CS3110 Fall 2013 Lecture 23: Prelim Review (11/19)

Robert Constable

1 Connections to Discrete Math CS2800 (a prerequisite)

- notations corresponding to set notations

  $\{x : A \,|\, B(x)\}$ or $\{x \in A \,|\, B(x)\}$

  $\{1, 2, 3, \ldots, n\}$

- operations on sets $\cup, \cap, -$ $A \cup B$, $A \cap B$, $A - B$
- relations in type theory
- functions and graphs
- lists as sets

There were many excellent questions (see the fix of bug), so we did not cover using `efix` to define recursive functions. This will be done in recitations.

2 Lectures

Lecture 20	(Mike)	folded into Lect 21 also we posted Kozen's notes
Lecture 21	Type Theory	Board Notes available since lecture, eventually pdf
Lecture 22	(Robbert)	Posted Kozen notes, *not on Prelim II*
Lecture 23	Prelim Review	
Lecture 24	Logic in Type Theory – might help in Prelim but no specific question on this topic	
Lecture 25	Types as Propositions	
Lecture 26	Review of Course	
Lecture 27	Yaron Minsky (Jane Street) guest lecture	

3 Connections to Discrete Math (CS2800)

Notations for sets

$\{1, 2, 3, 4\}$ $\quad$ $\{x \in \mathbb{N} \mid x = 1 \vee x = 2 \vee x = 3 \vee x = 4\}$

$\{x \in \mathbb{N} \mid P(x)\}$
$\{x \ : \mathbb{N} \mid P(x)\}$ $\quad$ $(x : nat \text{ where } b(x))$

$\{x : \mathbb{N} \mid Even(x) \ \& \ Odd(x^2)\}$

In OCaml SL this is ?

What is interesting about this type?

It is empty!
Note, in OCaml, there are NO empty types since each has a diverging element.

4 Sets vs Types continued

$\{x : \mathbb{R}\ list \,|\, len(x) = 2\}$

What are these?

Define in OCaml SL

$\mathbb{R}$ defined before

Can we define vectors this way?

```
let vec (n : nat) : type = (v : real list where len v = n)
```

This uses type valued function as opposed to *parameterized* type, `vec(n)`.

$$\texttt{circle = (v : vec(2) where let [x;y] = v in } \underbrace{\texttt{x}^2\texttt{ + y}^2\texttt{ = 1}}_{\text{ERROR SEE BELOW (Bad Type Def)}}\texttt{)}$$

radius 1

We saw the type Euclid before. Here is the type *Goldbach*.

```
 n : (x : even where x >= 6) -> (p : nat * nat where
                                 let (p1, p2) = p in
                                 prime(p1) & prime(p2) &
                                 odd(p1) & odd(p2) & n = p1 + p2)
```

Bad Type Definition

The type giving a circle of radius 1 has an error because $x^2 + y^2 = 1$ is not an equality that can be reduced to a Boolean! Equality must be defined on real, as we have done in lecture. But to use this equality in a "where" clause, it must be decidable. Equality on reals is not decidable.

To get a better type for vectors we should use finite precision as with rational numbers.

Comment on parameterized types

Notice that when we treat vectors as parameterized types, say of rationals, `rat`,

```
(v : rat list where len v = n),
```

call this `rat_vec(n)`, the parameter `n` becomes bound when we use this in a complete type such as

$$\underbrace{\texttt{n : nat -> rat\_vec(n)}}_{\text{binds the parameter}}$$

A function of this type, say `f`, on input 17 will produce a 17 dimensional vector.

5 Lists as sets

In Discrete Math the textbooks discuss sets not types. We can represent finite sets of numbers, strings, characters, etc. as lists. This idea works for any type with a decidable equality. A list represents a finite set iff

1. There are no duplicates among the list's elements and
2. Lists are considered equal regardless of order.

This type can be defined in OCaml SL because we are required to provide the equality relation. Note that equality as lists is decidable if the elements have decidable equality.

In Discrete Math, functions are defined in terms of relations (the single-valued ones) on a *set* of ordered pairs, say $A \times B$. The definition is subtle.

In OCaml SL, functions are a basic type, either $\alpha \to \beta$ or $x : \alpha \to \beta(x)$. We can define a corresponding set if the types α, β are finite. It is a list of ordered pairs.

In a *lazy language*, we can define *streams* (see Kozen notes), and represent graphs of unbounded functions, say `int -> int`.