CS3110 Fall 2013 Lecture 10: The Type of Rational Numbers (in OCaml and Mathematics) (10/1)

Robert Constable

1 Lecture Plan

- Binding and scope revisited
- Abstract syntax for λ-calculus
- Mathematical type for rational numbers
  $\mathbb{Z}$, $\mathbb{Q}$
- OCaml type for rational numbers $\mathbb{Q}$
  using *int*
  requires a relation as well
- Algebraic properties of $\mathbb{Z}$ and $\mathbb{Q}$
  modules and algebraic structure
  further properties of $\mathbb{Q}$
  equality
- Canonical form and greatest common divisor
  GCD definition
  GCD theorem
  gcd in OCaml

2 Binding and scope revisited

In Lecture 9 we looked at an hypothetical version of OCaml with lazy evaluation, and we discussed the capture problem.

There is a way to model lazy evaluation and you can see how OCaml responds to problems of this sort.

Here is the example from lecture with eager evaluation

$$\textbf{let } x = 5 \text{ in } \big(\lambda y.\,(\lambda x.\,y * x)\ x\big)\ 3 \ \downarrow 15$$

Here is a simulation of lazy evaluation

$$\textbf{let } x = \lambda x.\,5 \text{ in } \big(\lambda y.\,(\lambda x.\,y * x\ ())\big)\ 3$$

capture

warning 26: unused variable x

$$\textbf{let } x = \lambda x.\,5 \text{ in } \big(\lambda y.\,(\lambda x.\,y * x\ ())\ x\big)\ 3 \ \downarrow 15$$

```
(* NORMAL USE OF LET, NO CAPTURE, EAGER EVALUATION *)

# let x = 5 in (fun y -> (fun x -> y * x) x ) 3 ;;
- : int = 15

(* CAPTURE OF let x = 5 BY INNER (fun x -> y * x)  *)

# let x = 5  in (fun y -> (fun x -> y * x) ) 3 ;;
Characters 4-5:
  let x = 5  in (fun y -> (fun x -> y * x) ) 3 ;;
      ^
Warning 26: unused variable x.
- : int -> int = <fun>


(* SIMULATING LAZY EVALUATION IN OCaml     *)

# let x = (fun x -> 5)  in (fun y -> (fun x -> y * x () ) x ) 3 ;;
- : int = 15


(* CAPTURE IN LAZY EVALUATION *)

# let x = (fun x -> 5)  in (fun y -> (fun x -> y * x () ) ) 3 ;;
Characters 4-5:
  let x = (fun x -> 5)  in (fun y -> (fun x -> y * x () ) ) 3 ;;
      ^
Warning 26: unused variable x.
- : (unit -> int) -> int = <fun>

(* FIXING THE PROBLEM BY CHANGING SCOPE *)
                              ^

# let x = (fun x -> 5)  in (fun y -> (fun x -> y * x () ) x ) 3 ;;
- : int = 15


(* SIMPLE CASE *)

# (fun y -> ( (fun x -> y * x ()) ) ) 3 ;;
- : (unit -> int) -> int = <fun>
```

fun x → 3 * x ()

3 Abstract syntax for the Lambda Calculus

Abstract syntax examples

```
type id = Id of string;;

# type term = Lambda of (id * term) | Ap of (term * term)
              | Var of string ;;
type term = Lambda of (id * term) | Ap of (term * term)
  | Var of string

# Lambda ( Id "x1", Ap (Var "x1", Var "x1"));;
- : term = Lambda ( Id "x1", Ap (Var "x1", Var "x1"))
```

4 Rational Numbers in Mathematical Types

A standard way to define the type of rational numbers, denoted $\mathbb{Q}$, is as "fractions" p/q where

$p \in \mathbb{Z} = \{0, -1, +1, -2, +2, \ldots\}$ and
$q \in \mathbb{Z} - \{0\} = \{-1, +1, -2, +2, \ldots\}$.

$$\begin{array}{lllll} \text{So } \mathbb{Q} = \{ & \langle\ 0,1\rangle, & \langle\ 0,-1\rangle, & \langle\ 0,2\rangle, & \langle\ 0,-2\rangle, \cdots \\ & \langle\ 1,1\rangle, & \langle\ 1,-1\rangle, & \langle\ 1,2\rangle, & \langle\ 1,-2\rangle, \cdots \\ & \langle -1,1\rangle, & \langle -1,-1\rangle, & \langle -1,2\rangle, & \langle -1,-2\rangle, \cdots \\ & \langle\ 2,1\rangle, & \langle\ 2,-1\rangle, & \langle\ 2,2\rangle, & \langle\ 2,-2\rangle, \cdots \\ & \langle -2,1\rangle, & \langle -2,-1\rangle, & \langle -2,2\rangle, & \langle -2,-2\rangle, \cdots \\ & \vdots & & & \\ & & & & \} \end{array}$$

We know how to perform arithmetic on rational numbers and compare them with $=$, $<$, $>$.

Typically we represent rational numbers in reduced form, factoring out common factors, e.g.

$$2/4 = 1/2\ , \quad 50/100 = 1/2\ , \quad 6/8 = 3/4$$

How can we represent this clean mathematical type in OCaml?

5 Rational Numbers in OCaml

We can almost capture the mathematical type with `int * int` in place of $\mathbb{Z} \times \mathbb{Z} - \{0\}$ but we have anomalies such as $(0,0), (1,0), \cdots$
We can create a Boolean check to eliminate these

```
r:  int * int      rational(r) = snd(r) <> 0
```

e.g. (p, q) is rational iff $q \neq 0$

Arithmetic of rationals

addition $a/b + c/d = \dfrac{a*d + b*c}{b*d}$

e.g. $1/6 + 3/8 = \dfrac{26}{48} = \dfrac{13}{24}$

multiplication $a/b * c/d = \dfrac{a*c}{b*d}$

$1/6 * 3/8 = 3/48 = 1/16$

Both operations have **inverses**

subtraction $a/b - c/d = \dfrac{a*d - b*c}{b*d}$

division $a/b \div c/d = a/b * d/c = \dfrac{a*d}{b*c}$

$1/6 \div 1/2 = 2/6 = 1/3$

6 Algebraic Properties of $\mathbb{Z}$ and $\mathbb{Q}$

The integers $\mathbb{Z}$ have the properties of an **integral domain**, namely these 11 properties

$+$ is a binary function $\quad \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$

$+$ is associative
$a + (b + c) = (a + b) + c$

$+$ is commutative
$a + b = b + a$

$*$ is a binary function $\quad \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$

$*$ is associative
$a * (b * c) = (a * b) * c$

$*$ is commutative
$a * b = b * a$

$*$ distributes over $+$
$a * (b + c) = a * b + a * c$

Zero is an additive identity
$0 + a = a$

Unit is a non-zero multiplicative identity
$1 * a = a$

Additive inverse
$\forall a : \mathbb{Z}. \exists (-a) : \mathbb{Z}. \big(a + (-a) = 0 \big)$

Cancellation
If $c \neq 0$ and $c * a = c * b$, then $a = b$

6.1 Algebraic Properties of $\mathbb{Q}$

$\mathbb{Q}$ has the properties of a field.

A **field** is an integral domain such that for every $a \neq 0$ there is an inverse element a^{-1} such that $a^{-1} * a = 1$.

We denote a^{-1} as $1/a$.

In our construction of $\mathbb{Q}$ from $\mathbb{Z} \times \mathbb{Z} - \{0\}$ a^{-1} is $1/a$ and in general
$b * x = a$ is solved by $x = a/b$ the quotient of a by b.

6.2 Modules and Algebraic Structures

In PS4 we will see how OCaml modules organize types and operations in a manner similar to algebraic structures such as Integral Domains and Fields.

We will see in Lecture 11 that real numbers, $\mathbb{R}$, also form a field. In a similar way OCaml modules can have several different instances.

6.3 Further Properties of $\mathbb{Q}$

Mathematical types such as $\mathbb{Q}$ and $\mathbb{Z}$ define equality relations

$$x = y \text{ in } \mathbb{Z} \qquad\qquad x = y \text{ in } \mathbb{Q}$$

On $\mathbb{Z}$ two integers a and b are equal exactly when they have the same canonical forms, e.g.

$0 * 1 = 0$ since $0 * 1 \downarrow 0$

$2 * 3 = 6$ since $2 * 3 \downarrow 6$

What is equality on $\mathbb{Q}$?

$a/b = c/d \quad \textit{iff} \quad a * d = b * c$

$50/100 = 1/2 \quad \text{since} \quad 2 * 50 = 100$

The "fraction" $1/2$ is a canonical form of a rational number because it can't be further reduced.

To put a/b in canonical form, we want to remove common factors, e.g.

$$\frac{165}{462} = \frac{5}{14} \quad \text{since} \quad \frac{3 * 5 * 11}{2 * 3 * 7 * 11} = \frac{5}{2 * 7}$$

6.4 Equality on $\mathbb{Q}$

We reduce a/b to a canonical form by removing the common factors in a and b. We can do this by dividing a and b by their **greatest common divisor**, gcd.

For $\dfrac{3 * 5 * 11}{2 * 3 * 7 * 11}$ the gcd is $3 * 11$.

There is a simple algorithm to find the gcd of two integers. We can in fact find it from an inductive proof that the gcd exists.

Definition Integer d is the greatest common divisor of integers a and b iff
d divides a, e.g. $a = d * a_1$
d divides b, e.g. $b = d * b_1$
and for any common divisor c of a and b, e.g. c divides a and c divides b, we know that c divides d.
We say $\text{GCD}(a, b, d)$.

Theorem $\forall a, b : \mathbb{Z}. \exists\, d : \mathbb{Z}.\ \text{GCD}(a, b, d)$

7 GCD Theorem

$\forall n, m : \mathbb{N}. \exists\, g : \mathbb{N}.\ \mathrm{GCD}(m, n, g)$

Proof by induction on n.

See Anne Trostle's web page referenced in the web notes.

Qed

From the proof the proof assistant can **extract** an algorithm to compute the gcd. It is essentially this

```
let rec gcd (n : int) (m : int) : int =
  if n = 0 then m else gcd (m - (m / n) * n) n
```

where m/n is the OCaml integer divide, thus $m - (m/n) * n$ computes $m\ rem\ n$, the remainder of m divided by n, e.g. $5 - (5/2) * 2 = 1$.