



Theorembeweiserpraktikum

Quotients, Axioms, Booleans

Jakob von Raumer, Sebastian Ullrich | SS 2021



THEOREM PROVER

General Notes on Last Week's Exercises

- maybe a bit long, sorry about that!
- make sure that your proofs are properly structured:
 - `simp / simp_all` (without `only`) only at the end of a tactic block (e.g. using `have`)
 - don't close multiple goals at different points of a single tactic block (e.g. using `focus / case`)
- prefer `have` over `theorem` for use-once subproofs
 - see also Zulip for some improvements around it, among others
- doing case analysis on something like `h : a ∈ b :: bs` is surprisingly hard for Lean!
 - `match` can only do it if the inductive *indices* are variables (e.g. on `h : a = b`)
 - uses `cases` instead
- make sure your proofs and terms are properly indented
- as always, see the sample solution for concise solutions and other tricks

Reminder: Projects Are upon Us

- 1.6. – 5.7.2021
- in groups of two or three people
- verification of a simple compiler optimization
- alternative for mathematicians: Eulerian graphs

Quotients: Motivation

- Sometimes, not enough elements of a type are equal.
- Want to give an arbitrary binary predicate and *make* them equal.
- In mathematics: The real numbers as a quotient on Cauchy sequences.

Quotients in Lean

Quotients in Lean work like in set theory: Given

- A type $\alpha : \text{Sort } u$ and
- a binary relation $r : \alpha \rightarrow \alpha \rightarrow \text{Prop}$ on α ,

we get

- the quotient type $\text{Quot } r$,
- with a constructor $\text{Quot.mk } r : \alpha \rightarrow \text{Quot } r$,
- an axiom stating

```
axiom sound :  $\forall \{ \alpha : \text{Sort } u \} \{ r : \alpha \rightarrow \alpha \rightarrow \text{Prop} \} \{ a \ b : \alpha \}, r \ a \ b \rightarrow \text{Quot.mk } r \ a = \text{Quot.mk } r \ b$ 
```

- and an induction principle which *lifts* functions on the base type to functions on the quotient:

```
Quot.lift :  $\{ \alpha : \text{Sort } u_1 \} \rightarrow \{ r : \alpha \rightarrow \alpha \rightarrow \text{Prop} \} \rightarrow \{ \beta : \text{Sort } u_2 \} \rightarrow$   

 $(f : \alpha \rightarrow \beta) \rightarrow (\forall (a \ b : \alpha), r \ a \ b \rightarrow f \ a = f \ b) \rightarrow \text{Quot } r \rightarrow \beta$ 
```

Quotients in Lean

- Note that we didn't require r to be an equivalence relation.
- Objects of $\text{Quot } r$ are still the equivalence classes of the *reflexive-transitive-symmetric* closure of r since equality itself is reflexive, transitive, and symmetric.
- Main advantage of using $\text{Quot } r$ and $=$: We cannot rewrite along the relation r , but along $=$.

Propositional Extensionality

- sound for quotients was an axiom postulating an equality which otherwise could not be proved.
- The concept of *extensionality*: Two objects should be equal if they have the same observable properties.
- The only observable property of a proposition is its truth value, so *propositional extensionality* takes the form

```
axiom propext {a b : Prop} : (a ↔ b) → a = b
```

Functional Extensionality

- The observable property of a function is that we can evaluate it at any point.
- So we have *functional extensionality* equating functions which are *pointwise equal*:

theorem funext $\{f_1\ f_2 : \forall (x : \alpha), \beta\ x\} (h : \forall x, f_1\ x = f_2\ x) : f_1 = f_2 := \text{---}\dots$

- Proof idea:

- 1 Quotient $\gamma := \forall (x : \alpha), \beta\ x$ by pointwise equality \sim .
- 2 Construct a function $e : \gamma / \sim \rightarrow \gamma$ such that $e([f]) \equiv f$.
- 3 Have

$$f_1 \equiv e([f_1]) = e([f_2]) \equiv f_2.$$

The Axiom of Choice

- Set-theoretic version: For every set X of non-empty sets there exists a function f such that $f(A) \in A$ for every $A \in X$.
- If we want to formalize this in type theory, we fail since non-emptiness is a proposition which we cannot match on if our goal is not a proposition!
- So in type theory, the axiom states exactly that we can *extract the element of a non-empty type*:

```
namespace Classical  
  
axiom choice { $\alpha$  : Sort u} : Nonempty  $\alpha \rightarrow \alpha$ 
```

- The law of excluded middle `em` is then derived via a construction called *Diaconescu's theorem*.

A Remark on `Bool` and `Prop`

- In programming, the type `Bool` is used for propositions, why can't we do that?
- Constructive logic with `Bool` is always decidable!
- Lean still has `Bool` valued logical connectives `or`, `and`, `...`
- There is a `Bool` valued equality consisting of a type class

```
class BEq ( $\alpha$  : Type u) where  
  beq :  $\alpha \rightarrow \alpha \rightarrow \text{Bool}$ 
```

- It is used by some of Lean's own implementations like `filter`.
- Can be overwritten with a new instance, to modify it.
- If equality on a type is decidable, we have the instance

```
instance [DecidableEq  $\alpha$ ] : BEq  $\alpha$  where  
  beq a b := decide (Eq a b)
```

Other Helpful Stuff: Options

```
set_option pp.notation false
#check [] ++ [1, 2] -- HAppend.hAppend List.nil (List.cons 1 (List.cons 2 List.nil)) : List Nat
```

pp.explicit	show implicit parameters using @
pp.all	show everything
trace.Meta.Tactic.simp	show applied simp theorems can be used to move from simp to simp only

Can also be done inside a term or tactic block using `set_option opt val in`