Computer Supported Modeling and Reasoning

David Basin, Achim D. Brucker, Jan-Georg Smaus, and Burkhart Wolff

April 2005

http://www.infsec.ethz.ch/education/permanent/csmr/

HOL: Basic Library

Burkhart Wolff

Mathematics in the Isabelle/HOL Library: Introduction

Isabelle/HOL at Work

We have seen how the mechanism of conservative extensions works in principle.

For several lectures, we will now look at theories of the Isabelle/HOL library, all built by conservative extensions and modelling significant portions of mathematics.

Sets: The Basis of Principia Mathematica

Discoveries in mathematics:

Sets: The Basis of Principia Mathematica

Discoveries in mathematics:

- 17th century: geometry can be reduced to numbers (Descartes, Leibniz, . . .)

Sets: The Basis of Principia Mathematica

Discoveries in mathematics:

- 17th century: geometry can be reduced to numbers (Descartes, Leibniz, . . .)
- 19th century: numbers can be reduced to sets (Cantor, Peano, Frege [Fre93, Fre03], . . .)

Sets: The Basis of Principia Mathematica

Discoveries in mathematics:

- 17th century: geometry can be reduced to numbers (Descartes, Leibniz, . . .)
- 19th century: numbers can be reduced to sets (Cantor, Peano, Frege [Fre93, Fre03], . . .)
- 20th century: sets can be represented in logics (Zermelo/Fränkel, Russel/Whitehead [WR25], Gödel/Bernays, . . .)

Sets: The Basis of Principia Mathematica

Discoveries in mathematics:

- 17th century: geometry can be reduced to numbers (Descartes, Leibniz, . . .)
- 19th century: numbers can be reduced to sets (Cantor, Peano, Frege [Fre93, Fre03], . . .)
- 20th century: sets can be represented in logics (Zermelo/Fränkel, Russel/Whitehead [WR25], Gödel/Bernays, . . .)

As a collection of formalized theories, we call this the Principia Mathematica Structure [WR25].

Interesting: The libraries of modern theorem provers follow this structure . . .

The Roadmap

- Orders
- Sets
- Functions
- (Least) fixpoints and induction
- (Well-founded) recursion
- Arithmetic
- Datatypes

Orders

The Roadmap

We are looking at how the different parts of mathematics are encoded in the Isabelle/HOL library.

- Orders
- Sets
- Functions
- (Least) fixpoints and induction
- (Well-founded) recursion
- Arithmetic
- Datatypes

The Roadmap

We are looking at how the different parts of mathematics are encoded in the Isabelle/HOL library.

- Orders
- Sets
- Functions
- (Least) fixpoints and induction
- (Well-founded) recursion
- Arithmetic
- Datatypes

Three Order Classes

We first define a syntactic class `ord`. It is the class of types for which symbols $<$ and $\leq$ exist.

Three Order Classes

We first define a syntactic class `ord`. It is the class of types for which symbols $<$ and $\leq$ exist.

We then define two axiomatic classes order and linorder for which $<$ and $\leq$ are required to have certain properties, that of being a partial order, or a linear order, resp.

Orders (in `HOL.thy`)

axclass

ord < type

consts

"op <" :: ['a :: ord, 'a] ⇒ bool

"op ≤" :: ['a :: ord, 'a] ⇒ bool

constdefs

min :: "['a :: ord, 'a] ⇒ 'a"

"min a b ≡ (if a ≤ b then a else b)"

max :: "['a :: ord, 'a] ⇒ 'a"

"max a b ≡ (if a ≤ b then b else a)"

Recall **constdefs** syntax and note two uses of <.

Orders in HOL.thy (Cont.)

axclass order < ord

order_refl "$x \leq x$"

order_trans "$[\![x \leq y;\ y \leq z]\!] \Longrightarrow x \leq z$"

order_antisym "$[\![x \leq y;\ y \leq x]\!] \Longrightarrow x = y$"

order_less_le "$x < y = (x \leq y \wedge x \ \tilde{}= y)$"

axclass linorder < order

linorder_linear "$x \leq y \vee y \leq x$"

Least Elements

In HOL.thy, least elements used to be defined as:

$$\text{Least} \;::\; \text{"}(\text{'a} :: \text{ord} \Rightarrow \text{bool}) \Rightarrow \text{'a"}$$

$$\text{Least\_def} \;\text{"Least } P \equiv \text{THE } x.\; P(x) \wedge (\forall\; y.\; P(y) \Longrightarrow x \leq y)\text{"}$$

Monotonicity

In HOL.thy, monotonicity used to be defined as:

$$\text{mono} \quad :: \quad [\text{'a} :: \text{ord} \Rightarrow \text{'b} :: \text{ord}] \Rightarrow \text{bool}$$

$$\text{mono\_def} \quad \text{"mono(f)} \equiv (\forall \text{A B. A} \leq \text{B} \Longrightarrow \text{f(A)} \leq \text{f(B))}$$

Some Theorems about Orders

`monoI`	$(\bigwedge A B. A \le B \Longrightarrow f\,A \le f\,B) \Longrightarrow mono\,f$
`monoD`	$[\![mono\,f; A \le B]\!] \Longrightarrow f\,A \le f\,B$
`order_eq_refl`	$x = y \Longrightarrow x \le y$
`order_less_irrefl`	$\neg x < x$
`order_le_less`	$(x \le y) = (x < y \vee x = y)$
`linorder_less_linear`	$x < y \vee x = y \vee y < x$
`linorder_neq_iff`	$(x \ne y) = (x < y \vee y < x)$
`min_same`	$min\,x\,x = x$
`le_min_iff_conj`	$(z \le min\,x\,y) = (z \le x \wedge z \le y)$

Discussion of Orders

Type classes are a structuring mechanism in Isabelle:

- Syntactic classes (e.g. $t :: \alpha :: ord$ as in Haskell [HHPW96]): merely a mechanism to structure visibility of operations.

Discussion of Orders

Type classes are a structuring mechanism in Isabelle:

- Syntactic classes (e.g. $t :: \alpha :: ord$ as in Haskell [HHPW96]): merely a mechanism to structure visibility of operations.
- Axiomatic classes (e.g. $t :: \alpha :: order$): a mechanism for structuring semantic knowledge in types (foundation to be discussed later).

Sets

The Roadmap

We are still looking at how the different parts of mathematics are encoded in the Isabelle/HOL library.

- Orders
- Sets
- Functions
- (Least) fixpoints and induction
- (Well-founded) recursion
- Arithmetic
- Datatypes

The Roadmap

We are still looking at how the different parts of mathematics are encoded in the Isabelle/HOL library.

- Orders
- Sets
- Functions
- (Least) fixpoints and induction
- (Well-founded) recursion
- Arithmetic
- Datatypes

Set.thy

```
theory Set = HOL:
typedecl 'a set
instance set :: (type) ord ..
consts
  "{}"      :: 'a set ("{}")
  UNIV      :: 'a set
  Collect   :: ('a => bool) => 'a set
  "op :"    :: "'a => 'a set => bool"
```

Note that `Collect` and ":" (alias: $\in$) correspond to Abs_{set} and Rep_{set}.

Set.thy: More Constant Declarations

insert	::	['a, 'a set] ⇒ 'a set
∪, ∩	::	['a set, 'a set] ⇒ 'a set
Ball, Bex	::	['a set, 'a ⇒ bool] ⇒ bool
UNION, INTER	::	['a set, 'a ⇒ 'b set] ⇒ 'b set
⋃, ⋂	::	(('a set) set) ⇒ 'a set

There is the equivalent syntax:

$\{x, y, z\}$	for	`insert` x (`insert` y (`insert` z $\{\}$))
$\forall\, x : A.\, S x$	for	`Ball` A S
$\exists\, x : A.\, S x$	for	`Bex` A S
$\bigcup_{x \in A} .\, S\, x$	for	`UNION` A S
$\bigcap_{x \in A} .\, S\, x$	for	`INTER` A S

Set.thy: Constant Definitions

empty_def: $"\{\} \equiv \{x.\ False\}"$

UNIV_def: $"UNIV \equiv \{x.\ True\}"$

Un_def: $"A \cup B \equiv \{x.\ x \in A \vee x \in B\}"$

Int_def : $"A \cap B \equiv \{x.\ x \in A \wedge x \in B\}"$

insert_def : $"\mathit{insert}\ a\ B \equiv \{x.\ x = a\} \cup B"$

Ball_def : $"Ball\ A\ P \equiv \forall x.\ x \in A \longrightarrow P(x)"$

Bex_def: $"Bex\ A\ P \equiv \exists x.\ x \in A \wedge P(x)"$

Set.thy: Constant Definitions (2)

subset_def : "$A \leq B \equiv \forall x \in A.\ x \in B$"

Compl_def: "$-A \equiv \{x.\ \neg x \in A\}$"

set_diff_def : "$A - B \equiv \{x.\ x \in A \wedge \neg x \in B\}$"

UNION_def: "$\text{UNION}\ A\ B \equiv \{y.\ \exists x \in A.\ y \in B(x)\}$"

INTER_def: "$\text{INTER}\ A\ B \equiv \{y.\ \forall x \in A.\ y \in B(x)\}$"

Set.thy: Constant Definitions (2)

subset_def : "$A \leq B \equiv \forall\, x{\in}A.\ x{\in}B$"

Compl_def: "$-\ A \equiv \{x.\ \neg x{\in}A\}$"

set_diff_def : "$A\ -\ B \equiv \{x.\ x{\in}A \wedge \neg x{\in}B\}$"

UNION_def: "UNION A B $\equiv \{y.\ \exists x{\in}A.\ y{\in}\ B(x)\}$"

INTER_def: "INTER A B $\equiv \{y.\ \forall x{\in}A.\ y{\in}B(x)\}$"

Note use of $\leq$ instead of $\subseteq$!

Set.thy: Constant Definitions (3)

Union_def: "$\bigcup$ S $\equiv$($\bigcup$ x$\in$S. x)"

Inter_def : "$\bigcap$ S $\equiv$(INT x$\in$S. x)"

Pow_def: "Pow A $\equiv$\{B. B $\leq$A\}"

image_def: "f‘A $\equiv$\{y. $\exists$ x$\in$A. y = f(x)\}"

Some Theorems in Set.thy

CollectI	$P\,a \Longrightarrow a \in \{x.P\,x\}$
CollectD	$a \in \{x.P\,x\} \Longrightarrow P\,a$
set_ext	$(\bigwedge x.(x \in A) = (x \in B)) \Longrightarrow A = B$
subsetI	$(\bigwedge x.x \in A \Longrightarrow x \in B) \Longrightarrow A \subseteq B$
eqset_imp_iff	$A = B \Longrightarrow (x \in A) = (x \in B)$
UNIV_I	$x \in \mathtt{UNIV}$
subset_UNIV	$A \subseteq \mathtt{UNIV}$
empty_subsetI	$\{\} \subseteq A$
Pow_iff	$(A \in Pow\,B) = (A \subseteq B)$
IntI	$[\![c \in A; c \in B]\!] \Longrightarrow c \in A \cap B$

More Theorems in Set.thy

insert_iff	$(a \in insert\, b\, A) = (a = b \vee a \in A)$
image_Un	$f`(A \cup B) = f`A \cup f`B$
Inter_lower	$B \in A \Longrightarrow \bigcap A \subseteq B$
Inter_greatest	$(\bigwedge X. X \in A \Longrightarrow C \subseteq X) \Longrightarrow C \subseteq \bigcap A$

Discussion of Sets

Rich and powerful set theory available in HOL:

- No problems with consistency
- Weaker than ZF (since typed set-theory:) there is no "union of sets"; but: complement-closed
- Good mechanical support for many set tautologies (both for classical reasoning as well as simplification)
- Powerful basis for many problems in modeling.

Functions

The Roadmap

We are still looking at how the different parts of mathematics are encoded in the Isabelle/HOL library.

- Orders
- Sets
- Functions
- (Least) fixpoints and induction
- (Well-founded) recursion
- Arithmetic
- Datatypes

The Roadmap

We are still looking at how the different parts of mathematics are encoded in the Isabelle/HOL library.

- Orders
- Sets
- Functions
- (Least) fixpoints and induction
- (Well-founded) recursion
- Arithmetic
- Datatypes

Fun.thy

The theory Fun.thy defines some important notions on functions, such as concatenation, the identity function, the image of a function, etc.

We look at it briefly.

Two Extracts from Fun.thy

Composition and the identity function:

constdefs

```
 id  :: "'a ⇒ 'a"
"id ≡ λ x. x"

 comp :: "['b ⇒ 'c, 'a ⇒ 'b, 'a] ⇒ 'c"
"f o g ≡ λ x. f(g(x))"
```

There are also definitions for function update, function override and theorems for concepts such as injectivity, surjectivity and bijectivity.

Compare the syntax for `constdefs`.

Instantiating an Axiomatic Class

Sets are partial orders: `set` is an instance of the axiomatic class `order`.

instance set :: (type) order
  apply (auto) **done**

- Axiomatic classes result in proof obligations.
- These are discharged whenever instance is stated.
- Type-checking has access to the established properties.

Conclusion of Orders, Sets, and Functions

- conservative extensions can be used to build consistent libraries.

Conclusion of Orders, Sets, and Functions

- conservative extensions can be used to build consistent libraries.
- Sets as one important package of Isabelle/HOL library:
  - Set theory is typed, but very rich and powerfully supported.

Conclusion of Orders, Sets, and Functions

- conservative extensions can be used to build consistent libraries.
- Sets as one important package of Isabelle/HOL library:
  - Set theory is typed, but very rich and powerfully supported.
  - Sets are instance of `ord` and `order` type class, demonstrates type classes as structuring mechanism in Isabelle.

Conclusion of Orders, Sets, and Functions

- conservative extensions can be used to build consistent libraries.
- Sets as one important package of Isabelle/HOL library:
  - Set theory is typed, but very rich and powerfully supported.
  - Sets are instance of `ord` and `order` type class, demonstrates type classes as structuring mechanism in Isabelle.

More Detailed Explanations

Different uses of $<$

Note: the $<$ may occur in different lexical categories, e.g.:

axclass order $<$ ord

in the theory file states that `order` is a subclass of `ord`.

Compare to the declaration

"op <" :: ['a :: ord, 'a] $\Rightarrow$ bool ("(_ < _)" [50, 51] 50)

where a constant $<$ with a certain type is introduced.

Semantic Classes for Semantic Knowledge

The Isabelle type system records for any type variable what class constraints there are for this type variable. These class constraints may arise from the types of the constants used in an expression, or they may be given explicitly by the user in a goal. E.g. one might type

lemma "(x::'a :: order)<y ⟹x≤y";

to specify that x must be of a type in the type class `order`.

The axioms of an axiomatic class can only be applied if any constant declared in the axiomatic class (or a syntactic superclass) is applied to arguments of a type in the axiomatic class. E.g. order_refl can only be used to prove y ≤y if the type of y is in the type class order.

In this sense the type information (y is of type in class order) is semantic knowledge (y ≤y holds).

$\leq$ instead of $\subseteq$

Sets are an instance of the type class ord, where the generic constant $\leq$ is the subset relation in this particular case.

In fact, the subset relation is reflexive, transitive and anti-symmetric, and so sets are an instance of the axiomatic class order. This is non-obvious and must be proven as part of the **instance** statement.

Union of Arbitrary Sets?

In typed set theory (what we have here in HOL), it is not possible to form the union of two sets of different type. This is in contrast to ZF.

Typed Sets Are Complement-Closed

The complement of a typed set A, i.e.

$$\{x \mid x \notin A\}$$

is again a set, whose type is the same as the type of A. In ZF, the complement construction is not generally allowed since it opens the door to Russell's Paradox.

Proof Obligations

To claim that a type is an instance of an axiomatic class, it has to be proven that the axioms (in the case pf order: order_refl , order_trans , order_antisym, and order_less_le are indeed fulfilled by that type.

Discharge Obligations

The Isabelle mechanism is such that the line

instance set :: (type) order
  apply(auto) **done**

instructs Isabelle to prove the axioms using the previously proven theorems subset_refl , subset_trans , subset_antisym, and psubset_eq.

References

[Fre93] Gottlob Frege. *Grundgesetze der Arithmetik*, volume I. Verlag Hermann Pohle, 1893. Translated in part in [Fur64].

[Fre03] Gottlob Frege. *Grundgesetze der Arithmetik*, volume II. Verlag Hermann Pohle, 1903. Translated in part in [Fur64].

[Fur64] Montgomery Furth. *The Basic Laws of Arithmetic*. Berkeley: University of California Press, 1964. Translation of [Fre03].

[HHPW96] Cordelia V. Hall, Kevin Hammond, Simon L. Peyton Jones, and Philipp Wadler. Type classes in Haskell. *ACM Transactions on Programming Languages and Systems*, 18(2):109–138, 1996.

[WR25] Alfred N. Whitehead and Bertrand Russell. *Principia Mathematica*, volume 1. Cambridge University Press, 1925. 2nd edition.