Theorem Prover Lab

Episode 4: Higher-Order Logic

Terru Stübinger | 2025-11-26

KASTEL

Today

1. Homework
2. A couple more proof patterns
3. What is HOL, anyways?
4. A little bit about type classes

Isar: Collecting facts with `moreover`

Facts can be collected:

```
proof -
  have "P₁"
  moreover have "P₂"
  moreover have "P₃"
  ultimately
    have "..."
```

$\approx$

```
proof -
  have fact1: "P₁"
  have fact2: "P₂"
  have fact3: "P₃"
  from fact1 fact2 fact3
    have "..."
```

Facts are collected in the name `calculation`

Isar: abbreviations with `let`

```
let ?t = "some-big-formula"
⋮
have "... ?t ..."
```

Isar: pattern matching with `(is ...)`

Abbreviations can also be given by matching on a term:

```
lemma "t₁ = t₂" (is "?a = ?b")
proof
  assume ?a
  ⋮
  show ?b
next
  assume ?b
  ⋮
  show ?a
qed
```

Isar: managing facts explicitely with `note`

```
lemma assumes "..." shows "..."
proof cases
  case True
  have "P₁"
  note facts = assms True this
  ⋮
  from facts have "..."
```

Isar playground: `notepad`

`notepad` begins a proof in which anything can be assumed

```
notepad begin
  assume "a ∨ ¬ a"
  ⋮
end
```

Isar playground: `notepad`

`notepad` begins a proof in which anything can be assumed

```
notepad begin
  assume "a ∧ ¬ a"
  ⋮
end
```

Isar playground: `notepad`

`notepad` begins a proof in which anything can be assumed

```
notepad begin
  assume "a ∧ ¬ a"
   ⋮
end
```

… but anything inside will be discarded afterwards

Today: Higher-Order Logic

So far we've been using Isabelle/HOL, and also mentioned a metalogic

... but how does it work "under the hood?"

Isabelle has two main "layers":

- Isabelle/PURE, the meta logic
- an object logic we're working in (usually: HOL, based on Gordon 1988)

→ Nothing about HOL is built-in, it's all just axioms & definitions!

Literature

[1] Michael JC Gordon. “HOL: A proof generating system for higher-order logic”. In: *VLSI specification, verification and synthesis*. Springer, 1988, pp. 73–128.

[2] Ondej Kunar and Andrei Popescu. “Comprehending Isabelle/HOLs consistency”. In: *European Symposium on Programming*. Springer. 2017, pp. 724–749.

[3] Tobias Nipkow and Simon RoSSkopf. “Isabelle’s Metalogic: Formalization and Proof Checker.”. In: *CADE*. 2021, pp. 93–110.

[4] Lawrence C. Paulson. “Zermelo Fraenkel Set Theory in Higher-Order Logic”. In: *Archive of Formal Proofs* (Oct. 2019). `https://isa-afp.org/entries/ZFC_in_HOL.html`, Formal proof development. ISSN: 2150-914x.

[5] Andrew Pitts. “The HOL Logic”. In: ed. by Michael JC Gordon and Tom F Melham. Cambridge University Press, 1993.

The Metalogic $\mathcal{M}$ (Isabelle/PURE)

Before we can talk about logic, we need a language to talk *in!*

The Metalogic $\mathcal{M}$ (Isabelle/PURE)

Before we can talk about logic, we need a language to talk *in!*

Simply-typed λ-calculus with polymorphic types, and:

- a type `prop` of "propositions"
- operators $\Longrightarrow$ (implication), $\bigwedge$ (universal quantification) and $\equiv$ (equality)
- (and `&&&`, defined via $\Longrightarrow$)

The Metalogic $\mathcal{M}$ (Isabelle/PURE)

Before we can talk about logic, we need a language to talk *in!*

Simply-typed λ-calculus with polymorphic types, and:

- a type `prop` of "propositions"
- operators $\implies$ (implication), $\bigwedge$ (universal quantification) and $\equiv$ (equality)
- (and `&&&`, defined via $\implies$)
- frequently used implicitly (e.g. by **assume**, **fix**, **and**, **definition**, in the goal state ...)

The Metalogic $\mathcal{M}$ (Isabelle/PURE)

Before we can talk about logic, we need a language to talk *in!*

Simply-typed λ-calculus with polymorphic types, and:

- a type `prop` of "propositions"
- operators $\Longrightarrow$ (implication), $\bigwedge$ (universal quantification) and $\equiv$ (equality)
- (and &&&, defined via $\Longrightarrow$)
- frequently used implicitly (e.g. by **`assume`**, **`fix`**, **`and`**, **`definition`**, in the goal state ...)

But: structure of `prop` left undefined in $\mathcal{M}$

$\rightarrow$ to do anything useful, we need an *object logic* which gives meaning to `prop`!

Object Logics

So far we've been using HOL, but there are others!

- CTT — Constructive Type Theory
- ZFC — Zermelo-Fraenkel Set Theory (with Choice)
- FOL — First-Order Logic
- ...

Object logics don't have to be part of the distribution

- HoTT — Homotopy Type Theory
- & other experiments ...
- & many extensions to HOL in the AfP
- & for this course: MiniHOL

`MiniHOL.thy`

Defining HOL: Equality

Two axioms characterise equality, one to work with it:

$$(x = x) :: bool \tag{refl}$$

$$(\bigwedge x. fx = gy) \Longrightarrow (\lambda x.\ f\ x) = (\lambda x.\ g\ x) \tag{ext}$$

$$s = t \Longrightarrow Ps \Longrightarrow Pt \tag{subst}$$

Defining HOL: Equality

Two axioms characterise equality, one to work with it:

$$(x = x) :: bool \tag{refl}$$

$$(\bigwedge x. fx = gy) \Longrightarrow (\lambda x.\ f\ x) = (\lambda x.\ g\ x) \tag{ext}$$

$$s = t \Longrightarrow Ps \Longrightarrow Pt \tag{subst}$$

Then we can define:

$$\text{True} \equiv \lambda(x :: \text{bool}).\ x = \lambda x.\ x$$

And immediately have that true holds:

```
lemma TrueI: True
  unfolding True_def
  by (rule refl)
```

Further axioms: Relating to the metalogic

HOL's implication and equality should be the same as in $\mathcal{M}$

$$(P \Longrightarrow Q) \Longrightarrow P \longrightarrow Q \tag{impl}$$

$$P \longrightarrow Q \Longrightarrow P \Longrightarrow Q \tag{mp}$$

$$a = b \Longrightarrow a \equiv b \tag{eq_reflection}$$

What about $a \equiv b \Longrightarrow a = b$?

Further axioms: Relating to the metalogic

HOL's implication and equality should be the same as in $\mathcal{M}$

$$(P \Longrightarrow Q) \Longrightarrow P \longrightarrow Q \tag{impl}$$

$$P \longrightarrow Q \Longrightarrow P \Longrightarrow Q \tag{mp}$$

$$a = b \Longrightarrow a \equiv b \tag{eq\_reflection}$$

What about $a \equiv b \Longrightarrow a = b$?
$\rightarrow$ Proof!

Aside: Structuring rules

Isabelle/HOL is modelled after *Natural Deduction*

- Similar to sequent calculus
- but only has one goal in each rule

Rules come in two flavours:

- *introduction* explains how to prove a connective
- *elimination* explains what you can prove using a connective

$$\frac{A \quad B}{A \wedge B} \qquad \frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B}$$

Aside: Structuring rules

Isabelle/HOL is modelled after *Natural Deduction*

- Similar to sequent calculus
- but only has one goal in each rule

Rules come in two flavours:

- *introduction* explains how to prove a connective
- *elimination* explains what you can prove using a connective

$$\frac{A \quad B}{A \wedge B}\ \wedge\text{-intro} \qquad \frac{A \wedge B}{A}\ \wedge\text{-elim1} \qquad \frac{A \wedge B}{B}\ \wedge\text{-elim2}$$

Naming convention: `conjI` vs `conjE`

Aside: Structuring rules

Isabelle/HOL is modelled after *Natural Deduction*

- Similar to sequent calculus
- but only has one goal in each rule

Rules come in two flavours:

- *introduction* explains how to prove a connective
- *elimination* explains what you can prove using a connective

$$\frac{A}{A \lor B} \qquad \frac{B}{A \lor B} \qquad \frac{A \lor B \quad \begin{matrix}[A] \\ P\end{matrix} \quad \begin{matrix}[B] \\ P\end{matrix}}{P}$$

Naming convention: `conjI` vs `conjE`

Aside: Structuring rules

Isabelle/HOL is modelled after *Natural Deduction*

- Similar to sequent calculus
- but only has one goal in each rule

Rules come in two flavours:

- *introduction* explains how to prove a connective
- *elimination* explains what you can prove using a connective

$$\frac{A}{A \lor B}\ \lor\text{-intro1} \qquad \frac{B}{A \lor B}\ \lor\text{-intro2} \qquad \frac{A \lor B \quad \begin{matrix}[A]\\ P\end{matrix} \quad \begin{matrix}[B]\\ P\end{matrix}}{P}\ \lor\text{-elim}$$

Naming convention: `conjI` vs `conjE`

Applying rules

If we have an introduction rule, we can apply it:

```
apply (rule ...)
```

For elimination rules:

```
apply (erule ...)
```

Some rules can be useful for both!

Frequently this leaves us with goals like $\mathtt{a} \Longrightarrow \mathtt{b} \Longrightarrow \mathtt{a}$

```
apply assumption
```

Eliminating equals true

Eliminating equals true

```
lemma eqTrueE: "P = True ⟹ P"
 ...
```

Eliminating equals true

```
lemma eqTrueE: "P = True ⟹ P"
proof (erule subst)
  show "P = True ⟹ True = P"
    by (erule sym)
next
  show "P = True ⟹ True"
    using TrueI by assumption
qed
```

Meta logic & object logic: `prop` vs `bool`

Pure's `prop` is set to HOL's `bool`, with implicit coercions as needed

```
judgement Trueprop :: "bool ⇒ prop"  (‹_› 5)
```

Meta logic & object logic: `prop` vs `bool`

Pure's `prop` is set to HOL's `bool`, with implicit coercions as needed

```
judgement Trueprop :: "bool ⇒ prop"  (‹_› 5)
```

But: this is only a one-way conversion!

Meta logic & object logic: `prop` vs `bool`

Pure's `prop` is set to HOL's `bool`, with implicit coercions as needed

```
judgement Trueprop :: "bool ⇒ prop"  (<_> 5)
```

But: this is only a one-way conversion!

$$\texttt{P} \;=\; \texttt{True} \Longrightarrow \texttt{P}$$
$$\texttt{P} \;\equiv\; \texttt{True} \Longrightarrow \texttt{P}$$
$$\texttt{P} \;\equiv\; (\texttt{True} \Longrightarrow \texttt{P})$$

are allowed

Meta logic & object logic: `prop` vs `bool`

Pure's `prop` is set to HOL's `bool`, with implicit coercions as needed

```
judgement Trueprop :: "bool ⇒ prop"  (<_> 5)
```

But: this is only a one-way conversion!

$$\texttt{P} = \texttt{True} \Longrightarrow \texttt{P}$$
$$\texttt{P} \equiv \texttt{True} \Longrightarrow \texttt{P}$$
$$\texttt{P} \equiv (\texttt{True} \Longrightarrow \texttt{P})$$

are allowed

$$\texttt{P} = (\texttt{True} \Longrightarrow \texttt{P})$$
$$\forall \texttt{x}.\ \texttt{x} \equiv \texttt{x}$$
$$\forall \texttt{x}.\ (\texttt{x} \Longrightarrow \texttt{True})$$

are syntactic nonsense

Meta logic & object logic: `prop` vs `bool`

Pure's `prop` is set to HOL's `bool`, with implicit coercions as needed

```
judgement Trueprop :: "bool ⇒ prop"  (‹_› 5)
```

But: this is only a one-way conversion!

$$\texttt{P = True} \Longrightarrow \texttt{P}$$
$$\texttt{P} \equiv \texttt{True} \Longrightarrow \texttt{P}$$
$$\texttt{P} \equiv (\texttt{True} \Longrightarrow \texttt{P})$$

are allowed

$$\texttt{P = } (\texttt{True} \Longrightarrow \texttt{P})$$
$$\forall \texttt{x. x} \equiv \texttt{x}$$
$$\forall \texttt{x. } (\texttt{x} \Longrightarrow \texttt{True})$$

are syntactic nonsense

Meta-level $\bigwedge$, $\Longrightarrow$ and $\equiv$ can only talk about `prop`; use $\forall$, $\longrightarrow$ and $=$ instead if inside terms.

$\forall$ and `False`

`True` can be used to define $\forall$

$$\forall P \equiv P = (\lambda x.\ \mathsf{True})$$

$$\mathsf{False} \equiv \forall P.\ P$$

Excluded Middle

$$(P :: \mathsf{bool}) = \mathsf{False} \lor P = \mathsf{True} \qquad \text{(true\_or\_false)}$$

Effectively collapses `bool` into just the two values.

Now we can prove:

- `iffI: (a ⟹ b)⟹ (b ⟹ a)⟹ a = b`
- Finally also `eqTrueI: P ⟹ P = True`
- And the $\forall$-intro rule

`MiniHOL.thy`

Definite description: THE

Axiom:

$$(\text{THE } x.\ x = a) = a \qquad \text{(the\_eq\_trivial)}$$

What does THE $x.\ P\ x$ mean?

Definite description: THE

Axiom:

$$(\mathrm{THE}\ x.\ x = a) = a \qquad \text{(the\_eq\_trivial)}$$

What does THE x. P x mean?
"Give me the unique x for which P x holds"

Definite description: `THE`

Axiom:

$$(\text{THE } x.\ x = a) = a \qquad \text{(the\_eq\_trivial)}$$

What does THE x. $P\ x$ mean?
"Give me the unique x for which $P\ x$ holds"

Used e.g. to define `if`:

$$\text{if } P \text{ then } x \text{ else } y \equiv (\text{THE } z ::' a.\ (P = \mathit{True} \rightarrow z = x) \wedge (P = \mathit{False} \rightarrow z = y))$$

Definite description: THE

Axiom:

$$(\text{THE } x.\ x = a) = a \qquad \text{(the\_eq\_trivial)}$$

What does THE $x.\ P\ x$ mean?
"Give me the unique x for which $P\ x$ holds"

Used e.g. to define `if`:

$$\text{if } P \text{ then } x \text{ else } y \equiv (\text{THE } z ::' a.\ (P = \mathit{True} \to z = x) \wedge (P = \mathit{False} \to z = y))$$

What if no such x exists?

Definite description: THE

Axiom:

$$(\mathrm{THE}\ x.\ x = a) = a \qquad \text{(the\_eq\_trivial)}$$

What does THE $x.\ P\ x$ mean?
"Give me the unique x for which $P\ x$ holds"

Used e.g. to define `if`:

$$\text{if } P \text{ then } x \text{ else } y \equiv (\mathrm{THE}\ z ::' a.\ (P = \mathit{True} \rightarrow z = x) \wedge (P = \mathit{False} \rightarrow z = y))$$

What if no such x exists?
$\rightarrow$ no applicable rule, nothing provable, "stuck"

Undefinedness

Undefinedness has its own axiom:

$$\text{undefined} \; :: \; {}'a \qquad \text{(undef)}$$

This is *always* stuck, like `THE x. False`

Undefinedness

Undefinedness has its own axiom:

$$\text{undefined} :: {}'a \tag{undef}$$

This is *always* stuck, like `THE x. False`

Why is this safe?

Undefinedness

Undefinedness has its own axiom:

$$\text{undefined} :: {}'a \tag{undef}$$

This is *always* stuck, like `THE x. False`

Why is this safe?

$\rightarrow$ need to ensure all types are non-empty

Undefinedness

Undefinedness has its own axiom:

$$\text{undefined} :: {}'a \qquad \text{(undef)}$$

This is *always* stuck, like `THE x. False`

Why is this safe?

$\rightarrow$ need to ensure all types are non-empty

What can you prove about `undefined`?

Existential Quantification

$$\exists x.\ P\ x \ \equiv \forall Q.\ (\forall x.\ P\ x \longrightarrow Q) \longrightarrow Q$$

Existential Quantification

$$\exists x.\ P\ x\ \equiv \forall Q.\ (\forall x.\ P\ x \longrightarrow Q) \longrightarrow Q$$

No such x exists

Existential Quantification

$$\exists x.\ P\ x\ \equiv \forall Q.\ (\forall x.\ P\ x \longrightarrow Q) \longrightarrow Q$$

No such x exists

$\forall x.\ P\ x \longrightarrow Q$ vacuously true

$\forall Q. Q \equiv \text{False}$

Existential Quantification

$$\exists x.\ P\ x\ \equiv \forall Q.\ (\forall x.\ P\ x \longrightarrow Q) \longrightarrow Q$$

No such x exists

$\forall x.\ P\ x \longrightarrow Q$ vacuously true

$\forall Q.Q \equiv$ False

Some x exists

Existential Quantification

$$\exists x.\ P\ x \equiv \forall Q.\ (\forall x.\ P\ x \longrightarrow Q) \longrightarrow Q$$

No such x exists

$\forall x.\ P\ x \longrightarrow Q$ vacuously true

$\forall Q. Q \equiv \text{False}$

Some x exists

$\forall x.\ P\ x \longrightarrow Q \equiv Q$

$\forall Q.\ Q \longrightarrow Q \equiv \text{True}$

Existential Quantification

$$\exists x.\ P\ x\ \equiv \forall Q.\ (\forall x.\ P\ x \longrightarrow Q) \longrightarrow Q$$

No such x exists

$\forall x.\ P\ x \longrightarrow Q$ vacuously true

$\forall Q.Q \equiv \text{False}$

Some x exists

$\forall x.\ P\ x \longrightarrow Q \equiv Q$

$\forall Q.\ Q \longrightarrow Q \equiv \text{True}$

Note: this differs from Gordon's original formulation

Indefinite description (Hilbert's epsilon)

`SOME` is an even stronger version of `THE`:

$$P\ x \Longrightarrow P(\mathsf{SOME}\ x.\ P\ x) \tag{someI}$$

Indefinite description (Hilbert's epsilon)

`SOME` is an even stronger version of `THE`:

$$P\ x \Longrightarrow P(\mathsf{SOME}\ x.\ P\ x) \tag{someI}$$

(this is equivalent to the Axiom of Choice)

Indefinite description (Hilbert's epsilon)

`SOME` is an even stronger version of `THE`:

$$P\ x \Longrightarrow P(\mathsf{SOME}\ x.\ P\ x) \qquad \text{(someI)}$$

(this is equivalent to the Axiom of Choice)

Can define $\exists x.\ P\ x \equiv P(\mathsf{SOME}\ x.\ P\ x)$ (e.g. in Gordon 1988).

Included in Isabelle/HOL, but no longer used for core definitions.

A Note on `value`

Isabelle/HOL makes no claim to be computable!

$\rightarrow$ `value` command fails for things like `THE`, `SOME`, `undefined`, ...

A Note on `value`

Isabelle/HOL makes no claim to be computable!

$\rightarrow$ **`value`** command fails for things like `THE`, `SOME`, `undefined`, …

… but we can still use it "most of the time"

A Note on `value`

Isabelle/HOL makes no claim to be computable!

$\rightarrow$ `value` command fails for things like `THE`, `SOME`, `undefined`, …

… but we can still use it "most of the time"

`value` exports your definitions to Standard ML

A Note on `value`

Isabelle/HOL makes no claim to be computable!

→ `value` command fails for things like `THE`, `SOME`, `undefined`, ...

... but we can still use it "most of the time"

`value` exports your definitions to Standard ML

- for non-computable things it will complain

A Note on `value`

Isabelle/HOL makes no claim to be computable!

→ `value` command fails for things like `THE`, `SOME`, `undefined`, ...

... but we can still use it "most of the time"

`value` exports your definitions to Standard ML

- for non-computable things it will complain
- but: can help by giving *code equations*: `lemma[code]: "f a = ..."`

A Note on `value`

Isabelle/HOL makes no claim to be computable!

→ `value` command fails for things like `THE`, `SOME`, `undefined`, …

… but we can still use it "most of the time"

`value` exports your definitions to Standard ML

- for non-computable things it will complain
- but: can help by giving *code equations*: `lemma[code]: "f a = ..."`
- code generator is outside the trusted inference kernel ("oracle")

A Note on `value`

Isabelle/HOL makes no claim to be computable!

→ `value` command fails for things like `THE`, `SOME`, `undefined`, …

… but we can still use it "most of the time"

`value` exports your definitions to Standard ML

- for non-computable things it will complain
- but: can help by giving *code equations*: `lemma[code]: "f a = ..."`
- code generator is outside the trusted inference kernel ("oracle")

Technically, no guarantee that `value` agrees with the logic!

Sets

Sets are "just" `'a ⇒ bool` inside a new type:

$\texttt{x} \in \texttt{A}$	$\{\texttt{x}.\ \texttt{P x}\}$
$\approx$	$\approx$
$\texttt{A x}$	$\lambda\texttt{x}.\ \texttt{P x}$

Use two axioms to populate the new type.

- these are "small" sets (not sets as in ZF)
- (full ZFC sets can be added to HOL Paulson 2019, but not needed in practice)

Defining new types

So far all types were simply **typedecl**'d and then populated by axioms.

Safer way to define types:

```
typedef foo = "{x. ...}"
  by (...)
```

- new (sub-)type containing the elements of the given set
- need to prove that the set is non-empty!
- explicit coercions: `Abs_foo` and `Rep_foo`

Infinity & Natural Numbers

Axiom of infinity:

$$\exists (s :: \text{ind} \Rightarrow \text{ind})(z :: \text{ind}).\ (\forall x\ y.\ s\ x = s\ y \longrightarrow x = y) \land (\forall x. s\ x \neq z))$$

What does this do?

Infinity & Natural Numbers

Axiom of infinity:

$$\exists (s :: \mathsf{ind} \Rightarrow \mathsf{ind})(z :: \mathsf{ind}).\ (\forall x\ y.\ s\ x = s\ y \longrightarrow x = y) \wedge (\forall x. s\ x \neq z))$$

With z and s we can encode natural numbers!

Infinity & Natural Numbers

Axiom of infinity:

$$\exists (s :: \mathrm{ind} \Rightarrow \mathrm{ind})(z :: \mathrm{ind}).\ (\forall x\ y.\ s\ x = s\ y \longrightarrow x = y) \wedge (\forall x. s\ x \neq z))$$

With z and s we can encode natural numbers!

```
inductive Nat :: "ind ⇒ bool" where
    Zero_RepI: "Nat z"
  | Suc_RepI: "Nat i ⟹ Nat (s i)"
```

Infinity & Natural Numbers

Axiom of infinity:

$$\exists (s :: \mathsf{ind} \Rightarrow \mathsf{ind})(z :: \mathsf{ind}).\ (\forall x\ y.\ s\ x = s\ y \longrightarrow x = y) \land (\forall x. s\ x \neq z))$$

With z and s we can encode natural numbers!

```
inductive Nat :: "ind ⇒ bool" where
    Zero_RepI: "Nat z"
  | Suc_RepI: "Nat i ⟹ Nat (s i)"

typedef nat = "{n :: ind. Nat n}"
  morphisms Rep_Nat Abs_Nat using Nat.Zero_RepI by auto

definition "Zero ≡ Abs_Nat z"
definition "Suc n ≡ Abs_Nat (s (Rep_Nat n))"
```

Back to “normal” Isabelle/HOL: datatypes

The `datatype.thy` theory provides an easier way to do such constructions

```
datatype nat = Zero | Suc n
```

Internally, performs a (more general) construction & proves additional lemmas

```
datatype 'a list = Nil | Cons 'a "'a list"

term "Rep_list :: 'a list ⇒ 'a list_IITN_list
      ⇒ (nat + unit + nat × nat) set ⇒  'a + bool"
```

… and with that, we're “back” to what we know

Why trust these axioms?

Why trust these axioms?

Classicly, consistency is shown by providing a semantics:

- The classic semantics for Gordon's HOL given by Andrew Pitts Pitts 1993
- Newer version to reflect Isabelle/HOL's ad-hoc overloading Kunar and Popescu 2017

Why trust these axioms?

Classicly, consistency is shown by providing a semantics:

- The classic semantics for Gordon's HOL given by Andrew Pitts Pitts 1993
- Newer version to reflect Isabelle/HOL's ad-hoc overloading Kunar and Popescu 2017

Both of these are pen-and-paper, and not part of Isabelle!

Why trust these axioms?

Classicly, consistency is shown by providing a semantics:

- The classic semantics for Gordon's HOL given by Andrew Pitts Pitts 1993
- Newer version to reflect Isabelle/HOL's ad-hoc overloading Kunar and Popescu 2017

Both of these are pen-and-paper, and not part of Isabelle!

- Maintainer: "We did not get a report of genuine problems in the Isabelle inference kernel for so many years, that I begin to feel uneasy"

Why trust these axioms?

Classicly, consistency is shown by providing a semantics:

- The classic semantics for Gordon's HOL given by Andrew Pitts Pitts 1993
- Newer version to reflect Isabelle/HOL's ad-hoc overloading Kunar and Popescu 2017

Both of these are pen-and-paper, and not part of Isabelle!

- Maintainer: "We did not get a report of genuine problems in the Isabelle inference kernel for so many years, that I begin to feel uneasy"
- (however: oracles work outside the inference kernel, use `thm_oracles` to check)

Why trust these axioms?

Classicly, consistency is shown by providing a semantics:

- The classic semantics for Gordon's HOL given by Andrew Pitts Pitts 1993
- Newer version to reflect Isabelle/HOL's ad-hoc overloading Kunar and Popescu 2017

Both of these are pen-and-paper, and not part of Isabelle!

- Maintainer: "We did not get a report of genuine problems in the Isabelle inference kernel for so many years, that I begin to feel uneasy"
- (however: oracles work outside the inference kernel, use `thm_oracles` to check)
- Proof Checker for metalogic $\mathcal{M}$ formalised within Isabelle Nipkow and RoSSkopf 2021

Conclusion

You should be able to answer now:

- How the core of HOL works
- How HOL and the meta logic interact
- How to manually apply rules

Until next week:

- Download and work on the fourth exercise sheet
- Submit your solution to ILIAS

See you next week! :)