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)**
Context

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:

  \[ \text{proof} \quad \equiv \quad \text{gesture sequence} \]

- Different modes of reasoning with a single "syntax":

  \begin{align*}
  \text{Click} & \iff \text{introduction/elimination} \\
  \text{Drag-and-Drop} & \iff \text{backward/forward}
  \end{align*}

  \text{Sound and complete for iFOL!}
INTEGRATION WITH COQ
$G$ : goal list

Kernel Plugin Source

1
$\mathcal{G}$ : goal list

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

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

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

- **Kernel**
- **Plugin**
- **Source**

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

Kernel

\[ 1 \]

\[ 4 \]

Plugin

\[ 2 \]

\[ 3 \]

Source

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

\[ \text{compile}(\mathcal{A}) : \text{tactic} \]
Protocol

\[ \text{translate}(\mathcal{G}) : \text{agoal list} \]

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

Plugin

Source
Protocol (non-interactive)

Plugin

Graphical Proof Database
Protocol (non-interactive)

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

1. Plugin

Graphical Proof Database
Protocol (non-interactive)

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

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

1. Plugin
2. Graphical Proof Database

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

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

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

$G$ : goal list

Plugin ➔ 1 ➔ Actema

User
Protocol (interactive)

$G$: goal list

render($G$): HTMLDom

1. Plugin
2. Actema

User
Protocol (interactive)

$G$: goal list

$\text{render}(G)$: HTMLDom

$S$: gesture start
Protocol (interactive)

$G$: goal list

render(actions(S)): HTMLDom

S: gesture start
Protocol (interactive)

$\mathcal{G}$ : goal list
render(actions(S)) : HTMLDom

Plugin → Actema → User

1 4 5

E : gesture end
$\mathcal{G}$ : goal list

render($\text{actions}(S)$) : HTMLDom

(A, n) : action * int

E : gesture end

Plugin  Actema  User
Protocol (interactive)

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

(A, n) : action * int

E : gesture end
Deep Inference Semantics
• Socrates example:

Backward \iff \text{apply } H1

Forward \iff \text{apply } H1 \text{ in } H2

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

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

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

**switch**

\[ A \land B \vdash \fbox{B \land (A \lor C)} \land D \]

\[ B \land (A \land B \vdash \fbox{A \lor C}) \land D \]

\[ B \land (A \land B \vdash \fbox{A \lor C}) \land D \]

\[ B \land (A \land B \vdash \text{A} \lor C) \land D \]

**identity**

\[ B \land ((B \Rightarrow \text{A} \vdash \text{A}) \lor C) \land D \]

\[ B \land ((B \Rightarrow \top) \lor C) \land D \]

\[ B \land ((B \Rightarrow \top) \lor C) \land D \]

**unit elimination**

\[ B \land \top \land D \]

\[ B \land D \]
1. **Unify** linked subformulas

2. **Instantiate** unified variables

3. **Switch** uninstantiated quantifiers

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

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

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

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

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

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

\[
\top
\]
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

\[ \forall x'. \exists y'. R(x', y') \vdash \exists y. \forall x. R(x, y) \]
Add 4 rules → rewrite for free!

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

Compositional with semantics of connectives:

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

\[
\forall x. x \neq 0 \Rightarrow f(x) = g(x) \vdash \exists y. A(f(y)) \lor B(y)
\]

\[
\triangleright^* \quad \exists y. (y \neq 0 \land A(g(y))) \lor B(y)
\]
Compiling actions

(B. Werner, K. Chaudhuri)

- **Click** actions: standard Coq tactics
- **Drag-and-Drop** actions: \( \sim 3000 \) lines of Coq/Ltac
  - Deep embedding of goal \( \Gamma \vdash C \) in FOL
  - Subterm selection as *paths*, i.e. \textit{list} \textit{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 invokations (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!
REFERENCES

