

Proof automation in set theory

Bohua Zhan

Technical University of Munich

zhan@in.tum.de

June 21, 2018

Table of Contents

- 1 Introduction
- 2 Set theory foundation in Isabelle
- 3 Automation for set theory
 - Introduction to auto2
 - Abstraction of Definitions
 - Properties
 - Well-formed terms
- 4 Application
- 5 Conclusion

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

$$\text{is_ab_group}(R) \implies x \in . R \implies y \in . R \implies z \in . R \implies \\ (x +_R y) +_R z = x +_R (y +_R z)$$

Main thesis

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

Table of Contents

- 1 Introduction
- 2 Set theory foundation in Isabelle
- 3 Automation for set theory
 - Introduction to auto2
 - Abstraction of Definitions
 - Properties
 - Well-formed terms
- 4 Application
- 5 Conclusion

Overview of the logic of Isabelle/FOL

- Primitive types: i for sets and o for propositions.
- Function types: $i \rightarrow o$, $i \rightarrow i$, $(i \rightarrow o) \rightarrow o$, 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:

$$\begin{aligned} P(0) \implies \forall m \in \mathit{nat}. P(m) \longrightarrow P(\mathit{Suc}(m)) \\ \implies n \in \mathit{nat} \implies P(n) \end{aligned}$$

Higher-order constructions

- Induction on natural numbers:

$$\begin{aligned} P(0) \implies \forall m \in \mathit{nat}. P(m) \longrightarrow P(\mathit{Suc}(m)) \\ \implies n \in \mathit{nat} \implies P(n) \end{aligned}$$

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

$$\mathit{Fun} :: "i \Rightarrow i \Rightarrow (i \Rightarrow i) \Rightarrow i"$$

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

The ZFC axioms

extension: $\forall z. z \in x \longleftrightarrow z \in y \implies x = y$

empty_set: $x \notin \emptyset$

collect: $x \in \text{Collect}(A,P) \longleftrightarrow (x \in A \wedge P(x))$

upair: $x \in \text{Upair}(y,z) \longleftrightarrow (x = y \vee x = z)$

union: $x \in \bigcup C \longleftrightarrow (\exists A \in C. x \in A)$

power: $x \in \text{Pow}(S) \longleftrightarrow x \subseteq S$

replacement: $\forall x \in A. \forall y z. P(x,y) \wedge P(x,z) \longrightarrow y = z \implies$
 $b \in \text{Replace}(A,P) \longleftrightarrow (\exists x \in A. P(x,b))$

foundation: $x \notin \emptyset \implies \exists y \in x. y \cap x = \emptyset$

infinity: $\emptyset \in \text{Inf} \wedge (\forall y \in \text{Inf}. \text{succ}(y) \in \text{Inf})$

choice: $\exists x. x \in S \implies \text{Choice}(S) \in S$

Table of Contents

- 1 Introduction
- 2 Set theory foundation in Isabelle
- 3 Automation for set theory**
 - Introduction to auto2
 - Abstraction of Definitions
 - Properties
 - Well-formed terms
- 4 Application
- 5 Conclusion

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

$$\text{is_ab_group}(R) \implies x \in . R \implies y \in . R \implies z \in . R \implies \\ (x +_R y) +_R z = x +_R (y +_R z)$$

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 " $\langle a, b \rangle = \{\{a\}, \{a, b\}\}$ "

definition " $fst(p) = (THE\ a.\ \exists b.\ p = \langle a, b \rangle)$ "

definition " $snd(p) = (THE\ b.\ \exists a.\ p = \langle a, b \rangle)$ "

Example: ordered pairs

First, make the standard definitions:

definition " $\langle a, b \rangle = \{\{a\}, \{a, b\}\}$ "

definition " $\text{fst}(p) = (\text{THE } a. \exists b. p = \langle a, b \rangle)$ "

definition " $\text{snd}(p) = (\text{THE } b. \exists a. p = \langle a, b \rangle)$ "

Next, show the basic properties:

lemma pair_eqD : " $\langle a, b \rangle = \langle c, d \rangle \implies a = b \wedge c = d$ "

lemma fst_conv : " $\text{fst}(\langle a, b \rangle) = a$ "

lemma snd_conv : " $\text{snd}(\langle a, b \rangle) = b$ "

Example: ordered pairs

First, make the standard definitions:

definition " $\langle a, b \rangle = \{\{a\}, \{a, b\}\}$ "

definition " $\text{fst}(p) = (\text{THE } a. \exists b. p = \langle a, b \rangle)$ "

definition " $\text{snd}(p) = (\text{THE } b. \exists a. p = \langle a, b \rangle)$ "

Next, show the basic properties:

lemma pair_eqD : " $\langle a, b \rangle = \langle c, d \rangle \implies a = b \wedge c = d$ "

lemma fst_conv : " $\text{fst}(\langle a, b \rangle) = a$ "

lemma snd_conv : " $\text{snd}(\langle a, b \rangle) = b$ "

Then, the definitions are “removed” from the automation ...

Example: ordered pairs (continued)

So the automation only sees the following:

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

```
lemma fst_conv [rewrite]: " $\text{fst}(\langle a, b \rangle) = a$ "
```

```
lemma snd_conv [rewrite]: " $\text{snd}(\langle a, b \rangle) = b$ "
```

Example: ordered pairs (continued)

So the automation only sees the following:

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

```
lemma fst_conv [rewrite]: " $\text{fst}(\langle a, b \rangle) = a$ "
```

```
lemma snd_conv [rewrite]: " $\text{snd}(\langle a, b \rangle) = b$ "
```

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

- Make sense of the statement $x \in \langle a, b \rangle$.
- Rewrite $\text{fst}(p)$ 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

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

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

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

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

$$\text{is_ab_group}(R) \implies x \in . R \implies y \in . R \implies z \in . R \implies \\ (x +_R y) +_R z = x +_R (y +_R z)$$

Properties

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

Properties

- The concept of *properties* simulates type classes in Isabelle/HOL.
- Examples include *is_group*, *is_ring*, and *is_field* (all terms of type $i \rightarrow o$).
- 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: $is_ab_group(G)$, $K = G \times H$, $a \in . K$, $b \in . K$, $H = \dots$

Goal: $a \cdot_K b \cdot_K a^{-1} = b$

Term	Properties
G	abelian group, group
H	
$G \times H$	
K	

Example of property propagation

Facts: $is_ab_group(G)$, $K = G \times H$, $a \in . K$, $b \in . K$, $H = \dots$

Goal: $a \cdot_K b \cdot_K a^{-1} = b$

Term	Properties
G	abelian group, group
H	group
$G \times H$	group
K	group

have “ $is_group(H)$ ”

Example of property propagation

Facts: $is_ab_group(G)$, $K = G \times H$, $a \in . K$, $b \in . K$, $H = \dots$

Goal: $a \cdot_K b \cdot_K a^{-1} = b$

Term	Properties
G	abelian group, group
H	abelian group, group
$G \times H$	abelian group, group
K	abelian group, group

have “ $is_ab_group(H)$ ”

Example of property propagation

Facts: $is_ab_group(G)$, $K = G \times H$, $a \in . K$, $b \in . K$, $H = \dots$

Goal: $a \cdot_K b \cdot_K a^{-1} = b$ (OK).

Term	Properties
G	abelian group, group
H	abelian group, group
$G \times H$	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:

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

- Function composition:

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

- Operators:

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

More examples of well-formedness conditions

- Division:

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

- Inverse:

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

- Intersection:

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

- Supremum:

$\text{sup}(R, X)$ requires $\text{has_sup}(R, X)$.

- etc, etc...

Example of type checking: subtypes

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

Goal: $x = 0$.

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

Example of type checking: subtypes

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

Goal: $x = 0$.

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

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

Example of type checking: equality between types

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

Goal: $\text{append}(\text{append}(x,y),z) = \text{append}(x,\text{append}(y,z))$.

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

Example of type checking: equality between types

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

Goal: $\text{append}(\text{append}(x,y),z) = \text{append}(x,\text{append}(y,z))$.

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

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, $x \rightarrow 1/x$ is not a function $\mathbb{R} \rightarrow \mathbb{R}$, but only a function $(\mathbb{R} - \{0\}) \rightarrow \mathbb{R}$.

Table of Contents

- 1 Introduction
- 2 Set theory foundation in Isabelle
- 3 Automation for set theory
 - Introduction to auto2
 - Abstraction of Definitions
 - Properties
 - Well-formed terms
- 4 Application
- 5 Conclusion

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 $is_homotopy :: "[i, i, i] \Rightarrow o$ where

$is_homotopy(f,g,F) \longleftrightarrow$

(let $S = source_str(f)$ in let $T = target_str(f)$ in
 $continuous(f) \wedge continuous(g) \wedge$

$S = source_str(g) \wedge T = target_str(g) \wedge$

$F \in S \times_T I \rightarrow_T T \wedge$

$(\forall x \in carrier(S). F^{\wedge}\langle x, 0_{\mathbb{R}} \rangle = f^{\wedge}x \wedge F^{\wedge}\langle x, 1_{\mathbb{R}} \rangle = g^{\wedge}x))$ "

definition $is_path_homotopy :: "[i, i, i] \Rightarrow o$ where

$is_path_homotopy(f,g,F) \longleftrightarrow$

$is_path(f) \wedge is_path(g) \wedge is_homotopy(f,g,F) \wedge$

$(\forall t \in carrier(I). F^{\wedge}\langle 0_{\mathbb{R}}, t \rangle = f^{\wedge}(0_{\mathbb{R}}) \wedge F^{\wedge}\langle 1_{\mathbb{R}}, t \rangle = f^{\wedge}(1_{\mathbb{R}}))$ "

definition $path_homotopic :: "i \Rightarrow i \Rightarrow o$ where

$path_homotopic(f,g) \longleftrightarrow (\exists F. is_path_homotopy(f,g,F))$ "

Loop spaces

definition `loop_space` :: " $i \Rightarrow i \Rightarrow i$ " where
"`loop_space(X,x) =`
`{f ∈ I →T X. f^(0ℝ) = x ∧ f^(1ℝ) = x}`"

definition `loop_space_rel` :: " $i \Rightarrow i \Rightarrow i$ " where
"`loop_space_rel(X,x) =`
`Equiv(loop_space(X,x), λf g. path_homotopic(f,g))`"

definition `loop_classes` :: " $i \Rightarrow i \Rightarrow i$ " where
"`loop_classes(X,x) =`
`loop_space(X,x) // loop_space_rel(X,x)`"

Fundamental group

definition *fundamental_group* :: "i \Rightarrow i \Rightarrow i" ("π₁") **where**
"π₁(X,x) = (let \mathcal{R} = loop_space_rel(X,x) in
Group(loop_classes(X,x),
equiv_class(\mathcal{R} , const_mor(I,X,x)),
 $\lambda f g$. equiv_class(\mathcal{R} , rep(\mathcal{R} ,f) \star rep(\mathcal{R} ,g))))"

lemma *fundamental_group_is_group*:
"is_top_space(X) \Longrightarrow x \in carrier(X) \Longrightarrow
is_group(π₁(X,x))"

Morphisms

definition `induced_mor` :: "i \Rightarrow i \Rightarrow i" **where**

```
"induced_mor(k,x) =  
  (let X = source_str(k), Y = target_str(k),  
      R = loop_space_rel(X,x), S = loop_space_rel(Y,k\ $\backslash$ x)  
  in Mor( $\pi_1$ (X,x),  $\pi_1$ (Y,k\ $\backslash$ x),  
         $\lambda f.$  equiv_class(S, k  $\circ_m$  rep(R,f))))"
```

lemma `induced_mor_is_homomorphism`:

```
"continuous(k)  $\implies$  X = source_str(k)  $\implies$   
  Y = target_str(k)  $\implies$  x  $\in$  source(k)  $\implies$   
  induced_mor(k,x)  $\in$   $\pi_1$ (X,x)  $\rightarrow_T$   $\pi_1$ (Y,k\ $\backslash$ x)"
```

Functoriality

lemma *induced_mor_id*:

*"is_top_space(X) \implies x \in . X \implies
induced_mor(id_mor(X), x) = id_mor(π_1 (X, x))"*

lemma *induced_mor_comp*:

*"continuous(k) \implies continuous(h) \implies
target_str(k) = source_str(h) \implies x \in source(k) \implies
induced_mor(h \circ_m k, x) =
induced_mor(h, k`x) \circ_m induced_mor(k, x)"*

Table of Contents

- 1 Introduction
- 2 Set theory foundation in Isabelle
- 3 Automation for set theory
 - Introduction to auto2
 - Abstraction of Definitions
 - Properties
 - Well-formed terms
- 4 Application
- 5 Conclusion

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.