CS3110 Spring 2016 Lecture 13: Developing Constructive Analysis Continued

This lecture will cover two topics:

1. Specifying problems about real numbers.
2. Prelim structure and review.

1 Specifying problems about real numbers

We want to examine how to specify computational problems using types and logical formulas. Because Bishop's logical formulas all have a computational meaning, they are closely related to the logical operators we defined in OCaml, namely $\alpha * \beta$ for $\alpha \,\&\, \beta$, $\alpha \to \beta$ for $\alpha \Rightarrow \beta$, and $L\alpha|R\beta$ for $\alpha \vee \beta$.

We will extend these types to include types not yet in OCaml but used in the proof assistants Agda, Coq, and Nuprl. These are called *dependent types*. They are $\forall x : \mathbb{R}.\alpha(x)$ and $\exists x : \mathbb{R}.\alpha(x)$. We will gradually build up our knowledge of these logical formulas as types.

$\forall x{:}\mathbb{R}.\alpha(x)$ is $x{:}\mathbb{R} \to \alpha(x)$, and a *function* type.
$\exists x{:}\mathbb{R}.\alpha(x)$ is $x{:}\mathbb{R} * \alpha(x)$, a *product* type.

Properties of the order relations on $\mathbb{R}$

Proposition 2.16 in Bishop:

$$\forall x_1, ..., x_n : \mathbb{R}.(x_1 + ... + x_n > 0) \Rightarrow \exists i{:}\mathbb{N}.(1 \leq i \leq n \,\&\, x_i > 0)$$

Proof:

By Lemma 2.15, if $x < y$ then we can find a rational number α such that $x < \alpha < y$. Therefore there is a rational α such that $0 < \alpha < (x_1 + ... + x_n)$. We can find rationals a_i such that $|x_i - a_i| < \frac{1}{2n} * \alpha$.

Therefore $a_i > \dfrac{\alpha}{2n}$ for some i. (See Bishop page 26 for more detail.)

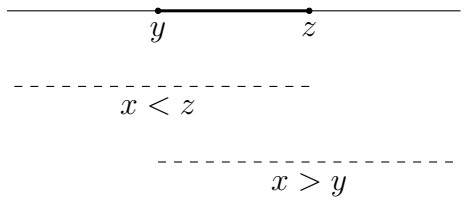
For this i we therefore know $x_i \geq (a_i - |x_i - a_i|) > 0$

QED

Corollary 2.17:

$$\forall x, y, z : \mathbb{R}.(y < z \Rightarrow x < z \ \vee \ x > y)$$

Remember that the constructive interpretation of 'or', $x < z \vee x > y$, is that we computationally know which inequality holds, $x < z$ or $x > y$.



Notice that as our types become more complex, the nature of relations such as $=$, $<$, $\leq$, etc. become more complex.

For integers, $\mathbb{Z}$, the meaning of equality, $x = y$ in $\mathbb{Z}$ or $x =_{\mathbb{Z}} y$, is a simple concept. Indeed we can *decide* on equality on $\mathbb{Z}$ and $\mathbb{Q}$.

$$0 = 0, \sim(0 = 1), 1 = 1, \sim(1 = -1), \sim (0 = 1), ...$$

For $\mathbb{Q}$ equality can be reduced to $=_{\mathbb{Z}}$, namely $\frac{2}{4} = \frac{1}{2}$ by "cross-multiplication", $2 * 2 = 1 * 4$.

But equality on $\mathbb{R}$ is a complex relation, see Bishop page 18.

Definition 2.1.2. $(x_n) = (y_n)$ iff $|x_n - y_n| \leq \frac{2}{n}$ *for all* $n \in \mathbb{Z}^+$

This relation is *not decidable*, and we cannot "just compute" whether $(x_n) = (y_n)$. So we cannot prove $\forall x, y : \mathbb{R}.(x = y \vee \sim(x = y))$.

Can you think of a way to prove this?

We can prove $\forall x, y : \mathbb{Q}.(x = y \vee \sim(x = y))$.

A constructive proof of this proposition actually provides a *decision procedure* for equality.

That is, given q_1, q_2 in $\mathbb{Q}$, we *can decide* whether $q_1 = q_2$ or $\sim (q_1 = q_2)$.

Bishop believes that all mathematics should have *numerical meaning*. This is an idea that was gradually refined in the late 1800's. We have mentioned that it was an idea tested in the development of analysis. L.E.J. Brouwer in 1907 showed that there is a natural logic for keeping track of numerical and computational meaning. He named this approach as *intuitionism*, suggesting that this is a natural way to reason about the computational processes that underlie mathematics.

We have already seen glimpses of this idea in the OCaml type system. We have noted that &, $\vee$, $\Rightarrow$, and not ($\sim$) can be captured well using types.

Following Bishop, we want to extend this interpretation to the quantifiers, $\forall x : T.\alpha(x)$ and $\exists x : T.\alpha(x)$. We need to exploit an idea not treated in OCaml explicitly. It is the idea of a parameterized type such as $\alpha(x)$ or a *dependent type*.

We want to be able to define the computational meaning of $\forall x, y : \mathbb{Q}.(x = y) \vee \sim(x = y)$ and $\forall x, y : \mathbb{R}.(x = y) \vee \sim(x = y)$.

We'll see that the statement about $\mathbb{Q}$ is "computationally justified' but the second, about $\mathbb{R}$, is not.

2 Prelim structure and review

The prelim will follow this pattern:

1. Syntax of OCaml
2. Evaluation rules and computation
3. Typing rules and type inference

4. Logical Specifications

5. Programming specifications and verification