Formally Verified Differential Dynamic Logic (in Isabelle/HOL and Coq)

Brandon Bohrer¹, Vincent Rahli², Ivana Vukotic², Marcus Völп², André Platzer¹
Thanks to: Johannes Hölzl, Fabian Immler, Tobias Nipkow, et. al.³
+ Coquelicot team

1 Carnegie Mellon University
2 University of Luxembourg
3 Technical University Munich
Safety-Critical Control Software is Everywhere

How can we design cyber-physical systems people can bet their lives on?
– Jeanette Wing
Formal Modeling and Verification Provide Safety

- *Differential Dynamic Logic* enables constructing and verifying hybrid models
- *KeYmaera X* theorem prover implements differential dynamic logic

**Example theorem:** \[ v \geq 0 \land A() \geq 0 \rightarrow [a := A(); \{v' = a, x' = v \land \text{true}\}]v \geq 0 \]
Formal Modeling and Verification Provide Safety

- *Differential Dynamic Logic* enables constructing and verifying hybrid models
- *KeYmaera X* theorem prover implements differential dynamic logic

**Example theorem:** \( v \geq 0 \& A() \geq 0 \rightarrow [a := A(); \{v' = a, x' = v \& true\}]v \geq 0 \)

- Correct proof requires correct prover
Formal Modeling and Verification Provide Safety

- *Differential Dynamic Logic* enables constructing and verifying hybrid models.
- *KeYmaera X* theorem prover implements differential dynamic logic.

**Example theorem:** $v \geq 0 \land A() \geq 0 \rightarrow [a := A(); \{v' = a, x' = v \land \text{true}\}]v \geq 0$

- Correct proof requires correct prover.
- **Goal:** Ensure correctness of *KeYmaera X*.
KeYmaera X Depends on a Big Stack

Pick part of the stack, try to improve it
KeYmaera X Depends on a Big Stack

Implemented in Scala: Large stack
(Not specific to hybrid systems proving)
KeYmaera X Depends on a Big Stack

LCF Approach:

- Abstract type of proofs isolates critical code
- Majority of code untrusted

<table>
<thead>
<tr>
<th>Prover</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOL Light</td>
<td>400</td>
</tr>
<tr>
<td>KeYmaera 3</td>
<td>66,000</td>
</tr>
<tr>
<td><strong>KeYmaera X</strong></td>
<td>1,700 (2%)</td>
</tr>
<tr>
<td>Isabelle/Pure</td>
<td>8,000</td>
</tr>
<tr>
<td>Coq</td>
<td>20,000</td>
</tr>
<tr>
<td>NuPrl</td>
<td>15,000 + 50,000</td>
</tr>
<tr>
<td>PHAVer</td>
<td>30,000</td>
</tr>
<tr>
<td>SpaceEx</td>
<td>100,000</td>
</tr>
</tbody>
</table>
Small Core Requires Simple Proof Calculus

Uniform Substitution (US) Simplifies Provers

\[ [t := 0; t' = 1] t \geq 0 \leftrightarrow [t := 0][t' = 1] t \geq 0 \]

Axiom Instance

\[ [a ; b]P \leftrightarrow [a][b]P \]

Axiom

Axioms are just data

Substitution \( \sigma = \{ a \mapsto t := 0, b \mapsto t' = 1, P \mapsto t \geq 0 \} \)
Uniform Substitution is Easy

Substitution $\sigma$ maps symbols to replacements
Replace recursively (Some cases not primitive recursive!)

Example: $\sigma=\{f \mapsto x + 1, p(y) \mapsto y \neq x\}$

$\sigma(p(f))$

$= \sigma(p(x + 1))$

$= x + 1 \neq x$
Uniform Substitution is Easy

Simple Proof Rule: \[ \text{US} \quad \frac{\varphi}{\sigma(\varphi)} \]

- Correctness
- Compiler
- Theory
- Implementation
- You

1,700 lines
Uniform Substitution is Easy

Simple Proof Rule:

\[
\text{US} \quad \frac{\varphi}{\sigma(\varphi)}
\]

Example:

\[
\text{US} \quad \frac{[x := f]p(x) \leftrightarrow p(f)}{[x := \_\_\_] \leftrightarrow \_\_\_}
\]

Substitution \( \sigma = \{ f \mapsto x + 1, p(y) \mapsto y \neq x \} \)

Correctness

Compiler

Theory

Implementation

You

1,700 lines
Uniform Substitution is Easy

Correctness

Compiler

Theory

Implementation

You

Simple Proof Rule: \[ \begin{array}{c} \varphi \\ \sigma(\varphi) \end{array} \]

Example:

\[ [x := f]p(x) \leftrightarrow p(f) \]

\[ [x := x + 1]_\sigma \leftrightarrow _\sigma(x + 1) \]

Substitution \( \sigma = \{ f \mapsto x + 1, p(y) \mapsto y \neq x \} \)

1,700 lines
Uniform Substitution is Easy

Simple Proof Rule: \[ \text{US} \quad \frac{\varphi}{\sigma(\varphi)} \]

Example:

\[ \text{US} \quad \frac{[x := f]p(x) \leftrightarrow p(f)}{[x := x + 1]x \neq x \leftrightarrow _{(x + 1)}} \]

Substitution \(\sigma=\{f \mapsto x + 1, \ p(y) \mapsto y \neq x\}\)
Uniform Substitution is Easy

Simple Proof Rule: \[
\text{US} \quad \varphi \\
\sigma(\varphi)
\]

Example:

\[
\text{US} \\
[x := f]p(x) \leftrightarrow p(f) \\
[x := x + 1]x \neq x \leftrightarrow x + 1 \neq x
\]

Substitution \(\sigma = \{f \mapsto x + 1, p(y) \mapsto y \neq x\}\)
Uniform Substitution is Hard

Simple Proof Rule: \[ \text{US} \quad \varphi \quad \sigma(\varphi) \]

Example:
\[ [x := f]p(x) \leftrightarrow p(f) \]
\[ [x := x + 1]x \neq x \leftrightarrow x + 1 \neq x \]
False \leftrightarrow True

Substitution \( \sigma = \{ f \mapsto x + 1, p(y) \mapsto y \neq x \} \)

**Naive Substitution:** UNSOUND!
Uniform Substitution is Hard

Admissibility checks determine when substitution is sound

Example: $\sigma([\alpha]\phi) = [\sigma(\alpha)]\sigma(\phi)$

If $\text{FV}(\sigma) \cap \text{BV}(\alpha) = \emptyset$

$\text{FV}(\sigma) = \{x\}$
$\text{BV}(x := f) = \{x\}$

$\text{FV}(\sigma) \cap \text{BV}(x := f) = \{x\} \neq \emptyset$

Clash Detected
Justify the Theory with Formal Verification

Related Work:

<table>
<thead>
<tr>
<th>Verified Theorem Prover</th>
<th>Verified Using</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOL Light</td>
<td>Self*-verified</td>
</tr>
<tr>
<td>HOL Light</td>
<td>HOL 4</td>
</tr>
<tr>
<td>Milawa</td>
<td>HOL 4</td>
</tr>
<tr>
<td>Subset of Coq</td>
<td>Coq</td>
</tr>
<tr>
<td>Theory of NuPRL</td>
<td>Coq</td>
</tr>
</tbody>
</table>

Let’s do it for hybrid systems!

Correctness

Compiler

Theory

Implementation

You

1,700 lines
Verification: Independent, Trustworthy Justification

Three reasons to trust the theory

1. Correctness
   - Compiler
2. Theory
   - Papers
   - Isabelle (8K/15K)
3. Implementation
   - 1,700 lines
4. You
   - Coq (20K/25K)
Formalization of dL in Isabelle/HOL

(Also applies to Coq version)
What’s in the Formalization?

Syntax

Dynamics

Axioms

Statics

Substitution

Renaming

Proof Checker

Syntax ⇒ Datatypes

\[
\alpha, \beta ::= \\
\quad \alpha \\
\quad | x := \theta \\
\quad | x' := \theta \\
\quad | ?\varphi \\
\quad | (\text{ODE} & \varphi) \\
\quad | (\alpha \cup \beta) \\
\quad | (\alpha; \beta) \\
\quad | \alpha^* \\
\]
What’s in the Formalization?

Syntax

Dynamics

Axioms

Statics

Substitution

Renaming

Proof Checker

Semantic Functions ⇒ Isabelle Functions

\[
[[\alpha \cup \beta]]_I^v = [[\alpha]]_I^v \cup [[\beta]]_I^v
\]

fun HPsem :: "interp → hp → (state * state) set"
where
| "HPsem I (Pvar p) = Programs I p"
| "HPsem I (Assign x t) = \{(v, \omega). \omega = v (x:= (\thetasem I t \nu))\}"
| "HPsem I (DiffAssign x t) = \{(v, \omega). \omega = v (x':= (\thetasem I t \nu))\}"
| "HPsem I (Test \phi) = \{(v, \nu) | \nu. \nu \in fml\_sem I \phi\}"
| "HPsem I (Choice \alpha \beta) = HPsem I \alpha \cup HPsem I \beta"
| "HPsem I (Sequence \alpha \beta) = HPsem I \alpha \circ HPsem I \beta"
| "HPsem I (Loop \alpha) = (HPsem I \alpha)\^*"
What’s in the Formalization?

Syntax
Dynamics
Axioms
Statics
Substitution
Renaming
Proof Checker

Axiom ⇒ Definition + Validity Lemma

\[ p() \rightarrow [a]p() \]

**Definition**
Vaxiom :: formula

where "Vaxiom ≡ ($φ p ()) \rightarrow ([[$α a]]($φ p ()))"

**Theorem**
V_valid: "valid Vaxiom"

by (auto simp: valid_def Vaxiom_def)
Differential Induction is Harder

\[ f \geq 0 \rightarrow [x' = \theta \& \varphi] \text{deriv}(f) \geq 0 \rightarrow [x' = \theta \& \varphi]f \geq 0 \]

\[ f(t) \geq 0 \]

(by Mean-Value Theorem)

\[ f(0) \geq 0 \]

\[ \text{deriv}(f)(s) \geq 0 \text{ (for all } s) \]
Compute FV(e), BV(e) by structural recursion

**Theorem** (Coincidence): Expressions depend only free variables

**Theorem** (Bound Effect): Programs affect only bound variables

Definitions verified, not trusted
Static Semantics Enable Substitution

\[
\begin{align*}
\sigma(\theta \geq \eta) &= \sigma(\theta) \geq \sigma(\eta) \\
\sigma(\forall x \phi) &= \forall x \sigma(\phi) \quad \text{if } \sigma \{x\}\text{-admissible for } \phi \\
\sigma([\alpha]\phi) &= [\sigma(\alpha)]\sigma(\phi) \quad \text{if } \sigma \text{ BV}(\sigma(\alpha))\text{-admissible for } \phi
\end{align*}
\]

\textbf{Syntax}

\textbf{Axioms}

\textbf{Statics}

\textbf{Substitution}

\textbf{Renaming}

\textbf{Proof Checker}

\begin{align*}
\text{primrec} & \quad \text{FOTsubst::"trm } \Rightarrow \text{ FOsubst } \Rightarrow \text{ trm"} \\
\text{primrec} & \quad \text{Tsubst::"trm } \Rightarrow \text{ subst } \Rightarrow \text{ trm"} \\
\text{inductive} & \quad \text{FOTadmit :: "FOsubst } \Rightarrow \text{ trm } \Rightarrow \text{ bool"} \\
\text{inductive} & \quad \text{Tadmit :: "subst } \Rightarrow \text{ trm } \Rightarrow \text{ bool"}
\end{align*}

Formalize primitive recursive variant instead
(First-order substitution for arguments)
Static Semantics Enable Substitution

Lemma subst_fml_valid:
assumes valid:"valid φ"
assumes Fadmit:"Fadmit σ φ"
shows "valid (Fsubst φ σ)"
New: Uniform Renaming

\[
\begin{align*}
\text{UR} & \quad \varphi \\
& \quad \varphi\{x \leftrightarrow y\}
\end{align*}
\]

\[
\text{lemma URename_sound:}
\]
\[
\text{assumes "valid } \varphi \text{
shows "valid (FUrename } x \ y \varphi)"}
\]

No Admissibility!
New: Bound Renaming Renames Destinations

Syntax
Dynamics
Axioms
Statics
Substitution
Renaming
Proof Checker

lemma BRename_sound:
assumes valid:"valid ([[Assign x θ]]φ)"
assumes FVF:"\{y, y'\} \cap FVF \ φ = {}"
shows "valid([[Assign y θ]]FUrename x y φ)"

Yes Admissibility!
lemma B Rename sound:
assumes valid:"valid ([[Assign x θ]]φ)"
assumes FVF:{y, y', x'} ∩ FV φ = {}"
shows "valid([[Assign y θ]]FUrename x y φ)"
Proof Checker - Step Toward Implementation

Syntax
Dynamics
Axioms
Statics
Substitution
Renaming

KeYmaera X

\[ \Gamma \vdash \varphi \quad \Gamma \vdash \psi \]

\[ \Gamma \vdash \varphi \land \psi \]

fun seq2fml :: " sequent ⇒ formula"
where
"seq2fml (ante,succ) = Implies (foldr And ante TT) (foldr Or succ FF)"

type pf = "sequent * derivation"

type rule = "sequent list * sequent"
Examples Validate Proof Checker

**Example 1:** A minimal hybrid system example (~ 100 proof steps)
\[
v \geq 0 & A() \geq 0 \rightarrow \{v' = A(), x' = v \& true\}v \geq 0
\]

**Example 2:** A derived case of differential induction (~ 60 proof steps)
\[
P \rightarrow [a]P' \rightarrow [a]P \quad Q \rightarrow [a]Q' \rightarrow [a]Q
\]

\[P \& Q \rightarrow [a](P' \& Q') \rightarrow [a](P \& Q)\]
Future Work
Future Work: Further Reduce Stack

Isabelle: 8KLOC core + 15 KLOC proof
Coq: 20KLOC core + 25 KLOC proof

Correctness
Compiler
Theory
Implementation
You

Theory
US
1,700 lines

Theory
Isabelle (8K/15K)
Theory
Coq (20K/25K)
Future Work: Further Reduce Stack

Code Extraction: Theory -> Implementation

Verified… one day

Isabelle: 8KLOC core + 15 KLOC proof
Coq: 20KLOC core + 25 KLOC proof
Future Work: Further Reduce Stack

- Correctness
- Compiler
- Theory
- Implementation
- You

Verified Compilation

Theory

Isabelle (8K/15K)

Coq (20K/25K)

Isabelle: 8KLOC core + 15 KLOC proof
Coq: 20KLOC core + 25 KLOC proof
## Future Work: KeYmaera X Integration

### KeYmaera X Integration
- Extract prover core from proof
- Export proof terms from KeYmaera X
- Import and check proof
- Verify theorems of real arithmetic
  - E.g. $\forall x. x^2 \geq 0$
Questions?

Comparison of Formalizations/Presentations

<table>
<thead>
<tr>
<th>Feature</th>
<th>CADE'15 [38]</th>
<th>JAR'16 [39]</th>
<th>KeYmaera X</th>
<th>Isabelle</th>
<th>Coq</th>
</tr>
</thead>
<tbody>
<tr>
<td>systems of ODEs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>multiple argument functions</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>infinite number of identifiers</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>explicit set representation</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>higher-order differentials</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>uniform substitution defns.</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>uniform variable renaming</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>bound variable renaming</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>sequent calculus</td>
<td>[36]</td>
<td>✔</td>
<td>(✓)</td>
<td>(✓)</td>
<td></td>
</tr>
<tr>
<td>DG</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>DG+DE: ODE systems</td>
<td>[37]</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>