeling tool uses existing Eclipse plugins to connect to a variety of backend source code repositories and online project management tools. As source code repository we utilize Subversion[6] and the Eclipse plugin Subclipse[7]. Currently, Mylyn[8] and its connectors are used for accessing online project management tools (e. g., Bugzilla[9], Redmine[10], or any web-based tool via Mylyn's Generic Web templates connector) and exchanging tickets (i. e., requests for verification). These tickets are the organizational means for collaborating on verification problems and tasks within a working group. Exchange of models and proofs may then be conducted either by attaching files to tickets, or by linking tickets directly to models and proofs in the source code repository. In the latter case, one benefits from the model and proof comparison capabilities of Sφnx. Verification tools (currently KeYmaera), are linked to the modeling tool by implementing extensions to the Eclipse launch configuration. These extensions hook into the context menu of Eclipse (models in dℒ and proof files in KeY in our case) and, on selection, launch an external program.

As a vision of extending collaboration support, it is planned to integrate Wikis and other online collaboration tools (currently, we use Redmine both as project management repository and for knowledge exchange) for exchanging knowledge on proof tactics. Additionally, collaboration with experts outside the own organization can be fostered by linking to Web resources, such as MathOverflow[11] and Amazon Mechanical-Turk[12]. Especially interesting, in this respect, is the possibility to create a social-network-like expert platform. In such a platform, requests could be forwarded to those experts whose knowledge matches the verification problem best.

[6] `subversion.apache.org`
[7] `subclipse.tigris.org`
[8] `www.eclipse.org/mylyn`
[9] `www.bugzilla.org`
[10] `www.redmine.org`

4 Application Example

With the increased introduction of autonomous robotic ground vehicles as consumer products—such as autonomous hovers and lawn mowers, or even accepting driverless cars on regular roads in California—we face an increased need for ensuring product safety not only for the good of our consumers, but also for the sake of managing manufacturer liability. One important aspect in building such systems is to make them scrutable, in order to mitigate unrealistic expectations and increase trust [38]. In the design stage of such systems, formal verification techniques ensure correct functioning w.r.t. some safety condition, and thus, increase trust. In the course of this, formal verification techniques can help to make assumptions explicit and thus clearly define what can be expected from the system under which circumstances.

We discuss a model of an autonomous robotic ground vehicle and its proof (to increase trust), and describe how we can derive bounds on the behavior of that vehicle. The sample autonomous robotic ground vehicle used in this paper operates on predefined tracks and, thus, cannot steer freely (i. e., single wheel drive with angular velocity zero). The control options of such vehicles are limited to choosing the value of acceleration and result in sequences of straight lines as trajectories. The trajectories are thus akin to those produced by our previous car models [23],[24].

In this example, navigation of autonomous robotic ground vehicles is considered safe, if such vehicles are able to stay within their assigned area (e. g., on a track) and do not actively crash with obstacles. Since we cannot guarantee reasonable behavior of obstacles, however, autonomous ground vehicles are allowed to passively crash (i. e., while obstacles might run into the robot, the robot will never move into a position where the obstacle could not avoid a collision).

[11] `mathoverflow.net`
[12] `www.mturk.com`

Model 1 Single wheel drive without steering (one-dimensional robot navigation)

$$swd \equiv (ctrl; dyn)^* \tag{1}$$
$$ctrl \equiv (ctrl_r \parallel ctrl_o) \tag{2}$$
$$ctrl_r \equiv (a_r := -b) \tag{3}$$
$$\cup\ (?safe;\ a_r := *;\ ?-b \leq a_r \leq A) \tag{4}$$
$$\cup\ (?v_r = 0;\ a_r := 0;\ o_r := *;\ ?o_r^2 = 1) \tag{5}$$
$$safe \equiv x_{\underline{b}} + \frac{1-o_r}{2} \cdot \left(\frac{v_r^2}{2b} + \left(\frac{A}{b}+1\right) \cdot \left(\frac{A}{2} \cdot \varepsilon^2 + \varepsilon \cdot v_r\right)\right) < x_r < x_{\overline{b}} - \frac{1+o_r}{2} \cdot \left(\frac{v_r^2}{2b} + \left(\frac{A}{b}+1\right) \cdot \left(\frac{A}{2} \cdot \varepsilon^2 + \varepsilon \cdot v_r\right)\right) \tag{6}$$
$$\wedge\ \|x_r - x_o\| \geq \frac{v_r^2}{2b} + \left(\frac{A}{b}+1\right) \cdot \left(\frac{A}{2} \cdot \varepsilon^2 + \varepsilon \cdot v_r\right) + V \cdot \left(\varepsilon + \frac{v_r + A \cdot \varepsilon}{b}\right) \tag{7}$$
$$ctrl_o \equiv \left(?v_o = 0;\ o_o := *;\ ?o_o^2 = 1\right) \tag{8}$$
$$\cup\ (v_o := *;\ ?0 \leq v_o \leq V) \tag{9}$$
$$dyn \equiv (t := 0;\ x_r' = o_r \cdot v_r,\ v_r' = a_r,\ x_o' = o_o \cdot v_o,\ t' = 1\ \&\ v_r \geq 0 \wedge v_o \geq 0 \wedge t \leq \varepsilon) \tag{10}$$

4.1 Modeling

Model 1 shows the model of a hybrid system comprising the control choices of an autonomous robotic ground vehicle, the control choices of a moving obstacle, and the continuous dynamics of the system. The system represents the common controller-plant model: it repeatedly executes control choices followed by dynamics, cf. (1). The control of the robot is executed in parallel to that of the obstacle, cf. (2).

The robot has three options: It is always allowed to brake, as expressed by cf. (3) having no test condition. If its current state is safe (defined by (6)), then the robot may accelerate with any rate within its physical bounds, cf. (4). For this, we utilize the modeling pattern introduced above: we assign an arbitrary value to the robot's acceleration state ($a_r := *$), which is then restricted to any value from the interval $(-b, A)$ using a test ($?-b \leq a_r \leq A$). Finally, if the robot is stopped, it may choose to remain in its current spot and may or may not change its orientation while doing so, cf. (5). This is expressed again by arbitrary assignment with subsequent test: this time, the test $?o_r^2 = 1$, however, restricts the orientation value to either forwards or backwards ($o_r \in \{1, -1\}$).

For always remaining safely inside its area, the robot must account for (i) its own braking distance ($\frac{v^2}{2b}$), (ii) the distance it may travel with its current velocity ($\varepsilon \cdot v_r$) until it is able to initiate braking, and (iii) the distance needed to compensate the acceleration A that may have been chosen in the worst case, cf. (6). Note, that the safety margin applies to either the upper or the lower bound of the robot's area, depending on the robot's orientation: when driving forward (i. e., towards the upper bound), we do not need a safety margin towards the lower bound, and vice versa. This is expressed by the factors $\frac{1-o_r}{2}$ and $\frac{1+o_r}{2}$, which mutually evaluate to zero (e. g., $\frac{1-o_r}{2} = 0$ when driving forward with $o_r = 1$). The distance between the robot and the obstacle must be large enough to (i) allow the robot to brake to a stand-still, (ii) compensate its current velocity and worst-case acceleration, and (iii) account for the obstacle moving towards the robot with worst-case velocity V while the robot is still not stopped, cf. (7). Note, that w.r.t. the obstacle we have to be more conservative than towards the bounds, because we want to be able to come to a full stop even when the obstacle approaches the robot from behind.

The obstacle, essentially, has similar control options as the robot (with the crucial difference of not having to care about safety): it may either remain in a spot and possibly change its orientation (8), or choose any velocity up to V, cf. (9).

4.2 Verification

We verify the safety of acceleration and orientation choices as modeled in Model 1 above, using a formal proof calculus for $\mathsf{d}\mathcal{L}$ [28, 30]. The robot is safely within its assigned area and at a safe distance to the obstacle, if it is able to brake to a complete stop at all times[13]. The following condition captures this requirement as an invariant that we want to hold at all times during the execution of the model:

$$\begin{aligned} r \text{ stoppable } (o,b) \equiv\ & \|x_r - x_o\| \ \geq\ \frac{v_r^2}{2b} + \frac{v \cdot V}{b} \\ & \wedge\ x_{\underline{b}} + \frac{1-or}{2} \cdot \frac{v_r^2}{2b} < x_r < x_{\overline{b}} - \frac{1+or}{2} \cdot \frac{vr^2}{2b} \\ & \wedge\ v_r \geq 0 \wedge o_r^2 = 1 \\ & \wedge\ o_o^2 = 1 \wedge 0 \leq v_o \leq V \end{aligned}$$

The formula states that the distance between the robot to both the obstacle and the bounds is safe, if there is still enough distance for the robot to brake to a complete stop before it reaches either. Also, the robot must drive with positive velocity, the chosen directions of robot and obstacle must be either forwards ($o_r = 1$) or backwards ($o_r = -1$), and the obstacle must use only positive velocities up to V.

Theorem 1 (Safety of single wheel drive)**.** *If a robot is inside its assigned area and at a safe distance from the obstacle's*

[13] The requirement that the robot has to ensure an option for the obstacle to avoid a collision is ensured trivially, since the obstacle in this model can choose its velocity directly. In a more realistic model the obstacle would choose acceleration instead; then the robot had to account for the braking distance of the obstacle, too

position x_o initially, then it will not actively collide with the obstacle and stay within its area while it follows the `swd` *control model (Model 1), as expressed by the provable* dL *formula:*

$$r \text{ stoppable } (o,b) \rightarrow [swd]\big((v>0 \rightarrow \|p_r - p_o\| > 0) \\ \wedge x_{\underline{b}} < x_r < x_{\overline{b}}\big)$$

We proved Theorem 1 using KeYmaera. With respect to making autonomous systems more scrutable, such a proof may help in a twofold manner: on the one hand, it may increase trust in the implemented robot (given the assumption that the actual implementation can be traced back to the abstract model). On the other hand, it makes the behavior of the robot more understandable. In this respect, the most interesting properties of the proven model are the definition of $safe$ and the invariant, which allow us to analyze design tradeoffs and tell us what is always true about the system regardless of its state. As an example, let us consider the distance between the robot and the obstacle that is considered safe: $\|x_r - x_o\| \geq \frac{v_r^2}{2b} + \big(\frac{A}{b}+1\big)\cdot\big(\frac{A}{2}\cdot\varepsilon^2 + \varepsilon\cdot v_r\big) + V\cdot\big(\varepsilon + \frac{v_r + A\cdot\varepsilon}{b}\big)$. This distance can be interpreted as the minimum distance that the robot's obstacle detection sensors are required to cover; it is a function of other robot design parameters (maximum velocity, braking power, worst-case acceleration, sensor/processor/actuator delay) and the parameters expected in the environment (obstacle velocity). $\|x_r - x_o\|$ can be optimized w.r.t. different aspects: for example, to find the most cost-efficient combination of components that still guarantees safety, to specify a safe operation environment given a particular robot configuration, or to determine time bounds for algorithm optimization.

With respect to the manual guidance and collaboration needed in such a proof, we had to apply knowledge in hybrid systems and in-depth understanding of the robot model to find a system invariant, which is the most important manual step in the proof above. We further used arithmetic interactions, such as the hiding of superfluous terms to reduce arithmetic complexity, transforming and replacing terms (e. g., substitute the absolute function with two cases, one for negative and one for positive values).

4.3 Model Variants and Proof Structure

Since it is hard to come up with a fully verifiable model that includes all the details right from the beginning, the models discussed in the previous section are the result of different modeling and verification variants. In the process of creating these models, different assumptions and simplifications were applied until we reached the version in Model 1. For example, one can make explicit restrictions on particular variables, such as first letting the robot start in a known direction (instead of an arbitrary direction). Such assumptions and simplifications, of course, are not without implications on the proof. While in some aspect a proof may become easier, it may become more laborious or more complex in another. In this section, we discuss five variants of the single wheel drive model (without obstacle) to demonstrate implications on the proof structure and on the entailed manual guidance needed to complete a proof in KeYmaera.

The following model variants are identical in terms of the behavior of the robot. However, assumptions on the starting direction were made in the antecedent of a provable dL formula, and the starting direction as well as the orientation of the robot were explicitly distinguished by disjunction or nondeterministic choice, or implicitly encoded in the arithmetic, as described below.

Assumed starting direction, orientation by disjunction In the first variant, the robot is assumed to start in a known direction, specified in the antecedent of $o_r = 1 \ldots \rightarrow [swd](x_{\underline{b}} < x_r < x_{\overline{b}})$. Also, the orientation of the robot is explicitly distinguished by disjunction in $safe \equiv (o_r = -1 \wedge x_{\underline{b}} + \ldots < x_r) \vee (o_r = 1 \wedge x_r < x_{\overline{b}} - \ldots)$, and the robot had an explicit choice on turning during stand-still $(?v_r = 0;\ o_r := -o_r; \ldots) \cup (?v_r = 0; \ldots)$.

Orientation by arithmetic In the second variant, we kept the assumed starting direction of the first variant. However, the orientation by disjunction in the definition of $safe$ was replaced by using o_r as discriminator value encoded in the arithmetic, as in $safe \equiv x_{\overline{b}} - \frac{1+o_r}{2}\cdot(\ldots) < x_r < x_{\underline{b}} + \frac{1-o_r}{2}\cdot(\ldots)$.

Arbitrary starting direction by disjunction The third variant relaxes the assumption on the starting direction by introducing a disjunction of possible starting directions in the antecedent of the provable formula $(o_r = 1 \vee o_r = -1) \ldots \rightarrow [swd](x_{\underline{b}} < x_r < x_{\overline{b}})$.

Arbitrary starting direction by arithmetic The fourth variant replaces the disjunction in the antecedent by stating the two orientation options as $o_r^2 = 1$ in $o_r^2 = 1 \ldots \rightarrow [swd](x_{\underline{b}} < x_r < x_{\overline{b}})$.

Replace non-deterministic choice with arithmetic Finally, we replace the non-deterministic turning choice with $(?v_r = 0;\ o_r := *;\ ?o_r^2 = 1; \ldots)$.

Table 2 summarizes the proof structures of the five variants. Unsurprisingly—when considering the rules of the dL proof calculus [29] as listed in Table 2—disjunctions in the antecedent ($\vee l$) or in tests of hybrid programs, as well as non-deterministic choices ($[\cup]$) increase the number of proof branches and with it the number of manual proof steps. The number of proof branches can be reduced, if we can replace disjunctions in the antecedent (but also conjunctions in the consequent) or non-deterministic choices in the hybrid program by an equivalent arithmetic encoding. Conversely, this means that some arithmetic problems can be traded for easier ones with additional proof branches.

5 Conclusion

In this paper, we gave a vision of a verification-driven engineering toolset including hybrid and arithmetic verification tools, and introduced modeling and collaboration tools with the goal of making formal verification of hybrid systems accessible to a broader audience. The current implementation features textual and graphical modeling editors, integration of KeYmaera as a hybrid systems verification tool, model and proof comparison, and connection to various collaboration backend systems. The VDE toolset is currently being tested in a collaborative verification setting between Carnegie Mellon University, the University of Cambridge, and the University of Edinburgh.

Table 2: Nodes, branches, and manual proof steps of variants

	Variant	Nodes	Branches	Manual steps	Avoids
(i)	Assumed starting direction, orientation by disjunction	387	34	24	
(ii)	Orientation by arithmetic	331	28	25	$(\vee l)$
(iii)	Arbitrary starting direction by disjunction	650	56	44	
(iv)	Arbitrary starting direction by arithmetic	185	17	22	$(\vee l)$
(v)	Replace non-deterministic choice	160	14	29	$([\cup])(\wedge r)$

$$\frac{\Gamma, \phi \vdash \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \vee \psi \vdash \Delta}\ (\vee l) \qquad \frac{[a]\phi \wedge [b]\phi}{[a \cup b]\phi}\ ([\cup]) \qquad \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta}\ (\wedge r)$$

Acknowledgments

This material is based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246, NSF EXPEDITION CNS-0926181, and under Grant Nos. CNS-1035800 and CNS-0931985, by DARPA under agreement number FA8750-12-2-0291, and by the US Department of Transportation's University Transportation Center's TSET grant, award# DTRT12GUTC11. Passmore was also supported by the UK's EPSRC [grants numbers EP/I011005/1 and EP/I010335/1]. Mitsch was also supported by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under grant FFG FIT-IT 829589, FFG BRIDGE 838526, and FFG Basisprogramm 838181.

References

[1] Rajeev Alur, 'Formal verification of hybrid systems', in *Proceedings of the 11th International Conference on Embedded Software (EMSOFT)*, eds., Samarjit Chakraborty, Ahmed Jerraya, Sanjoy K. Baruah, and Sebastian Fischmeister, pp. 273–278. ACM, (2011).

[2] Manas Bajaj, Andrew Scott, Douglas Deming, Gregory Wickstrom, Mark De Spain, Dirk Zwemer, and Russell Peak, 'Maestro—a model-based systems engineering environment for complex electronic systems', in *Proceedings of the 22nd Annual INCOSE International Symposium*, Rome, Italy, (2012). INCOSE.

[3] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB standard: Version 2.0. `http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.0-r12.09.09.pdf`, 2012. (last accessed: 2013-01-09).

[4] Maria Christakis, Peter Müller, and Valentin Wüstholz, 'Collaborative verification and testing with explicit assumptions', in *FM*, eds., Dimitra Giannakopoulou and Dominique Méry, volume 7436 of *LNCS*, 132–146, Springer, (2012).

[5] Pieter Collins and John Lygeros, 'Computability of finite-time reachable sets for hybrid systems', in *44th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pp. 4688–4693. IEEE, (dec. 2005).

[6] James H. Davenport and Joos Heintz, 'Real quantifier elimination is doubly exponential', *J. Symb. Comput.*, **5**(1-2), 29–35, (February 1988).

[7] Leonardo De Moura and Nikolaj Bjørner, 'Z3: an efficient smt solver', in *TACAS*, pp. 337–340, Budapest, Hungary, (2008). Springer-Verlag.

[8] Bart De Schutter, W.P.M.H. Heemels, Jan Lunze, and Christophe Prieur, 'Survey of modeling, analysis, and control of hybrid systems', in *Handbook of Hybrid Systems Control – Theory, Tools, Applications*, eds., Jan Lunze and Françoise Lamnabhi-Lagarrigue, chapter 2, 31–55, Cambridge University Press, Cambridge, UK, (2009).

[9] Patricia Derler, Edward A. Lee, and Alberto Sangiovanni-Vincentelli, 'Modeling cyber-physical systems', *Proceedings of the IEEE*, **100**(1), 13 –28, (jan. 2012).

[10] Johannes Faber, Sven Linker, Ernst-Rüdiger Olderog, and Jan-David Quesel, 'Syspect - modelling, specifying, and verifying real-time systems with rich data', *International Journal of Software and Informatics*, **5**(1-2), 117–137, (2011).

[11] Goran Frehse, 'PHAVer: Algorithmic verification of hybrid systems past HyTech', in *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings*, eds., Manfred Morari and Lothar Thiele, volume 3414 of *LNCS*, pp. 258–273. Springer, (2005).

[12] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler, 'SpaceEx: Scalable verification of hybrid systems', in *CAV*, ed., Shaz Qadeer Ganesh Gopalakrishnan, LNCS. Springer, (2011).

[13] Aniruddha S. Gokhale, Krishnakumar Balasubramanian, Arvind S. Krishna, Jaiganesh Balasubramanian, George Edwards, Gan Deng, Emre Turkay, Jeff Parsons, and Douglas C. Schmidt, 'Model driven middleware: A new paradigm for developing distributed real-time and embedded systems', *Sci. Comput. Program.*, **73**(1), 39–58, (2008).

[14] Timothy Gowers and Michael Nielsen, 'Massively collaborative mathematics', *Nature*, **461**, 879–881, (2009).

[15] Thomas C. Hales, John Harrison, Sean McLaughlin, Tobias Nipkow, Steven Obua, and Roland Zumkeller, 'A revision of the proof of the Kepler conjecture', *Discrete & Computational Geometry*, **44**(1), 1–34, (2010).

[16] Matthew Clayton Hause and Francis Thom, 'An integrated MDA approach with SysML and UML', in *Proc. of the 13th Intl. Conference on Engineering of Complex Computer Systems*, ICECCS '08, pp. 249–254, Washington, DC, USA, (2008). IEEE Computer Society.

[17] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Marco A. A. Sanvido, 'Extreme model checking', in *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, ed., Nachum Dershowitz, volume 2772 of *LNCS*, pp. 332–358. Springer, (2003).

[18] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev, 'ATL: A model transformation tool', *Sci. Comput. Program.*, **72**(1-2), 31–39, (2008).

[19] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood, 'seL4: formal verification of an OS kernel', in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 207–220, New York, NY, USA, (2009). ACM.

[20] Fabrice Kordon, Jérôme Hugues, and Xavier Renault, 'From model driven engineering to verification driven engineering', in *Proc. of the 6th IFIP int. workshop on Software Technologies for Embedded and Ubiquitous Systems*, pp. 381–393. Springer, (2008).

[21] Yanni Kouskoulas, David Renshaw, André Platzer, and Peter Kazanzides, 'Certifying the safe design of a virtual fixture control algorithm for a surgical robot', in *HSCC*, eds., Calin Belta and Franjo Ivancic. ACM, (2013).

[22] Orna Kupferman and Moshe Y. Vardi, 'Modular model checking', in *Revised Lectures from the International Symposium on Compositionality: The Significant Difference*, COM-

POS'97, pp. 381–401, London, UK, (1998). Springer.
[23] Sarah M. Loos, André Platzer, and Ligia Nistor, 'Adaptive cruise control: Hybrid, distributed, and now formally verified', in *FM*, volume 6664 of *LNCS*, pp. 42–56. Springer, (2011).
[24] Stefan Mitsch, Sarah M. Loos, and André Platzer, 'Towards formal verification of freeway traffic control', in *Proc. of the 2nd Int. Conference on Cyber-Physical Systems (ICCPS)*, ed., Chenyang Lu, pp. 171–180. IEEE, (2012).
[25] Wojciech Mostowski, 'The KeY syntax', in *Verification of Object-Oriented Software. The KeY Approach*, eds., Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt, volume 4334 of *Lecture Notes in Computer Science*, 599–626, Springer Berlin Heidelberg, (2007).
[26] Grant Olney Passmore, *Combined Decision Procedures for Nonlinear Arithmetics, Real and Complex*, Ph.D. dissertation, University of Edinburgh, 2011.
[27] Grant Olney Passmore, Lawrence C. Paulson, and Leonardo Mendonça de Moura, 'Real algebraic strategies for MetiTarski proofs', in *AISC/MKM/Calculemus*, eds., Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, volume 7362 of *Lecture Notes in Computer Science*, pp. 358–370. Springer, (2012).
[28] André Platzer, 'Differential dynamic logic for hybrid systems.', *J. Autom. Reas.*, **41**(2), 143–189, (2008).
[29] André Platzer, 'Differential-algebraic dynamic logic for differential-algebraic programs', *J. Log. Comput.*, **20**(1), 309–352, (2010).
[30] André Platzer, *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*, Springer, Heidelberg, 2010.
[31] André Platzer, 'Stochastic differential dynamic logic for stochastic hybrid programs', in *CADE*, pp. 431–445, (2011).
[32] André Platzer, 'A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems', *Logical Methods in Computer Science*, **8**(4), 1–44, (2012). Special issue for selected papers from CSL'10.
[33] André Platzer, 'The complete proof theory of hybrid systems', in *LICS*, pp. 541–550. IEEE, (2012).
[34] André Platzer and Edmund M. Clarke, 'Computing differential invariants of hybrid systems as fixedpoints', *Formal Methods in System Design*, **35**(1), 98–120, (2009).
[35] André Platzer and Edmund M. Clarke, 'Formal verification of curved flight collision avoidance maneuvers: A case study', in *FM*, eds., Ana Cavalcanti and Dennis Dams, volume 5850 of *LNCS*, pp. 547–562. Springer, (2009).
[36] André Platzer and Jan-David Quesel, 'KeYmaera: A hybrid theorem prover for hybrid systems.', in *IJCAR*, eds., Alessandro Armando, Peter Baumgartner, and Gilles Dowek, volume 5195 of *LNCS*, pp. 171–178. Springer, (2008).
[37] André Platzer and Jan-David Quesel, 'European Train Control System: A case study in formal verification', in *ICFEM*, eds., Karin Breitman and Ana Cavalcanti, volume 5885 of *LNCS*, pp. 246–265. Springer, (2009).
[38] Nava Tintarev, Nir Oren, Kees Van Deemter, Roman Kutlak, Matt Green, Judith Masthoff, and Wamberto Vasconcelos, 'SAsSy—scrutable autonomous systems', in *to appear in: Proceedings of the 2013 Workshop on Enabling Domain Experts to use Formalised Reasoning (Do-Form)*, (2013).