Implicit and Explicit Proof Management in KeYmaera X*

Stefan Mitsch
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA
smitsch@cs.cmu.edu

Hybrid systems theorem proving provides strong correctness guarantees about the interacting discrete and continuous dynamics of cyber-physical systems. The trustworthiness of proofs rests on the soundness of the proof calculus and its correct implementation in a theorem prover. Correctness is easier to achieve with a soundness-critical core that is stripped to the bare minimum, but, as a consequence, proof convenience has to be regained outside the soundness-critical core with proof management techniques. We present modeling and proof management techniques that are built on top of the soundness-critical core of KeYmaera X to enable expanding definitions, parametric proofs, lemmas, and other useful proof techniques in hybrid systems proofs. Our techniques steer the uniform substitution implementation of the differential dynamic logic proof calculus in KeYmaera X to allow users choose when and how in a proof abstract formulas, terms, or programs become expanded to their concrete definitions, and when and how lemmas and sub-proofs are combined to a full proof. The same techniques are exploited in implicit sub-proofs (without making such sub-proofs explicit to the user) to provide proof features, such as temporarily hiding formulas, which are notoriously difficult to get right when implemented in the prover core, but become trustworthy as proof management techniques outside the core. We illustrate our approach with several useful proof techniques and discuss their presentation on the KeYmaera X user interface.

1 Introduction

Hybrid systems theorem proving provides strong correctness guarantees about the interacting discrete and continuous dynamics of cyber-physical systems. Theorem proving is most valuable early in the design of a system, since it is not merely a technique to prove the correctness of an already correct system, but also shines when analyzing a system in all its subtleties to discover unknown or only partially known properties of the system. The trustworthiness of proofs and analysis results, however, rests on the soundness of the proof calculus and its correct implementation in a theorem prover. Typical theorem prover implementations often opt for directly representing the rules of a proof calculus in the theorem prover (e.g., with axiom schemata in [1,18], or with trusted implementations of rules (e.g., KeYmaeraD [20]) or decision procedures (e.g., invariant computation [8] in the HHL prover [24]). The downside of such an approach is not only that implementations of rules and their side conditions become soundness-critical, but also that additional features often result in increasing the size of the soundness-critical code base of the theorem prover. Correctness is easier to achieve with an LCF-style approach that strips the soundness-critical core to the bare minimum, but, as a consequence, proof convenience has to be regained outside the soundness-critical core with proof management techniques. The KeYmaera X [5] theorem prover for hybrid systems takes an LCF-style approach; previous techniques expanded the capabilities of KeYmaera X primarily by providing tactics [4], e.g., for certifying solutions of differential equations [16],

*This material is based upon work supported by the Air Force Office of Scientific Research under grant number FA9550-16-1-0288 and FA8750-18-C-0092. Any opinions, finding, and conclusion or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

Submitted to:
F-IDE 2021

for certifying safety and liveness properties of differential equations [19,21], for stability proofs [22], for code synthesis [3], for component-based modeling and verification [12], and for monitor synthesis [10].

In this paper, we present modeling and proof management techniques that are built on top of the soundness-critical core of KeYmaera X to enable structuring and modularizing models with definitions and modularizing proofs with lemmas. These modeling and proof management techniques were developed primarily with interactive proofs in mind, but may also be beneficial for automation (e.g., hierarchical definitions may serve as proof hints). Useful proof techniques for explicit proof management include expanding definitions of the model during a proof, parametric proofs to make progress in proofs despite unknown system properties (e.g., loop invariants), and creating and applying lemmas. Our techniques steer the uniform substitution implementation of the differential dynamic logic proof calculus in KeYmaera X to allow users choose when in a proof and how abstract formulas, terms, or programs become expanded to their concrete definitions, and when and how lemmas and sub-proofs are combined to a full proof. The same techniques are exploited in implicit sub-proofs (without making such sub-proofs explicit to the user) to hide technicalities of the prover implementation whose details are irrelevant to the user, or to provide proof features, such as temporarily hiding formulas, which are notoriously difficult to get right when implemented in the prover core, but become trustworthy as proof management techniques outside the core. On the user interface, we attempt to make such proof features available as part of the usual user interactions: for example, when a tactic asks for input (e.g., a loop invariant), users start a parametric proof simply by using uninterpreted function and predicate symbols as tactic inputs, which then appear like elements of the input model whose concrete interpretations can be defined and expanded at a later point in the proof. That way, users can focus on exploring and understanding a system by way of formal proof to provide insight to the theorem prover when it becomes available during the proof.

The remainder of this paper is structured as follows: Section 2 introduces differential dynamic logic and the relevant core and user interface features of KeYmaera X. Section 3 gives an example proof that combines and illustrates several of the desired proof management techniques, Section 4 and Section 5 discuss the underlying lemma application and proof techniques and their appearance on the user interface, and Section 6 concludes the paper with a discussion of related and future work.

2 Preliminaries

Differential Dynamic Logic by Example Differential dynamic logic dL [16, 17] is a specification language and verification calculus for hybrid systems written as hybrid programs. The syntax of *hybrid programs* (HP) is described by the following grammar where α, β are hybrid programs, x is a variable and $e, f(x)$ are terms, Q is a logical formula:

$$\alpha, \beta \ ::= \ x := e \mid ?Q \mid x' = f(x) \,\&\, Q \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$$

Assignments $x := e$ and tests $?Q$ (to abort execution and discard the run if Q is not true) are as usual. Differential equations $x' = f(x) \,\&\, Q$ are followed along a solution of $x' = f(x)$ for any duration as long as the evolution domain constraint Q is true at every moment along the solution. Nondeterministic choice $\alpha \cup \beta$ runs either α or β, sequential composition $\alpha; \beta$ first runs α and then β on the resulting states of α, and nondeterministic repetition α^* runs α any natural number of times.

For example, hybrid program

$$\Big(\underbrace{\text{if } (x < 1)\{y := -x\} \text{ else } \{y := *; ?y > 2\}}_{ctrl}; \ \underbrace{x' = -xy}_{ode}\Big)^*$$

repeats program *ctrl* followed by differential equation *ode* arbitrarily often; program *ctrl* is a choice between setting y to the value of $-x$ when $x < 1$ or else picking any $y > 2$. The combined effect of *ctrl* and *ode* is an exponential increase/decay of x with a rate depending on the choice of y. When programs become more complicated, it is useful to literally modularize hybrid programs into *ctrl*, *ode* etc. using program symbols and use definitions as a structuring mechanism for models.

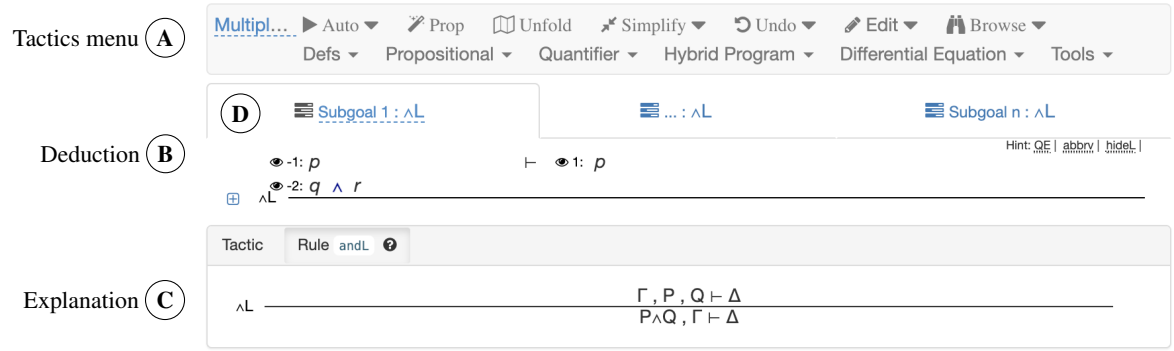
The formulas of dL describe properties of hybrid programs, summarized by the following grammar where P, Q are formulas, $e, \tilde{e}$ are terms, x is a variable and α is a hybrid program:

$$P, Q ::= e \geq \tilde{e} \mid \neg P \mid P \wedge Q \mid P \vee Q \mid P \rightarrow Q \mid P \leftrightarrow Q \mid \forall x P \mid \exists x P \mid [\alpha]P \mid \langle\alpha\rangle P$$

The operators of first-order real arithmetic are as usual with quantifiers ranging over the reals. Formula $[\alpha]P$ is true in a state iff formula P is true after all ways of running hybrid program α, which is useful for expressing safety properties. Dually, liveness properties are expressed with $\langle\alpha\rangle P$, which is true in a state iff P is true after at least one run of α.

$$\frac{\Gamma_1 \vdash \Delta_1 \ (\text{subgoal}_1) \qquad \ldots \qquad \Gamma_n \vdash \Delta_n \ (\text{subgoal}_n)}{\Gamma \vdash \Delta \ (\text{conclusion})}$$

(a) A `Provable` representing proof state in the KeYmaera X core



(b) User interface displays the open subgoals of a proof. Tactics menu (A) lists proof tactics and automation, deduction view (B) lists the open subgoals (one tab per subgoal), explanation (C) illustrates the last applied proof step and lists the recorded proof history as a tactic. Subgoal 1 is selected (D) and its sequent $p, q \wedge r \vdash p$ is displayed.

Figure 1: Proof state data structure and rendering on the user interface.

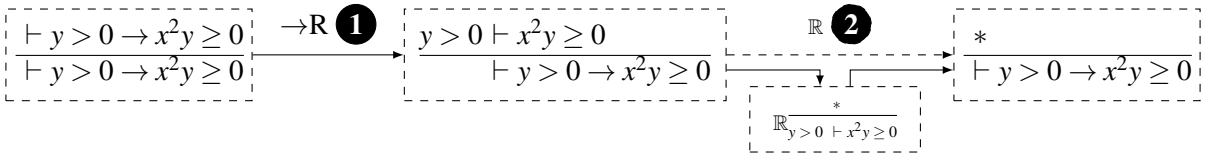
Proofs in the KeYmaera X Core The KeYmaera X prover core represents proof state as derived rules called `Provables`, which lists the conclusion to prove and the open subgoals, as illustrated in Fig. 1a. Conclusion and subgoals are each represented with a sequent of the form $\Gamma \vdash \Delta$: assumptions are in Γ, whereas Δ lists the alternatives to prove. The meaning of sequent $\Gamma \vdash \Delta$ is that of dL formula $\bigwedge_{p \in \Gamma} p \rightarrow \bigvee_{q \in \Delta} q$. Validity of the subgoals justifies validity of the conclusion; a proof is closed when there are no more open subgoals. The user interface of KeYmaera X in Fig. 1b displays proof state in its deduction view (B), provides automation and tactics (A) to progress in the proof, and lists proof step explanations (C) as well as a tactic summarizing the recorded proof history. Proofs can be started from an initial conjecture, as well as from a (partial) tactic that advances the proof state according to the tactic steps and displays the remaining proof goals (further manual steps are then recorded and extend the provided tactic). The deduction view strives for close mnemonic similarity to text books [9, 11] while

maximizing screen estate use (it displays open subgoals in tabs to utilize the full screen width for each subgoal). In principle, the user interface could use typesetting libraries such as MathJax, to resemble textbook appearance even more closely, but such attempts were abandoned for performance reasons.

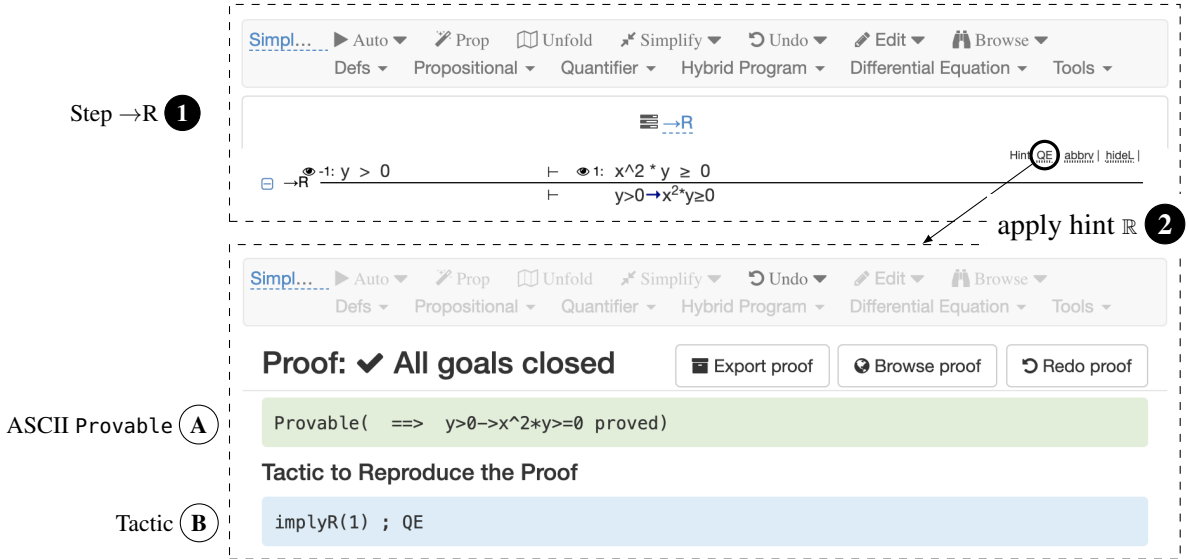
The KeYmaera X core is stateless, it does not keep track of proof state. Instead, tactics and proof management outside the core keep track of `Provables` and instruct the core to apply operations on `Provables` to transform proof state, see [11] for a description of how tactics combine axioms and a comparison to alternative implementation approaches. Major core operations are to

- create a `Provable`, which is allowed only from a small number of sources, the most important ones are $\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta}$, $\mathbb{R}\dfrac{*}{\Gamma \vdash \Delta}$, $\dfrac{*}{\vdash \text{dL axiom}}$ (from left to right: starting a proof by justifying the conjecture from itself, real arithmetic facts, and dL axioms);
- apply another `Provable`, whose conjecture matches a subgoal to replace the existing subgoal with the subgoals of the other `Provable`, which we exploit for applying lemmas.
- apply uniform substitution to replace predicate symbols with formulas, function symbols with terms, and program symbols with hybrid programs, which is useful to support definitions.

A typical proof, illustrated in Fig. 2, retrieves an initial `Provable` from the KeYmaera X core and then proceeds by handing back the `Provable` to the core together with a proof rule to retrieve a follow-up



(a) A simple proof instructing the core to create a new `Provable`, apply rule →R in step 1, and apply ℝ in step 2, which obtains a `Provable` from a trusted solver and applies it to the remaining subgoal to close the proof.



(b) Presentation of step →R 1 on the user interface as a sequent proof. The final `Provable` is printed directly from the KeYmaera X core in ASCII syntax, the tactic below lists the steps to reproduce the proof from the conjecture.

Figure 2: Steps in a proof in the core vs. presentation on the user interface

$$\frac{p \vdash [a]q \qquad p \vdash [b]q}{\vdash p \to [a \cup b]q} \;\text{US}\; \textcircled{1} \;\rightarrow\; \frac{x>0 \vdash [x'=-x]x>0 \qquad x>0 \vdash [(x:=\frac{x}{2})^*]x>0}{\vdash x>0 \to [x'=-x \cup (x:=\frac{x}{2})^*]x>0}$$

Figure 3: Uniform substitution $\sigma = \{p \mapsto x>0, a \mapsto x'=-x, b \mapsto (x:=\frac{x}{2})^*, q \mapsto x>0\}$ on an entire `Provable` has uniform effect across subgoals and conclusion [16, Thm. 27]

`Provable`. This process is repeated until all subgoals are either reduced to dL axioms or valid formulas in real arithmetic, so no more subgoals remain. At any point in this process can proof state be stored and used later as a lemma (even in other proofs). This entire process is hidden from the user, who instead is presented a sequent proof.

KeYmaera X proofs appeal to uniform substitution from dL axioms [16]: for example, the test axiom $[?q]p \leftrightarrow (q \to p)$, which is an ordinary dL formula, and uniform substitution $\sigma = \{q \mapsto x>0, p \mapsto [x'=-x]x \geq 0\}$ can be used to obtain a concrete instance of this axiom during a proof as follows.

$$\text{US}\frac{[?q]p \leftrightarrow (q \to p)}{[?x>0][x'=-x]x \geq 0 \leftrightarrow (x>0 \to [x'=-x]x \geq 0)}$$

Uniform substitution is mainly used as a mechanism to instantiate axioms soundly, but through [16, Thm. 27] it is also useful to replace symbols in entire `Provables` soundly, as illustrated in Fig. 3. In this paper, we are going to exploit its application to entire `Provables` in order to implement proof features such as expanding definitions during a proof and an extended lemma mechanism that is able to bridge syntactic differences between the lemma conclusion and its application target.

3 Implicit and Explicit Proof Management by Example

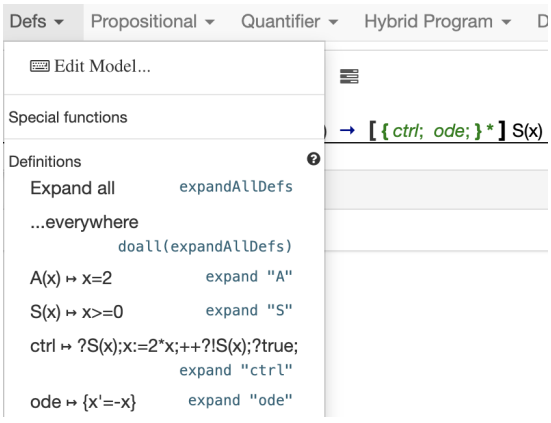
The main motivation for proof management is to allow users expand definitions and structure proofs at their discretion, as well as to enable future automated definition expansion and contraction [26]. For example, consider the KeYmaera X input file in Fig. 4a that uses predicate definitions $A(x) \equiv x=2$ to capture assumptions about initial values of x and $S(x) \equiv x \geq 0$ to describe the desired safety property, as

```
Definitions
  Bool A(Real x) <-> x=2;
  Bool S(Real x) <-> x>=0;
  HP ctrl ::= { if (S(x)) x:=2*x; }; /* ?S(x);
      ↪ x:=2*x; ++ ?!S(x);?true; */
  HP ode  ::= { {x'=-x} };
End.

ProgramVariables Real x; End.

Problem  A(x) -> [{ctrl;ode;}*]S(x) End.
```

(a) Definitions in the KeYmaera X input syntax



(b) Definitions menu

Figure 4: Predicate and program definitions in KeYmaera X

well as program definitions *ctrl*, which doubles the value of any non-negative x, and the differential equation *ode*, which models exponential decay. The definitions populate the "Defs" menu in KeYmaera X that allows users to expand definitions collectively (`expandAllDefs`) or selectively (e.g., `expand "ctrl"`) during a proof, see Fig. 4b. The menu automatically adjusts to the symbols of the currently selected subgoal (in the background in Fig. 4b).

As a safety question example, we want to answer whether repeated execution of $ctrl;ode$ keeps the value of x non-negative when started at $x = 2$. In the proof, we want control over when to expand definitions, and we want to structure the proof into a main theorem and supporting lemmas. Fig. 5 illustrates the proof steps.

$$\text{ode}\ [?],\to\text{R}\ \frac{*\ \dfrac{}{x\geq 0 \vdash [x' = -x]x \geq 0}}{\vdash [?x\geq 0][x' = -x]x \geq 0}$$

(a) Exponential decay lemma

$$\text{id using S(x)}\ \frac{*\ \dfrac{}{S(x) \vdash P, S(x)}}{\dfrac{\neg\text{L}\ \dfrac{}{S(x), \neg S(x) \vdash P}}{\dfrac{[?],\to\text{R}\ \dfrac{}{S(x) \vdash [?\neg S(x)]P}}{\to\text{R}\ \dfrac{}{\vdash S(x) \to [?\neg S(x)]P}}}}$$

(b) Unsatisfiable control guard lemma

$$\dfrac{\dfrac{\text{MR}\ \dfrac{\text{by}\ \dfrac{}{S(x) \vdash [ode]S(x)}\ (\textit{Fig. 5a}) \qquad \text{auto}\ \dfrac{*\ \dfrac{}{S(x) \vdash [?S(x);x := 2x]S(x)}}{S(x) \vdash [?S(x);x := 2x][ode]S(x)}}{[\cup],\wedge\text{R}\ \dfrac{}{S(x) \vdash [?S(x);x := 2x \cup ?\neg S(x)][ode]S(x)}} \qquad \text{by}\ \dfrac{}{S(x) \vdash [?\neg S(x)][ode]S(x)}\ (\textit{Fig. 5b})}{\text{expand}\ \dfrac{}{S(x) \vdash [ctrl][ode]S(x)}}}{[;]\ \dfrac{}{S(x) \vdash [ctrl;ode]S(x)}}$$

(c) Induction step lemma, appeals to exponential decay lemma and unsatisfiable guard lemma

$$\dfrac{\text{loop}\ \dfrac{\text{expand}\ \dfrac{\mathbb{R}\ \dfrac{*\ \dfrac{}{x = 2 \vdash x \geq 0}}{A(x) \vdash S(x)} \qquad \text{by}\ \dfrac{}{S(x) \vdash [ctrl;ode]S(x)}\ (\textit{Fig. 5c}) \qquad \text{id}\ \dfrac{*\ \dfrac{}{S(x) \vdash S(x)}}{}}{A(x) \vdash [(ctrl;ode)^*]S(x)}}{\to\text{R}\ \dfrac{}{\vdash A(x) \to [(ctrl;ode)^*]S(x)}}$$

(d) Main theorem proves loop induction base case and use case, appeals to Fig. 5c for the induction step.

Figure 5: The substitutions collected during `expand` and `by` are $\sigma = \{A(\cdot) \mapsto \cdot = 2, S(\cdot) \mapsto \cdot \geq 0, ctrl \mapsto ?S(x) \cup ?\neg S(x), ode \mapsto x' = -x, P \mapsto [ode]S(x)\}$, so the proof shows validity of the concrete formula $x = 2 \to [\big((?x \geq 0 \cup ?\neg x \geq 0); \{x' = -x\}\big)^*]x \geq 0$.

The proof proceeds from the initial conjecture $A(x) \vdash [(ctrl;ode)^*]S(x)$ bottom-to-top, with proof step justifications annotated to the left of the horizontal bars. Validity transfers top-to-bottom, so validity of the sequents (subgoals) above a horizontal bar justifies validity of the conclusion below the horizontal bar. The first step →R makes the left-hand side of the implication available as assumptions $A(x)$. Next, loop induction splits the proof into three subgoals: the base case $A(x) \vdash S(x)$, which closes by real arithmetic $\mathbb{R}$ after expanding the definitions, the use case $S(x) \vdash S(x)$ that is trivially true by id, and the induction step. The induction step in Fig. 5c first addresses the sequential composition with [;] to isolate *ctrl* from *ode*, then expands *ctrl* to split into its two cases: (i) in the left branch, the condition of

if $(S(x))$ is true (represented with $?S(x)$) and preserved by the program $?S(x);x := 2$ as witnessed by a monotonicity step MR and, thus, the exponential decay lemma applies; (ii) in the right branch with its leading test $?\neg S(x)$ the unsatisfiable control guard lemma applies.

The main proof management features used in the proof are expand to expand definitions, by to apply a lemma, and `using` to temporarily restrict reasoning to certain formulas. To users, the proof in Fig. 5 appears as if they were working on a single `Provable` and the proof steps were combined immediately. Doing so, however, would require extensive changes to the soundness-critical core and violate the local nature of its reasoning. Behind the scenes, this proof therefore requires a shift from operating on a single provable to keeping track of loosely connected sub-proofs outside the prover core; these sub-proofs fit together only after applying the substitutions collected during the proof. In the following sections, we provide details on explicit proof management that structures proofs into lemmas and implicit proof management that delays merging `Provables` and applying uniform substitutions.

4 Explicit Proof Management with Lemmas

The KeYmaera X input format allows explicit proof management in the input format for users to structure their problem descriptions into lemmas that are shared between proofs, and theorems, which appeal to lemmas to show some of their subgoals. For example, the "Exponential decay" and "Unsatisfiable control guard" lemmas from Fig. 5a and Fig. 5b are expressed in the KeYmaera X ASCII input syntax below, recorded from the steps of the interactive proofs in Fig. 5a and Fig. 5b. Optional "/" in lemma names structure the lemmas into folders on both the user interface and the file system.

```
Lemma "FIDE21/Exponential decay"
  ProgramVariables Real x; End.
  Problem x>=0 -> [{x'=-x}]x>=0 End.
  Tactic "Recorded" implyR('R=="x>=0 -> [{x'=-x}]x>=0"); ODE('R=="[{x'=-x}]x>=0") End.
  Tactic "Automated proof" auto End.
End.
Lemma "FIDE21/Unsatisfiable control guard"
  Definitions    /* constants, functions, properties, programs */
    Bool S(Real x);
    Bool P(Real x);
  End.
  ProgramVariables Real x; End.       /* variables */
  Problem S(x) -> [?!S(x);]P(x) End.  /* specification in dL */
  Tactic "Interactive proof"
    implyR('R=="S(x) -> [?!S(x);]P(x)");
    testb('R=="[?!S(x);]P(x)");
    implyR('R=="!S(x) -> P(x)");
    notL('L=="!S(x)");
    id using "S(x)"
  End.
  Tactic "Automated proof" auto End.
End.
```

The "Induction step" lemma below uses the earlier two lemmas in its proof. It follows the steps in Fig. 5c largely verbatim, but the specific lemma application steps are worth noting. Applying the "Exponential decay" lemma is straightforward by auto, since it uses $S(x)$ and *ode* in their expanded form as the only difference between the subgoal and the lemma conclusion. The "Unsatisfiable control guard"

lemma, however, introduces a new predicate symbol $P(x)$, which is not present in the induction step (nor in the original conjecture). We, therefore, use substitution $\sigma = \{P(x) \mapsto [x := x][?true][x' = -x]S(x)\}$ to tell the lemma application mechanism how to resolve $P(x)$.[1]

```
Lemma "FIDE21/Induction step"
  Definitions
    Bool S(Real x) <-> x>=0;
    HP ctrl ::= { if (S(x)) { x:=2*x; } };
    HP ode ::= { {x'=-x} };
  End.
  ProgramVariables Real x; End.
  Problem S(x) -> [ctrl;ode;]S(x) End.
  Tactic "Proof induction step"
    implyR('R=="S(x)->[ctrl;ode;]S(x)");
    composeb('R=="[ctrl;ode;]S(x)");
    expand "ctrl";
    choiceb('R=="[?S(x);x:=2*x;++?!S(x);?true;][ode;]S(x)");
    andR('R=="[?S(x);x:=2*x;][ode;]S(x) & [?!S(x);?true;][ode;]S(x)"); <(
      "[?S(x);x:=2*x;][ode;]S(x)":
        MR("S(x)", 'R=="[?S(x);x:=2*x;][ode;]S(x)"); <(
          "Use Q->P": expand "S"; auto,
          "Show [a]Q": useLemma("FIDE21/Exponential decay", "US({`S(x)~>x>=0 :: ode;~>{x
              ↪ '=-x} :: nil`});unfold;id")
        ),
      "[?!S(x);?true;][ode;]S(x)":
        composeb('R=="[?!S(x);?true;][ode{|^@|};]S(x)");
        useLemma("FIDE21/Unsatisfiable control guard", "US({`P(x)~>[x:=x;][?true;][{x'=-x
            ↪ }]S(x) :: nil`});unfold;id")
    )
  End.
End.
```

The main theorem of Fig. 5d is expressed in KeYmaera X ASCII syntax below. Its proof uses the "Induction step" lemma in a straightforward way.

```
Theorem "FIDE21/Combine lemmas"
  Definitions
    Bool A(Real x) <-> x=2;
    Bool S(Real x) <-> x>=0;
    HP ctrl ::= { if (S(x)) { x:=2*x; } };
    HP ode ::= { {x'=-x} };
  End.
  ProgramVariables Real x; End.
  Problem A(x) -> [{ctrl;ode;}*]S(x) End.
  Tactic "Interactive proof"
    implyR('R=="A(x)->[{ctrl;ode;}*]S(x)");
    loop("S(x)", 'R=="[{ctrl;ode;}*]S(x)"); <(
      "Init": expandAllDefs; QE,
      "Post": id,
      "Step": expandAllDefs; useLemma("FIDE21/Induction step", "prop")
```

[1] The leading self-assignment $x := x$ is a necessary technicality to make variable x must-bound because the differential equation $x' = -x$ may run for duration 0.

```
    )
  End.
End.
```

Structuring proofs into lemmas and theorems need not necessarily be done when creating the input file. As an alternative, the KeYmaera X user interface allows users to start lemmas from any proof state; lemma proofs remain linked from the tabs representing open subgoals in the main proof until finished. Other pre-existing lemmas can be searched and applied from the user interface as in Fig. 6. Techniques for implicit proof management and delayed substitution (used in the proofs above) are discussed next.

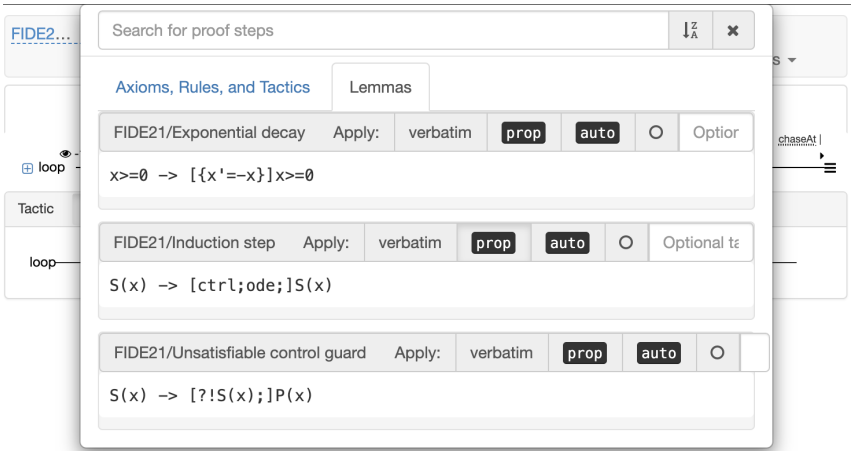


Figure 6: Applying a lemma from the lemma search dialog in KeYmaera X

5 Implicit Proof Management with Delayed Substitution

In this section, we discuss the fundamental proof management technique of delayed proof composition and delayed uniform substitution, and then devise several applications of it for enabling parametric proofs and delayed modeling, and for temporary sub-proofs focusing on some select aspects of a subgoal.

5.1 Delayed Proof Composition and Delayed Uniform Substitution

As illustrated in Section 2, the KeYmaera X core creates and modifies proof state without keeping track of the steps of the proof. KeYmaera X records proof steps outside the core with a separate `Provable` per proof step. This trace of `Provables` not only is the basis for rendering and navigating the sequent calculus proof on the user interface, but also gives us freedom to choose when to combine `Provables`. Instead of combining provables and applying uniform substitutions immediately at every step in the proof, those separate `Provables` are combined to a single proof once all the steps are finished. The advantage of delayed merging is that we can postpone the uniform effect [16, Thm. 27] of uniform substitution across subgoals of a `Provable`. Without delayed merging, symbols that are expanded on one branch would immediately be expanded uniformly across all other branches of the proof, even if those other branches would prefer to continue using symbols in their unexpanded form. The different points of expanding symbols are then reconciled in the final proof checking pass that combines `Provables`: uniform substitutions that originate from explicit user interactions (e.g., from expanding definitions) are combined with substitutions found through unification.

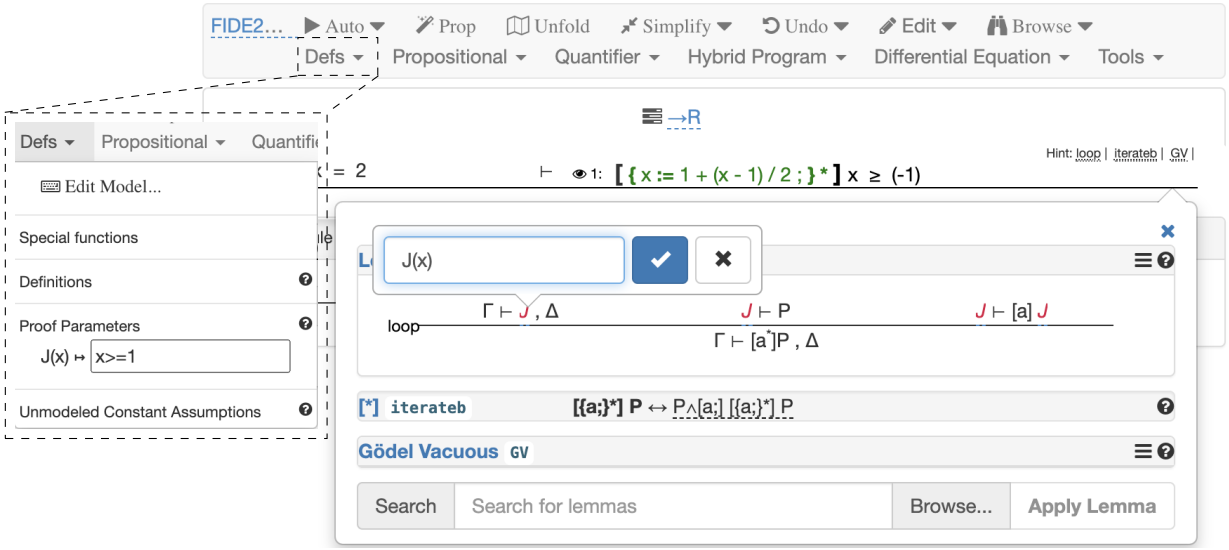
5.2 Parametric Proofs and Delayed Modeling

Theorem proving is not merely a tool to just obtain a correctness proof about an already correct system; it is a tool to explore and understand a system in all its subtleties and with all its corner cases thoroughly, to discover properties of the system that are not or only partially known, and to discover and fix correctness bugs in the process. We, therefore, frequently want to make progress in a proof without yet committing to specific inputs or even without supplying a finished model and/or conjecture. For example, we may want to analyze a loop, but do not yet know a concrete loop invariant that we could use in the proof. An obvious technique is to use loop unrolling to debug the behavior of the loop body in an attempt to manually identify a loop invariant candidate. However, this is usually not a suitable technique to prove safety of a loop, and so requires duplicate proof effort once a loop invariant candidate is identified through debugging (and perhaps several rounds of alternating debugging and proof attempts).

Parametric Proofs A powerful alternative technique are parametric proofs [16] to advance in a proof without committing to concrete inputs early in a proof.

$$\dfrac{\mathbb{R}\ \dfrac{*}{x=2\vdash x\geq 1}\ \text{expand}\quad \mathbb{R}\ \dfrac{\dfrac{*}{x\geq 1\vdash 1+\frac{x-1}{2}\geq 1}}{J(x)\vdash J(1+\frac{x-1}{2})}\ \text{expand}\quad \mathbb{R}\ \dfrac{*}{x\geq 1\vdash x\geq -1}\ \text{expand}}{x=2\vdash [(x:=1+\frac{x-1}{2})^*]x\geq -1}\ \text{loop}\ \ \dfrac{}{J(x)\vdash [x:=1+\frac{x-1}{2}]J(x)}\ [:=]$$

(a) Parametric sequent proof



(b) Starting a parametric proof in KeYmaera X by using an uninterpreted predicate symbol as tactic input; the proof parameter is then definable from a text field in the "Defs" menu.

Figure 7: Parametric loop induction with abstract invariant $J(x)$. The substitution $\sigma = \{J(\cdot) \mapsto \cdot \geq 1\}$ supplies a concrete loop invariant to close all three branches of the proof, but any other substitution $\sigma = \{J(\cdot) \mapsto \cdot \sim y\}$ with $\sim \in \{\geq, >\}$ and $y \in [-1, 1]$ would work as well.

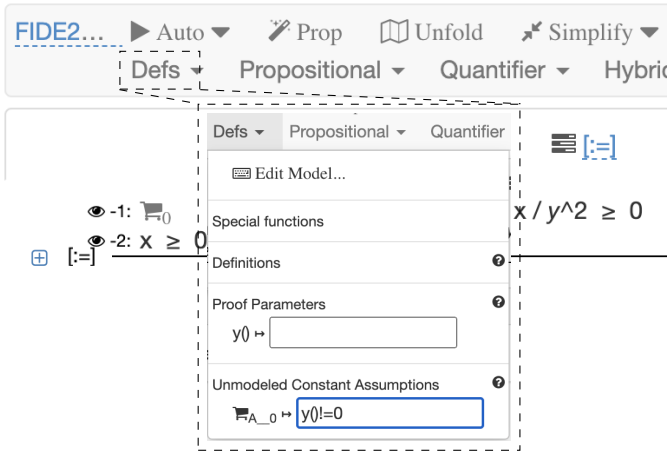
Parametric proofs allow users to proceed with abstract terms or formulas whose concrete shape is discovered later during the proof. Delayed merging of `Provables` and delayed uniform substitution allow users to supply concrete terms, formulas, and programs for function symbols, predicate symbols, and program symbols at any point in the proof, which then get automatically applied to all prior proof steps upon composition of the final `Provable`.

In Fig. 7, the loop step uses an uninterpreted predicate symbol $J(x)$ instead of a concrete formula as a loop invariant. That way, we can advance the proof on all three branches until we find that $J(x)$ simultaneously has to fit $x=2 \vdash J(x)$ in the induction base case, $J(x) \vdash J(1+\frac{x-1}{2})$ in the induction step, and $J(x) \vdash x \geq -1$ in the induction use case. At this point, we can experiment with different choices of $J(x)$ and, ultimately, settle for $\sigma = \{J(\cdot) \mapsto \cdot \geq 1\}$. Uniformly substituting this choice into the entire `Provable` concludes the proof by $\mathbb{R}$ on all branches.

Delayed Modeling Delayed merging of `Provables` and delayed substitution are also helpful to address a common nuisance in proofs: missing assumptions about model parameters (e.g., to avoid division by zero) are easily forgotten and their absence becomes apparent often only rather late in a proof. Using an arity 0 predicate symbol in the model allows users to supply missing assumptions during a proof as they are discovered, without requiring to redo the proof. For example, the proof in Fig. 8 uses an arity 0 predicate symbol $p_{\text{🛒}}$ that can be used to collect missing assumptions as they become apparent in the proof. The effect of collecting assumptions during the proof is achieved by simply augmenting the concrete assumption with another fresh $p_{\text{🛒}i}$.

$$\dfrac{\dfrac{\dfrac{*}{x \geq 0, y \neq 0 \vdash \frac{x}{y^2} \geq 0}\,\mathbb{R}}{x \geq 0, p_{\text{🛒}} \vdash \frac{x}{y^2} \geq 0}\,\text{expand}}{x \geq 0, p_{\text{🛒}} \vdash [x := \frac{x}{y^2}]x \geq 0}\,[:=]$$

(a) Predicate $p_{\text{🛒}}$ to supply assumptions during the proof. The substitution $\sigma = \{p_{\text{🛒}} \mapsto y \neq 0\}$ results in the proof showing validity of the sequent $x \geq 0, y \neq 0 \vdash [(x := \frac{x}{y^2})^*]x \geq 0$.



(b) Providing assumptions with the "Defs" menu

Figure 8: Delayed modeling by using uninterpreted predicate symbols in the input

Note that a proof parameter $J(x)$ cannot be used to introduce the missing assumption $y \neq 0$ because that fact was not even available in the original conjecture and therefore would not be provable in the base case of the induction proof. We use $p_{\text{🛒}}$ to allow limited fixing of model mistakes during the proof; in the proof in Fig. 8 it is important that $p_{\text{🛒}}$ is an arity 0 predicate symbol whose free variables do not overlap with the variables bound in the loop, so that it stays available in the induction step of the proof. The conjecture of Fig. 8 is expressed below in KeYmaera X ASCII input syntax.

```
Problem A__0() -> x>=0 -> [x:=x/y()2;]x>=0 End.
```

$$\dfrac{\dfrac{\dfrac{*}{\mathbb{R}\ \not\!\!\text{\textcircled{\scriptsize$\vee$}}x<0\vee x=0,y=x\vdash xy\le y^2}\qquad \dfrac{\dfrac{*}{\mathbb{R}\ x<0\vee x=0,y>0\vdash xy\le y^2}}{\text{WLi}\ \not\!\!\text{\textcircled{\scriptsize$\vee$}}x<0\vee x=0,y>0\vdash xy\le y^2}}{\vee\text{L}\ \not\!\!\text{\textcircled{\scriptsize$\vee$}}x<0\vee x=0,y=x\vee y>0\vdash xy\le y^2}}{\text{WL}\ x<0\vee x=0,y=x\vee y>0\vdash xy\le y^2}$$

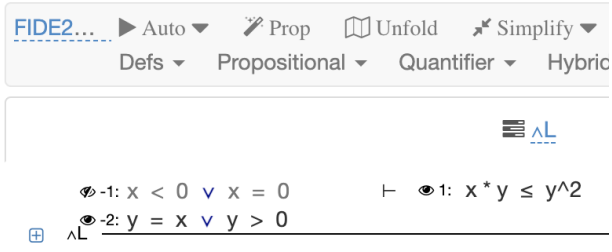
(a) Wanted effect of temporarily hiding a formula: ignore for some proof steps, re-introduce when needed

$$\dfrac{\dfrac{\dfrac{*}{\text{id}\ x=0,x=1\vdash x=0}}{\text{WLi}\ \not\!\!\text{\textcircled{\scriptsize$\vee$}}x=0,x=1\vdash x=0}}{\lightning\ \not\!\!\text{\textcircled{\scriptsize$\vee$}}x=0\vdash[x:=1]x=0}}{\text{WL}\ x=0\vdash[x:=1]x=0}$$

(b) Unsoundly ignoring temporarily hidden formula

$$\dfrac{\dfrac{x_0=0,x=1\vdash x=0}{\circlearrowleft\text{US}\ P_{\not\!\!\text{\textcircled{\scriptsize$\vee$}}}(x_0),x=1\vdash x=0}}{[:=]\ P_{\not\!\!\text{\textcircled{\scriptsize$\vee$}}}(x)\vdash[x:=1]x=0}}{\circlearrowleft\text{US}\ x=0\vdash[x:=1]x=0}$$

(c) Sub-proof with substitution $\sigma=\{P_{\not\!\!\text{\textcircled{\scriptsize$\vee$}}}(\cdot)\mapsto\cdot=0\}$



(d) Hiding formulas temporarily on the user interface

Figure 9: Sub-proof allows hiding explicit formula structure to temporarily focus tactic application on relevant formulas without sacrificing soundness.

5.3 Temporary Implicit Sub-Proofs with Select Formulas

Applying tactics, axioms, and proof rules has permanent effect on the proof state. For example, weakening assumptions permanently removes formulas from the proof state; if weakening is done for convenience to focus on specific aspects of the proof, we cannot undo the effect of weakening when the hidden formulas become useful again later in the proof. Not focusing, however, is not an option either, because the mere presence of additional assumptions and formulas may result in duplicate proof effort or intractable proofs (e.g., when applying real arithmetic decision procedures with non-trivial complexity).

We want to keep temporary operations separate from the soundness-critical prover core, because their effect is not compatible with its local isolated reasoning principles. It would be unsound to temporarily exclude formulas so that they are not affected by tactic applications. Fig. 9a illustrates an example with the wanted effect of temporarily hiding formulas, but Fig. 9b shows that care needs to be taken to not unsoundly exclude formulas temporarily from being affected by, e.g., the assignment axiom. In order to fix Fig. 9b, the prover core would need to know which axiom or rule can soundly ignore temporarily hidden facts under what conditions. With an implicit sub-proof as in Fig. 9c we can temporarily focus tactic application on some proof aspects without extending the KeYmaera X core or sacrificing soundness. As an additional benefit, the abbreviations $P_{\not\!\!\text{\textcircled{\scriptsize$\vee$}}}(x)$ have simpler structure than the fully expanded formulas, which makes tactic applications less expensive even if they have to operate on $P_{\not\!\!\text{\textcircled{\scriptsize$\vee$}}}(x)$.

In tactics, temporarily focusing on a subset of the sequent formulas is supported with the notation `using`. For example, the proof of Fig. 9a is expressed as follows:

```
(orL('L)*; <(QE, skip)) using "y=x|y>0 :: x*y<=y^2 :: nil"; QE
```

This script advances the proof fully on the left branch, but postpones the final $\mathbb{R}$ (`QE`) step of the right branch using `skip` until after the `using` block.

6 Conclusion

Uniform substitution in hybrid systems is a powerful technique for implementing hybrid systems theorem provers in an LCF-style approach. We illustrated how several proof management features can be implemented on top of uniform substitution outside the soundness-critical core of a theorem prover by following certain modeling conventions and corresponding treatment in the user interface.

This approach of using modeling conventions and making proof steps implicit through other user interactions sits somewhat between auto-active verifiers and full interactive theorem proving. Auto-active verifiers, such as Dafny [6, 7] and AutoProof [23] hide verification and interaction with the verification tool entirely behind annotations in the analyzed code. Interactive theorem provers, such as Coq [2] and Isabelle/HOL [14], primarily interact with users through scripts, such as structured proofs in Isabelle/Isar [13] and hide only little of the proof complexity behind other means of presentation even in advanced editors [25] or when proofs are found automatically, e.g., with Sledgehammer [15]. Many (hybrid systems) theorem provers (e.g., [1, 18, 20, 24]) opt for implementing their proof calculus using axiom schemata or with trusted rules, which renders the features presented here soundness-critical.

For future work, we plan to automate unification steps in applying lemmas to bridge the syntactic differences between lemma conclusion and target subgoal, and seek to exploit uniform substitution for further proof techniques.

References

[1] Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Richard Bubel, Martin Giese, Reiner Hähnle, Wolfram Menzel, Wojciech Mostowski, Andreas Roth, Steffen Schlager & Peter H. Schmitt (2005): *The KeY Tool*. *Software and System Modeling* 4(1), pp. 32–54, doi:10.1007/s10270-004-0058-x.

[2] Yves Bertot & Pierre Castéran (2004): *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series, Springer, doi:10.1007/978-3-662-07964-5.

[3] Brandon Bohrer, Yong Kiam Tan, Stefan Mitsch, Magnus O. Myreen & André Platzer (2018): *VeriPhy: Verified Controller Executables from Verified Cyber-Physical System Models*. In Dan Grossman, editor: *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018*, ACM, pp. 617–630, doi:10.1145/3192366.3192406.

[4] Nathan Fulton, Stefan Mitsch, Brandon Bohrer & André Platzer (2017): *Bellerophon: Tactical Theorem Proving for Hybrid Systems*. In Mauricio Ayala-Rincón & César A. Muñoz, editors: *ITP*, *LNCS* 10499, Springer, pp. 207–224, doi:10.1007/978-3-319-66107-0_14.

[5] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp & André Platzer (2015): *KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems*. In Amy Felty & Aart Middeldorp, editors: *CADE*, *LNCS* 9195, Springer, Berlin, pp. 527–538, doi:10.1007/978-3-319-21401-6_36.

[6] K. Rustan M. Leino (2013): *Developing verified programs with Dafny*. In David Notkin, Betty H. C. Cheng & Klaus Pohl, editors: *35th Int. Conf. on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, IEEE Computer Soc., pp. 1488–1490, doi:10.1109/ICSE.2013.6606754.

[7] K. Rustan M. Leino & Valentin Wüstholz (2014): *The Dafny Integrated Development Environment*. In Catherine Dubois, Dimitra Giannakopoulou & Dominique Méry, editors: *Proceedings 1st Workshop on Formal Integrated Development Environment, F-IDE 2014, Grenoble, France, April 6, 2014.*, *EPTCS* 149, pp. 3–15, doi:10.4204/EPTCS.149.2.

[8] Jiang Liu, Naijun Zhan & Hengjun Zhao (2011): *Computing semi-algebraic invariants for polynomial dynamical systems*. In: *EMSOFT 2011*, ACM, New York, NY, USA, pp. 97–106.

[9] Stefan Mitsch & André Platzer (2016): *The KeYmaera X proof IDE: Concepts on usability in hybrid systems theorem proving*. In Catherine Dubois, Paolo Masci & Dominique Méry, editors: *3rd Workshop on Formal Integrated Development Environment*, *EPTCS* 240, pp. 67–81, doi:10.4204/EPTCS.240.5.

[10] Stefan Mitsch & André Platzer (2016): *ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models*. *Form. Methods Syst. Des.* 49(1-2), pp. 33–74, doi:10.1007/s10703-016-0241-z. Special issue of selected papers from RV'14.

[11] Stefan Mitsch & André Platzer (2020): *A Retrospective on Developing Hybrid Systems Provers in the KeYmaera Family - A Tale of Three Provers*. In Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, Reiner Hähnle & Matthias Ulbrich, editors: *Deductive Software Verification: Future Perspectives - Reflections on the Occasion of 20 Years of KeY*, *LNCS* 12345, Springer, pp. 21–64, doi:10.1007/978-3-030-64354-6_2.

[12] Andreas Müller, Stefan Mitsch, Werner Retschitzegger, Wieland Schwinger & André Platzer (2018): *Tactical Contract Composition for Hybrid System Component Verification*. *STTT* 20(6), pp. 615–643, doi:10.1007/s10009-018-0502-9. Special issue for selected papers from FASE'17.

[13] Tobias Nipkow (2002): *Structured Proofs in Isar/HOL*. In Herman Geuvers & Freek Wiedijk, editors: *Types for Proofs and Programs, 2nd Int. Workshop, TYPES 2002, Berg en Dal, The Netherlands, April 24-28, 2002, Selected Papers*, *LNCS* 2646, Springer, pp. 259–278, doi:10.1007/3-540-39185-1_15.

[14] Tobias Nipkow, Lawrence C. Paulson & Markus Wenzel (2002): *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. *LNCS* 2283, Springer.

[15] Lawrence C. Paulson (2010): *Three Years of Experience with Sledgehammer, a Practical Link between Automatic and Interactive Theorem Provers*. In Renate A. Schmidt, Stephan Schulz & Boris Konev, editors: *Proceedings of the 2nd Workshop on Practical Aspects of Automated Reasoning, PAAR-2010, Edinburgh, Scotland, UK, July 14, 2010*, *EPiC Series* 9, EasyChair, pp. 1–10.

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

[17] André Platzer (2018): *Logical Foundations of Cyber-Physical Systems*. Springer, Cham, doi:10.1007/978-3-319-63588-0.

[18] André Platzer & Jan-David Quesel (2008): *KeYmaera: A Hybrid Theorem Prover for Hybrid Systems*. In Alessandro Armando, Peter Baumgartner & Gilles Dowek, editors: *IJCAR*, *LNCS* 5195, Springer, Berlin, pp. 171–178, doi:10.1007/978-3-540-71070-7_15.

[19] André Platzer & Yong Kiam Tan (2020): *Differential Equation Invariance Axiomatization*. *J. ACM* 67(1), pp. 6:1–6:66, doi:10.1145/3380825.

[20] David W. Renshaw, Sarah M. Loos & André Platzer (2011): *Distributed Theorem Proving for Distributed Hybrid Systems*. In Shengchao Qin & Zongyan Qiu, editors: *ICFEM*, *LNCS* 6991, Springer, pp. 356–371, doi:10.1007/978-3-642-24559-6_25.

[21] Andrew Sogokon, Stefan Mitsch, Yong Kiam Tan, Katherine Cordwell & André Platzer: *Pegasus: Sound Continuous Invariant Generation*. *Form. Methods Syst. Des.*, doi:10.1007/s10703-020-00355-z. Special issue for selected papers from FM'19.

[22] Yong Kiam Tan & André Platzer (2021): *Deductive Stability Proofs for Ordinary Differential Equations*. In Jan Friso Groote & Kim G. Larsen, editors: *Tools and Algorithms for the Construction and Analysis of*

Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Proceedings, *LNCS* 12652, Springer, doi:10.1007/978-3-030-72013-1_10.

[23] Julian Tschannen, Carlo A. Furia, Martin Nordio & Nadia Polikarpova (2015): *AutoProof: Auto-Active Functional Verification of Object-Oriented Programs*. In Christel Baier & Cesare Tinelli, editors: *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, London, UK, April 11-18, 2015. Proceedings*, *LNCS* 9035, Springer, pp. 566–580, doi:10.1007/978-3-662-46681-0.

[24] Shuling Wang, Naijun Zhan & Liang Zou (2015): *An Improved HHL Prover: An Interactive Theorem Prover for Hybrid Systems*. In: *Formal Methods and Software Engineering*, Springer, pp. 382–399.

[25] Makarius Wenzel (2012): *Isabelle/jEdit - A Prover IDE within the PIDE Framework*. In Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel & Volker Sorge, editors: *Intelligent Computer Mathematics - 11th International Conference, AISC 2012, 19th Symp., Calculemus 2012, 5th Int. Workshop, DML 2012, 11th Int. Conf., MKM 2012, Systems and Projects, Held as Part of CICM 2012, Bremen, Germany, July 8-13, 2012. Proc.*, *LNCS* 7362, Springer, pp. 468–471, doi:10.1007/978-3-642-31374-5.

[26] Larry Wos (1987): *The Problem of Definition Expansion and Contraction*. *J. Autom. Reason.* 3(4), pp. 433–435, doi:10.1007/BF00247438.