# A Case Study in Dependent Type Theory: Extracting a Certified Program from the Formal Proof of its Specification

Andreas Salhus Bakseter

Department of Informatics
University of Bergen

June 9, 2023

# Overview

# Proofs

- proofs are an important part of mathematics

# Proofs

- proofs are an important part of mathematics
- two kinds of proofs:

# Proofs

- proofs are an important part of mathematics
- two kinds of proofs:
  - **informal proofs**, natural language, by humans for humans

# Proofs

- proofs are an important part of mathematics
- two kinds of proofs:
  - **informal proofs**, natural language, by humans for humans
  - **formal proofs**, formal language, hard for humans easy for computers

# Proofs

- proofs are an important part of mathematics
- two kinds of proofs:
  - **informal proofs**, natural language, by humans for humans
  - **formal proofs**, formal language, hard for humans easy for computers
- we can tranfsorm informal proofs into formal proofs (formalization)

# Proofs

- proofs are an important part of mathematics
- two kinds of proofs:
  - **informal proofs**, natural language, by humans for humans
  - **formal proofs**, formal language, hard for humans easy for computers
- we can tranfsorm informal proofs into formal proofs (formalization)
- using computers & proof assistants we can check formal proofs (verification)

# Proofs

- proofs are an important part of mathematics
- two kinds of proofs:
  - **informal proofs**, natural language, by humans for humans
  - **formal proofs**, formal language, hard for humans easy for computers
- we can tranfsorm informal proofs into formal proofs (formalization)
- using computers & proof assistants we can check formal proofs (verification)
- we can also extract programs from formal proofs

# (Dependent) Type Theory & Propositions as Types

- foundation of mathematics

# (Dependent) Type Theory & Propositions as Types

- foundation of mathematics
- every object has a type, $a : A$, must know its construction

# (Dependent) Type Theory & Propositions as Types

- foundation of mathematics
- every object has a type, $a : A$, must know its construction

# (Dependent) Type Theory & Propositions as Types

- foundation of mathematics
- every object has a type, $a : A$, must know its construction
- provides us with rules of inference for manipulating types & objects

# (Dependent) Type Theory & Propositions as Types

- foundation of mathematics
- every object has a type, $a : A$, must know its construction
- provides us with rules of inference for manipulating types & objects
- **propositions as types**:

# (Dependent) Type Theory & Propositions as Types

- foundation of mathematics
- every object has a type, $a : A$, must know its construction
- provides us with rules of inference for manipulating types & objects
- **propositions as types**:
  - propositions are types

# (Dependent) Type Theory & Propositions as Types

- foundation of mathematics
- every object has a type, $a : A$, must know its construction
- provides us with rules of inference for manipulating types & objects
- **propositions as types**:
  - propositions are types
  - proofs are objects

# (Dependent) Type Theory & Propositions as Types

- foundation of mathematics
- every object has a type, $a : A$, must know its construction
- provides us with rules of inference for manipulating types & objects
- **propositions as types**:
  - propositions are types
  - proofs are objects
  - proof of a proposition is an object of that type

# (Dependent) Type Theory & Propositions as Types

- foundation of mathematics
- every object has a type, $a : A$, must know its construction
- provides us with rules of inference for manipulating types & objects
- **propositions as types**:
  - propositions are types
  - proofs are objects
  - proof of a proposition is an object of that type
- **dependent types**:

# (Dependent) Type Theory & Propositions as Types

- foundation of mathematics
- every object has a type, $a : A$, must know its construction
- provides us with rules of inference for manipulating types & objects
- **propositions as types**:
  - propositions are types
  - proofs are objects
  - proof of a proposition is an object of that type
- **dependent types**:
  - gives us more expressive types

# (Dependent) Type Theory & Propositions as Types

- foundation of mathematics
- every object has a type, $a : A$, must know its construction
- provides us with rules of inference for manipulating types & objects
- **propositions as types**:
  - propositions are types
  - proofs are objects
  - proof of a proposition is an object of that type
- **dependent types**:
  - gives us more expressive types
  - $\Sigma$-types to model existential quantification

# (Dependent) Type Theory & Propositions as Types

- foundation of mathematics
- every object has a type, $a : A$, must know its construction
- provides us with rules of inference for manipulating types & objects
- **propositions as types**:
  - propositions are types
  - proofs are objects
  - proof of a proposition is an object of that type
- **dependent types**:
  - gives us more expressive types
  - $\Sigma$-types to model existential quantification
  - $\Pi$-types to model universal quantification

# Proof Assistants

- software tools for constructing & verifying formal proofs

# Proof Assistants

- software tools for constructing & verifying formal proofs
- based on type theory, propositions as types

# Proof Assistants

- software tools for constructing & verifying formal proofs
- based on type theory, propositions as types
- **Coq**:

# Proof Assistants

- software tools for constructing & verifying formal proofs
- based on type theory, propositions as types
- **Coq**:
  - based on the type theory *Calculus of Inductive Constructions*

# Proof Assistants

- software tools for constructing & verifying formal proofs
- based on type theory, propositions as types
- **Coq**:
  - based on the type theory *Calculus of Inductive Constructions*
  - uses *Gallina* as its specification language

# Proof Assistants

- software tools for constructing & verifying formal proofs
- based on type theory, propositions as types
- **Coq**:
  - based on the type theory *Calculus of Inductive Constructions*
  - uses *Gallina* as its specification language
  - uses *Ltac* as its tactic language, for ease of use
  - supports extraction of programs

# Overview of case

- two problems in lattice theory solved by Bezem & Coquand

# Overview of case

- two problems in lattice theory solved by Bezem & Coquand
- we want to answer these questions:

# Overview of case

- two problems in lattice theory solved by Bezem & Coquand
- we want to answer these questions:
  - are the results from Bezem & Coquand correct?

# Overview of case

- two problems in lattice theory solved by Bezem & Coquand
- we want to answer these questions:
  - are the results from Bezem & Coquand correct?
  - is it feasible to formalize complex proofs, such as these?

# Overview of case

- two problems in lattice theory solved by Bezem & Coquand
- we want to answer these questions:
  - are the results from Bezem & Coquand correct?
  - is it feasible to formalize complex proofs, such as these?
  - is the formalization process worth the effort?

# Relevant parts of the paper

- join-semilattices: partially ordered set where any two elements have a least upper bound, called their join, denoted $\vee$

# Relevant parts of the paper

- join-semilattices: partially ordered set where any two elements have a least upper bound, called their join, denoted $\vee$
-