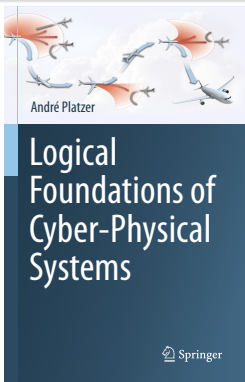


# A Tale of Three Provers

Logical Foundations of Cyber-Physical Systems



Stefan Mitsch



- 1 Prover Implementation Challenges
- 2 KeYmaera 3
- 3 KeYmaeraD
- 4 KeYmaera X
- 5 Core Comparison
- 6 User Interfaces
- 7 Summary

- 1 Prover Implementation Challenges
- 2 KeYmaera 3
- 3 KeYmaeraD
- 4 KeYmaera X
- 5 Core Comparison
- 6 User Interfaces
- 7 Summary

# How to Put it All into Theorem Proving Practice?

## Design decisions

- Axioms: what is the minimal set needed?
- Rules and derived rules
- Arithmetic procedures: use at all? how to interoperate reliably?
- Invariant generators: make non-soundness critical
- Proof management, replay, visualization

## Proof repeatability challenges

- Timeouts
- External tools
- What is a robust proof?  
`auto` vs. `implR(1)` vs. `implR(R=="x>0->x>=0")`?

## Term language

Should be minimal to make correctness obvious, but then makes tracking other information difficult (e.g., annotations, definitions, archives)

- Terms
- Formulas
- Programs

## Proof data structure

- Conclusion and premises
- Operations on proofs?

## Essential tools

- Parser: how to make not soundness-critical?
- Arithmetic tools: bridge data structure differences (numbers!)

## Infrastructure

- Unification
- Store proofs
- Suspend and resume proofs

## Automation vs. interaction

- Fully interactive vs. fully automatic?
- “Auto-active” with annotations in the code?
- Interrupt automation + interactive?
- Tactic-based? Interpreted or hosted tactic language?
- Proof robustness?

- 1 Prover Implementation Challenges
- 2 KeYmaera 3**
- 3 KeYmaeraD
- 4 KeYmaera X
- 5 Core Comparison
- 6 User Interfaces
- 7 Summary

Build on top of an existing dynamic logic prover

## Benefits

- Inherit user interface
- Inherit infrastructure (help resources, proof archives)

## Implementation Reality: Non-deterministic Choice

```
1 box_choice_right {
2   \find (==> \modality{#boxtr}#dl ++ #dl2\endmodality(post))
3   \replacewith (==> \modality{#boxtr}#dl\endmodality(post));
4   \replacewith (==> \modality{#boxtr}#dl2\endmodality(post))
5   \heuristics (simplify_prog)
6   \displayname "[++] choice"
7 };
```



## Successes

- Easy to add rules to proof search
- Simple proof management: single proof tree

## Challenge: Limit applicability of proof rules

KeYmaera 3 proof rule mechanism adapts rules to *any* context

$$\begin{array}{c}
 \frac{\Gamma \vdash \mathcal{U}[\alpha]\phi, \Delta \quad \Gamma \vdash \mathcal{U}[\beta]\phi, \Delta}{\Gamma \vdash \mathcal{U}[\alpha \cup \beta]\phi, \Delta} \\
 \text{[U]}
 \end{array}
 \quad
 \frac{\Gamma \vdash \mathcal{U}J, \Delta \quad \Gamma, \mathcal{U}J \vdash \mathcal{U}\phi, \Delta \quad \Gamma, \mathcal{U}J \vdash \mathcal{U}[\alpha]J, \Delta}{\Gamma \vdash \mathcal{U}[\alpha^*]\phi, \Delta}$$

```

1 loop_inv_box_quan {
2   \find (==> \modality{#boxtr}#dl*\endmodality(post))
3   "Invariant Initially Valid":
4     \replacewith (==> inv);
5   "Use Case":
6     \replacewith (==> #UnivCl(\[#dl\]true, inv->post, false));
7   "Body Preserves Invariant":
8     \replacewith (==> #UnivCl( ... ))
9 };

```

## Successes

- Easy to add rules to proof search
- Simple proof management: single proof tree

## Challenge: Limit applicability of proof rules

KeYmaera 3 proof rule mechanism adapts rules to *any* context

$$\begin{array}{c}
 \frac{\Gamma \vdash \mathcal{U}[\alpha]\phi, \Delta \quad \Gamma \vdash \mathcal{U}[\beta]\phi, \Delta}{\Gamma \vdash \mathcal{U}[\alpha \cup \beta]\phi, \Delta} \\
 \text{[U]}
 \end{array}
 \quad
 \frac{\Gamma \vdash \mathcal{U}J, \Delta \quad \Gamma, \mathcal{U}J \vdash \mathcal{U}\phi, \Delta \quad \Gamma, \mathcal{U}J \vdash \mathcal{U}[\alpha]J, \Delta}{\Gamma \vdash \mathcal{U}[\alpha^*]\phi, \Delta}$$

```

1 loop_inv_box_quan {
2   \find (==> \modality{#boxtr}#dl*\endmodality(post))
3   "Invariant Initially Valid":
4     \replacewith (==> inv);
5   "Use Case":
6     \replacewith (==> #UnivCl(\[#dl\]true, inv->post, false));
7   "Body Preserves Invariant":
8     \replacewith (==> #UnivCl( ... ))
9 };

```

## Successes

- Easy to add rules to proof search
- Simple proof management: single proof tree

## Challenge: Limit applicability of proof rules

KeYmaera



### Challenge

Meta-operator `#UnivCl` soundness-critical (800LoC)  
Similar meta-operators needed to compute Lie derivatives  
etc.

```
Γ ⊢ ℚ  
[U] ————— α]J, Δ  
  
1 loop_invariant( ... )  
2 \find (==> \modality{#boxtr}#dl*\endmodality(post))  
3 "Invariant Initially Valid":  
4 \replacewith (==> inv);  
5 "Use Case":  
6 \replacewith (==> #UnivCl(\[#dl\]true, inv->post, false));  
7 "Body Preserves Invariant":  
8 \replacewith (==> #UnivCl( ... ))  
9 };
```

- 1 Prover Implementation Challenges
- 2 KeYmaera 3
- 3 KeYmaeraD**
- 4 KeYmaera X
- 5 Core Comparison
- 6 User Interfaces
- 7 Summary

## Bare-bones prover with direct dL rendition in Scala

## Benefits

- And/Or-proof tree
- Pattern matching in Scala replaces rule application mechanism
- Directly program proofs in Scala

## Implementation Reality: Non-deterministic Choice

```
1 val choose = new ProofRule("choose") {  
2   def apply(p: Position) = sq => {  
3     lookup(p, sq) match {  
4       case Modality(Box, Choose(h1, h2), phi) =>  
5         val fm1 = Modality(Box, h1, phi)  
6         val fm2 = Modality(Box, h2, phi)  
7         Some((List(replace(p, sq, Binop(And, fm1, fm2))), Nil))  
8       case _ => None  
9     }  
10 }
```

## Successes

- Parallel proof search
- Low-level control over proof

## Challenges

- Simple rules put burden on user, for example assignment

$$[:=]_{\text{eq}} \frac{\Gamma, y = e \vdash P_x^y, \Delta}{\Gamma \vdash [x := e]P, \Delta} \quad (y \text{ fresh})$$

$$x = y \vdash [x := x + 1][x := 2x][x := x - 2]x = 2y$$

produces  $x = y, x_0 = x + 1, x_1 = 2x_0, x_2 = x_1 - 2 \vdash x_2 = 2y$

- More clever rules require considerable admissibility-checking
- Every extension of proof rules is soundness-critical

- 1 Prover Implementation Challenges
- 2 KeYmaera 3
- 3 KeYmaeraD
- 4 KeYmaera X**
- 5 Core Comparison
- 6 User Interfaces
- 7 Summary

## Small-core prover

## Benefits

- Proof search decoupled from proof checking
- Rule application in context

## Implementation Reality: Axioms

```
1 Axiom "[++] choice"  
2   [a; ++b;]p(II) <-> [a;]p(II) & [b;]p(II)  
3 End.  
4 Axiom "[:=] assign"  
5   [x:=f();]p(x) <-> p(f())  
6 End.  
7 Axiom "[:=] assign equality"  
8   [x:=f();]p(II) <-> \forall x (x=f() -> p(II))  
9 End.
```



## Assignment

```
1 @Tactic("[:=]", conclusion = "__[x:=e]p(x)___<->p(e) ")
2 val assignb = anon by { w =>
3   useAt("[:=] assign")(w)
4   | useAt("[:=] self assign")(w)
5   | asgnEq(w)
6 }
```

## Interpreted vs. builtin tactics

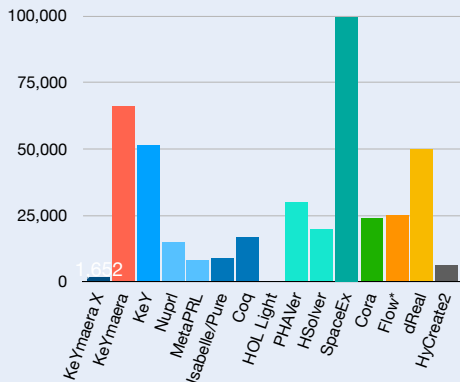
- Tactic `assignb` is implemented as an interpreted tactic
- Benefit: can record inner steps
- Challenge: performance
- Challenge: debugging
- Challenge: result depends on proof state

## Equational assignment

```
1 @Tactic(  
2   names = "[:]==",  
3   premises = "G, x=e |- P, D",  
4   //      [:=]= -----  
5   conclusion = "G |- [x:=e]P, D",  
6   displayLevel = "all"  
7 )  
8 val asgnEq = anon by ((w, seq) => seq.sub(w) match {  
9   case Some(Box(Assign(x, t), p)) =>  
10    val y = freshNamedSymbol(x, seq)  
11    boundRenaming(x, y)(w) &  
12    useAt("[:]= assign equality")(w) &  
13    uniformRenaming(y, x) &  
14    (if (w.isTopLevel&&w.isSucc) allR(w) & implyR(w) else ident)  
15  })
```

## Successes

- Core size
- Extensibility with tactics
- Modularization
- Extension with non-trusted generators



## Challenges

- Performance
- Handover between automation and interaction

- 1 Prover Implementation Challenges
- 2 KeYmaera 3
- 3 KeYmaeraD
- 4 KeYmaera X
- 5 Core Comparison**
- 6 User Interfaces
- 7 Summary

## Assignments

$$x = 1 \vdash \{x := x + 1\} [?x \geq 1] x \geq 0 \quad \mathbf{K3} \text{ (update)}$$

$$x = 1, x_1 = x + 1 \vdash [?x_1 \geq 1] x_1 \geq 0 \quad \mathbf{KD} \text{ (equation)}$$

$$x = 1 \vdash [?x + 1 \geq 1] 2 \geq 0 \quad \mathbf{KX} \text{ (substitution)}$$

$$\frac{[:=]}{x = 1 \vdash [x := x + 1] [?x \geq 1] x \geq 0}$$

## The surprising subtlety of assignments

$$\frac{\vdash [x := 3] x \geq 3}{[:=] \vdash [x := 2] [x := 3] x \geq 3}$$

$$\frac{\vdash [x := 2 + 1] x \geq 3}{[:=] \vdash [x := 2] [x := x + 1] x \geq 3}$$

$$\frac{\vdash [\{x' = -x^2\}] (x + 1)^2 \geq 2}{[:=] \vdash [\{x' = -x^2\}] [x := x + 1] x^2 \geq 2}$$

$$\frac{x_0 = 1, x = 2 \vdash [\{x' = x\}] x \geq 2}{[:=] \vdash x = 1 \vdash [x := 2] [\{x' = x\}] x \geq 2}$$

$$\frac{\vdash [x := 2 + 1 \cup x := 3] [(x := x + 1)^*] x \geq 3}{[:=] \vdash [x := 2] [x := x + 1 \cup x := 3] [(x := x + 1)^*] x \geq 3}$$

$$\frac{\vdash [(x := x + 1)^*] \forall x (x = 2 \rightarrow [\{x' = 3\}] x \geq 2)}{[:=] \vdash [(x := x + 1)^*] [x := 2] [\{x' = 3\}] x \geq 2}$$

## Loops

**K3**  $x = 1, b > 0 \vdash x \geq 1$                       **K3**  $x = 1, b > 0 \vdash \forall x (x \geq 1 \rightarrow x \geq 0)$

**KD**  $x = 1, b > 0 \vdash x \geq 1 \wedge b > 0$    **KD**  $x \geq 1 \wedge b > 0 \vdash x \geq 0$

**KX**  $x = 1, b > 0 \vdash x \geq 1$                       **KX**  $b > 0, x \geq 1 \vdash x \geq 0$

(a) Base case

(b) Use case

**K3**  $x = 1, b > 0 \vdash \forall x (x \geq 1 \rightarrow [x := x + 1/b]x \geq 1)$

**KD**  $x \geq 1 \wedge b > 0 \vdash [x := x + 1/b](x \geq 1 \wedge b > 0)$

**KX**  $b > 0, x \geq 1 \vdash [x := x + 1/b]x \geq 1$

(c) Induction step

	Base case (1a)	Use case (1b)	Induction step (1c)
loop	$x = 1, b > 0 \vdash [(x := x + 1/b)^*]x \geq 0$		

## Differential Induction

(init)	(step)
<b>K3</b> $x^2 + y^2 = 1 \vdash x^2 + y^2 = 1$	$x^2 + y^2 = 1 \vdash \forall x \forall y (2xy + 2y(-x) = 0)$
<b>KD</b> $x^2 + y^2 = 1 \vdash x^2 + y^2 = 1$	$\vdash 2xy + 2y(-x) = 0$
<b>KX</b> $x^2 + y^2 = 1 \vdash x^2 + y^2 = 1 \checkmark$	$x_0^2 + y_0^2 = 1 \vdash [x' := y][y' := -x]2xx' + 2yy' = 0 \checkmark$
$\text{DI} \frac{\quad}{x^2 + y^2 = 1 \vdash [x' = y, y' = -x]x^2 + y^2 = 1}$	

- KeYmaera X provides several versions of differential induction (fully automatic to fully manual) to accommodate examples of varying complexity
- Fully manual differential induction is challenging in axiomatic prover, easy in rule-based prover

# Proof Management: Central Proof Tree vs. Separate Steps

$$\begin{array}{c}
 \text{IR} \frac{\text{WL} \frac{\text{VL} \frac{\text{IR} \frac{\text{WL} \frac{x < 0 \vee x = 0, y = x \vdash xy \leq y^2}{*}}{\text{WL} \frac{x < 0 \vee x = 0, y = x \vee y > 0 \vdash xy \leq y^2}{*}}{\text{VL} \frac{x < 0 \vee x = 0, y = x \vee y > 0 \vdash xy \leq y^2}{*}}{\text{WL} \frac{x < 0 \vee x = 0, y = x \vee y > 0 \vdash xy \leq y^2}{*}}}{\text{WL} \frac{x < 0 \vee x = 0, y = x \vee y > 0 \vdash xy \leq y^2}{*}}
 \end{array}$$

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

$$\begin{array}{c}
 \text{id} \frac{*}{x = 0, x = 1 \vdash x = 0} \\
 \text{WLi} \frac{\text{WL} \frac{x = 0, x = 1 \vdash x = 0}{\text{WL} \frac{x = 0 \vdash [x := 1]x = 0}}{\text{WL} \frac{x = 0 \vdash [x := 1]x = 0}}{\text{WL} \frac{x = 0 \vdash [x := 1]x = 0}}
 \end{array}$$

(b) Unsoundly ignoring temporarily hidden formula

$$\begin{array}{c}
 \text{OUS} \frac{x_0 = 0, x = 1 \vdash x = 0}{P_{\text{O}}(x_0), x = 1 \vdash x = 0} \\
 \text{OUS} \frac{[:=] \frac{P_{\text{O}}(x) \vdash [x := 1]x = 0}}{x = 0 \vdash [x := 1]x = 0}}{x = 0 \vdash [x := 1]x = 0}
 \end{array}$$

(c) Sub-proof with substitution  
 $\sigma = \{P_{\text{O}}(\cdot) \mapsto \cdot = 0\}$

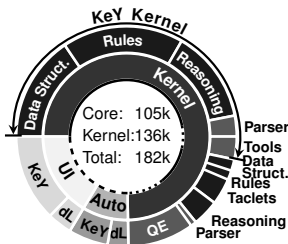


# Code Statistics

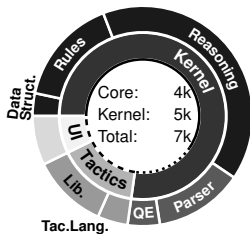
■ Soundness-critical core  
■ Non-critical prover code

■ Correctness-critical tools  
■ Non-critical user-facing infrastructure

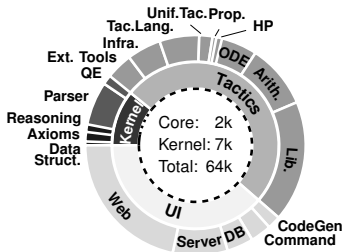
## KeYmaera 3



## KeYmaeraD



## KeYmaera X



- 1 Prover Implementation Challenges
- 2 KeYmaera 3
- 3 KeYmaeraD
- 4 KeYmaera X
- 5 Core Comparison
- 6 User Interfaces**
- 7 Summary

# Challenges

- Easy to lose track of remaining proof effort
- Visualize large trees

The screenshot shows the KeYmaera Prover interface. At the top, the title bar reads "KeYmaera -- Prover". Below the title bar is a toolbar with buttons for "Start", "Goal Back", "Reuse", and icons for file operations. The main window is divided into three panes:

- Tasks:** Shows the environment "Env. with no model #1" and a key file "bouncing-ball-simplest6543601653448115689.key".
- Proof:** A tree view showing the current state of the proof. The root is "Proof Tree", which branches into "1:→r imply right", "2:ind loop invariant", "Invariant Initially Valid", "Use Case", "Body Preserves Invariant", "5:Update Simplification", "6:vr all right", "12:vr all right", "13:→r imply right", and "14:OPEN GOAL".
- Current Goal:** A list of logical formulas. The top goal is  $(v_2)^2 \leq 2 \cdot g \cdot (H - h_2) \wedge h_2 \geq 0$ . Below it is a conjunction of conditions:  $g > 0 \wedge h \geq 0 \wedge v^2 \leq 2 \cdot g \cdot (H - h) \wedge H \geq 0 \wedge (0 \leq c \wedge c < 1)$ . This is followed by a block of code:  $\{h := h_2 \mid \mid v := v_2\}$ , a block of code:  $\{h' = v, v' = -g\};$   $(?h = 0;$   $v := (-c \cdot v))$   $u ?h \geq 0$ , and finally a block of code:  $\} (v^2 \leq 2 \cdot g \cdot (H - h) \wedge h \geq 0)$ .

# Challenges

- Easy to lose track of remaining proof effort
- Visualize large trees

The screenshot displays the KeyMaeraD proof assistant interface. The window title is "KeyMaeraD". The menu bar includes "File", "View", and "Prove".

**Left Panel (Proof Tree):** A large, inverted tree structure representing the proof. The root node is a green checkmark with the text "87 | substitute". The tree branches downwards, with nodes labeled with numbers and actions like "substitute", "assign", "diffsolve", "andright", and "andleft". The bottom-most node is labeled "DoneNode: 127".

**Right Panel (Code Editor):** A code editor showing the formal proof script. The code is as follows:

```
(qx0 - px0) * nx0 + (qy0 - py0) * ny0 >= 0
|-
[[fyp0 := *;
 fyp0 := *;
 fnp0 := fxp0 * nx0 + fyp0 * ny0;
 fn0 := fx0 * nx0 + fy0 * ny0;
 d00 := (qx0 - px0) * nx0 + (qy0 - py0) * ny0;
 dist0 := d00 + K0 * (fn0 * e0 + fnp0 * e0^2 * (1 / 2));
 disc0 := (K0 * fn0)^2 - 2 * K0 * fnp0 * d00;
 (((((?fnp0 <= 0 & dist0) >= 0;
 g0 := 0) ++
 (?fnp0 <= 0 & dist0) <= 0;
 tmp0 := *;
 ?tmp0 * K0 * e0 = 1;
 g0 := fn0 + (d00 + 1 / 2 * K0 * fnp0 * e0^2) * tmp0) ++
 (?fnp0 >= 0 & fn0 <= 0 & disc0) <= 0;
 g0 := 0) ++
 (?fnp0 >= 0 & fn0 <= 0 & disc0) >= 0 & fn0 + fnp0 * e0 >= 0;
 tmp0 := *;
 ?tmp0 * tmp0 * K0 = 2 * d00 * fnp0 & tmp0 >= 0;
 g0 := fn0 - tmp0) ++
 (?fnp0 >= 0 & fn0 <= 0 & disc0) >= 0 & fn0 + fnp0 * e0 <= 0 & dist0 <= 0;
 tmp0 := *;
 ?tmp0 * K0 * e0 = 1;
 gt0 := fn0 + (d00 + 1 / 2 * K0 * fnp0 * e0^2) * tmp0;
 tmp20 := *;
 (?fnp0 > 0 & fnp0 * tmp20 = -(1 * (fn0 - gt0));
 tmp30 := *;
 (?tmp20 <= e0 & tmp30 * tmp30 * K0 = 2 * d00 * fnp0 & tmp30) >= 0;
 g0 := fn0 - tmp30) ++
 (?tmp20 >= e0;
 g0 := gt0) ++
 (?fnp0 = 0;
 g0 := gt0) ++
 (?fnp0 >= 0 & fn0 <= 0 & disc0) >= 0 & fn0 + fnp0 * e0 <= 0 & dist0) >= 0;
 g0 := 0) ++
 (?fnp0 >= 0 & fn0) >= 0;
 g0 := 0;
 t0 := 0;
 [qx0' = K0 * (fx0 - g0) * nx0, qy0' = K0 * (fy0 - g0) * ny0, fxp0' = fxp0, fyp0' = fyp0, t0' = 1, t0 <= e0]]*)
(qx0 - px0) * nx0 + (qy0 - py0) * ny0 >= 0
```

# Challenges

- Easy to lose track of remaining proof effort
- Visualize large trees

KeYmaera X

Theme Help ? ⏻ ↻

07: Bo...

▶ Auto Prop Unfold Simplify Undo Edit Browse  
Defs Propositional Quantifier Hybrid Program Differential Equation Tools

Post : Induction

Init : Induction

Step : [i]

Hint: ODE | solve | dC | dl | dW | dG | GV | MR

$2 * g * x =$   
-1:  $2 * g * H - v^2 \wedge$   
 $x \geq 0$   
-2:  $g > 0$   
-3: 1

$\vdash$   $[ \{ x' = v, v' = -g \ \& \ x \geq 0 \} ]$   
 $[ ?x = 0; v := -c * v; \cup ?x \neq 0; ] (2 * g * x =$   
 $2 * g * H - v^2 \wedge x \geq 0)$

$\ominus$  [i] -4: c  
 $2 * g$  [u] choiceb  $[a; \cup b;] p \leftrightarrow [a;] p \wedge [b;] p$   
 $g >$   
 $1 =$  [∧] boxAnd  $[a;] (p \wedge q) \leftrightarrow [a;] p \wedge [a;] q$   
loop  $c \geq$  Gödel Vacuous gv  
 $(x \geq$   
 $\rightarrow R$   $g >$  Monotonicity MR ...  
chaseAt chaseAt

**choiceb** splits a nondeterministic choice  $a \cup b$  between running  $a$  or  $b$  into its alternatives by reducing it to a conjunction  $[a]P \wedge [b]P$  that establishes the safety of  $a$  and of  $b$  separately.

Tactic Rule

Search Search for lemmas

Browse... Apply Lemma

- 1 Prover Implementation Challenges
- 2 KeYmaera 3
- 3 KeYmaeraD
- 4 KeYmaera X
- 5 Core Comparison
- 6 User Interfaces
- 7 Summary**

## Lessons Learned

- Small core makes it significantly easier to advance development
- Uniform substitution calculus increases flexibility in proofs
- Common proof state data structure helps interleaving automation and interaction

## Advice for Future Development

- Invest in common infrastructure early
- Invest in the ability of visualizing/debugging tactics
- Strive for deterministic tactic behavior (timeouts in automation are challenging)
- Invest in good error messages



Stefan Mitsch and André Platzer.

A retrospective on developing hybrid system provers in the KeYmaera family - A tale of three provers.

*In Deductive Software Verification: Future Perspectives - Reflections on the Occasion of 20 Years of KeY*, pages 21–64. 2020.

[doi:10.1007/978-3-030-64354-6\\\_2](https://doi.org/10.1007/978-3-030-64354-6\_2).



Stefan Mitsch.

Implicit and explicit proof management in KeYmaera X.

*In Proceedings of the 6th Workshop on Formal Integrated Development Environment, F-IDE@NFM 2021*, pages 53–67, 2021.

[doi:10.4204/EPTCS.338.8](https://doi.org/10.4204/EPTCS.338.8).



Stefan Mitsch and André Platzer.

The KeYmaera X proof IDE - concepts on usability in hybrid systems theorem proving.

*In Proceedings of the Third Workshop on Formal Integrated Development Environment, F-IDE@FM 2016, Limassol, Cyprus, November 8, 2016*, pages 67–81, 2016.

[doi:10.4204/EPTCS.240.5](https://doi.org/10.4204/EPTCS.240.5).