Proof automation in set theory

Bohua Zhan

Technical University of Munich

zhan@in.tum.de

June 21, 2018

Table of Contents

Formalization of mathematics

- Produce machine-checkable proofs of mathematical theorems.

Formalization of mathematics

- Produce machine-checkable proofs of mathematical theorems.
- Possible choices of foundation:
  - Set theory
  - Simple type theory
  - Dependent type theory
  - Homotopy type theory
  - ...

Formalization of mathematics

- Produce machine-checkable proofs of mathematical theorems.
- Possible choices of foundation:
  - Set theory
  - Simple type theory
  - Dependent type theory
  - Homotopy type theory
  - ...
- Choice of foundation can affect definitions of concepts and style of proof throughout the formalization.

Why set theory?

- Set theory is the "standard" foundation of modern mathematics
  - Usual definitions in mathematics can be used without change.
  - Friendly to working mathematicians.

Why set theory?

- Set theory is the "standard" foundation of modern mathematics
  - Usual definitions in mathematics can be used without change.
  - Friendly to working mathematicians.
- Simple treatment of subtypes and partial functions, equality between types, etc.

Why set theory?

- Set theory is the "standard" foundation of modern mathematics
  - Usual definitions in mathematics can be used without change.
  - Friendly to working mathematicians.
- Simple treatment of subtypes and partial functions, equality between types, etc.
- Certain advanced constructions in mathematics are done in a particularly "type-free" way (e.g. algebraic closure of an arbitrary field).

Problems with set theory

- Everything is a set, so it is possible to form non-sensical statements. E.g. $2 \in 3$, $\{a\} \in \langle a, b \rangle$, etc.

Problems with set theory

- Everything is a set, so it is possible to form non-sensical statements. E.g. $2 \in 3$, $\{a\} \in \langle a, b \rangle$, etc.
- Statement of theorems become longer: instead of

$$(x + y) + z = x + (y + z)$$

we may have

$$is\_ab\_group(R) \Longrightarrow x \in . R \Longrightarrow y \in . R \Longrightarrow z \in . R \Longrightarrow (x +_R y) +_R z = x +_R (y +_R z)$$

Main thesis

Problems with using set theory should be addressed by improvements in proof automation.

Table of Contents

Overview of the logic of Isabelle/FOL

- Primitive types: i for sets and o for propositions.
- Function types: $i \to o$, $i \to i$, $(i \to o) \to o$, etc.
- Enough higher-order features to state induction rules, etc.
- However, no equality except for types i and o. Any functions that we wish to consider as first-class objects should be defined as set-theoretic functions (of type i).

Higher-order constructions

- Induction on natural numbers:

$$P(0) \Longrightarrow \forall m \in nat.\ P(m) \longrightarrow P(Suc(m)) \Longrightarrow n \in nat \Longrightarrow P(n)$$

Higher-order constructions

- Induction on natural numbers:

$$P(0) \Longrightarrow \forall m{\in}nat.\ P(m) \longrightarrow P(Suc(m)) \Longrightarrow n \in nat \Longrightarrow P(n)$$

- Conversion from meta-function to set-theoretic function:

$$Fun :: "i \Rightarrow i \Rightarrow (i \Rightarrow i) \Rightarrow i"$$

where $Fun(S,T,f)$ is the set-theoretic function with domain S, codomain T, and mapping given by f.

The ZFC axioms

$$
\begin{aligned}
&\mathit{extension}: && \forall z.\ z \in x \longleftrightarrow z \in y \Longrightarrow x = y \\
&\mathit{empty\_set}: && x \notin \emptyset \\
&\mathit{collect}: && x \in \mathit{Collect}(A,P) \longleftrightarrow (x \in A \wedge P(x)) \\
&\mathit{upair}: && x \in \mathit{Upair}(y,z) \longleftrightarrow (x = y \vee x = z) \\
&\mathit{union}: && x \in \bigcup C \longleftrightarrow (\exists A \in C.\ x \in A) \\
&\mathit{power}: && x \in \mathit{Pow}(S) \longleftrightarrow x \subseteq S \\
&\mathit{replacement}: && \forall x \in A.\ \forall y\ z.\ P(x,y) \wedge P(x,z) \longrightarrow y = z \Longrightarrow \\
& && \quad b \in \mathit{Replace}(A,P) \longleftrightarrow (\exists x \in A.\ P(x,b)) \\
&\mathit{foundation}: && x \notin \emptyset \Longrightarrow \exists y \in x.\ y \cap x = \emptyset \\
&\mathit{infinity}: && \emptyset \in \mathit{Inf} \wedge (\forall y \in \mathit{Inf}.\ \mathit{succ}(y) \in \mathit{Inf}) \\
&\mathit{choice}: && \exists x.\ x \in S \Longrightarrow \mathit{Choice}(S) \in S
\end{aligned}
$$

Table of Contents

Introduction to auto2

- Saturation-based search: iteratively add new facts that can be derived from the initial assumptions.

Introduction to auto2

- Saturation-based search: iteratively add new facts that can be derived from the initial assumptions.
- List of rules (*proof steps*) for deriving new facts from old ones. Rules can be added and removed in-between proofs.

Introduction to auto2

- Saturation-based search: iteratively add new facts that can be derived from the initial assumptions.
- List of rules (*proof steps*) for deriving new facts from old ones. Rules can be added and removed in-between proofs.
- Several tables maintained during the proof: equality, properties, well-formedness data.

Introduction to auto2

- Saturation-based search: iteratively add new facts that can be derived from the initial assumptions.
- List of rules (*proof steps*) for deriving new facts from old ones. Rules can be added and removed in-between proofs.
- Several tables maintained during the proof: equality, properties, well-formedness data. These tables are updated whenever new facts are available.

Problems with set theory

- Everything is a set, so it is possible to form non-sensical statements. E.g. $2 \in 3$, $\{a\} \in \langle a, b \rangle$, etc.
- Statement of theorems become longer: instead of

$$(x + y) + z = x + (y + z)$$

we may have

$$is\_ab\_group(R) \Longrightarrow x \in . R \Longrightarrow y \in . R \Longrightarrow z \in . R \Longrightarrow (x +_R y) +_R z = x +_R (y +_R z)$$

Abstraction of definitions

- Concepts such as ordered pairs and natural numbers are represented as sets, but we never make use of their representations except when they are introduced.

Abstraction of definitions

- Concepts such as ordered pairs and natural numbers are represented as sets, but we never make use of their representations except when they are introduced.
- We can abstract away the underlying representation by *removing the corresponding definitions* from the automation after they are no longer useful.

Example: ordered pairs

First, make the standard definitions:

definition `"⟨a,b⟩ = {{a}, {a,b}}"`
definition `"fst(p) = (THE a. ∃b. p = ⟨a,b⟩)"`
definition `"snd(p) = (THE b. ∃a. p = ⟨a,b⟩)"`

Example: ordered pairs

First, make the standard definitions:

definition `"⟨a,b⟩ = {{a}, {a,b}}"`
definition `"fst(p) = (THE a. ∃b. p = ⟨a,b⟩)"`
definition `"snd(p) = (THE b. ∃a. p = ⟨a,b⟩)"`

Next, show the basic properties:

lemma *pair_eqD:* `"⟨a,b⟩ = ⟨c,d⟩ ⟹ a = b ∧ c = d"`
lemma *fst_conv:* `"fst(⟨a,b⟩) = a"`
lemma *snd_conv:* `"snd(⟨a,b⟩) = b"`

Example: ordered pairs

First, make the standard definitions:

definition "⟨a,b⟩ = {{a}, {a,b}}"
definition "fst(p) = (THE a. ∃b. p = ⟨a,b⟩)"
definition "snd(p) = (THE b. ∃a. p = ⟨a,b⟩)"

Next, show the basic properties:

lemma pair_eqD: "⟨a,b⟩ = ⟨c,d⟩ ⟹ a = b ∧ c = d"
lemma fst_conv: "fst(⟨a,b⟩) = a"
lemma snd_conv: "snd(⟨a,b⟩) = b"

Then, the definitions are "removed" from the automation ...

Example: ordered pairs (continued)

So the automation only sees the following:

lemma *pair_eqD [forward]:* "$\langle a,b\rangle = \langle c,d\rangle \Longrightarrow a = b \wedge c = d$"

lemma *fst_conv [rewrite]:* "$fst(\langle a,b\rangle) = a$"
lemma *snd_conv [rewrite]:* "$snd(\langle a,b\rangle) = b$"

Example: ordered pairs (continued)

So the automation only sees the following:

lemma *pair_eqD [forward]:* "⟨a,b⟩ = ⟨c,d⟩ ⟹ a = b ∧ c = d"

lemma *fst_conv [rewrite]:* "fst(⟨a,b⟩) = a"
lemma *snd_conv [rewrite]:* "snd(⟨a,b⟩) = b"

With this setup, the automation is unable to ...

- Make sense of the statement $x \in \langle a,b \rangle$.
- Rewrite *fst(p)* unless p is known to be an ordered pair.

... but this is exactly what we want.

Example: structures

Represent *structures* in mathematics as partial functions from natural numbers. For example, a group with carrier set S, identity u, and multiplication f is given by

$$[0 \to S,\ 1 \to u,\ 2 \to f]$$

Example: structures

Represent *structures* in mathematics as partial functions from natural numbers. For example, a group with carrier set S, identity u, and multiplication f is given by

$$[0 \to S,\ 1 \to u,\ 2 \to f]$$

Set up names of fields:

```
definition "carrier_name = 0"
definition "one_name = 1"
definition "times_fun_name = 2"

definition "carrier(G) = graph_eval(G, carrier_name)"
definition "one(G) = graph_eval(G, one_name)"
definition "times_fun(G) = graph_eval(G, times_fun_name)"
```

Example: structures (continued)

Constructor for groups:

definition *"Group(S,u,f) =*
{⟨carrier_name, S⟩, ⟨one_name, u⟩, ⟨times_fun_name, f⟩}"

From this definition, we can immediately derive:

lemma *Group_eval [rewrite]:*
"carrier(Group(S,u,f)) = S"
"one(Group(S,u,f)) = u"
"times_fun(Group(S,u,f)) = f"

Example: structures (continued)

Constructor for groups:

definition *"Group(S,u,f) =*
{⟨carrier_name, S⟩, ⟨one_name, u⟩, ⟨times_fun_name, f⟩}"

From this definition, we can immediately derive:

lemma *Group_eval [rewrite]:*
"carrier(Group(S,u,f)) = S"
"one(Group(S,u,f)) = u"
"times_fun(Group(S,u,f)) = f"

Next, the definition of *Group* is removed from use in the automation. The term *Group* can be used to construct groups, with expected evaluation on fields.

More examples

- Set-theoretic functions (as structures containing domain, codomain, and set of pairs).
- Natural numbers (von-Neumann construction).
- Integers, rationals (as quotients).
- Real numbers (as equivalence classes of Cauchy sequences).

Problems with set theory

- Everything is a set, so it is possible to form non-sensical statements. E.g. $2 \in 3$, $\{a\} \in \langle a, b \rangle$, etc.
- Statement of theorems become longer: instead of

$$(x + y) + z = x + (y + z)$$

we may have

$$is\_ab\_group(R) \Longrightarrow x \in . R \Longrightarrow y \in . R \Longrightarrow z \in . R \Longrightarrow (x +_R y) +_R z = x +_R (y +_R z)$$

Properties

- The concept of *properties* simulates type classes in Isabelle/HOL.
- Examples include `is_group`, `is_ring`, and `is_field` (all terms of type $i \to o$).

Properties

- The concept of *properties* simulates type classes in Isabelle/HOL.
- Examples include `is_group`, `is_ring`, and `is_field` (all terms of type $i \to o$).
- During proof, the *property table* maintains the list of known properties about existing terms. Dependencies between properties are automatically propagated.

Example of property propagation

Facts: $is\_ab\_group(G)$, $K = G \times H$, $a \in . K$, $b \in . K$, $H = \ldots$
Goal: $a \cdot_K b \cdot_K a^{-1} = b$

Term	Properties
G	abelian group, group
H	
G × H	
K	

Example of property propagation

Facts: $\mathtt{is\_ab\_group}(G)$, $K = G \times H$, $a \in . K$, $b \in . K$, $H = \ldots$
Goal: $a \cdot_K b \cdot_K a^{-1} = b$

Term	Properties
G	abelian group, group
H	group
G × H	group
K	group

have "$\mathtt{is\_group}(H)$"

Example of property propagation

Facts: $is\_ab\_group(G)$, $K = G \times H$, $a \in . K$, $b \in . K$, $H = \ldots$
Goal: $a \cdot_K b \cdot_K a^{-1} = b$

Term	Properties
G	abelian group, group
H	abelian group, group
G × H	abelian group, group
K	abelian group, group

have "$is\_ab\_group(H)$"

Example of property propagation

Facts: $is\_ab\_group(G)$, $K = G \times H$, $a \in . K$, $b \in . K$, $H = \ldots$
Goal: $a \cdot_K b \cdot_K a^{-1} = b$ (OK).

Term	Properties
G	abelian group, group
H	abelian group, group
G × H	abelian group, group
K	abelian group, group

Well-formed terms

- The concept of *well-formedness data* simulates types and type-checking in Isabelle/HOL.

Well-formed terms

- The concept of *well-formedness data* simulates types and type-checking in Isabelle/HOL.
- For any meta-function, we can register *well-formedness conditions*. These are conditions that should be satisfied by the arguments of the function.

Well-formed terms

- The concept of *well-formedness data* simulates types and type-checking in Isabelle/HOL.
- For any meta-function, we can register *well-formedness conditions*. These are conditions that should be satisfied by the arguments of the function.
- During proof, the *well-formedness table* maintains the list of known well-formedness conditions of existing terms.

Basic examples of well-formedness conditions

- Function application:

  $f \setminus x$ requires $x \in \mathit{source}(f)$.

- Function composition:

  $f \circ g$ requires $\mathit{source}(f) = \mathit{target}(g)$.

- Operators:

  $a +_R b$ requires $a \in \mathit{carrier}(R)$ and $b \in \mathit{carrier}(R)$.

More examples of well-formedness conditions

- Division:

  $a /_R b$ requires $a \in \mathit{carrier}(R)$ and $b \in \mathit{units}(R)$.

- Inverse:

  $\mathit{inv}(a)$ requires $a \in \mathit{units}(R)$.

- Intersection:

  $\bigcap S$ requires $S \neq \emptyset$.

- Supremum:

  $\mathit{sup}(R, X)$ requires $\mathit{has\_sup}(R, X)$.

- etc, etc...

Example of type checking: subtypes

Facts: $x \in \mathit{real}$, $1/(x^2+1) + 1 = 2$.
Goal: $x = 0$.

Term	Types	Well-formedness data
x	real	
x^2	real	$x \in \mathit{real}$
x^2+1	real	$x^2 \in \mathit{real}$, $1 \in \mathit{real}$
$1/(x^2+1)$		$1 \in \mathit{real}$
$1/(x^2+1)+1$		$1 \in \mathit{real}$

Example of type checking: subtypes

Facts: $x \in \texttt{real}$, $1/(x^2+1) + 1 = 2$.
Goal: $x = 0$.

Term	Types	Well-formedness data
x	$\texttt{real}$	
x^2	$\texttt{real}$	$x \in \texttt{real}$
x^2+1	$\texttt{real}$, $\texttt{units}(\mathbb{R})$	$x^2 \in \texttt{real}$, $1 \in \texttt{real}$
$1/(x^2+1)$	$\texttt{real}$	$1 \in \texttt{real}$, $x^2+1 \in \texttt{units}(\mathbb{R})$
$1/(x^2+1)+1$	$\texttt{real}$	$1/(x^2+1) \in \texttt{real}$, $1 \in \texttt{real}$

have "$x^2 + 1 \neq 0$"

Example of type checking: equality between types

Facts: $x \in lists(S,k)$, $y \in lists(S,m)$, $z \in lists(S,n)$,
$k \in nat$, $m \in nat$, $n \in nat$.

Goal: $append(append(x,y),z) = append(x,append(y,z))$.

Term	Types
x	$lists(S,k)$
y	$lists(S,m)$
z	$lists(S,n)$
$append(x,y)$	$lists(S,k+m)$
$append(y,z)$	$lists(S,m+n)$
$append(append(x,y),z)$	$lists(S,(k+m)+n)$
$append(x,append(y,z))$	$lists(S,k+(m+n))$

Example of type checking: equality between types

Facts: $x \in lists(S,k)$, $y \in lists(S,m)$, $z \in lists(S,n)$,
$k \in nat$, $m \in nat$, $n \in nat$.

Goal: $append(append(x,y),z) = append(x,append(y,z))$.

Term	Types
x	$lists(S,k)$
y	$lists(S,m)$
z	$lists(S,n)$
$append(x,y)$	$lists(S,k+m)$
$append(y,z)$	$lists(S,m+n)$
$append(append(x,y),z)$	$lists(S,(k+m)+n)$, $lists(S,k+(m+n))$
$append(x,append(y,z))$	$lists(S,(k+m)+n)$, $lists(S,k+(m+n))$

have "$(k + m) + n = k + (m + n)$"

Aside: undefined values

- It is always possible to write down expressions like $1 /_{\mathbb{R}} 0$, but nothing can be proved about these expressions (except it is equal to itself, etc). In particular, we cannot prove that it is a real number.
- As a consequence, $x \to 1/x$ is not a function $\mathbb{R} \to \mathbb{R}$, but only a function $(\mathbb{R} - \{0\}) \to \mathbb{R}$.

Table of Contents

Mathematical library based on Isabelle/FOL

- Approx. 14,000 lines of theory files, 3,500 lines of ML code.
- Need to work with:
  - Algebraic and topological structures.
  - Quotients.
  - Induction (e.g. on natural numbers, finite sets, etc).
  - Arithmetic (e.g. for constructing the real numbers).

Definition of the fundamental group

- Given topological space X and a point x on X, the group $\pi_1(X, x)$ is defined on the set of loops based at x modulo path homotopy. The identity element is given by the constant loop at x, and multiplication is given by adjoining paths.

Path homotopy

definition *is_homotopy :: "[i, i, i] ⇒ o"* **where**
"is_homotopy(f,g,F) ⟷
(let S = source_str(f) in let T = target_str(f) in
continuous(f) ∧ continuous(g) ∧
S = source_str(g) ∧ T = target_str(g) ∧
F ∈ S ×T I ⇀T T ∧
(∀x∈carrier(S). F‘⟨x,0ℝ⟩ = f‘x ∧ F‘⟨x,1ℝ⟩ = g‘x))"

definition *is_path_homotopy :: "[i, i, i] ⇒ o"* **where**
"is_path_homotopy(f,g,F) ⟷
is_path(f) ∧ is_path(g) ∧ is_homotopy(f,g,F) ∧
(∀t∈carrier(I). F‘⟨0ℝ,t⟩ = f‘(0ℝ) ∧ F‘⟨1ℝ,t⟩ = f‘(1ℝ))"

definition *path_homotopic :: "i ⇒ i ⇒ o"* **where**
"path_homotopic(f,g) ⟷ (∃F. is_path_homotopy(f,g,F))"

Loop spaces

definition *loop_space :: "i ⇒ i ⇒ i"* **where**
"loop_space(X,x) =
{f ∈ I ⇀_T X. f`(0_ℝ) = x ∧ f`(1_ℝ) = x}"

definition *loop_space_rel :: "i ⇒ i ⇒ i"* **where**
"loop_space_rel(X,x) =
Equiv(loop_space(X,x), λf g. path_homotopic(f,g))"

definition *loop_classes :: "i ⇒ i ⇒ i"* **where**
"loop_classes(X,x) =
loop_space(X,x) // loop_space_rel(X,x)"

Fundamental group

definition *fundamental_group :: "i ⇒ i ⇒ i" ("π₁") where*
"π₁(X,x) = (let ℛ = loop_space_rel(X,x) in
Group(loop_classes(X,x),
equiv_class(ℛ,const_mor(I,X,x)),
λf g. equiv_class(ℛ,rep(ℛ,f) ⋆ rep(ℛ,g))))"

lemma *fundamental_group_is_group:*
"is_top_space(X) ⟹ x ∈ carrier(X) ⟹
is_group(π₁(X,x))"

Morphisms

definition *induced_mor :: "i ⇒ i ⇒ i"* **where**
"induced_mor(k,x) =
(let X = source_str(k), Y = target_str(k),
$\mathcal{R}$ = loop_space_rel(X,x), $\mathcal{S}$ = loop_space_rel(Y,k`x)
in Mor(π_1(X,x), π_1(Y,k`x),
λf. equiv_class($\mathcal{S}$, k $\circ_m$ rep($\mathcal{R}$,f))))"

lemma *induced_mor_is_homomorphism:*
"continuous(k) ⟹ X = source_str(k) ⟹
Y = target_str(k) ⟹ x ∈ source(k) ⟹
induced_mor(k,x) ∈ π_1(X,x) $\rightharpoonup_T$ π_1(Y,k`x)"

Functoriality

lemma *induced_mor_id:*
"is_top_space(X) $\Longrightarrow$ *x* $\in$*. X* $\Longrightarrow$
induced_mor(id_mor(X),x) = id_mor(π_1*(X,x))"*

lemma *induced_mor_comp:*
"continuous(k) $\Longrightarrow$ *continuous(h)* $\Longrightarrow$
target_str(k) = source_str(h) $\Longrightarrow$ *x* $\in$ *source(k)* $\Longrightarrow$
induced_mor(h $\circ_m$ *k, x) =*
induced_mor(h, k`*x) *$\circ_m$* *induced_mor(k, x)"*

Table of Contents

Conclusion

- Set theory has an advantage as the "standard" foundation of mathematics.

Conclusion

- Set theory has an advantage as the "standard" foundation of mathematics.
- Difficulties with using set theory can be addressed through improvements in automation:

Conclusion

- Set theory has an advantage as the "standard" foundation of mathematics.
- Difficulties with using set theory can be addressed through improvements in automation:
  - Abstraction of definitions.
  - Properties and well-formedness conditions.

Conclusion

- Set theory has an advantage as the "standard" foundation of mathematics.
- Difficulties with using set theory can be addressed through improvements in automation:
  - Abstraction of definitions.
  - Properties and well-formedness conditions.
- Examples such as the fundamental group can be expressed in a natural way using set theory.