

# Booleans, the original true-false dichotomy

Marc Coiffier

## Contents

Functions on Booleans . . . . . 1

'utils require import

- Required module: utils

Booleans can have two values, in any given universe. First, we define the Boolean context :

```
'Bool-context [ { Type ' .Bool } { .Bool ' .true } { .Bool ' .false } ] def
```

In this context, the type of booleans is simply the .Bool type in context, and the 'true' and 'false' values are respectively the 'true' and 'false' hypotheses.

```
'Bool Bool-context { .Bool } prods "Boolean" defconstr
'true Bool-context { .true } funs "true" defconstr
'false Bool-context { .false } funs "false" defconstr
[ 'Bool 'true 'false ] { export } each
```

We can test that *true* and *false* have the correct type :

- type of *true* : *Boolean*
- type of *false* : *Boolean*
- type of  $\lambda(b : \text{Boolean}).\mu(b) : \forall(b : \text{Boolean})(\text{Bool}^P : \text{Boolean} \rightarrow \text{Set}_1), \text{Bool}^P \text{ true} \rightarrow \text{Bool}^P \text{ false} \rightarrow \text{Bool}^P b$

## Functions on Booleans

Then, we can start defining first-level combinators, such as 'not', 'and' and 'or' :

```
'not { Bool 'b } Bool-context # { b ( .Bool .false .true ) } funs def
'and { Bool 'x } { Bool 'y } Bool-context # #
  { x ( .Bool y ( .Bool .true .false ) .false ) } funs def
'or { Bool 'x } { Bool 'y } Bool-context # #
  { x ( .Bool .true y ( .Bool .true .false ) ) } funs def
'implies { Bool 'x } { Bool 'y } Bool-context # #
  { x ( .Bool y ( .Bool .true .false ) .true ) } funs def
```

As always, we should verify the type of our combinators, and test whether they truly conform to their specification :

```
[ 'not 'or 'and 'implies ] { dup $ type swap " - $s : %l$\n" printf } each
```

- $not : Boolean \rightarrow Boolean$ 
  - $or : Boolean \rightarrow Boolean \rightarrow Boolean$
  - $and : Boolean \rightarrow Boolean \rightarrow Boolean$
  - $implies : Boolean \rightarrow Boolean \rightarrow Boolean$