

FSA seminar 2011, Type Theory and Proof Assistants

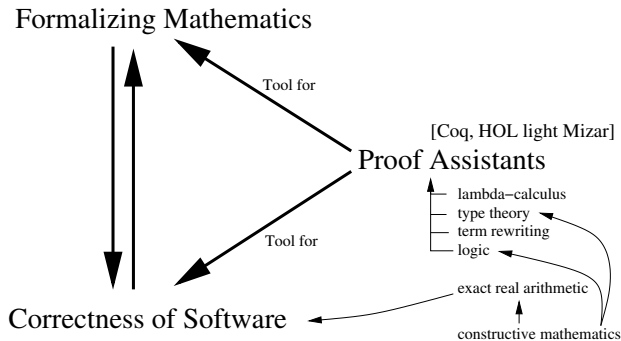
Herman Geuvers

Eindhoven University of Technology

Sep 15 2011

Type Theory and Proof Assistants

- ▶ Integration of computing and proving
- ▶ Prove programs correct
- ▶ Use computation (algorithms/programs) to speed up proving



Projects you can do

- ▶ **Formalizing**: proving correctness of programs in Coq; formalizing proofs in Coq.
- ▶ **Proof Assistant technology**: develop tactics, improve the interface.
- ▶ **Type Theory**: Study properties of type systems, develop new type systems.

Projects you can do

- ▶ **Formalizing**: proving correctness of programs in Coq; formalizing proofs in Coq.
Frank Staals: develop a library + tactics for proving equality between streams in Coq, using **coinduction**
- ▶ Proof Assistant technology: improve the interface to Coq.
- ▶ Type Theory: Study properties of type systems, develop new type systems.
Sepideh Saidi: study and rephrase Wand's principal typing algorithm for weakly polymorphic type theory.

Concrete Project Proof Assistant technology

Creating a **declarative input mode** for Coq

- ▶ **Procedural**: you tell the PA **what** to do
- ▶ **Declarative**: tell the PA **where** to go

Procedural versus Declarative

Procedural

start going east
after 65 m.
after 120 m.
after 1430 m.
after 1640 m.
etcetera

go left
go right
go right
go left

Declarative

at JF Kennedylaan
at Fellenoord
at Mauritsstraat
at Edenstraat
etcetera

go to Fellenoord
go to Mauritsstraat
go to Edenstraat
go to Akkerstraat

Procedural versus Declarative

Procedural

start going east
after 65 m.
after 120 m.
after 1430 m.
after 1640 m.
etcetera

go left
go right
go right
go left

Declarative

at JF Kennedylaan
at Fellenoord
at Mauritsstraat
at Edenstraat
etcetera

go to Fellenoord
go to Mauritsstraat
go to Edenstraat
go to Akkerstraat

Procedural versus Declarative in Coq

- ▶ Coq “mathematical mode” exists (Pierre Corbineau)
- ▶ Should be improved
- ▶ Declarative: readable proofs; Procedural: less typing.
- ▶ ??? Type tactics and get declarative proof
- ▶ ??? Write declarative proof with holes; Fill in with tactics

Procedural versus Declarative in Coq

- ▶ Coq “mathematical mode” exists (Pierre Corbineau)
- ▶ Should be improved
- ▶ Declarative: readable proofs; Procedural: less typing.
- ▶ ??? Type tactics and get declarative proof
- ▶ ??? Write declarative proof with holes; Fill in with tactics

Procedural versus Declarative in Coq

- ▶ Coq “mathematical mode” exists (Pierre Corbineau)
- ▶ Should be improved
- ▶ Declarative: readable proofs; Procedural: less typing.
- ▶ ??? Type tactics and get declarative proof
- ▶ ??? Write declarative proof with holes; Fill in with tactics

Concrete Project Type Theory

Extend the **principal type algorithm** to include recursive types and subtypes

Simple type theory: principal typing algorithm

- ▶ $\lambda x. \lambda y x$ has **principal type** $A \rightarrow B \rightarrow A$
- ▶ $\lambda x. x x$ has **no type**.
- ▶ **principal type algorithm**
input M (λ -term),
output principal type A , if M is typable, or 'fail' if M is not typable.

Concrete Project Type Theory

Extend the **principal type algorithm** to include recursive types and subtypes

Simple type theory: principal typing algorithm

- ▶ $\lambda x. \lambda y x$ has **principal type** $A \rightarrow B \rightarrow A$
- ▶ $\lambda x. x x$ has **no type**.
- ▶ **principal type algorithm**
input M (λ -term),
output principal type A , if M is typable, or 'fail' if M is not typable.

Concrete Project Type Theory

Extend the **principal type algorithm** to include recursive types and subtypes

Simple type theory: principal typing algorithm

- ▶ $\lambda x. \lambda y x$ has **principal type** $A \rightarrow B \rightarrow A$
- ▶ $\lambda x. x x$ has **no type**.
- ▶ **principal type algorithm**
input M (λ -term),
output principal type A , if M is typable, or 'fail' if M is not typable.

Concrete Project Type Theory

Principal type algorithm for recursive types

Computing a principal type for $\lambda x.x x$ yields the equation

$$A = A \rightarrow B$$

This is not **solvable** in simple type theory
but it is with recursive types:

$$A = A \rightarrow B \vdash \lambda x.x x : A \rightarrow B$$

With recursive types we work **modulo a set of type equations** E :

$$E \vdash M : C$$

Concrete Project Type Theory

Principal type algorithm for recursive types

Computing a principal type for $\lambda x.x x$ yields the equation

$$A = A \rightarrow B$$

This is not **solvable** in simple type theory
but it is with recursive types:

$$A = A \rightarrow B \vdash \lambda x.x x : A \rightarrow B$$

With recursive types we work **modulo a set of type equations** E :

$$E \vdash M : C$$

Concrete Project Type Theory

Principal type algorithm for recursive types

Computing a principal type for $\lambda x.x x$ yields the equation

$$A = A \rightarrow B$$

This is not **solvable** in simple type theory
but it is with recursive types:

$$A = A \rightarrow B \vdash \lambda x.x x : A \rightarrow B$$

With recursive types we work **modulo a set of type equations** E :

$$E \vdash M : C$$

Concrete Project Type Theory

Principal type algorithm for recursive types

- ▶ Is there a principal type algorithm for **recursive types** that
 - ▶ terminates
 - ▶ produces a “minimal set of equations” that types the term

Yes ... but there are various formulations of recursive types

...

?? Is this a variant of the “well-known” algorithm for simple types

- ▶ Is there a principal type algorithm for **subtypes**?

Concrete Project Type Theory

Principal type algorithm for recursive types

- ▶ Is there a principal type algorithm for **recursive types** that
 - ▶ terminates
 - ▶ produces a “minimal set of equations” that types the term

Yes ... but there are various formulations of recursive types

...

?? Is this a variant of the “well-known” algorithm for simple types

- ▶ Is there a principal type algorithm for **subtypes**?

Concrete Project Type Theory

Principal type algorithm for recursive types

- ▶ Is there a principal type algorithm for **recursive types** that
 - ▶ terminates
 - ▶ produces a “minimal set of equations” that types the term

Yes ... but there are various formulations of recursive types

...

?? Is this a variant of the “well-known” algorithm for simple types

- ▶ Is there a principal type algorithm for **subtypes**?

Me

- ▶ Course Proving with Computer Assistance
- ▶ Only at the TUE on Thursday (otherwise at RU Nijmegen)
- ▶ HG 6.88, `herman@cs.ru.nl`