The KeYmaera X Proof IDE

Concepts on Usability in Hybrid Systems Theorem Proving

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

André Platzer
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
aplatzer@cs.cmu.edu

Hybrid systems verification is quite important for developing correct controllers for physical systems, but is also challenging. Verification engineers, thus, need to be empowered with ways of guiding hybrid systems verification while receiving as much help from automation as possible. Due to undecidability, verification tools need sufficient means for intervening during the verification and need to allow verification engineers to provide system design insights.

This paper presents the design ideas behind the user interface for the hybrid systems theorem prover KeYmaera X. We discuss how they make it easier to prove hybrid systems as well as help learn how to conduct proofs in the first place. Unsurprisingly, the most difficult user interface challenges come from the desire to integrate automation and human guidance. We also share thoughts how the success of such a user interface design could be evaluated and anecdotal observations about it.

1 Introduction

Cyber-physical systems such as cars, aircraft, and robots combine computation and physics, and provide exceedingly interesting and important verification challenges. KeYmaera X [11] is a theorem prover for *hybrid systems*, i. e., systems with interacting discrete and continuous dynamics, which arise in virtually all mathematical models of cyber-physical systems.[1] It implements *differential dynamic logic* ($\mathsf{d}\mathcal{L}$ [23, 24, 27]) for *hybrid programs*, a program notation for hybrid systems. Differential dynamic logic provides compositional techniques for proving properties about hybrid systems. Despite the substantial advances in automation, user input is often quite important, since hybrid systems verification is not semidecidable [23]. Human insight is needed most notably for finding invariants for loop induction and finding differential invariants for unsolvable differential equations [24]. But even some of the perfectly decidable questions in hybrid systems verification are intractable in practice, such as the final step of checking the validity of formulas in real arithmetic[2] in a $\mathsf{d}\mathcal{L}$ proof.

To overcome those verification challenges, KeYmaera X combines automation and interaction capabilities to enable users to verify their applications even if they are still out of reach for state-of-the-art automation techniques. The central question in usability, thus, is how interaction and automation can jointly solve verification challenges. For isolated strategic aspects of the proofs, such user guidance is easily separated, for example when using system insights to provide loop invariants and differential invariants. Other aspects of human insights are more invasive, such as picking and transforming relevant

[1] KeYmaera X is available at `http://keymaeraX.org/`

[2] Decision procedures are doubly exponential in the number of quantifier alternations [9], and practical implementations doubly exponential in the number of variables.

formulas to make intractable arithmetic tractable. Users have to link the logical level of proving (e. g., the conjecture, available proof rules and axioms) to the abstract interaction level (e. g., visualization of formulas in a sequent) and decide about concrete interaction steps (e. g., where to click, what to type) [3]. Hence, going back and forth between automated proof tactics and user guidance poses several challenges:

- Users need to be provided with a way of understanding the proof state produced by automated tactics. What are the open proof goals? How are these goals related to the proof of the conclusion? And why did the automated tactic stop making progress?
- Users need to be provided with efficient ways of understanding the options for making progress. What tactics are available and where can they be applied? What input is needed? And what interactions provide genuinely new insights into a proof that the automation would not have tried?
- Users need efficient tools for executing proof steps. How to provide gradual progress for novices? How to let experienced users operate with minimal input? How to reuse proof steps across similar goals? And how to generalize a specific tactic script into a proof search procedure for similar problems?

Users may additionally benefit from picking an appropriate interaction paradigms, such as *proof as programming*, *proof by pointing*, or *proof by structure editing* [3].

This paper discusses how KeYmaera X addresses these challenges through its web-based user interface. The complexity of larger hybrid systems verification challenges require that users be granted significant control over how a proof is conducted and what heuristics are applied for proof search. To this end, KeYmaera X separates user interaction and proof search from the actual proof steps in the prover kernel to ensure soundness while providing reasoning flexibility [11]. All proof steps follow from a small set of axioms by uniform substitution [26, 27]. This paper further introduces an evaluation concept to determine how effectively alternative user interaction concepts implemented in KeYmaera X address these challenges, as well as whether or not the combination of these concepts compare favorably to our previous hybrid systems theorem prover KeYmaera [28]. The experiments are yet to be conducted; our reports on the effectiveness of the user interaction concept remain anecdotal based on feedback from external users and students.

2 Preliminaries: Differential Dynamic Logic

This section recalls the syntax and semantics of differential dynamic logic by example of the motion of a person on an escalator. This example serves for illustrating prover interaction throughout the paper.

Syntax and semantics by example. Suppose a person is standing on an escalator, which moves upwards with non-negative speed $v{\geq}0$, so the person's vertical position x follows the differential equation $x' = v$. If the person is not at the bottom-most step ($?x{>}1$), she may step down one step ($x := x - 1$) or may just continue moving upwards; since she may or may not step down, even if allowed, we use a non-deterministic choice ($\cup$). We want to prove that the person never falls off the bottom end of the escalator ($x{\geq}0$) when stepping down and moving upwards are repeated arbitrarily often (modeled with the repetition operator *).

$$\underbrace{x \geq 2 \wedge v \geq 0 \rightarrow}_{\text{initial conditions}} [\underbrace{\big((?x > 1; x := x - 1) \cup x' = v\big)^*}_{\text{hybrid program}}]\underbrace{x \geq 0}_{\text{safety condition}} \tag{1}$$

The formula (1) captures this example as a safety property in d$\mathscr{L}$. Suppose, the person is initially at some position $x \geq 2$ when the escalator turns on. From any state satisfying the initial conditions

$x \geq 2 \wedge v \geq 0$, all runs of the hybrid program $((?x > 1; x := x - 1) \cup x' = v)^*$ reach only states that satisfy the safety condition $x \geq 0$. More detailed examples on modeling hybrid systems are in [29].

Manual proof in d$\mathscr{L}$. Formulas in d$\mathscr{L}$, such as the simple example (1), can be proved with the d$\mathscr{L}$ proof calculus. Proofs in d$\mathscr{L}$ are sequent proofs: a sequent has the shape $\Gamma \vdash \Delta$, where we assume all formulas Γ in the antecedent (to the left of the turnstile $\vdash$) to show any of the formulas Δ in the succedent (right of the turnstile). The sequent notation works from the desired conclusion at the bottom toward the resulting subgoals at the top. While sometimes surprising for novices, this notation emphasizes how the truth of the conclusions follows from the truth of their respective premises top-down. This notation also highlights the current subquestion at the very top of the deduction. Steps in the sequent proof are visualized through a horizontal line, which separates the conclusion at the bottom from the premises at the top. The name of the deduction step is annotated to the left of the horizontal line. For example, the proof rule $[\cup]$ says that to conclude safety of a non-deterministic choice of programs $\alpha \cup \beta$ (below the bar) it suffices to prove safety of both α and β individually (above bar).

$$[\cup]\,\dfrac{\phi \vdash [\alpha]\psi \wedge [\beta]\psi}{\phi \vdash [\alpha \cup \beta]\psi}$$

Manual proof example. Starting from the formula (1) at the very bottom of the deduction, we develop a safety proof as follows. As first step, the rule $\rightarrow$R moves the assumptions from the left-hand side of an implication into the antecedent. Next $\wedge$L splits the conjunction $x \geq 2 \wedge v \geq 0$ into individual facts (i. e., we get to assume both, $x \geq 2$ and $v \geq 0$ individually). Then, we use loop induction with the invariant $x > 0$. We have to show three cases: the loop invariant must hold initially (base case $x \geq 2 \rightarrow x > 0$), it must be strong enough to entail safety (use case $x > 0 \rightarrow x \geq 0$), and it must be preserved by the loop body (induction step $[(?x > 1; x := x - 1) \cup x' = v]x > 0$).

$$[\cup]\,\dfrac{\wedge\mathsf{R}\,\dfrac{[;]\,\dfrac{[?],\!\rightarrow\!\mathsf{R}\,\dfrac{[:=]\,\dfrac{\mathsf{QE}\,\dfrac{*}{x>0, v\geq0, x>1 \vdash x-1>0}}{x>0, v\geq0, x>1 \vdash [x := x-1]x>0}}{x>0, v\geq0 \vdash [?x>1][x := x-1]x>0}}{x>0, v\geq0 \vdash [?x>1; x := x-1]x>0} \qquad \mathsf{ODE}\,\dfrac{\mathsf{QE}\,\dfrac{*}{x_0>0, v\geq0 \vdash \forall t\geq0\, \forall 0\leq s\leq t\ (x = x_0 + vs \rightarrow x>0)}}{x>0, v\geq0 \vdash [x' = v]x>0}}{x>0, v\geq0 \vdash [?x>1; x := x-1]x>0 \wedge [x' = v]x>0}}{\ldots}$$

$$\rightarrow\!\mathsf{R}\,\dfrac{\wedge\mathsf{L}\,\dfrac{\mathsf{loop}\,\dfrac{\mathsf{QE}\,\dfrac{\text{(base case)}\ *}{x\geq2, v\geq0 \vdash x>0} \qquad \mathsf{QE}\,\dfrac{\text{(use case)}\ *}{x>0 \vdash x\geq0} \qquad \dfrac{\nwarrow\text{(induction step)}}{x>0, v\geq0 \vdash [(?x>1; x := x-1) \cup x' = v)]x>0}}{x\geq2, v\geq0 \vdash [((?x>1; x := x-1) \cup x' = v)^*]x\geq0}}{x\geq2 \wedge v\geq0 \vdash [((?x>1; x := x-1) \cup x' = v)^*]x\geq0}}{\vdash x\geq2 \wedge v\geq0 \rightarrow [((?x>1; x := x-1) \cup x' = v)^*]x\geq0}$$

Here, the base case and use case can be shown easily using quantifier elimination QE. The induction step proceeds using the proof rule $[\cup]$ for non-deterministic choice that we saw above, followed by $\wedge$R to split the induction step proof into two branches. On the first branch, we first turn the sequential composition (;) into nested boxes ($[?x > 1][x := x - 1]x > 0$), and then use the test condition ($x > 1$) as an additional assumption using rule [?] followed by $\rightarrow$ R. We show safety of the assignment $x := x - 1$ using quantifier elimination QE to close the proof. On the second branch, we show safety of the differential equation $x' = v$ using the proof rule ODE followed by QE. □

3 KeYmaera X Proof Automation

As a basis for understanding how KeYmaera X searches for proofs and where and why it asks for user guidance, this section gives a high-level explanation of KeYmaera X tactics.

KeYmaera X automates the tedious task of proving steps that follow unambiguously from the structure of the conjecture. It further provides (heuristic) tactics to generate and explore invariant candidates for loop induction and differential equations. One might imagine KeYmaera X to try to solve differential equations and use the solution to guide a differential invariant proof, before it resorts to more involved differential invariant proofs. KeYmaera X provides proof tactics for propositional reasoning, reasoning about hybrid programs, and closing (arithmetic) proof goals, which are combined into a fully automated proof search tactic. For example, with a loop invariant candidate annotated in the KeYmaera X input file, the running example in this paper proves fully automated.

Propositional reasoning prop and program unfolding unfold of hybrid programs follows along propositional sequent rules and the axioms of $\mathsf{d}\mathcal{L}$. These tactics successively match on the shape of a formula to transform it into simpler parts, before the tactics descend into the resulting parts. Program unfolding focuses on the decidable fragment of reasoning about hybrid programs: it stops and asks for user guidance when it encounters loops or ODEs. The following proof snippet applies unfold to just the induction step of the escalator proof.

Induction step by unfold. The tactic unfold applies hybrid program axioms and splits conjunctions in the succedent into proof branches, but stops when it encounters loops or ODEs. The resulting two subgoals correspond to the two (logically unfolded) paths through our running example: we have to show safety of the discrete assignment $x := x - 1$ as well as of the differential equation $x' = v$.

$$\text{unfold}\,\frac{x{>}0, v{\geq}0, x{>}1 \vdash x-1 \geq 0 \qquad x{>}0, v{\geq}0 \vdash [x' = v]x{>}0}{x{>}0, v{\geq}0 \vdash [?x{>}1; x := x-1 \cup x' = v]x{>}0}$$

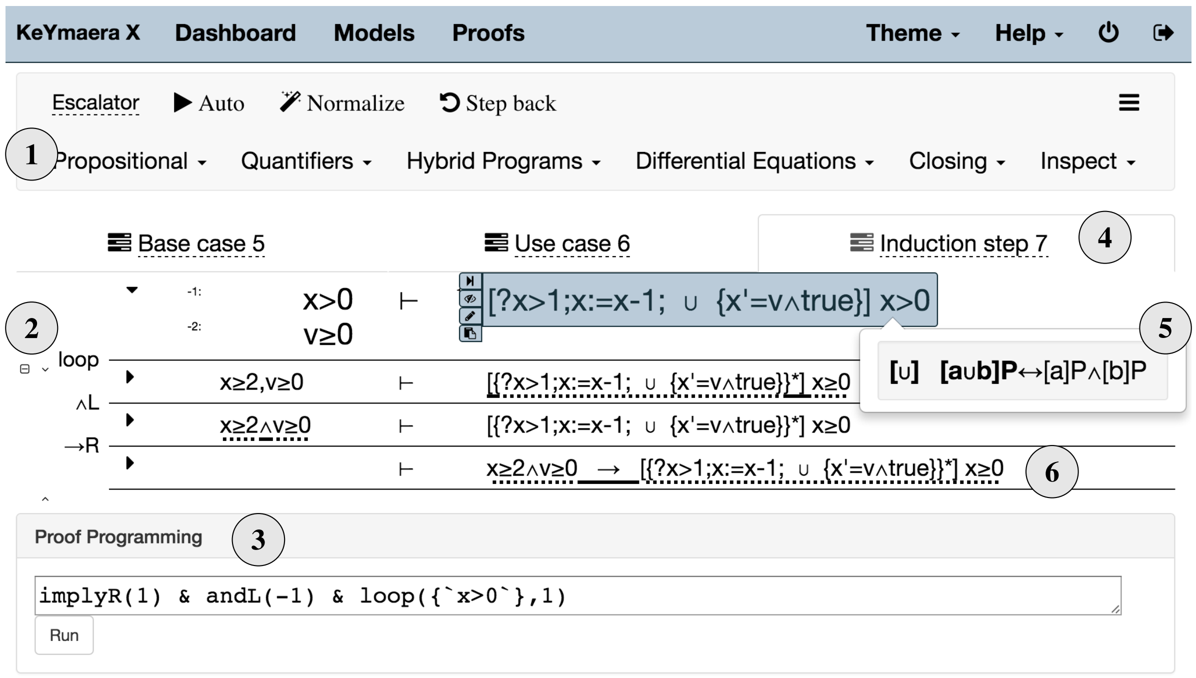
□

KeYmaera X ships with proof search tactic auto, which combines propositional reasoning with program unfolding, loop invariant exploration, certain automated proof techniques for differential equations, and proof closing by quantifier elimination. Even though the auto tactic finds proofs for important classes of hybrid systems automatically, it still may stop exploration and ask for user guidance in complicated cases (e. g., when none of the explored differential invariants helps closing the proof).

4 KeYmaera X User Interaction

When the automated tactics shipped with KeYmaera X fail to find a proof (due to a wrong model, missing loop or differential invariants, or intractable arithmetic), user interaction is needed to improve the model and make progress with the proof. This section introduces the KeYmaera X user interaction concepts and their implementation in a graphical web-based user interface. The user interface of KeYmaera X is based on these principles and hypotheses:

Familiarity Prover user interfaces benefit from a familiar look&feel that resembles how proofs are conducted in theoretical developments and that are compatible with the way that proof rules are presented.



```
implyR(1) & andL(-1) & loop({`x>0`},1)
```

Figure 1: Screenshot of KeYmaera X with annotated user interface elements: ① proof-by-search; ② sequent view; ③ proof programming and tactic extraction; ④ proof branch; ⑤ tactic suggestion; ⑥ tactic step highlighting.

Traceability Sophisticated verification challenges, especially in hybrid systems, need a way of mixing automation with user guidance in a way that the user can trace and understand the respective remaining questions.

Tutoring Interactive proof-by-pointing at formulas and terms are an efficient way of learning how to conduct proofs and help internalizing reasoning principles by observation. Tactic recording is an efficient way of learning how to write tactics by observing to which interactive proof steps they correspond.

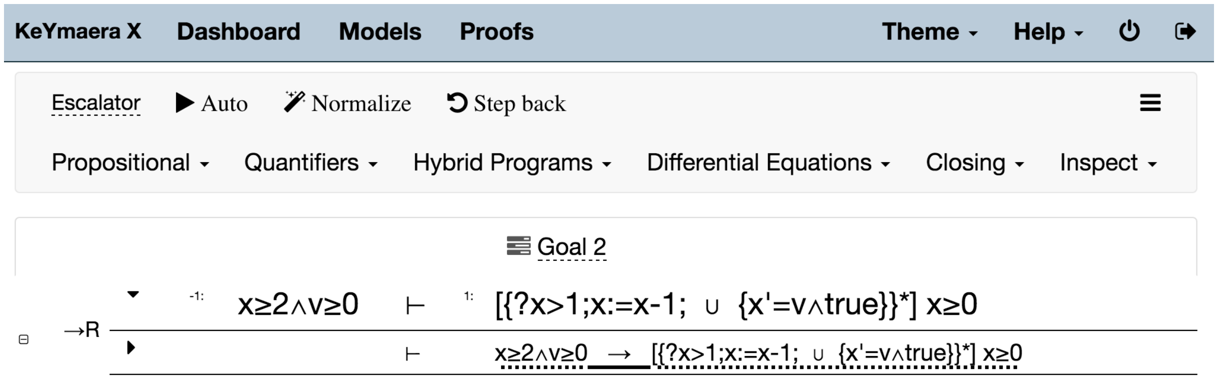
Flexibility Humans reason in more flexible ways than automation procedures. User interfaces should allow proof steps at any part of a proof as well as free-style transformations of formulas, and they should embrace multiple reasoning styles, such as explicit proofs, proof-by-pointing, and proof-by-search.

Experimentation A strict separation of prover core and prover interface not only helps soundness, but also enables more agile experimentation with new styles of conducting proofs and of interacting with provers.

Figure 1 shows a screenshot of the KeYmaera X user interface with user interface elements annotated by their interaction purpose. We discuss these design choices in more detail in the following paragraphs.

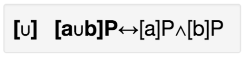
4.1 Familiarity

The KeYmaera X prover kernel implements Hilbert-style proofs by uniform substitution from a small set of locally sound axioms [27] together with a first-order sequent calculus [23]. The user interface of

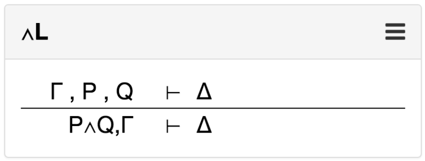


-1: x≥2∧v≥0 ⊢ 1: [{?x>1;x:=x-1; ∪ {x'=v∧true}}*] x≥0
→R
⊢ x≥2∧v≥0 → [{?x>1;x:=x-1; ∪ {x'=v∧true}}*] x≥0

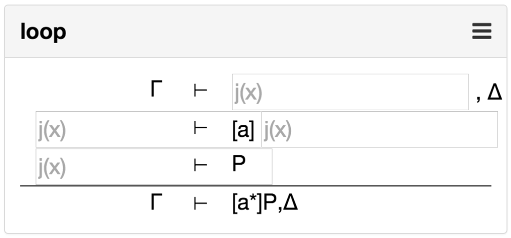
(a) Sequent proof with step $\rightarrow$R; the position where $\rightarrow$R was applied in the original formula (below the horizontal sequent line) is highlighted with a dotted line, the specific operator with a solid line.


[∪] [a∪b]P↔[a]P∧[b]P

(b) Axiom [$\cup$]


∧L
Γ, P, Q ⊢ Δ
P∧Q,Γ ⊢ Δ

(c) Sequent rule $\wedge$L


loop
Γ ⊢ j(x) , Δ
j(x) ⊢ [a] j(x)
j(x) ⊢ P
Γ ⊢ [a*]P,Δ

(d) Sequent rule loop

Figure 2: Sequent proof, axioms, and proof rules rendered with standard notation.

KeYmaera X presents proofs in sequent form as in Figure 2, which enables users to equivalently read the logical transformations as either sequent proof rule uses [23] or axiom uses [27]. The rendering is consistent with the notation used in Section 2 and in the *Foundations of Cyber-Physical Systems* course [25], which should make it easy to switch between proof development on paper and proving in KeYmaera X. Proof suggestions for tactics are rendered by their primary nature either as axioms (e. g., the d$\mathscr{L}$ axiom [$\cup$], see Figure 2b), proof rules (e. g., the propositional proof rule $\wedge$L, see Figure 2c), or proof rules with input (e. g., the sequent proof rule loop, which requires input, combines multiple axioms in a tactic to implement a proof rule, and creates multiple subgoals, see Figure 2d).

The hypothesis is that the ability to work from a small number of reasoning principles, which is crucial for a small prover core, helps human understanding as well. The experience with the *Foundations of Cyber-Physical Systems* course [25], in which KeYmaera and KeYmaera X have been used, suggests that equivalence axioms indeed make it easier for students to understand reasoning principles than explicit sequent proof rules [23], which obscure inherent dualities for novices. For example, the equivalence $[a\cup b]P \leftrightarrow [a]P \wedge [b]P$ characterizes nondeterministic choices under all circumstances. Its conjunction $\wedge$ and the rules for $\wedge$ make it apparent that such nondeterministic choices will branch in the succedent but not in the antecedent:

$$\text{[}\cup\text{]},\wedge\text{R}\,\frac{\Gamma\vdash [a]P,\Delta \quad \Gamma\vdash [b]P,\Delta}{\Gamma\vdash [a\cup b]P,\Delta} \qquad \text{[}\cup\text{]},\wedge\text{L}\,\frac{\Gamma,[a]P,[b]P\vdash\Delta}{\Gamma,[a\cup b]P\vdash\Delta}$$

Of course, it is exactly the same reasoning principle either way, but understanding the direct sequent proof rules still requires two logical principles at once compared to the single axiom.

4.2 Traceability

When switching from automated mode to user guidance, it is important to visualize just enough contextual information about the open proof goals to understand where the present subgoals came from and how they relate to the proof of the ultimate conclusion. KeYmaera X shows local views of proofs that illustrate the way how the current question came about and how it is related to the proof of the conclusion. The idea is that this enables traceability and gives local justifications while limiting information to the presently relevant part of the proof.

Visualizing the proof state. Visualizing the entire proof tree that unfolds from the original conjecture as tree root is not a viable option, since proofs in $\mathsf{d}\mathcal{L}$ unfold into many branches and therefore easily exceed the screen width when rendered in a single tree. Even simple models, such as the escalator example with only four branches (induction base case, induction use case, and stepping down plus moving upwards in the induction step) become hard to navigate and keep track of when both horizontal and vertical scrolling is needed. KeYmaera X, therefore, renders a proof tree in sequent deduction paths from the tree root representing the original conjecture at the bottom of the screen to a leave representing an open goal at the top, see Figure 1.

The sequent deduction paths are arranged in tabs; branching occurs when a deduction step has more than one premise, so that premises are spread over multiple tabs and interconnected with links between the tabs.

Proof navigation. Users can focus solely on the open goal by collapsing the entire deduction path (⊟), or they can keep some part of the proof structure visible, e. g., by collapsing only between branching points in the proof, see Figure 1. Triangles left of the sequent path identify groups: when uncollapsed, down/up arrows ($\asymp$) indicate the group borders; when collapsed ($\langle\rangle$), the steps in a group are abbreviated into "...". Additionally, sequents can be expanded over multiple lines (▼, one formula per line) or collapsed into a single line (▶).

When proof automation hands over to the user, the topmost line in a sequent deduction path represents an open goal. This means that either the goal cannot be proved at all because the model is wrong, or it is not yet proved because user guidance is needed. In the former case, the counterexample tool allows users to find concrete values that make all formulas on the left of the sequent true but violate all formulas on the right. In the latter case, users are interested in what to do next (see Section 4.3).

4.3 Tutoring

Suggesting possible proof steps. KeYmaera X analyzes the shape of formulas to suggest proof steps on demand. A tactic comes with a description of the shape of its conclusion (must match the current open goal so that the tactic is applicable), a description of the premises that remain to be proved after applying the tactic, and a description of required user input (such as invariants for loops). Such meta-information allows tactic suggestion in two different flavors, as depicted in Figure 3:

- When users know where to continue with the proof (i. e., exactly on which formula or term or part thereof), KeYmaera X displays a dialog with important applicable tactics and their required input, which resembles the user interaction of KeY [1, 2] and its hybrid systems descendant KeYmaera [28].
- When users know what to do next, KeYmaera X searches for formulas or terms where the desired tactic is applicable.

For example, when pointing to a loop, KeYmaera X suggests the induction tactic loop as well as loop unrolling $[^*]$ as shown in Figure 3. The tactic loop requires a loop invariant $j(x)$ as input and produces three branches that remain to be proved to conclude safety of the program with the loop. Loop unrolling $[^*]$ turns a formula of the shape in boldface (left-hand side $[\alpha^*]P$ of the equivalence) into a formula of the right-hand side $P \wedge [\alpha][\alpha^*]P$ of the equivalence.

For reasons of flexibility, KeYmaera X supports contextual reasoning to apply axioms deeply nested inside a formula, not only on the top-level operator as in Figure 3. As a consequence, users may have many options to work on a single formula. In the induction step of the escalator example

$$\vdash x>0 \rightarrow [(?x>1; x:=x-1) \cup x' = v]x>0$$

we used $\rightarrow$R top-level to get $x>0 \vdash [(?x>1; x:=x-1) \cup x' = v]x>0$ as a next goal. Alternatively, the non-deterministic choice tactic $[\cup]$ on the right-hand side of the implication would result in a conjunction $\vdash x>0 \rightarrow [?x>1; x:=x-1]x>0 \wedge [x' = v]x>0$.

For novice users, it is often easiest to focus on the top-level operator and work on formulas outside-in, i. e., apply $\rightarrow$R first and then $[\cup]$ next. But it quickly becomes more convenient to apply proof rules in any order anywhere in the middle of the formulas to follow whatever line of thought the user may have in mind. Such proofs in arbitrary order also often reduce the branching or repetition of proof steps in different branches.

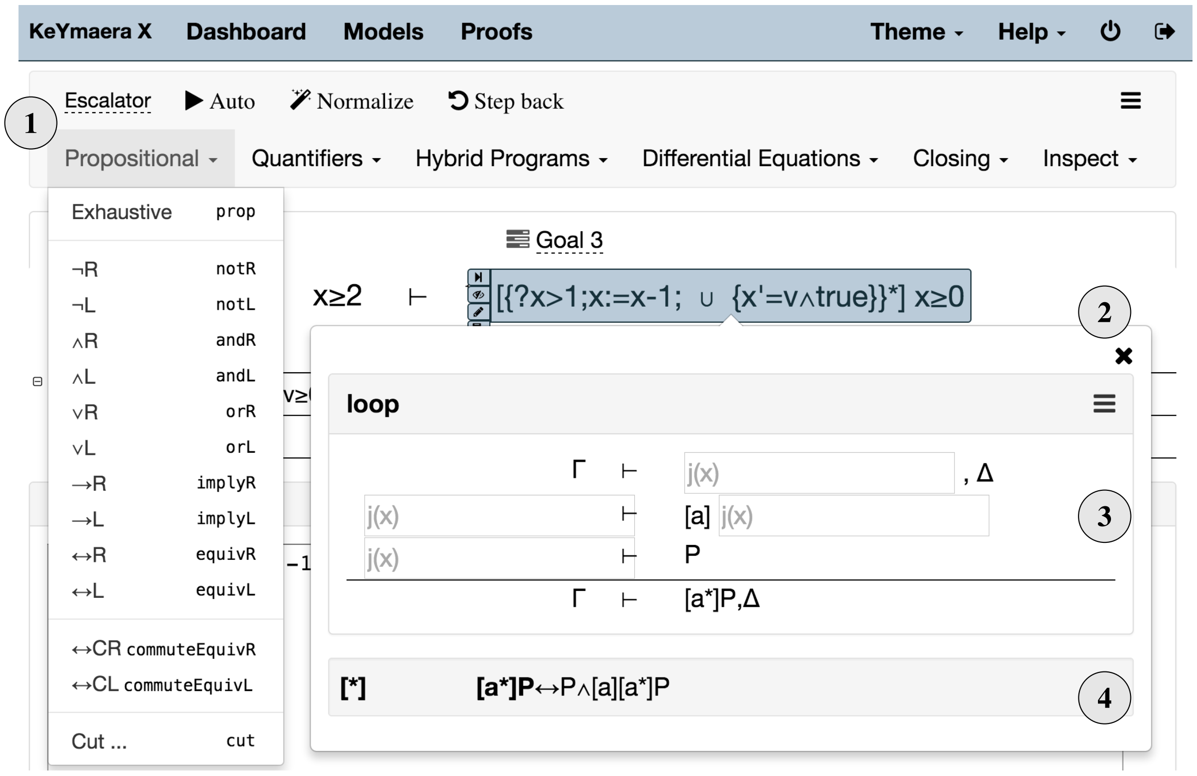
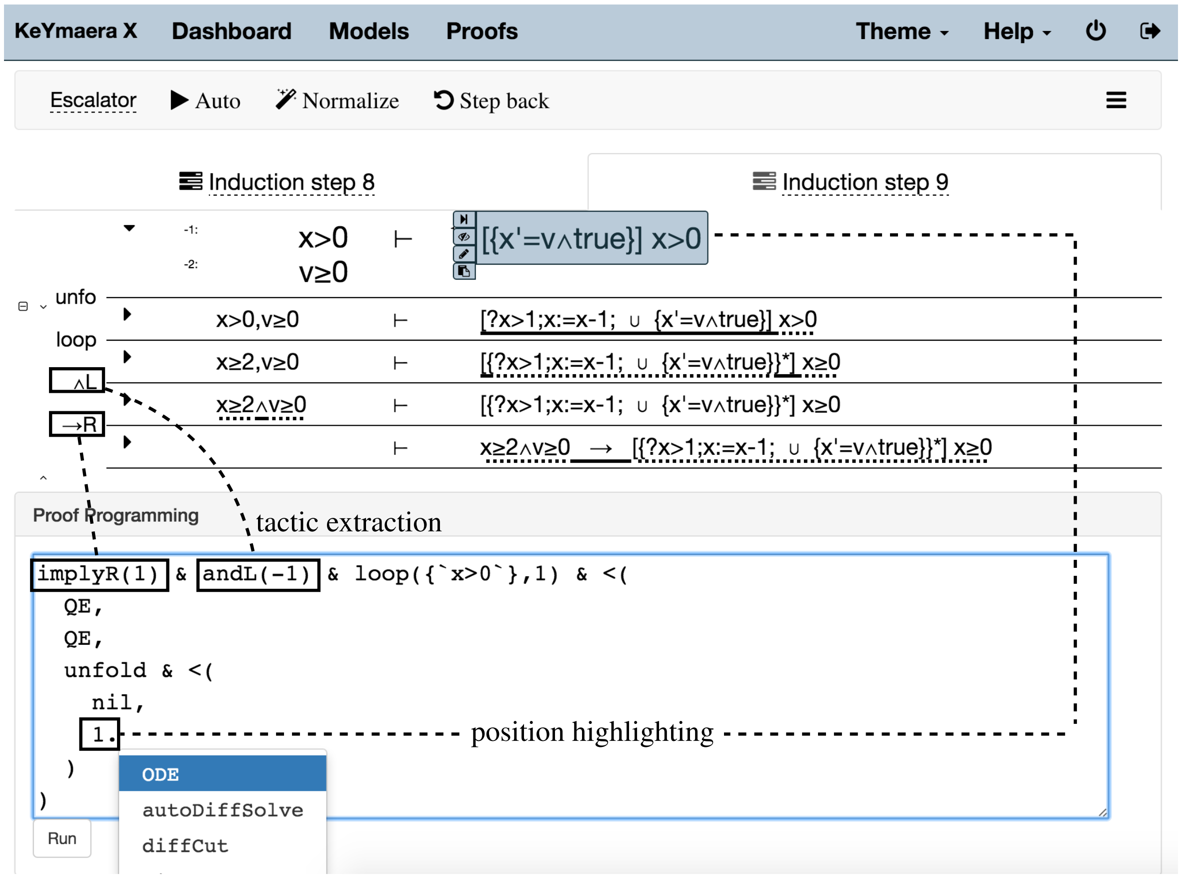


Figure 3: Two ways of asking KeYmaera X for help: ① Know what to do, but not sure where ⇝ search tactics in the toolbox menu instruct KeYmaera X to apply a specific tactic to the first applicable formula. ② Know where, but not sure what to do ⇝ tactic suggestion in the context menu of a formula: ③ tactic with input; ④ apply an axiom.



```
implyR(1) & andL(-1) & loop({`x>0`},1) & <(
  QE,
  QE,
  unfold & <(
    nil,
    1.
  )
)
```

Figure 4: Tactic editor below sequent view. The tactic is extracted automatically from the point-and-click interaction: the proof closed both the induction base case and the induction use case by quantifier elimination QE; the sequent view shows two open goals in the induction step, which result from using unfold (splits the choice, handles the test and the assignment, but stops at the differential equation). The tactic editor displays tactic suggestions akin to the tactic popover in the sequent view. Here, it display suggestions for the formula at position "1", which is also highlighted in the sequent view.

Tactic extraction. In order to conduct proofs effectively, interactive theorem provers typically ship with extensive tactic libraries (e. g., Coq [20], Isabelle [22]), accompanied with library documentation and examples to facilitate learning. Still, extensive tactic libraries incur a steep learning curve. We conjecture that observing how tactics evolve while doing a proof (similar to observing an expert) can reduce the steep learning curve associated with extensive tactic libraries. KeYmaera X automatically generates tactics that correspond to the point-and-click interaction that the user performed and displays these tactics below the graphical sequent view, see Figure 4.

The graphical sequent view and the tactic editor operate on the same proof, so that users can switch between both interaction concepts as they see fit. As a rule of thumb, the sequent view is designed for conducting specific steps at specific positions (e. g., use a specific equality $x = y$ to rewrite x into y in some other term), while the tactic editor is intended for describing proof search (e. g., repeat some step exhaustively ∧L('L)* or follow a high-level proof strategy such as unfold & ODE & QE).

4.4 Flexibility

To allow users to reason in flexible ways and reduce manual proof effort, KeYmaera X supports proof steps by pointing at formula parts and lets users transform and abbreviate formulas as well as execute tactic scripts. As underlying technique for proof-by-pointing and flexible reasoning anywhere in formulas, KeYmaera X provides contextual reasoning CQ and CE [27], so that questions about contextual equality or equivalence in a context $C\{\dots\}$ reduce to reasoning about arguments as follows.

$$\text{CQ}\,\frac{\vdash f() = g()}{\vdash C\{p(f())\} \leftrightarrow C\{p(g())\}} \qquad\qquad \text{CE}\,\frac{\vdash P \leftrightarrow Q}{\vdash C\{P\} \leftrightarrow C\{Q\}}$$

When contextual reasoning is combined with uniform substitution US and axiom lookup, axioms can be applied inside formulas, which enables proof by pointing at the desired formula part. Often, the next proof step follows unambiguously from just the shape of the pointed formula part by indexing [27], so that the proof advances without further user input.

$$\text{cut}\,\dfrac{\vdash y>2 \land 7>5 \qquad \text{CE}\,\dfrac{\text{US}\,\dfrac{\text{ax}\,[:=]\,\dfrac{*}{\vdash [x:=f()]p(x) \leftrightarrow p(f())}}{\vdash [x:=7]x>5 \leftrightarrow 7>5}}{\vdash y>2 \land [x:=7]x>5 \leftrightarrow y>2 \land 7>5}}{\vdash y>2 \land \underbrace{[x:=7]x>5}}$$

unifies with underlined key in axiom $\underline{[x:=f()]p(x)}\leftrightarrow p(f())$ $\rightsquigarrow$ replace with $p(f())$, which is 7>5

Note that side branches with contextual reasoning, uniform substitution, and axiom lookup (as in the proof above) close fully automatically. Hence, the user interface displays only the result of applying axioms by pointing, while it hides the minutia of the side deductions from the user as in the following example.

$$\text{useAt}\,\frac{\vdash P \to ([x'=2]x \geq 5) \lor Q}{\vdash P \to \underbrace{([x'=2 \cup x:=5]x \geq 5)}_{[\beta]B \to (\underline{[\alpha\cup\beta]B} \leftrightarrow [\alpha]B)} \lor Q}$$

4.5 Experimentation

The prover kernel and the prover interface are strictly separated, to the extent that the prover kernel only knows about making single deduction steps and combining them, but is completely oblivious of organizing these steps in a tree structure. For soundness, it suffices to know that any step between the original conjecture and the current subgoals must have been done in the prover kernel. Intermediate steps are only necessary to repeat a proof from the original conjecture, and can therefore be tracked outside the prover kernel. As a result, proof organization features (such as undoing proof steps, pruning the proof tree) can be implemented conveniently in the user interface without affecting soundness.

The separation between the prover core and the tactics becomes especially useful when adding compiled tactics to the server-side implementation. Complementing the interpreted tactics from the web-based user interface, server-side tactics can base on a rich implementation language (Scala) to try proof steps that are only sound in the usual cases and just rely on the prover core to catch when they happen to be applied in one of the unsound corner cases.

5 Evaluation Concept

Compared to its predecessor KeYmaera [28], the clean-slate implementation KeYmaera X introduces significant changes in user interaction. Although the user interaction changes are based on informal feedback from KeYmaera users and our own observations on how students used KeYmaera, it still remains to be checked by an experimental user study which style of user interaction is more effective. Additionally, KeYmaera X introduces new interaction concepts (e. g., tactic programming and tactic recording), which seem promising for experienced users to conduct large proofs, but are not yet backed by evidence that indeed prover interaction is improved compared to only mouse-based interaction. Similar user-related research questions were addressed in a recent controlled user experiment [13], which followed methods from empirical software engineering [32] to compare two very different user interfaces of the KeY theorem prover [1, 2]. The obtained results are encouraging pointers towards controlled user experiments being an appropriate method for testing theorem prover interaction.

Such controlled experiments, however, require a large number of participants with varying experience and deliberately exposing them to different user interaction concepts. In order to avoid adverse effects from pre-assigned interaction styles, we propose gathering data from normal operation on how often users rely on certain interaction patterns (e. g., clicking vs. automated tactic), how often they use a certain functionality during a proof, and how successful they were, even if that might lead to selection bias. Learnability of theorem provers can be measured based on cognitive dimensions [8], such as visibility (how easy are relevant steps accessed), juxtaposability (different notations side-by-side), viscosity (resistance to change), premature commitment (to an order how to do steps), error-proneness (how likely are errors), and consistency (similar information presented in similar ways). Theorem provers, especially in education and also in industrial applications, should have high visibility and juxtaposability, high consistency, and low viscosity [12]. The following metrics should be easily recordable from KeYmaera X without changing user interaction and give insights into cognitive dimensions.

Interaction concept Number of proof steps by clicking/tactic programming/automated proof search. Supporting evidence: number of proof steps at top-level/inside formulas; for clicking: number of steps executed by pointing/from tactic suggestion. Related to visibility and premature commitment.

Functionality Number of undo operations, length of pruned deduction paths with distance to next branching point above and below, number of find counterexample operations. Number of interactions in pruned proofs. Related to viscosity and error-proneness.

Time Proof duration (including estimates of user time and number of reproof attempts)

Trends Compare trends as students gain experience and examples become more complex

We believe this data should enable a study that tests the following hypotheses.

Interaction preference Novice users prefer automated proof search over clicking over tactic programming. Novice users prefer working top-level over working inside formulas. Experienced users balance interaction.

Functionality Novice users either undo short paths or entire proofs and end up with more open branches. Experienced users undo branching and exercise branching control techniques.

Trends Automated usage drops with increased example complexity and student experience, while use of tactic programming increases; the focus on applying steps top-level drops while applying steps inside formulas increases with experience.

Influence Students that engage more direct control over proofs also for simpler examples are more productive in conducting proofs of complex cases than students who relied on full automation earlier on.

The quantitative insights from these metrics could be augmented with qualitative evaluation techniques, as demonstrated being effective for theorem provers, e. g., with focus groups [7] (users subjectively express their perception of or opinion on the user interface), with questionnaires [6], or by co-operative evaluation [15] (users verbalize their interactions while using the theorem prover).

6 Related Work

The verification landscape spans a wide variety of approaches from automated theorem provers and reachability analysis tools to interactive theorem provers.

Automated theorem provers (e. g., Vampire [16]) and reachability analysis tools (e. g., SpaceEx [10]) strive for fully automatic verification without user interaction, but their scope is inherently limited in cases that are not semidecidable, such as hybrid systems.

Auto-active verifiers (e. g., Dafny [17, 18], AutoProof [30]) put the model or code first and hide the verification engine, but support user guidance through annotations in the code. The basic idea is to make verification an integral part of (software) development that should be performed in the background by an IDE, much like background compilation. The downside of such an approach is that the verification steps are hidden entirely from the user, which can make it hard to resolve proofs with additional annotations when the verifier is stuck because the proof state and the verifier's working principles are opaque. The KeY interactive verification debugger [14] combines code annotations with interactive verification in KeY [1, 2] to supplement manual proofs when automated proving fails at some goals.

Interactive theorem provers, such as Coq [20] and Isabelle [22], primarily interact with users through tactic scripts, such as structured proofs in Isabelle/Isar [21]. Their user interfaces (e. g., CoqIDE [20], ProofGeneral [5], jEdit [31]) focus on text editing support for writing tactics and let users inspect the proof state and open goals by placing the cursor in the tactic script. Navigation with cursors introduced a limited form of proof by pointing [4] to fold or unfold equations. KeYmaera X advances proof-by-pointing to transform all or parts of a formula following the shape of an axiom or fact. PeaCoq[3] aims to make the proof state more accessible to users by providing a proof tree.

KeYmaera X combines concepts from automated, auto-active, and interactive theorem proving: it comes with fully automated proof search tactics for hybrid systems of limited scope (i. e., when a loop invariant can be found, a symbolic solution of a differential equation can be found to serve as an oracle for differential invariants, and the resulting arithmetic is tractable); it supports annotations for loop invariants and differential invariants, since both serve as model documentation as well as proof guidance; and finally, it allows users to conduct and finish proofs themselves with tactics and a graphical user interface.

7 Conclusion

The user interface of KeYmaera X is based on the principles of familiarity, traceability, tutoring, flexibility, and experimentation. It supports several different user interaction styles to make progress in proofs. The web-based user interface applies proof steps when clicking on formulas in the sequent view, and it batch-runs proof steps by automated search tactics as well as through tactic programming.

[3] `http://goto.ucsd.edu/peacoq/`

Future work includes evaluation in controlled user experiments, as described in the evaluation concept. On the basis of user studies, we expect to gain insights into how to best teach hybrid systems theorem proving, tactic programming, tactic generalization, and proof search. We are working on improved proof exploration, e.g., with timeouts (e. g., allow QE 5s to close; if it does not close within the budgeted time, try something else). Failed proof attempts or expired timeouts need a robust approach to making proofs portable and repeatable.

Acknowledgments The authors thank Nathan Fulton, Brandon Bohrer, Jan-David Quesel, Ran Ji, and Marcus Völp for their support in implementing KeYmaera X, Sarah Loos, Jean-Baptiste Jeannin, João Martins, Khalil Ghorbal, and Sarah Grebing for feedback and discussions on the user interface, as well as the anonymous reviewers for feedback on the manuscript.
This material is based on research sponsored by DARPA under agreement DARPA FA8750-12-2-0291.

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] Wolfgang Ahrendt, Bernhard Beckert, Daniel Bruns, Richard Bubel, Christoph Gladisch, Sarah Grebing, Reiner Hähnle, Martin Hentschel, Mihai Herda, Vladimir Klebanov, Wojciech Mostowski, Christoph Scheben, Peter H. Schmitt & Mattias Ulbrich (2014): *The KeY Platform for Verification and Analysis of Java Programs*. In Dimitra Giannakopoulou & Daniel Kroening, editors: *Verified Software: Theories, Tools and Experiments - 6th International Conference, VSTTE 2014, Vienna, Austria, July 17-18, 2014, Revised Selected Papers*, *LNCS* 8471, Springer, pp. 55–71, doi:10.1007/978-3-319-12154-3_4.

[3] J. Stuart Aitken, Philip D. Gray, Thomas F. Melham & Muffy Thomas (1998): *Interactive Theorem Proving: An Empirical Study of User Activity*. *J. Symb. Comput.* 25(2), pp. 263–284, doi:10.1006/jsco.1997.0175.

[4] David Aspinall & Christoph Lüth (2004): *Proof General meets IsaWin: Combining Text-Based And Graphical User Interfaces*. *Electr. Notes Theor. Comput. Sci.* 103, pp. 3–26, doi:10.1016/j.entcs.2004.09.011.

[5] David Aspinall, Christoph Lüth & Daniel Winterstein (2007): *A Framework for Interactive Proof*. In Manuel Kauers, Manfred Kerber, Robert Miner & Wolfgang Windsteiger, editors: *Towards Mechanized Math. Assistants, 14th Symp., Calculemus, 6th Int. Conf., MKM, Hagenberg, Austria, June 27-30, 2007, Proc.*, *LNCS* 4573, Springer, pp. 161–175, doi:10.1007/978-3-540-73086-6_15.

[6] Bernhard Beckert & Sarah Grebing (2012): *Evaluating the Usability of Interactive Verification Systems*. In Vladimir Klebanov, Bernhard Beckert, Armin Biere & Geoff Sutcliffe, editors: *Proceedings of the 1st International Workshop on Comparative Empirical Evaluation of Reasoning Systems, Manchester, United Kingdom, June 30, 2012*, *CEUR Workshop Proceedings* 873, CEUR-WS.org, pp. 3–17.

[7] Bernhard Beckert, Sarah Grebing & Florian Böhl (2014): *A Usability Evaluation of Interactive Theorem Provers Using Focus Groups*. In Carlos Canal & Akram Idani, editors: *Software Engineering and Formal Methods - SEFM 2014 Collocated Workshops: HOFM, SAFOME, OpenCert, MoKMaSD, WS-FMDS, Grenoble, France, September 1-2, 2014, Revised Selected Papers*, *LNCS* 8938, Springer, pp. 3–19, doi:10.1007/978-3-319-15201-1_1.

[8] Alan F. Blackwell, Carol Britton, Anna Louise Cox, Thomas R. G. Green, Corin A. Gurr, Gada F. Kadoda, Maria Kutar, Martin Loomes, Chrystopher L. Nehaniv, Marian Petre, Chris Roast, Chris Roe, Allan Wong & R. Michael Young (2001): *Cognitive Dimensions of Notations: Design Tools for Cognitive Technology*. In Meurig Beynon, Chrystopher L. Nehaniv & Kerstin Dautenhahn, editors: *Cognitive Technology: Instruments of Mind, 4th International Conference, CT 2001, Warwick, UK, August 6-9, 2001, Proceedings*, *Lecture Notes in Computer Science* 2117, Springer, pp. 325–341, doi:10.1007/3-540-44617-6_31.

[9] James H. Davenport & Joos Heintz (1988): *Real Quantifier Elimination is Doubly Exponential*. J. Symb. Comput. 5(1/2), pp. 29–35, doi:10.1016/S0747-7171(88)80004-X.

[10] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang & Oded Maler (2011): *SpaceEx: Scalable Verification of Hybrid Systems*. In Ganesh Gopalakrishnan & Shaz Qadeer, editors: CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proc., LNCS 6806, Springer, pp. 379–395, doi:10.1007/978-3-642-22110-1_30.

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

[12] D. Diaper G. Kadoda, R. G. Stone (1999): *Desirable features of educational theorem provers - a cognitive dimensions viewpoint*. In: 11th Annual Workshop of Psychology of Programming Interest Group, PPIG, pp. 1–6.

[13] Martin Hentschel, Reiner Hähnle & Richard Bubel (2016): *An empirical evaluation of two user interfaces of an interactive program verifier*. In Lo et al. [19], pp. 403–413, doi:10.1145/2970276.2970303.

[14] Martin Hentschel, Reiner Hähnle & Richard Bubel (2016): *The interactive verification debugger: effective understanding of interactive proof attempts*. In Lo et al. [19], pp. 846–851, doi:10.1145/2970276.

[15] Michael J. Jackson (1997): *Evaluation of a Semi-Automated Theorem Prover (Part II)*.

[16] Laura Kovács & Andrei Voronkov (2013): *First-Order Theorem Proving and Vampire*. In Natasha Sharygina & Helmut Veith, editors: Computer Aided Verification - 25th Int. Conf., CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proc., LNCS 8044, Springer, pp. 1–35, doi:10.1007/978-3-642-39799-8_1.

[17] 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.

[18] 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.

[19] David Lo, Sven Apel & Sarfraz Khurshid, editors (2016): *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*. ACM, doi:10.1145/2970276.

[20] The Coq development team (2015): *The Coq proof assistant reference manual*. LogiCal Project. Available at `http://coq.inria.fr`. Version 8.5.

[21] 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.

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

[23] André Platzer (2008): *Differential Dynamic Logic for Hybrid Systems*. J. Autom. Reas. 41(2), pp. 143–189, doi:10.1007/s10817-008-9103-8.

[24] André Platzer (2012): *Logics of Dynamical Systems*. In: LICS, IEEE, pp. 13–24, doi:10.1109/LICS.2012.13.

[25] André Platzer (2013): *Teaching CPS Foundations With Contracts*. In: CPS-Ed, pp. 7–10.

[26] André Platzer (2015): *A Uniform Substitution Calculus for Differential Dynamic Logic*. In Amy P. Felty & Aart Middeldorp, editors: CADE, LNCS 9195, Springer, pp. 467–481, doi:10.1007/978-3-319-21401-6_32.

[27] André Platzer (2016): *A Complete Uniform Substitution Calculus for Differential Dynamic Logic*. J. Autom. Reas., doi:10.1007/s10817-016-9385-1.

[28] André Platzer & Jan-David Quesel (2008): *KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description)*. In Alessandro Armando, Peter Baumgartner & Gilles Dowek, editors: Automated Reasoning,

4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proc., *LNCS* 5195, Springer, pp. 171–178, doi:10.1007/978-3-540-71070-7_15.

[29] Jan-David Quesel, Stefan Mitsch, Sarah M. Loos, Nikos Arechiga & André Platzer (2016): *How to model and prove hybrid systems with KeYmaera: a tutorial on safety*. *STTT* 18(1), pp. 67–91, doi:10.1007/s10009-015-0367-0.

[30] 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.

[31] 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.

[32] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson & Björn Regnell (2012): *Experimentation in Software Engineering*. Springer, doi:10.1007/978-3-642-29044-2.