Compiling graphical actions with deep inference

Pablo Donato  Benjamin Werner  Kaustuv Chaudhuri

Partout team — LIX

Scalp 2023

CIRM
Goal: Make proof assistants easier to use

- Intuitive and discoverable for newcomers
- Productive and beautiful for experts
Goal: Make proof assistants easier to use

- Intuitive and discoverable for newcomers
- Productive and beautiful for experts

For now, focus on common logical heart:

*Intuitionistic First-Order Logic (iFOL)*
Goal: Make proof assistants easier to use

- Intuitive and discoverable for newcomers
- Productive and beautiful for experts

For now, focus on common logical heart:

**Intuitionistic First-Order Logic (iFOL)**

Disclaimer: WIP, still at an experimental stage...
GRAPHICAL PROOFS
“A demo is worth a thousand words...”
Paradigm

- Fully graphical: no textual proof language
- Both spatial and temporal:
  \[
  \text{proof} = \text{gesture sequence}
  \]
- Different modes of reasoning with a single “syntax”:
  
  Click \iff introduction/elimination
  Drag-and-Drop \iff backward/forward

*Sound and complete for iFOL!*
INTEGRATION WITH COQ
Architecture

Coq

Engine

Kernel

Plugin
  OCaml

IDE

User

Interface

Backend
  js_of_ocaml

Frontend
  HTML/CSS/JS

Server
  NodeJS

Actema
  Electron

Database
  File system
$\mathcal{G}$: goal list

Diagram:

- Kernel
- Plugin
- Source

Diagram edge:

1

Diagram note:

Kernel Plugin Source
$\mathcal{G}$ : goal list
$$\mathcal{G} : \text{goal list}$$

$$\text{translate}(\mathcal{G}) : \text{agoal list}$$
Kernel \xrightarrow{1} \text{Plugin} \xrightarrow{2} \text{Source} \xrightarrow{3}

\( \mathcal{G} : \text{goal list} \quad \text{translate}(\mathcal{G}) : \text{agoal list} \quad \mathcal{A} : (\text{action} \times \text{int}) \text{ list} \)
$G$ : goal list

$\text{translate}(G) : \text{agoal list}$

$\text{compile}(A) : \text{tactic}$

$A : \text{(action * int) list}$
translate(\mathcal{G}) : \text{aggoal list}

\mathcal{A} : (\text{action} \times \text{int}) \text{ list}
Protocol (non-interactive)

\[(G, \text{id}) : \text{agoal} \ast \text{string}\]
Protocol (non-interactive)

\[(G, id) : \text{agoal} * \text{string}\]

\[\mathcal{A} : (\text{action} * \text{int}) \text{ list}\]

Plugin

1

2

Graphical Proof Database
Protocol (non-interactive)

\[ (G, \text{id}) : \text{goal} \times \text{string} \]

\[ A : (\text{action} \times \text{int}) \text{ list} \]
Protocol (interactive)

$\mathcal{G}$ : goal list

Plugin \quad 1 \quad Actema

User

7/20
Protocol (interactive)

\[ G : \text{goal list} \quad \text{render}(G) : \text{HTMLDom} \]

1. Plugin
2. Actema

User
Protocol (interactive)

\[ G : \text{goal list} \]
\[ \text{render}(G) : \text{HTMLDom} \]

1. Plugin
2. Actema
3. User

\[ S : \text{gesture start} \]
Protocol (interactive)

\[ G : \text{goal list} \]
\[ \text{render} (\text{actions}(S)) : \text{HTMLDom} \]

1. Plugin
2. Actema
3. User
4. S : gesture start
Protocol (interactive)

\[ G : \text{goal list} \]

\[ \text{render}(\text{actions}(S)) : \text{HTMLDom} \]

User

Plugin

Actema

1

4

5

E : gesture end
Protocol (interactive)

\[ G : \text{goal list} \quad \text{render} \left( \text{actions}(S) \right) : \text{HTMLDom} \]

\[ (A, n) : \text{action} \times \text{int} \quad E : \text{gesture end} \]
Protocol (interactive)

\[\mathcal{G} : \text{goal list}\]

\[\text{render}(\text{actions}(S)) : \text{HTMLDom}\]

Plugin \(\xrightarrow{\text{act}}\) Actema

\((A, n) : \text{action} \times \text{int}\)

User

\(E : \text{gesture end}\)
Deep inference semantics
• Socrates example:

\[
\begin{align*}
\text{Backward} & \iff \text{apply } H1 \\
\text{Forward} & \iff \text{apply } H1 \text{ in } H2
\end{align*}
\]

• \( A \land B \vdash B \land (A \lor C) \land D \) is trickier...

\[
\frac{
\frac{A, B \vdash A}{A \lor R_1} \quad \frac{A, B \vdash A \lor C \quad A, B \vdash D}{A, B \vdash (A \lor C) \land D} \quad \frac{A, B \vdash B}{A \land B \vdash B \land (A \lor C) \land D} \quad \frac{A, B \vdash (A \lor C) \land D}{A \land B \vdash (A \lor C) \land D}
}{A \land B \vdash (A \lor C) \land D}
\]

destruct \( H \) as \([HA \; HB]\).
split.
* admit.
* split.
  - left. assumption.
  - admit.
Idea: instead of *destroying* connectives, *switch* them

**switch**

\[
\begin{align*}
A \land B & \vdash B \land (A \lor C) \land D \\
B \land (A \land B) & \vdash (A \lor C) \land D \\
B \land (A \land B) & \vdash A \lor C \land D \\
B \land ((A \land B) \vdash A) \lor C) & \land D
\end{align*}
\]

**identity**

\[
\begin{align*}
B \land ((B \Rightarrow (A \vdash A))) & \lor C) \land D \\
B \land ((B \Rightarrow T) \lor C) & \land D \\
B \land (T \lor C) & \land D \\
B & \land T \land D \\
B & \land D
\end{align*}
\]
1. **Unify** linked subformulas

2. **Instantiate** unified variables

3. **Switch** uninstantiated quantifiers

\[ \exists y. \forall x. R(x, y) \vdash \forall x'. \exists y'. R(x', y') \]

\[ \forall y. (\forall x. R(x, y) \vdash \forall x'. \exists y'. R(x', y')) \]

\[ \forall y. \forall x'. (\forall x. R(x, y) \vdash \exists y'. R(x', y')) \]

\[ \forall y. \forall x'. (\exists y. R(x', y) \vdash R(x', y)) \]

\[ \forall y. \forall x'. (\forall y. R(x', y) \vdash R(x', y)) \]

\[ \forall y. \forall x'. \top \]

\[ \top \]

\[ x \xrightarrow{} x' \]

\[ y \xleftarrow{} y' \]

\[ \ast \]
1. **Unify** linked subformulas
2. **Instantiate** unified variables
3. **Switch** uninstantiated quantifiers

\[
\forall x'. \exists y'. R(x', y') \vdash \exists y. \forall x. R(x, y)
\]
1. **Unify** linked subformulas
2. **Check** for $\forall \exists$ dependency cycles
3. **Instantiate** unified variables
4. **Switch** uninstantiated quantifiers

$$\forall x'. \exists y'. R(x', y') \vdash \exists y. \forall x. R(x, y)$$
Rewriting equalities

Add 4 rules $\Rightarrow$ rewrite for free!

\[
\begin{align*}
  t = u \vdash A & \quad \Rightarrow \quad A \{t := u\} &
  t = u \vdash A & \quad \Rightarrow \quad A \{u := t\} \\
  t = u \ast A & \quad \Rightarrow \quad A \{t := u\} &
  t = u \ast A & \quad \Rightarrow \quad A \{u := t\}
\end{align*}
\]

Compositional with semantics of connectives:

- **Quantifiers:** rewrite modulo unification
- **Implication:** conditional rewrite
- **Arbitrary** combinations are possible:

\[
\begin{align*}
  \forall x. x \neq 0 \Rightarrow f(x) = g(x) & \vdash \exists y. A(f(y)) \lor B(y) \\
  \Rightarrow^* \exists y. (y \neq 0 \land A(g(y))) \lor B(y)
\end{align*}
\]
• **Click** actions: standard Coq tactics

• **Drag-and-Drop** actions: \(\sim3000\) lines of Coq/Ltac
  
  • **Deep embedding** of goal \(\Gamma \vdash C\) in FOL
  
  • Subterm selection as **paths**, i.e. `list nat`

• **Computational reflection** for *deep inference* semantics [Donato et al. (2022b)]
  
  • Backward: new conclusion \(C'\)
  
  • Forward: new hypothesis \(A\)

• Final tactic = apply **soundness** theorem
  
  • Backward: \(\Gamma \Rightarrow C' \Rightarrow C\)
  
  • Forward: \(\Gamma \Rightarrow A\)
Conclusion
What are the most useful usecases of Actema?

- Proof exploration
- Educational setting
What were the *infrastructure* challenges/solutions?

- Interaction protocol that can handle *arbitrary goals and tactics* (still a WIP, because of FOL and notations)
- Generic protocol *independent of the specifics of Coq* (simpler with FOL)
- **Portable API with reusable boilerplate** for serialization on both sides (atdgen)
- **Linking external libraries** in Coq plugin, for serialization/HTTP (currently falls out of dune capabilities, need coq_makefile)
Related works (non-exhaustive)

- **Proof-by-Pointing** [Bertot et al. (1994)]

- **Subformula linking** [Chaudhuri (2013), Chaudhuri (2021)]

- **ProofWidgets** [Ayers et al. (2021)]
  - Framework for user-defined graphical notations
  - PA serves the GUI, instead of requesting from it
  - Relies on Lean’s metaprogramming capabilities
Future works

For more complex theories:

- Support arbitrary **Coq notations** (and more?)
- Selection-based **lemma search**
- Extend to **HOL**

For proof evolution:

- Translate graphical proof into *readable* and *reusable* **tactic invocations** (avoid paths)
- Replay/Edit graphical proof through **animations**
Future works

For more complex theories:

- Support arbitrary Coq notations (and more?)
- Selection-based lemma search
- Extend to HOL

For proof evolution:

- Translate graphical proof into readable and reusable tactic invokations (avoid paths)
- Replay/Edit graphical proof through animations

Thank you!
