Practical examples of writing programs and proving theorems in Idris.

Donovan Crichton

January 2020

Preliminaries

Propositional Logic

- Concerned with statements of verifiable facts.
- Used daily by programmers when reasoning about Boolean values.

Symbol	**Meaning**	**Example**
T, F	True, False	Boolean values.
$p, q, r, \dots$	Propositions	Let $p =$ "It is raining."
$\neg$	Negation (Not)	$\neg p$
$\wedge$	Conjuction (And)	$p \wedge q$
$\vee$	Disjunction (Or)	$p \vee q$
$\rightarrow$	Implication (If)	$p \rightarrow q$
$\leftrightarrow$	Bi Implication (Iff)	$p \leftrightarrow q$
$\equiv$	Equivalence	$p \equiv q$
$\top$	Tautology	$p \vee \neg p \equiv \top$
$\bot$	Contradiction	$p \wedge \neg p \equiv \bot$

Definitions of Connectives

Conjunction (And)

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Disjunction (Or)

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Negation (Not)

p	$\neg p$
T	F
F	T

Implication (If)

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Bi Implication (Iff)

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

Logical Equivalence

p	q	$p \equiv q$
T	T	T
T	F	F
F	T	F
F	F	T

Proof Techniques

By Exhaustion

Idea: Prove by enumerating all possible cases.

Prove: $(\neg p \lor q) \leftrightarrow (p \to q)$.

p	q	$\neg p$	$\neg p \lor q$	$p \to q$	$(\neg p \lor q) \leftrightarrow (p \to q)$
T	T	F	T	T	T
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

Proof Techniques

By Appeal to Lemma

Idea: Introduce pre-proven smaller proofs (called a Lemma) to prove a larger proof.

- **Lemma 1.** $p \vee \neg p \equiv \top$.
- **Lemma 2.** $(p \equiv q) \equiv (p \leftrightarrow q)$.

Prove: $(p \leftrightarrow q) \vee \neg(p \equiv q) \leftrightarrow \top$.

$$\begin{aligned} (p \leftrightarrow q) \vee \neg(p \equiv q) &\leftrightarrow \top && \text{Premise.} \\ (p \equiv q) \vee \neg(p \equiv q) &\equiv \top && \text{Lemma 2.} \\ \top &\equiv \top && \text{Lemma 1.} \end{aligned}$$

$\square$

First Order (or Predicate) Logic

Extends propositional logic from reasoning about propositions to reasoning about sets.

Symbol	**Meaning**	**Example**
$X, Y, Z, \ldots$	Set Variables	Let $Y = \{2, 3, 4\}$.
$x, y, z, \ldots$	Member Variables	Let $z = 2$.
$P(x), Q(y), \ldots$	Predicate Variables	Let $Q(y) = y > 1$.
$\forall x \in X, P(x)$	Universal Quantifier	$\forall y \in Y, Q(y)$
$\exists x \in X, P(x)$	Existential Quantifier	$\exists z \in Y, z = 2$

- Allows us to prove that a property $P(x)$ holds $\forall x \in X$. Provided X is well-founded.
- Informally well-founded means "no infinite decreasing chains".

Prove: $\forall n \in \mathbb{N}(\exists y \in \mathbb{N}, y = n + 1)$

Base Case. $n = 0$

$$\begin{aligned} y &= 0 + 1 \\ &= 1 \end{aligned}$$

$\square$

Inductive Step. $n = k + 1$

$$\begin{aligned} y &= (k + 1) + 1 \\ &= k + 2 \end{aligned}$$

$\square$

Why should I care?

- Types are just sets with flavour!
- **Bool** $= \{True, False\}$
- **Int** $= \{-\infty, ..., -2, -1, 0, 1, 2, 3, ..., \infty\}$
- Mixing of flavours is not allowed!
- $\{True, -2, "Hello", 1\}$ Can really only be said to be a "thing" flavoured set.

Propositions as Types. Proofs as Programs

- The Curry-Howard-Lambeck correspondence is well known amongst Haskell programmers for the correspondence between categories and programming.
- The correspondence with logic is less often discussed.
- Holds for any language that is based on a typed lambda calculus.

Idea: A type is a propostion.

What is Truth?

Propositional Logic and Predicate Logic consider truth to be the Boolean value "True". These logics also have a notion of vacuous truth.

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	T

In Predicate Logic: $\forall x \in \{\} P(x)$ is also true.

- If a type is a proposition, what does it mean for it to be true?
- A type is true iff it is inhabited with a value.

Curry-Howard in Idris

Logic Term	**Logic Symbol**	**Idris Symbol**	**Idris Type**
Implication	$p \Rightarrow q$	`p -> q`	Arrow
Conjunction	$p \wedge q$	`(p, q)`	Pair (Product)
Disjunction	$p \vee q$	`Either p q`	Enum (Sum)
Negation	$\neg\, p$	`p -> Void`	Void Type
IFF/Eq	$p \Leftrightarrow q$, $p \equiv q$	`(p -> q, q -> p)`	Pair Arrows
Universal	$\forall\, x.\ P\ x$	`p -> Type`	Π Type
Existential	$\exists\, x.\ P\ x$	`(x ** P x)`	Σ Type
$=$	$=$	`p = q`	Type Equality
$\top$	True	`()`	Unit Type
$\bot$	False	`Void`	Uninhabited

Natural Numbers

Let $\mathbb{N}$ denote the set of natural numbers where:
1. Zero (0) is a natural number.
2. If k is a natural number, then the successor of k is also a natural number.

```
data Nat : Type where
  Z : Nat
  S : (k : Nat) -> Nat
```

- **Nat : Type** is the type constructor.
- **S** and **K** are the value constructors.

Proving commutativity on addition in Idris

```
(+) : Nat -> Nat -> Nat
Z     + y = Z
(S k) + y = S (k + y)
```

$\forall x, y \in \mathbb{N}. x + y = y + x$

```
plusIsCommutative : (x, y : Nat) -> x + y = y + x
plusIsCommutative x y = ?what
```

$\frac{x,y:Nat}{what:x+y=y+x}$

```
plusIsCommutative : (x, y : Nat) -> x + y = y + x
plusIsCommutative Z y = ?t1
plusIsCommutative (S k) y = ?t2
```

$\frac{x,y:Nat}{t1:y=y+Z}$

$\frac{x,y:Nat}{t2:S(k+y)=y+(S\ k)}$