Proof-assistants for mathematics

Henk Barendregt
Nijmegen University
The Netherlands

1. The goal
2. Case studies
3. Comparisons
4. Vernacular

Mathematics

Given situation Γ we have A. Proof p QED

Mathematics

Given situation Γ we have A. Proof p QED

Logic

$\Gamma \vdash_L A$ Proof p ■

Mathematics

Given situation Γ we have A. Proof p QED

Logic

$\Gamma \vdash_L A$ Proof p ■

Type system

$\Gamma \vdash_{\lambda L} p : A$ (decidable using 'easy' algorithm)

How to construct a formal proof?

once we know there is one

Case studies

Gröbner paths

Venanzio Capretta

Case studies

Fundamental Theorem of Algebra

Herman Geuvers
Freek Wiedijk
Jan Zwanenburg
[Randy Pollack]
Milad Niqui

Case studies

Fundamental Theorem of Arithmetic

HB

Phases

Informal mathematics
Theory development
Proof development

Proofs by computation (reflection)

$\vdash$ p ‘t’ : t=s because

$\vdash$ p : $\forall$e:L.[[e]] = [[simpl e]]
[[‘t’]]$\rightarrow$* t
simpl ‘t’ $\rightarrow$* ‘s’

Proofs by computation

t, s stand for e.g. expressions over a ring i.e. polynomials

working with the FTA rational expressions are needed

Proofs by computation

How should one deal with undefined?

$$\frac{x+1}{x-1}$$

Rather than $[[e]] = t$ one formalises this as a relation $Sem(e,t)$

Comparisons

- Logical Frameworks
- Compilation
- Declarative reasoning
  Forward reasoning
- Ample Automation
- Flexible logic
- Flexible reduction

- Type Systems
- Interaction
- Procedural reasoning
  Backward reasoning
- Little automation
- Fixed logic
- Fixed reduction

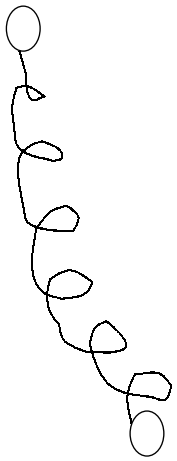
Assistants based on

Logical Frameworks — Isabelle/HOL

vs

Type Systems — Coq/Lego/Agda

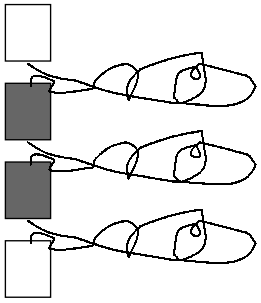
Logical Frameworks



proofscript p_o $\longrightarrow$

(large) transient proof p
can be checked on the fly;
size is large because traces of
computations are included

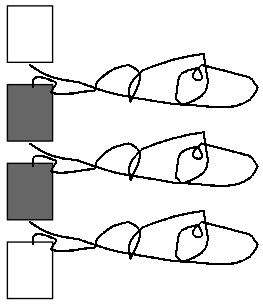
Type Systems



proofscript p_o $\longrightarrow$

proof-object p
can be checked;
size is smaller because traces of
computations are avoided
(Poincaré Principle)

Type Systems



computations are avoided

by using a fixed model of computation ($\beta\delta\iota$-reduction) and requiring that the program be given explicitly

Compilation vs Interaction

Compilation

Automath, Mizar

Interaction

Coq, Lego, Agda; Isabelle, HOL

compiler

OK

more efficient

interaction

--- ok

--- ok

--- ok

--- ok

easier

procedural vs declarative

..... ok → -----

..... ok → ------

..... ok → -----

..... ok → -----

proof-script

proof-object

Goal $\forall x{:}A\ (Px \rightarrow Qx)$

intros	|	Let x:A;
	|	assume Px
	|	(in order to prove Qx)
	|	
procedural		declarative

Forward reasoning

Goal $Px \rightarrow Qx$

$Px \rightarrow$

Forward reasoning

Goal $Px \rightarrow Qx$

$Px \rightarrow P'x$

Forward reasoning

Goal $Px \rightarrow Qx$

$Px \rightarrow P'x \rightarrow P''x$

Forward reasoning

Goal $Px \rightarrow Qx$

$Px \rightarrow P'x \rightarrow P''x \rightarrow \ldots.$

Forward reasoning

Goal $Px \rightarrow Qx$

$Px \rightarrow P'x \rightarrow P''x \rightarrow \ldots. \rightarrow Qx$

Backward reasoning

Goal $Px \rightarrow Qx$

$\rightarrow Qx$

Backward reasoning

Goal $Px \rightarrow Qx$

$Q'x \rightarrow Qx$

Backward reasoning

Goal $Px \rightarrow Qx$

$Q''x \rightarrow Q'x \rightarrow Qx$

Backward reasoning

Goal $Px \rightarrow Qx$

$\ldots. \rightarrow Q''x \rightarrow Q'x \rightarrow Qx$

Backward reasoning

Goal $Px \rightarrow Qx$

$Px \rightarrow \ldots. \rightarrow Q''x \rightarrow Q'x \rightarrow Qx$

Automation

Automated deduction can find proofs of puzzles by Smullyan

$\exists x.[Dx \rightarrow \forall y.Dy]$

Less automation

Mizar

A
then B
then C.
D, by B,
then E.

Less automation

Mizar

By assumption A
hence B
and therefore C.
We have D, by B,
thus E.

Logic

- A v B at set or prop level
- At the prop level (Coq), then a proof of

- $\forall x(Ax \vee \neg Ax)$

- does not give a decision method for A.

Logic

- Making our own logic in Coq at set level is a bit difficult, since one looses predefined tactics.

Reduction

Because proofs are often done via computation, one needs an efficient reduction in the proof engine. Either user definable strategies, or smart implementations, like strictness analysis.

Reduction

User specification of reduction strategy

Or automated

Mathematical vernacular

Nederpelt
based on ideas of de Bruijn

Vernacular for theory development

Vernacular

informal math

formal theory-development | 25-30% of the work

formal proof-development

Vernacular

informal math

$\Downarrow$ mathematician

formal theory-development

$\Downarrow$ automated

formal proof-development

Stylized informal math

nouns and adjectives

Every triangle is .....
Every equilateral triangle has equal ....

Not: $\forall$x:triangle. equilateral x $\rightarrow$ ....

Subtyping

Using coercions macros
(Baily, Saibi)

or

Other type system (Luo, Pollack)

Conclusions

- Logical Frameworks
- Compilation
- Declarative reasoning
- Forward reasoning
- Ample Automation
- Flexible logic
- Flexible reduction
-

- Type Systems
- Interaction
- Procedural reasoning
  Backward reasoning
- Right automation
- Fixed logic
- Fixed reduction

Vernacular

Important

also for proof development ….

Collaboration

Highly important