A Universal Proof Framework

Ali Assaf

INRIA Paris-Rocquencourt, Deducteam
Outline

Introduction

A closer look

My work
def binary_search(A, x, left, right):
    middle = (left + right) / 2
    if A[middle] > x:
        return binary_search(A, x, left, middle)
    elif A[middle] < x:
        return binary_search(A, x, middle, right)
    else:
        return middle

A = [2, 3, 5, 7]
binary_search(A, 7, 0, 3)
# Oops!
def binary_search(A, x, left, right):
    middle = (left + right) / 2
    if A[middle] > x:
        return binary_search(A, x, left, middle)
    elif A[middle] < x:
        return binary_search(A, x, middle, right)
    else:
        return middle

A = [2, 3, 5, 7]
binary_search(A, 7, 0, 3) # Oops!
Whack-A-Bug

def binary_search(A, x, left, right):
    middle = (left + right) / 2
    if A[middle] > x:
        return binary_search(A, x, left, middle - 1)
    elif A[middle] < x:
        return binary_search(A, x, middle + 1, right)
    else:
        return middle

A = [2, 3, 5, 7]
binary_search(A, 7, 0, 3)  # Oops!
Proof of correctness

Theorem
If \( x \in A \) and \( left \leq right \) and \( A[left] \leq x \leq A[right] \) then \( A[binary\_search(A, x, left, right)] = x \).

Proof.
By induction on \( (right - left) \) ...
Formal proof

Formal language: $\forall, \exists, \implies, \ldots$

Deductive system:

\[
\frac{\Gamma, A \vdash B}{\Gamma \vdash A \implies B}
\]

\[
\frac{\Gamma \vdash A \implies B \quad \Gamma \vdash A}{\Gamma \vdash B}
\]
Proof systems

Theoretical systems
► First-order logic
► Higher-order logic
► Calculus of constructions
Proof systems

Theoretical systems
- First-order logic
- Higher-order logic
- Calculus of constructions

Implementations
- Twelf
- HOL
- Coq
Why use them?

In software engineering: eliminate more bugs
  ▶ CompCert project
  ▶ L4.verified project

In mathematics: prove harder theorems
  ▶ 4-color theorem
  ▶ Kepler theorem
Outline

Introduction

A closer look

My work
Two problems

Proof checking:
- Given a proposition $A$ and a proof $D$, does $D$ prove $A$?
- $Check(D, A) = Yes$ or $No$
Two problems

Proof checking:
- Given a proposition $A$ and a proof $D$, does $D$ prove $A$?
- $\text{Check}(D, A) = \text{Yes}$ or $\text{No}$

Proof search:
- Given a proposition $A$, does there exist a proof $D$ of $A$?
- $\text{Search}(A) = D$ or $\text{None}$
Theorem foo: forall A B C : Prop,  
    (A -> B) -> (B -> C) -> (A -> C) :=  

fun A B C proof_of_A_B proof_of_B_C =>  
fun proof_of_A =>  

let proof_of_B := (proof_of_A_B proof_of_A) in  
let proof_of_C := (proof_of_B_C proof_of_B) in  
proof_of_C.
Writing proofs

Theorem foo: forall A B C : Prop, (A -> B) -> (B -> C) -> (A -> C) :=

fun A B C proof_of_A_B proof_of_B_C =>
fun proof_of_A =>

let proof_of_B := (proof_of_A_B proof_of_A) in
let proof_of_C := (proof_of_B_C proof_of_B) in
proof_of_C.
Theorem foo: forall A B C : Prop, 
   (A -> B) -> (B -> C) -> (A -> C) :=

fun A B C proof_of_A_B proof_of_B_C =>
fun proof_of_A =>

let proof_of_B := (proof_of_A_B proof_of_A) in
let proof_of_C := (proof_of_B_C proof_of_B) in
proof_of_C .
Proof development

1. Write your proof
2. Call the proof checker
3. Checker answers *Yes* or *No* and gives you an error
4. If answer is *No*, go back to step 1

Sounds familiar?
Proof development

1. Write your proof
2. Call the proof checker
3. Checker answers *Yes* or *No* and gives you an error
4. If answer is *No*, go back to step 1

Sounds familiar?
Program development

1. Write your programs
2. Call the compiler
3. Compiler answers Yes or No and gives you an error
4. If answer is No, go back to step 1

Sounds familiar?
Proofs are programs!

Curry-Howard correspondence

<table>
<thead>
<tr>
<th>Proof</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposition</td>
<td>Type</td>
</tr>
<tr>
<td>$A \iff B$</td>
<td>$A \rightarrow B$</td>
</tr>
<tr>
<td>Proof checking</td>
<td>Type checking</td>
</tr>
</tbody>
</table>
Proofs are programs!

Proof system = Programming language
Outline

Introduction

A closer look

My work
A Zoology of proof systems

- Twelf, HOL, Coq, Isabelle, PVS, NuPRL, Mizar, Agda, ProofPower, Lego, ACL2, ...
- Different properties
  - Intuitionistic/Classical
  - Top-down/Bottom-up
  - Sequents/Proof terms
  - ...
### Top 100 theorems

<table>
<thead>
<tr>
<th>System</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOL Light</td>
<td>86</td>
</tr>
<tr>
<td>Mizar</td>
<td>57</td>
</tr>
<tr>
<td>Isabelle</td>
<td>51</td>
</tr>
<tr>
<td>Coq</td>
<td>49</td>
</tr>
<tr>
<td>ProofPower</td>
<td>42</td>
</tr>
<tr>
<td>nqthm/ACL2</td>
<td>18</td>
</tr>
<tr>
<td>PVS</td>
<td>16</td>
</tr>
<tr>
<td>NuPRL/MetaPRL</td>
<td>8</td>
</tr>
</tbody>
</table>

The need for interoperability

- CompCert project: 50,000 lines (Coq)
- Four-color theorem: 60,000 lines (Coq)
- Jordan curve theorem: 75,000 lines (HOL)
- Odd order theorem: 170,000 lines (Coq)
- L4.verified project: 200,000 lines (Isabelle)
The need for interoperability

- CompCert project: 50,000 lines (Coq)
- Four-color theorem: 60,000 lines (Coq)
- Jordan curve theorem: 75,000 lines (HOL)
- Odd order theorem: 170,000 lines (Coq)
- L4.verified project: 200,000 lines (Isabelle)

Can we reuse them?
Logical Framework

**Idea:** Express all these proofs in a common logical framework.
Dedukti

- Dedukti: means “to deduce” in Esperanto
- Minimal formalism: $\lambda\Pi$-calculus modulo $=$
  First-order logic $+$ rewriting
Without rewriting

\[
\begin{align*}
4 & = 4 \\
4 + 1 & = 5 \\
4 + 2 & = 6 \\
4 + 3 & = 7 \\
4 + 4 & = 8
\end{align*}
\]
With rewriting

\[ x + 0 \quad \rightarrow \quad x \]
\[ x + (y + 1) \quad \rightarrow \quad (x + 1) + y \]
With rewriting

\[
\begin{align*}
  x + 0 & \quad \rightarrow \quad x \\
  x + (y + 1) & \quad \rightarrow \quad (x + 1) + y \\
  4 + 4 & \quad \rightarrow^* \quad 8
\end{align*}
\]
With rewriting

\[ x + 0 \quad \longrightarrow \quad x \]
\[ x + (y + 1) \quad \longrightarrow \quad (x + 1) + y \]

\[ 4 + 4 \quad \longrightarrow^{*} \quad 8 \]

\[ 8 = 8 \]
\[ 4 + 4 = 8 \]
Adding rewrite rules extends the logic.
Adding rewrite rules extends the logic.

Can express the proofs of a system $P$ in Dedukti.

\[ \Gamma \vdash_P A \implies \phi(\Gamma) \vdash_{\lambda\Pi R} \phi(A) \]
Encodings

- Adding rewrite rules extends the logic.
- Can express the proofs of a system $P$ in Dedukti.

\[
\begin{align*}
\Gamma \vdash_P A & \iff \phi(\Gamma) \vdash_{\lambda \Pi R} \phi(A) \\
\Gamma \vdash_P A & \iff \phi(\Gamma) \vdash_{\lambda \Pi R} \phi(A)
\end{align*}
\]
Adding rewrite rules extends the logic.

Can express the proofs of a system $P$ in Dedukti.

\[
\Gamma \vdash_P A \quad \Longrightarrow \quad \phi(\Gamma) \vdash_{\lambda \Pi R} \phi(A) \quad \text{(Completeness)}
\]

\[
\Gamma \vdash_P A \quad \Longleftrightarrow \quad \phi(\Gamma) \vdash_{\lambda \Pi R} \phi(A) \quad \text{(Soundness)}
\]
Encodings

- Need to find a careful balance between expressivity and consistency.

**Theorem (Assaf, Cousineau, Dowek)**

*Any pure type system (PTS) can be encoded in the $\lambda\Pi$-calculus modulo in a way that is sound and complete.*
Implementations

- Holide: HOL in Dedukti (Assaf, Burel)
- Coqine: Coq in Dedukti (Assaf, Boespflug, Burel)
- Focalide: Focalize in Dedukti (Cauderlier, Dubois)
Thank you!

https://www.rocq.inria.fr/deducteam/software.html
Thank you!

https://www.rocq.inria.fr/deducteam/software.html