Proofs About Type Classes

COS 441 Slides 07b

Agenda

- Last time
  - defining and using type classes
- This time:
  - proving properties of type classes

EQUALITY

Equality

- Haskell's equality type class:

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool
```

- Some basic axioms about equality:
  - Reflexivity: x == x
  - Transitivity: x == y and y == z implies x == z

Equality

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

- An instance:

```
data Bit = On | Off deriving (Show)

instance Eq Bit where
  (==) On  On   = True
  (==) Off Off  = True
  (==) On  Off  = False
  (==) Off On   = False
```

Equality

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
data Bit = On | Off deriving (Show)

instance Eq Bit where
  (==) On  On   = True
  (==) Off Off  = True
  (==) On  Off  = False
  (==) Off On   = False
```

- Reflexivity Proof (by cases on x):

```
case x = On:
  On == On          (unfold (==) at type Bit)

case x = Off:
  Off == Off        (unfold (==) at type Bit)
```

Equality

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
data Bit = On | Off deriving (Show)

instance Eq Bit where
  (==) On  On   = True
  (==) Off Off  = True
  (==) On  Off  = False
  (==) Off On   = False
```

- Transitivity Proof (by cases on x):

```
case x = On:
  (0) x = On                      (assumption for this case)
  (1) x == y                      (by assumption)
  (2) y == z                      (by assumption; now must prove x == z)
  (3) y = On                      (by (0,1) and (==) at type Bit)
  (4) z = On                      (by (2,1) and (==) at type Bit)
  (5) x == z                      (by (0,3) and (==) at type Bit)

case x is Off:  Similar to the case for x = Off.
```

Equality

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
data Pair a b = Pair a b deriving (Show)

instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

Equality

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
data Pair a b = Pair a b deriving (Show)

instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Reflexivity Proof (By Calculation):

  Must prove: p == p for any Pair a b such that Eq a and Eq b.
  What do such pairs look like?

Equality

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
data Pair a b = Pair a b deriving (Show)

instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Reflexivity Proof (By Calculation):

Must prove: p == p for any Pair a b such that Eq a and Eq b.
What do such pairs look like?
They must have the form p = Pair x y where x :: a and y :: b
Hence, we must prove:

```
Pair x y == Pair x y
```

Equality

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
data Pair a b = Pair a b deriving (Show)

instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Reflexivity Proof (By Calculation):

Must prove: p == p for any Pair a b such that Eq a and Eq b.
What do such pairs look like?
They must have the form p = Pair x y where x :: a and y :: b
Hence, we must prove:

```
   Pair x y == Pair x y
= (x == x) && (y == y)                (unfold == at type Pair a b)
```

Equality

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
data Pair a b = Pair a b deriving (Show)

instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Reflexivity Proof (By Calculation):

Must prove: p == p for any Pair a b such that Eq a and Eq b.
What do such pairs look like?
They must have the form p = Pair x y where x :: a and y :: b
Hence, we must prove:

```
   Pair x y == Pair x y
= (x == x) && (y == y)          (unfold == at type Pair a b)
= True && (y == y)              (by Eq reflexivity at type a)
```

use axioms at types for which Eq already proven

Equality

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
data Pair a b = Pair a b deriving (Show)

instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Reflexivity Proof (By Calculation):

Must prove: p == p for any Pair a b such that Eq a and Eq b.
What do such pairs look like?
They must have the form p = Pair x y where x :: a and y :: b
Hence, we must prove:

```
   Pair x y == Pair x y
= (x == x) && (y == y)          (unfold == at type Pair a b)
= True && (y == y)              (by Eq reflexivity at type a)
= True && True                  (by Eq reflexivity at type b)
```

use axioms at types for which Eq already proven

Equality

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
data Pair a b = Pair a b deriving (Show)

instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Reflexivity Proof (By Calculation):

use axioms at types for which Eq already proven

Must prove: p == p for any Pair a b such that Eq a and Eq b.
What do such pairs look like?
They must have the form p = Pair x y where x :: a and y :: b
Hence, we must prove:

```
   Pair x y == Pair x y
= (x == x) && (y == y)        (unfold == at type Pair a b)
= True && (y == y)            (by Eq reflexivity at type a)
= True && True                (by Eq reflexivity at type b)
= True                        (by unfold &&)
```

Equality

```
class Eq a where ...
  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Transitivity Proof (By Calculation):

  Must prove Pair x1 y1 == Pair x2 y2 and Pair x2 y2 == Pair x3 y3 implies Pair x1 y1 == Pair x3 y3 at type Pair a b.

Equality

```
class Eq a where ...
  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Transitivity Proof (By Calculation):

  Must prove Pair x1 y1 == Pair x2 y2 and Pair x2 y2 == Pair x3 y3 implies Pair x1 y1 == Pair x3 y3 at type Pair a b.

  (1) Pair x1 y1 == Pair x2 y2 (by assumption)
  (2) Pair x2 y2 == Pair x3 y3 (by assumption)
  (3) x1, x2, x3 :: a and Eq a (by assumption)
  (4) y1, y2, y3 :: b and Eq b (by assumption)

Equality

```
class Eq a where ...
  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Transitivity Proof (By Calculation):

  Must prove Pair x1 y1 == Pair x2 y2 and Pair x2 y2 == Pair x3 y3 implies Pair x1 y1 == Pair x3 y3 at type Pair a b.

(1)	Pair x1 y1 == Pair x2 y2	(by assumption)
(2)	Pair x2 y2 == Pair x3 y3	(by assumption)
(3)	x1, x2, x3 :: a and Eq a	(by assumption)
(4)	y1, y2, y3 :: b and Eq b	(by assumption)
(5)	(x1 == x2) && (y1 == y2)	(by (1), (==) at type Pair a b)
(6)	(x2 == x3) && (y2 == y3)	(by (2), (==) at type Pair a b)

Equality

```
class Eq a where ...
  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Transitivity Proof (By Calculation):

Must prove Pair x1 y1 == Pair x2 y2 and Pair x2 y2 == Pair x3 y3 implies Pair x1 y1 == Pair x3 y3 at type Pair a b.

(1)	Pair x1 y1 == Pair x2 y2	(by assumption)
(2)	Pair x2 y2 == Pair x3 y3	(by assumption)
(3)	x1, x2, x3 :: a and Eq a	(by assumption)
(4)	y1, y2, y3 :: b and Eq b	(by assumption)
(5)	(x1 == x2) && (y1 == y2)	(by (1), (==) at type Pair a b)
(6)	(x2 == x3) && (y2 == y3)	(by (2), (==) at type Pair a b)

Pair x1 y1 == Pair x3 y3

Equality

```
class Eq a where ...
  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Transitivity Proof (By Calculation):

Must prove Pair x1 y1 == Pair x2 y2 and Pair x2 y2 == Pair x3 y3 implies Pair x1 y1 == Pair x3 y3 at type Pair a b.

(1)	Pair x1 y1 == Pair x2 y2	(by assumption)
(2)	Pair x2 y2 == Pair x3 y3	(by assumption)
(3)	x1, x2, x3 :: a and Eq a	(by assumption)
(4)	y1, y2, y3 :: b and Eq b	(by assumption)
(5)	(x1 == x2) && (y1 == y2)	(by (1), (==) at type Pair a b)
(6)	(x2 == x3) && (y2 == y3)	(by (2), (==) at type Pair a b)

```
  Pair x1 y1 == Pair x3  y3
= (x1 == x3) && (y1 == y3)        (unfold == at type Pair a b)
```

Equality

```
class Eq a where ...
  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Transitivity Proof (By Calculation):

Must prove Pair x1 y1 == Pair x2 y2 and Pair x2 y2 == Pair x3 y3 implies Pair x1 y1 == Pair x3 y3 at type Pair a b.

```
(1)  Pair x1 y1 == Pair x2 y2         (by assumption)
(2)  Pair x2 y2 == Pair x3 y3         (by assumption)
(3)  x1, x2, x3 :: a  and Eq a        (by assumption)
(4)  y1, y2, y3 :: b and Eq b         (by assumption)
(5)  (x1 == x2) && (y1 == y2)         (by (1), (==) at type Pair a b)
(6)  (x2 == x3) && (y2 == y3)         (by (2), (==) at type Pair a b)

    Pair x1 y1 == Pair x3  y3
  = (x1 == x3) && (y1 == y3)          (unfold == at type Pair a b)
  = True && (y1 == y3)                (by (5), (6), transitivity at type a)
```

Equality

```
class Eq a where ...
  -- axiom:  x == x
  -- axiom:  x == y and y == z implies x == z
```

```
instance (Eq a, Eq b) => Eq (Pair a b)
where
  (==) (Pair x1 y1) (Pair x2 y2) =
    (x1 == x2) && (y1 == y2)
```

- Transitivity Proof (By Calculation):

Must prove Pair x1 y1 == Pair x2 y2 and Pair x2 y2 == Pair x3 y3 implies Pair x1 y1 == Pair x3 y3 at type Pair a b.

(1) Pair x1 y1 == Pair x2 y2 (by assumption)
(2) Pair x2 y2 == Pair x3 y3 (by assumption)
(3) x1, x2, x3 :: a and Eq a (by assumption)
(4) y1, y2, y3 :: b and Eq b (by assumption)
(5) (x1 == x2) && (y1 == y2) (by (1), (==) at type Pair a b)
(6) (x2 == x3) && (y2 == y3) (by (2), (==) at type Pair a b)

```
  Pair x1 y1 == Pair x3  y3
= (x1 == x3) && (y1 == y3)      (unfold == at type Pair a b)
= True && (y1 == y3)            (by (5), (6), transitivity at type a)
= True && True = True           (by (5), (6), transitivity at type b; by &&)
```

Lessons

- When proving things about type classes, be specific about the type at which you use a definition
  - eg: unfold == *at type Pair a b*
  - eg: unfold == *at type a*

Lessons

- What specific types have we proven have reflexive and transitive equality?
  - Bit
  - Pair Bit Bit
  - Pair (Pair Bit Bit) Bit
  - Pair (Pair (Pair Bit (Pair Bit Bit)) (Pair Bit Bit)) Bit
  - Pair … …
- Why?
  - We proved == at type Bit satisfies the axioms
  - We proved that if == at type a and type b satisfies the axioms then == at type Pair a b satisfies the axioms
  - This is a kind of induction!
  - It is *induction on the structure of types*.

Lessons

- Type class proofs are often achieved by *induction on the structure of the type*
  - Given: instance (T a) => T (Constructor a) where ...
  - Assume: the axioms for T hold for type a
  - Must prove: the axioms hold for type Constructor a
  - the axioms at the smaller type a are used as *inductive hypotheses* within the proofs of the axioms for Constructor a
  - If all your type classes have the form
    - instance (T a) => T (Constructor a) where ...
    - then your type class is uninhabited! You need some base cases.
  - Base cases arise when types unconditionally belong to the type class

Lessons

- When proving something with the form:
  - If A and B then C
- You may structure your proof by assuming A and B, then proving C:

Theorem: If A and B then C.
Proof: By calculation, or induction, or whatever else works.

(1) A (By assumption)
(2) B (By assumption)
(3) ...
(4) ...
(5) ...
(6) C (By 2, 3, 5)
QED.