The Booleans

Heshan Du
University of Nottingham Ningbo China

October 2024

Aims and Learning Objectives

- To be able to understand the definitions of booleans and functions on booleans.
- To be able to apply the tactics *change*, *contradiction*, *dsimp* in Lean.
- To be able to construct proofs for propositions involving booleans using Lean.

Reading

- Thorsten Altenkirch, *Introduction to Formal Reasoning*, 2023.
  - Chapter 5. The Booleans

The Booleans

- The logic we have introduced so far was very generic.
- We fix this by looking at a very simple type, the booleans *bool* which has just two elements *tt* (for *true*) and *ff* (for *false*), and functions on this type.
- Then we are going to use predicate logic to prove some simple theorems about booleans.

bool

In Lean, *bool* is defined as an inductive type:

```
inductive bool : Type
| ff : bool
| tt : bool
```

This declaration means:

- There is a new type *bool* : *Type*.
- There are two elements *tt ff* : *bool*.
- These are the only elements of *bool*.
- *tt* and *ff* are different elements of *bool*.

Functions on bool: bnot

- Let us define negation on booleans. This is a function $bnot : bool \to bool$.
- By a function here, we mean something which we can feed an element of the input type (here *bool*) and it will return an element of the output type (here *bool*).
- How to define the function *bnot*?

```
namespace bool

def bnot : bool → bool
| tt := 
| ff :=

end bool
```

Functions on bool: band

- To define a function with two inputs, like *and* for booleans *band*, we use *currying*.
- This is, *band* applied to a boolean returns a function which applied to the second boolean returns a boolean, hence $band : bool \rightarrow bool \rightarrow bool$.
- As we have already seen $\rightarrow$ is right associative, hence, putting the extra brackets in the type is $band : bool \rightarrow (bool \rightarrow bool)$.

Defining band

How to define *band*?

```
def band : bool → bool → bool
|
|
```

Defining bor

How to define *bor*?

```
def bor : bool → bool → bool
| 
|
```

Operations on bool

The lean prelude also introduces the standard infix notation for operations on *bool*:

```
x && y := band x y
```

and

```
x || y := bor x y
```

Evaluate boolean expressions

We can evaluate boolean expressions using #*reduce*:

```
namespace bool

#reduce ff && (tt || ff)
#reduce tt && (tt || ff)

end bool
```

Proving some basic properties

- To reason about *bool*, we can use *cases x* to analyze a variable $x : bool$, which means that there are two possibilities *tt* and *ff*.
- How to prove in predicate logic that every element of *bool* is either *tt* or *ff*?

```
namespace bool

example : ∀ x : bool, x = tt ∨ x = ff :=
begin
    
end

end bool
```

Proving $tt \neq ff$

How to prove that $tt \neq ff$?

```
namespace bool

theorem cons : tt ≠ ff :=
begin

end

end bool
```

Defining *is_tt*

- How to prove that $tt \neq ff$?
- The idea is to define a predicate $is\_tt \; : \; bool \to Prop$. How to define it?

The tactic *change*

- The tactic *change* replaces the current goal with another one that is definitionally the same.
- For example, since $\mathit{is\_tt}\ \mathit{ff} = \mathit{false}$, we use *change is_tt ff* to replace the goal *false*.

The tactic *contradiction*

- Since this is a common situation Lean provides the tactic *contradiction*, we can use it to prove $tt \neq ff$. How?
- The tactic *contradiction* will solve the current goal, if there is an inconsistent assumption like assuming that two different constructors of an inductive type are equal.

Proving equations about bool

- Next let us prove some interesting equalities.
- We are going to revisit our old friend, *distributivity*, but this time for booleans.
- How to prove $\forall\, x\ y\ z\ :\ bool, x\ \&\&\ (y\ ||\ z) = x\ \&\&\ y\ ||\ x\ \&\&\ z$ in Lean?

```
namespace bool

theorem distr_b : ∀ x y z : bool,
    x && (y || z) = x && y || x && z :=
begin

end

end bool
```

Tactic *dsimp*

- We can instruct Lean to use the definition of && (i.e., *band*) by saying *dsimp [band]*.
- Similarly, we can instruct Lean to use the definition of || (i.e., *bor*) by saying *dsimp [bor]*.
- Write down the proof for $\forall\ x\ y\ z\ : bool, x\ \&\&\ (y \mid\mid z) = x\ \&\&\ y \mid\mid x\ \&\&\ z$ using *dsimp*.
- Write down the proof for $\forall\ x\ y\ z\ : bool, x\ \&\&\ (y \mid\mid z) = x\ \&\&\ y \mid\mid x\ \&\&\ z$ *without* using *dsimp*.

Exercise 1

How to prove $\forall\ x\ y\ : bool,\ bnot(x \mid\mid y) = bnot\ x\ \&\&\ bnot\ y$?

```
namespace bool

theorem dm1_b : ∀ x y : bool, bnot (x || y) = bnot x && bnot y  :=
begin

end

end bool
```

Exercise 2

How to prove $\forall\ x\ y\ : bool,\ bnot(x\ \&\&\ y) = bnot\ x\ ||\ bnot\ y$?

```
namespace bool

theorem dm2_b : ∀ x y : bool, bnot (x && y) = bnot x || bnot y  :=
begin

end

end bool
```

Relating bool and Prop

- We seem to define logical operations twice: once for *Prop* and once for *bool*. How are the two related?
- Indeed, we can use *is_tt* for example to relate $\wedge$ and $\&\&$.
- How to prove $\forall\ x\ y\ : bool,\ is\_tt\ x \wedge is\_tt\ y \leftrightarrow is\_tt(x\&\&y)$?

```
namespace bool

def is_tt: bool → Prop
| tt := true
| ff := false

theorem and_thm : ∀ x y : bool, is_tt x ∧ is_tt y ↔ is_tt (x && y) :=
begin

end

end bool
```

Exercise 3

How to prove $\forall\ x\ : bool,\ \neg\ (is\_tt\ x) \leftrightarrow is\_tt\ (bnot\ x)$?

```
namespace bool

def is_tt: bool → Prop
| tt := true
| ff := false

theorem not_thm : ∀ x : bool, ¬ (is_tt x) ↔ is_tt(bnot x) :=
begin

end

end bool
```

Exercise 4

How to prove $\forall\ x\ y\ : bool,\ is\_tt\ x\ \vee\ is\_tt\ y \leftrightarrow is\_tt\ (x \mid\mid y)$?

```
namespace bool

def is_tt: bool → Prop
| tt := true
| ff := false

theorem or_thm : ∀ x y : bool, is_tt x ∨ is_tt y ↔ is_tt (x || y) :=
begin

end

end bool
```