A Coq Library for Mechanised First-Order Logic

Dominik Kirst, Johannes Hostert, Andrej Dudenhofner, Yannick Forster, Marc Hermes, Mark Koch, Dominique Larchey-Wendling*, Niklas Mück, Benjamin Peters, Gert Smolka, Dominik Wehr

Saarland University, Saarland Informatics Campus, Germany
*Université de Lorraine, CNRS, LORIA, Vandœuvre-lès-Nancy, France

The Coq Workshop’22, Haifa, Israel, August 12th
Contributors

Dominik Kirst  Johannes Hostert  Andrej Dudenhefner  Yannick Forster  Marc Hermes  Mark Koch

Dominique Larchey-Wendling  Niklas Mück  Benjamin Peters  Gert Smolka  Dominik Wehr
Background

- Merge of several developments concerned with first-order logic
- Published at several venues (CPP, ITP, IJCAR, LFCS, FSCD, TYPES, JAR, JLC, LMCS)
- Design of a core framework general enough to accommodate all results
- Import of developments based on earlier versions of the framework
- Developed in a fork of the Coq library of undecidability proofs (Forster et al. (2020))

Framework

Emerged over several projects with ideas from various contributors:

- Deep embedding of syntax, deduction systems, and semantics
- Combination of well-known techniques, most notably de Bruijn indices
- Tool support for easy interaction by external users

Took most inspiration from O'Connor (2009); Ilik (2010); Herbelin and Lee (2014); Han and van Doorn (2020); Laurent (2021)
Terms and formulas are represented as inductive types $\mathcal{T}$ and $\mathcal{F}$ over a signature $\Sigma = (\mathcal{F}_\Sigma, \mathcal{P}_\Sigma)$:

$$
t : \mathcal{T} ::= x_n | f \bar{t}$$

$$
\varphi, \psi : \mathcal{F} ::= \bot | P \bar{t} | \varphi \rightarrow \psi | \varphi \land \psi | \varphi \lor \psi | \forall \varphi | \exists \varphi
$$

- Syntax modular in type classes for binary connectives and quantifiers
- Common instances ($\rightarrow, \forall$) and ($\rightarrow, \land, \lor, \forall, \exists$) provided
- Availability of $\bot$ regulated via type class flag
- De Bruijn indices encode the number of quantifiers shadowing their relevant binder
- Capture-avoiding instantiation $t[\sigma]$ and $\varphi[\sigma]$ for parallel substitutions $\sigma : \mathbb{N} \rightarrow \mathcal{T}$
Framework: Syntax (Coq)

Context \{\text{sig\_funcs} : \text{func\_signature}\}.

\textbf{Inductive} term : Type :=
\begin{itemize}
\item var : nat \to term
\item func : \text{forall} (f : \text{syms}), \text{vec} term (\text{ar\_syms} f) \to term.
\end{itemize}

Context \{\text{sig\_preds} : \text{pred\_signature}\}.

\textbf{Inductive} falsity\_flag := falsity\_off | falsity\_on.
\textbf{Existing Class} falsity\_flag.

\textbf{Class} operators := \{\text{binop} : Type ; \text{quantop} : Type\}.
\textbf{Context} \{\text{ops} : \text{operators}\}.

\textbf{Inductive} form : falsity\_flag \to Type :=
\begin{itemize}
\item falsity : form falsity\_on
\item atom \{b\} : \text{forall} (P : \text{preds}), \text{vec} term (\text{ar\_preds} P) \to form b
\item bin \{b\} : \text{binop} \to form b \to form b \to form b
\item quant \{b\} : \text{quantop} \to form b \to form b.
\end{itemize}
Framework: Deduction Systems

Proof rules are represented as inductive predicates relating a context $\Gamma$ to a formula $\varphi$:

\[
\begin{array}{llll}
\Gamma[\uparrow] \vdash \varphi & \quad \text{(AI)} \\
\Gamma \vdash \forall \varphi & \quad \text{(AE)} \\
\Gamma \vdash \varphi[t] & \quad \text{(EI)} \\
\Gamma \vdash \exists \varphi & \quad \text{(EE)}
\end{array}
\]

- Quantifier rules use shifted contexts $\Gamma[\uparrow]$ so that $x_0$ acts as canonical free variable
- Trivialises structural properties like substitutivity and weakening
- Availability of classical rules regulated via type class flag
- Similar representation of sequent calculi and other systems
Framework: Deduction Systems (Coq)

Context \{\text{sig_funcs} : \text{funcs_signature}\}.

Context \{\text{sig_preds} : \text{preds_signature}\}.

Reserved Notation 'A \vdash phi' (at level 61).

Inductive peirce := class | intu.

Existing Class peirce.

Inductive prv : forall (ff : falsity_flag) (p : peirce), list form -> form -> Prop :=
\[
\begin{align*}
&\text{II} \{ff\} \{p\} \ A \ \phi \ \psi : \phi : : A \vdash \psi \rightarrow A \vdash \phi \rightarrow \psi \\
&\text{IE} \{ff\} \{p\} \ A \ \phi \ \psi : A \vdash \phi \rightarrow \psi \rightarrow A \vdash \phi \rightarrow A \vdash \psi \\
&\text{AllI} \{ff\} \{p\} \ A \ \phi : \text{map} (\text{subst_form} \uparrow) A \vdash \phi \rightarrow A \vdash \forall \phi \\
&\text{AllE} \{ff\} \{p\} \ A \ t \ \phi : A \vdash \forall \phi \rightarrow A \vdash \phi[t..] \\
&\text{Exp} \{p\} \ A \ \phi : \text{prv} \ p \ A \ \text{falsity} \rightarrow \text{prv} \ p \ A \ \phi \\
&\text{Ctx} \{ff\} \{p\} \ A \ \phi : \text{phi el} A \rightarrow A \vdash \phi \\
&\text{Pc} \{ff\} \ A \ \phi \ \psi : \text{prv} \ \text{class} \ A (((\phi \rightarrow \psi) \rightarrow \phi) \rightarrow \phi) \rightarrow \phi
\end{align*}
\]
where 'A \vdash phi' := (prv _ A phi).
Framework: Semantics

Tarski models $\mathcal{M}$ are represented as a domain type $D$ and symbol interpretations:

- Interpretation of terms and formulas based on assignments $\rho : \mathbb{N} \rightarrow D$
- Term evaluation $\hat{\rho} t$ defined recursively, main rule $\hat{\rho} (f \vec{t}) := f^\mathcal{M} (\hat{\rho} \vec{t})$
- Formula satisfaction $\rho \models \varphi$ defined recursively, main rule $\rho \models P \vec{t} := P^\mathcal{M} (\hat{\rho} \vec{t})$
- Induces the logical entailment relation $\Gamma \models \varphi$
Framework: Semantics (Coq)

Context `{domain : Type}.`

Class interp := B_I
{ i_func : forall f : syms, vec domain (ar_syms f) -> domain ;
  i_atom : forall P : preds, vec domain (ar_preds P) -> Prop ; }.

Definition env := nat -> domain.

Context `{I : interp}.`

Fixpoint eval (rho : env) (t : term) : domain := match t with
| var s => rho s
| func f v => i_func (Vector.map (eval rho) v) end.

Fixpoint sat {ff : falsity_flag} (rho : env) (phi : form) : Prop := match phi with
| atom P v => i_atom (Vector.map (eval rho) v)
| falsity => False
| bin Impl phi psi => sat rho phi -> sat rho psi
| quant All phi => forall d : domain, sat (d .: rho) phi end.

The Coq Workshop’22
A Coq Library for Mechanised First-Order Logic
Framework: Axiom Systems

Concrete axiom systems $\mathcal{A}$ are modelled as predicates of formulas over a specific signature.

For the example of Peano arithmetic (PA), we instantiate to the arithmetical signature

$$(O, S_-, _, +_, _\times_; _\equiv_$$

and collect the usual axioms, with the induction scheme represented as all instances of

$$\varphi[O] \rightarrow (\forall x. \varphi[x] \rightarrow \varphi[Sx]) \rightarrow \forall x. \varphi[x].$$

- Include fragments of PA like Robinson’s Q, also several variants of ZF set theory
- Equality $\equiv$ seen as axiomatised symbol of the signature rather than a logical primitive
- Axiom systems $\mathcal{A}$ induce relatives deductive and semantic theories $\mathcal{A} \vdash \varphi$ and $\mathcal{A} \models \varphi$
Framework: Tool Support

Tools presented at last year’s Coq Workshop (Hostert et al. (2021)):

- HOAS-input language
  - Concrete formulas can be written with Coq binders instead of de Bruijn indices
  - Eases interaction with the syntax

- Proof mode (inspired by Iris proof mode, Krebbers et al. (2017))
  - Tactic and notation layer hiding the proof rules
  - Eases interaction with the deduction systems

- Reification tactic (employing MetaCoq, Sozeau et al. (2020))
  - Extracts first-order formulas from Coq predicates
  - Eases interaction with the semantics
Framework: Tool Support (Proof Mode)

```coq
205  rewrite (ax_add_zero y).
206  fapply ax_refl.
207  - fintros "x" "IH" "y".
208  rewrite (ax_add_rec (\ y) x).
209  rewrite ("IH" y).
210  rewrite (ax_add_rec y x). fapply ax_refl.
211  Qed.

213  Lemma add_comm :
214  FAI |- \forall x y, x \cdot y == y \cdot x.
215  (*Proof.*
216  fstart. fapply ((ax_induction (\ x, \forall y, x \cdot y == y \cdot x))).
217  - fintros.
218  rewrite (ax_add_zero x).
219  rewrite (add_zero_r x).
220  fapply ax_refl.
221  - fintros "x" "IH" "y".
222  rewrite (add_succ_r y x).
223  rewrite \IH x.
224  rewrite (ax_add_rec y x).
225  fapply ax_refl.
226  Qed.

228  Lemma pa_eq_dec :
229  FAI |- \forall x y, (x == y) v \neg (x == y).
230  (Proof.
231  fstart.
232  fapply ((ax_induction (\ x, \forall y, (x == y) v \neg (x == y)))).
233  --fapply
```

```
1 goal
p : peirce
x, y : term

 FAI
"IH" : \forall x0, x'[\tau] \cdot x0 == x0 \cdot x'[\tau]

[\sigma x \cdot y == y \cdot \sigma x]
```

Framework: Tool Support (Reification Tactic)

```
Lemma add_comm a b : a ⊕ b ⊕ b ⊕ a = a ⊕ b ⊕ a.
Proof.
  elim a using PA_induction.
  - represent.
  - eapply ieq_trans. 1:apply (add_zero_l (is b)).
    eapply ieq_congr_succ, ieq_sym, add_zero_l.
  - intros d IH.
    eapply ieq_trans. 1:apply (add_succ_l d (is b)).
    eapply ieq_congr_succ, eapply ieq_trans.
    + apply IH.
    + eapply ieq_sym, add_succ_l.
  Qed.

Lemma add_comm a b : a ⊕ b ⊕ i = b ⊕ i ⊕ a.
Proof.
  elim a using PA_induction.
  - represent.
  - eapply ieq_trans.
    + eapply (add_zero_l b).
    + eapply ieq_sym, (add_zero_r b).
  - intros a' IH.
    eapply ieq_trans. 2:eapply ieq_trans.
    + eapply (add_succ_l a' b).
    + eapply ieq_congr_succ, IH.
    + eapply ieq_sym, add_succ_r.
  Qed.
```

Framework: Evolution

Forster, Kirst, Smolka (2019) at CPP’19:
- Concrete signature, small logical fragment, named variables
- Among the initial projects constituting the undecidability library

Forster, Kirst, and Wehr (2021) at LFCS’20/JLC’21:
- Arbitrary signature, both logical fragments, de Bruijn encoding
- Use of Autosubst 2 (Stark et al. (2019)) for de Bruijn boilerplate

Kirst and Larchey-Wendling (2020) at IJCAR’20/LMCS’22:
- Parametric in logical fragment, merged into undecidability library
- Refrains from Autosubst 2 mostly due to dependency on function extensionality

Kirst and Hermes (2021) at ITP’21/JAR’22:
- Compromise of previous developments, merged into undecidability library
- Still no explicit code generation with Autosubst 2 but identical design
## Framework: Comparison

<table>
<thead>
<tr>
<th>Development</th>
<th>Signature</th>
<th>Binding</th>
<th>(AI)-Rule</th>
<th>Weakening</th>
</tr>
</thead>
<tbody>
<tr>
<td>O’Connor</td>
<td>arbitrary</td>
<td>named</td>
<td>side-condition</td>
<td>n.a.</td>
</tr>
<tr>
<td>Ilik</td>
<td>monadic</td>
<td>locally-nameless</td>
<td>co-finite</td>
<td>easy</td>
</tr>
<tr>
<td>Herbelin et al.</td>
<td>dyadic</td>
<td>locally-named</td>
<td>side-condition</td>
<td>needs renaming</td>
</tr>
<tr>
<td>Han and van Doorn</td>
<td>arbitrary</td>
<td>de Bruijn</td>
<td>shifting</td>
<td>easy</td>
</tr>
<tr>
<td>Laurent</td>
<td>full</td>
<td>anti-loc.-namel.</td>
<td>shifting</td>
<td>easy</td>
</tr>
<tr>
<td>Our framework</td>
<td>arbitrary</td>
<td>de Bruijn</td>
<td>shifting</td>
<td>easy</td>
</tr>
</tbody>
</table>
Overview:

- Many metamathematical results: completeness, undecidability, incompleteness
- Many interdependencies, based on the Coq library of undecidability proofs
- Many possible projects/collaborations: syntactic cut-elimination, Hilbert systems, Löwenheim-Skolem theorems, resolution, tableaux, constructible hierarchy, ... 

Shared methods:

- Constructive meta-theory where possible
- Synthetic approach to computability results
In which situations does $\Gamma \models \varphi$ imply $\Gamma \vdash \varphi$?

Based on the publication Forster et al. (2021):

- Constructively extremely subtle topic, extensive related literature
- Model-theoretic semantics (Tarski/Kripke) yield connections to MP and LEM
- Fully constructive proofs for algebraic and dialogical semantics
Which decision problems of first-order logic are undecidable?

Library includes all common undecidability results:

- Validity, provability, satisfiability (Forster et al. (2019))
- Finite satisfiability (Kirst and Larchey-Wendling (2020))
- Strongest versions regarding binary signatures (Hostert et al. (2022))
- Several variants of PA and ZF (Kirst and Hermes (2021))
- Post’s theorem on the arithmetical hierarchy (Kirst et al. (2022))
Which axiom systems $\mathcal{A}$ satisfy $\mathcal{A} \vdash \varphi$ or $\mathcal{A} \vdash \neg \varphi$ for all $\varphi$?

Library exploiting the connection to undecidability:

- Incompleteness of several variants of PA and ZF (Kirst and Hermes (2021))
- Essential incompleteness of Q (Peters and Kirst (2022))
- Tennenbaum’s theorem on computable models of PA (Hermes and Kirst (2022))
Current Status: Overview

- Completed core framework ✓
- Main completeness, undecidability, and incompleteness results imported ✓
- Essential incompleteness, Tennenbaum’s theorem, and Post’s theorem pending ✓
- Signature transformations and further computability results planned to be imported ✗
- Total: about 25k lines of code (8500 spec, 15500 proofs, 1000 comments), 110 files
# Current Status: Structure

<table>
<thead>
<tr>
<th>Component</th>
<th>Task Details</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetics</td>
<td>Rename Deduction -&gt; ND to prepare for Sequent</td>
<td>2 months ago</td>
</tr>
<tr>
<td>Completeness2</td>
<td>Start using Asimpi in places</td>
<td>5 days ago</td>
</tr>
<tr>
<td>Deduction</td>
<td>Finish Kripke Completeness, work on atom substitution</td>
<td>10 days ago</td>
</tr>
<tr>
<td>Incompleteness</td>
<td>Tarski Constructions ported</td>
<td>2 months ago</td>
</tr>
<tr>
<td>Proofmode</td>
<td>Fix Proofmode MinZF demo</td>
<td>4 days ago</td>
</tr>
<tr>
<td>Refutation</td>
<td>Tarski Constructions ported</td>
<td>2 months ago</td>
</tr>
<tr>
<td>Semantics</td>
<td>Add validity facts, remove &quot;not_strong&quot; preservation</td>
<td>10 days ago</td>
</tr>
<tr>
<td>Sets</td>
<td>Rename Deduction -&gt; ND to prepare for Sequent</td>
<td>2 months ago</td>
</tr>
<tr>
<td>Syntax</td>
<td>Start using Asimpi in places</td>
<td>5 days ago</td>
</tr>
<tr>
<td>Undecidability</td>
<td>Start using Asimpi in places</td>
<td>5 days ago</td>
</tr>
<tr>
<td>Utilis</td>
<td>Port FOLP reduction, refactor PCP decidabilities</td>
<td>2 months ago</td>
</tr>
<tr>
<td>FragmentSyntax.v</td>
<td>Start using Asimpi in places</td>
<td>5 days ago</td>
</tr>
<tr>
<td>FullSyntax.v</td>
<td>Start using Asimpi in places</td>
<td>5 days ago</td>
</tr>
</tbody>
</table>
Current Status: Pending Contributions

**Filters**
- is:pr is:open

**2 Open**
- **WIP: Add further incompleteness results**
  - #4 opened 4 days ago by bn-peters • Draft • 4 tasks
- **Add PrenexNormalForm and ArithmeticalHierarchy**
  - #3 opened 6 days ago by SohnyBohny

**Filters**
- is:issue is:open

**1 Open**
- **Proofmode bugs**
  - #2 opened 8 days ago by bn-peters
Current Status: Activity

History for coq-library-undecidability / theories / FOL

Committed on Jul 22, 2022

Fix Proofmode MinZF demo
- mark-koch committed 4 days ago

Committed on Jul 21, 2022

Remove Require in section
- JoJoDeveloping committed 5 days ago

Start using Asimpi in places
- JoJoDeveloping committed 5 days ago

Merge branch 'fol-library' of github.com:dominik-kirst/coq-library-und...  
- mark-koch committed 5 days ago

Fix ProofMode
- mark-koch committed 5 days ago

Committed on Jul 20, 2022

Working and somewhat efficient ASimpi tactic
- JoJoDeveloping committed 6 days ago

The Coq Workshop'22
A Coq Library for Mechanised First-Order Logic
August 12th, 2022
Future Plans

1. Finish importing the remaining developments
2. Possible round of refactoring (proof mode performance, falsity flags)
3. Decide on a plan how to integrate with the undecidability library
4. Follow the release cycle of the undecidability library, itself following Coq
5. Possible timeline: opam package for Coq 8.16, add to Coq CI for Coq 8.17
6. At any time: help new users get started and contribute their developments!

Thanks for listening!


Han, J. and van Doorn, F. (2020). A formal proof of the independence of the continuum hypothesis. In *9th International Conference on Certified Programs and Proofs*.


Bibliography II


