Quantified Class Constraints in Haskell

KU Leuven

Gert-Jan Bottu

Tom Schrijvers

George Karachalias

University of Hong Kong

Bruno C. d. S. Oliveira

University of Edinburgh

Philip Wadler

Type Classes[1]

[1] P. Wadler and S. Blott 1989. How to Make Ad-hoc Polymorphism Less Ad Hoc

Type Classes[1]

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

[1] P. Wadler and S. Blott 1989. How to Make Ad-hoc Polymorphism Less Ad Hoc

Type Classes[1]

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

instance Eq Char where
    x == y = eqChar x y
```

[1] P. Wadler and S. Blott 1989. How to Make Ad-hoc Polymorphism Less Ad Hoc

Type Classes[1]

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

instance Eq Char where
    x == y = eqChar x y

instance Eq Bool where
    True  == True  = True
    False == False = True
    _     == _     = False
```

[1] P. Wadler and S. Blott 1989. How to Make Ad-hoc Polymorphism Less Ad Hoc

Type Classes[1]

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

[1] P. Wadler and S. Blott 1989. How to Make Ad-hoc Polymorphism Less Ad Hoc

Type Classes[1]

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


instance Eq a => Eq [a] where
    []      == []      = True
    (h1:t1) == (h2:t2) = (h1 == h2) && (t1 == t2)
    _       == _       = False
```

[1] P. Wadler and S. Blott 1989. How to Make Ad-hoc Polymorphism Less Ad Hoc

Type Classes[1]

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

```
instance Eq a => Eq [a] where
    []      == []      = True
    (h1:t1) == (h2:t2) = (h1 == h2) && (t1 == t2)
    _       == _       = False
```

[1] P. Wadler and S. Blott 1989. How to Make Ad-hoc Polymorphism Less Ad Hoc

Type Classes[1]

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

```
instance Eq a => Eq [a] where
    []        == []        = True
    (h1:t1)   == (h2:t2)   = (h1 == h2) && (t1 == t2)
    _         == _         = False
```

[1] P. Wadler and S. Blott 1989. How to Make Ad-hoc Polymorphism Less Ad Hoc

Type Classes

```
instance Eq a => Eq [a] where
    []        == []        = True
    (h1:t1)   == (h2:t2)   = (h1 == h2) && (t1 == t2)
    _         == _         = False
```

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

Type Classes

```
instance Eq a => Eq [a] where
    []        == []        = True
    (h1:t1)   == (h2:t2)   = (h1 == h2) && (t1 == t2)
    _         == _         = False
```

```
forall a. Eq a => Eq [a]
```

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

Type Classes

```
instance Eq a => Eq [a] where
    []      == []      = True
    (h1:t1) == (h2:t2) = (h1 == h2) && (t1 == t2)
    _       == _       = False
```

```
forall a. Eq a => Eq [a]
```

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

```
forall a. Ord a => Eq a
```

Entailment

Entailment

$$P \models Eq\ (Nat, Bool)$$

Entailment

$$\frac{\forall a, b. (Eq\ a, Eq\ b) \Rightarrow Eq\ (a, b) \in P}{P \models Eq\ (Nat, Bool)}$$

Entailment

$$\frac{\forall a, b.(Eq\ a, Eq\ b) \Rightarrow Eq\ (a, b) \in P \qquad P \models Eq\ Nat \qquad P \models Eq\ Bool}{P \models Eq\ (Nat, Bool)}$$

Entailment

$$\frac{\forall a, b.(Eq\ a, Eq\ b) \Rightarrow Eq\ (a, b) \in P \qquad \dfrac{Eq\ Nat \in P}{P \models Eq\ Nat} \qquad P \models Eq\ Bool}{P \models Eq\ (Nat, Bool)}$$

Entailment

$$\dfrac{\forall a, b.(Eq\ a, Eq\ b) \Rightarrow Eq\ (a, b) \in P \qquad \dfrac{Eq\ Nat \in P}{P \models Eq\ Nat} \qquad \dfrac{Eq\ Bool \in P}{P \models Eq\ Bool}}{P \models Eq\ (Nat, Bool)}$$

Quantified Class Constraints

Quantified Class Constraints

$$C \;::=\; TC\ \tau$$

Quantified Class Constraints

$$C \ ::= \ TC\ \tau$$

$$\Downarrow$$

$$C \ ::= \ TC\ \tau \mid C_1 \Rightarrow C_2 \mid \forall a.C$$

Motivation

- Precise specifications
- Terminating (co)recursive resolution

Motivation: Precise Specifications

Motivation: Precise Specifications

```
class Trans t where
    lift :: Monad m => m a -> (t m) a
```

Motivation: Precise Specifications

```
class Trans t where
    lift :: Monad m => m a -> (t m) a
```

```
newtype (t1 * t2) m a = C { runC :: t1 (t2 m) a } 2
```

[2] Tom Schrijvers and Bruno C.d.S. Oliveira. 2011. Monads, Zippers and Views: Virtualizing the Monad Stack

Motivation: Precise Specifications

```
class Trans t where
    lift :: Monad m => m a -> (t m) a


newtype (t1 * t2) m a = C { runC :: t1 (t2 m) a } [2]


instance (Trans t1, Trans t2) => Trans (t1 * t2) where
    lift x = C (lift (lift x))
```

[2] Tom Schrijvers and Bruno C.d.S. Oliveira. 2011. Monads, Zippers and Views: Virtualizing the Monad Stack

Motivation: Precise Specifications

```
class Trans t where
    lift :: Monad m => m a -> (t m) a


newtype (t1 * t2) m a = C { runC :: t1 (t2 m) a } [2]


instance (Trans t1, Trans t2) => Trans (t1 * t2) where
    lift x = C (lift (lift x))

                           m a
```

[2] Tom Schrijvers and Bruno C.d.S. Oliveira. 2011. Monads, Zippers and Views: Virtualizing the Monad Stack

Motivation: Precise Specifications

```
class Trans t where
    lift :: Monad m => m a -> (t m) a
```

```
newtype (t1 * t2) m a = C { runC :: t1 (t2 m) a } [2]
```

```
instance (Trans t1, Trans t2) => Trans (t1 * t2) where
    lift x = C (lift (lift x))
```

$$\texttt{lift x = C (lift }\underbrace{\texttt{(lift x)}}_{\texttt{(t2 m) a}}\texttt{)}$$

[2] Tom Schrijvers and Bruno C.d.S. Oliveira. 2011. Monads, Zippers and Views: Virtualizing the Monad Stack

Motivation: Precise Specifications

```
class Trans t where
    lift :: Monad m => m a -> (t m) a


newtype (t1 * t2) m a = C { runC :: t1 (t2 m) a } [2]


instance (Trans t1, Trans t2) => Trans (t1 * t2) where
    lift x = C (lift (lift x))
                  t1 (t2 m) a
```

[2] Tom Schrijvers and Bruno C.d.S. Oliveira. 2011. Monads, Zippers and Views: Virtualizing the Monad Stack

Motivation: Precise Specifications

```
class Trans t where
    lift :: Monad m => m a -> (t m) a


newtype (t1 * t2) m a = C { runC :: t1 (t2 m) a } [2]


instance (Trans t1, Trans t2) => Trans (t1 * t2) where
    lift x = C (lift (lift x))

                    Monad (t2 m)
```

[2] Tom Schrijvers and Bruno C.d.S. Oliveira. 2011. Monads, Zippers and Views: Virtualizing the Monad Stack

Motivation: Precise Specifications

```
class (forall m. Monad m => Monad (t m)) => Trans t where
    lift :: Monad m => m a -> (t m) a


newtype (t1 * t2) m a = C { runC :: t1 (t2 m) a }[2]


instance (Trans t1, Trans t2) => Trans (t1 * t2) where
    lift x = C (lift (lift x))
```

[2] Tom Schrijvers and Bruno C.d.S. Oliveira. 2011. Monads, Zippers and Views: Virtualizing the Monad Stack

Motivation:
Terminating (Co)recursive Resolution

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation: Terminating (Co)recursive Resolution

```
data Rose a = Branch a [Rose a]
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation: Terminating (Co)recursive Resolution

```
data Rose a = Branch a [Rose a]

data GRose f a = GBranch a (f (GRose f a))
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation:
Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation:
Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, Show (f (GRose f a))) => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation: Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, Show (f (GRose f a))) => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs
```

```
Show (GRose [] Bool)
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation:
Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, Show (f (GRose f a))) => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs
```

```
Show (GRose [] Bool)
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation: Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, Show (f (GRose f a))) => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs
```

```
Show (GRose [] Bool)
-> Show Bool, Show [GRose [] Bool]
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation: Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, Show (f (GRose f a))) => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs

instance Show a => Show [a] where ...


Show (GRose [] Bool)
-> Show Bool, Show [GRose [] Bool]
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation: Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, Show (f (GRose f a))) => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs

instance Show a => Show [a] where ...


Show (GRose [] Bool)
-> Show Bool, Show [GRose [] Bool]
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation:
Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, Show (f (GRose f a))) => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs

instance Show a => Show [a] where ...


Show (GRose [] Bool)
-> Show Bool, Show [GRose [] Bool]
-> Show Bool, Show (GRose [] Bool)
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation: Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, Show (f (GRose f a))) => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs

instance Show a => Show [a] where ...


Show (GRose [] Bool)
-> Show Bool, Show [GRose [] Bool]
-> Show Bool, Show (GRose [] Bool)
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation: Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, Show (f (GRose f a))) => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs

instance Show a => Show [a] where ...


Show (GRose [] Bool)
-> Show Bool, Show [GRose [] Bool]
-> Show Bool, Show (GRose [] Bool)
-> ...
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation:
Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, forall x. Show x => Show (f x))
        => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation: Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, forall x. Show x => Show (f x))
        => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs


Show (GRose [] Bool)
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation: Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, forall x. Show x => Show (f x))
         => Show (GRose f a) where
     show (GBranch x xs) = show x ++ "-" ++ show xs


Show (GRose [] Bool)
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation: Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, forall x. Show x => Show (f x))
        => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs


Show (GRose [] Bool)
-> Show Bool, forall x. Show x => Show [x]
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Motivation: Terminating (Co)recursive Resolution

```
data GRose f a = GBranch a (f (GRose f a))

instance (Show a, forall x. Show x => Show (f x))
        => Show (GRose f a) where
    show (GBranch x xs) = show x ++ "-" ++ show xs



Show (GRose [] Bool)
-> Show Bool, forall x. Show x => Show [x]
```

[3] Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Intermezzo: Cycle-aware constraint resolution [4]

[4] Ralf Lämmel and Simon Peyton Jones. 2005. Scrap Your Boilerplate with Class: Extensible Generic Functions

Intermezzo: Cycle-aware constraint resolution [4]

- Only cyclic resolutions

[4] Ralf Lämmel and Simon Peyton Jones. 2005. Scrap Your Boilerplate with Class: Extensible Generic Functions

Motivation:
Terminating (Co)recursive Resolution

Motivation: Terminating (Co)recursive Resolution

```
data Perfect a = Zero a | Succ (Perfect (a , a))
```
[5]

[5] Ralf Hinze. 2000. Perfect trees and bit-reversal permutations

Motivation:
Terminating (Co)recursive Resolution

```
data Perfect a = Zero a | Succ (Perfect (a , a))
```
[5]

```
data Mu h a = In { out :: h (Mu h) a }
```
[6]

[5] Ralf Hinze. 2000. Perfect trees and bit-reversal permutations
[6] Ralf Hinze. 2010. Adjoint Folds and Unfolds: Or: Scything Through the Thicket of Morphisms

Motivation:
Terminating (Co)recursive Resolution

```
data Perfect a = Zero a | Succ (Perfect (a , a)) [5]

data Mu h a = In { out :: h (Mu h) a } [6]

data HPerf f a = HZero a | HSucc (f (a , a))
```

[5] Ralf Hinze. 2000. Perfect trees and bit-reversal permutations

[6] Ralf Hinze. 2010. Adjoint Folds and Unfolds: Or: Scything Through the Thicket of Morphisms

Motivation: Terminating (Co)recursive Resolution

```
data Perfect a = Zero a | Succ (Perfect (a , a)) [5]

data Mu h a = In { out :: h (Mu h) a } [6]

data HPerf f a = HZero a | HSucc (f (a , a))

type Perfect = Mu HPerf
```

[5] Ralf Hinze. 2000. Perfect trees and bit-reversal permutations

[6] Ralf Hinze. 2010. Adjoint Folds and Unfolds: Or: Scything Through the Thicket of Morphisms

Motivation: Terminating (Co)recursive Resolution

```
data Mu h a = In { out :: h (Mu h) a }

data HPerf f a = HZero a | HSucc (f (a , a))

type Perfect = Mu HPerf
```

Motivation:
Terminating (Co)recursive Resolution

```
data Mu h a = In { out :: h (Mu h) a }

data HPerf f a = HZero a | HSucc (f (a , a))

type Perfect = Mu HPerf

instance (Show (h (Mu h) a)) => Show (Mu h a) where
    show (In x) = show x

instance (Show a, Show (f (a , a))) => Show (HPerf f a) where
    show (HZero a ) = "(Z" ++ show a  ++ ")"
    show (HSucc xs) = "(S" ++ show xs ++ ")"
```

Motivation: Terminating (Co)recursive Resolution

```
instance (Show (h (Mu h) a)) => Show (Mu h a) where ...
instance (Show a, Show (f (a , a))) => Show (HPerf f a) where ...
```

Motivation: Terminating (Co)recursive Resolution

```
instance (Show (h (Mu h) a)) => Show (Mu h a) where ...
instance (Show a, Show (f (a , a))) => Show (HPerf f a) where ...

Show (Perfect Int)
```

Motivation: Terminating (Co)recursive Resolution

```
instance (Show (h (Mu h) a)) => Show (Mu h a) where ...
instance (Show a, Show (f (a , a))) => Show (HPerf f a) where ...
```

```
Show (Perfect Int)
-> Show (Mu HPerf Int)
```

Motivation: Terminating (Co)recursive Resolution

```
instance (Show (h (Mu h) a)) => Show (Mu h a) where ...
instance (Show a, Show (f (a , a))) => Show (HPerf f a) where ...

Show (Perfect Int)
-> Show (Mu HPerf Int)
```

Motivation:
Terminating (Co)recursive Resolution

```
instance (Show (h (Mu h) a)) => Show (Mu h a) where ...
instance (Show a, Show (f (a , a))) => Show (HPerf f a) where ...

Show (Perfect Int)
-> Show (Mu HPerf Int)
-> Show (HPerf (Mu HPerf) Int)
```

Motivation:
Terminating (Co)recursive Resolution

```
instance (Show (h (Mu h) a)) => Show (Mu h a) where ...
instance (Show a, Show (f (a , a))) => Show (HPerf f a) where ...

Show (Perfect Int)
-> Show (Mu HPerf Int)
-> Show (HPerf (Mu HPerf) Int)
```

Motivation:
Terminating (Co)recursive Resolution

```
instance (Show (h (Mu h) a)) => Show (Mu h a) where ...
instance (Show a, Show (f (a , a))) => Show (HPerf f a) where ...

Show (Perfect Int)
-> Show (Mu HPerf Int)
-> Show (HPerf (Mu HPerf) Int)
-> Show Int, Show (Mu HPerf (Int , Int))
```

Motivation: Terminating (Co)recursive Resolution

```
instance (Show (h (Mu h) a)) => Show (Mu h a) where ...
instance (Show a, Show (f (a , a))) => Show (HPerf f a) where ...

Show (Perfect Int)
-> Show (Mu HPerf Int)
-> Show (HPerf (Mu HPerf) Int)
-> Show Int, Show (Mu HPerf (Int , Int))
```

Motivation:
Terminating (Co)recursive Resolution

```
instance (Show (h (Mu h) a)) => Show (Mu h a) where ...
instance (Show a, Show (f (a , a))) => Show (HPerf f a) where ...

Show (Perfect Int)
-> Show (Mu HPerf Int)
-> Show (HPerf (Mu HPerf) Int)
-> Show Int, Show (Mu HPerf (Int , Int))
-> Show Int, Show (HPerf (Mu HPerf) (Int , Int))
```

Motivation:
Terminating (Co)recursive Resolution

```
instance (Show (h (Mu h) a)) => Show (Mu h a) where ...
instance (Show a, Show (f (a , a))) => Show (HPerf f a) where ...
```

```
Show (Perfect Int)
-> Show (Mu HPerf Int)
-> Show (HPerf (Mu HPerf) Int)
-> Show Int, Show (Mu HPerf (Int , Int))
-> Show Int, Show (HPerf (Mu HPerf) (Int , Int))
-> Show Int, Show (Int , Int),
   Show (Mu HPerf ((Int , Int) , (Int , Int)))
```

Motivation: Terminating (Co)recursive Resolution

```
instance (Show (h (Mu h) a)) => Show (Mu h a) where ...
instance (Show a, Show (f (a , a))) => Show (HPerf f a) where ...

Show (Perfect Int)
-> Show (Mu HPerf Int)
-> Show (HPerf (Mu HPerf) Int)
-> Show Int, Show (Mu HPerf (Int , Int))
-> Show Int, Show (HPerf (Mu HPerf) (Int , Int))
-> Show Int, Show (Int , Int),
   Show (Mu HPerf ((Int , Int) , (Int , Int)))
-> ...
```

Motivation: Terminating (Co)recursive Resolution

```
instance (Show a,
forall f x. (Show x, forall y. Show y => Show (f y)) => Show (h f x))
        => Show (Mu h a) where
    show (In x) = show x

instance (Show a, forall x. Show x => Show (f x))
        => Show (HPerf f a) where
    show (HZero a)  = "(Z " ++ show a ++ ")"
    show (HSucc xs) = "(S " ++ show xs ++ ")"
```

Motivation: Terminating (Co)recursive Resolution

```
instance (Show a,
forall f x. (Show x, forall y. Show y => Show (f y)) => Show (h f x))
        => Show (Mu h a) where ...

instance (Show a, forall x. Show x => Show (f x))
        => Show (HPerf f a) where ...
```

Motivation: Terminating (Co)recursive Resolution

```
instance (Show a,
forall f x. (Show x, forall y. Show y => Show (f y)) => Show (h f x))
        => Show (Mu h a) where ...

instance (Show a, forall x. Show x => Show (f x))
        => Show (HPerf f a) where ...


Show (Perfect Int)
```

Motivation:
Terminating (Co)recursive Resolution

```
instance (Show a,
forall f x. (Show x, forall y. Show y => Show (f y)) => Show (h f x))
        => Show (Mu h a) where ...

instance (Show a, forall x. Show x => Show (f x))
        => Show (HPerf f a) where ...


Show (Perfect Int)
-> Show (Mu HPerf Int)
```

Motivation: Terminating (Co)recursive Resolution

```
instance (Show a,
forall f x. (Show x, forall y. Show y => Show (f y)) => Show (h f x))
        => Show (Mu h a) where ...

instance (Show a, forall x. Show x => Show (f x))
        => Show (HPerf f a) where ...


Show (Perfect Int)
-> Show (Mu HPerf Int)
```

Motivation: Terminating (Co)recursive Resolution

```
instance (Show a,
forall f x. (Show x, forall y. Show y => Show (f y)) => Show (h f x))
        => Show (Mu h a) where ...

instance (Show a, forall x. Show x => Show (f x))
        => Show (HPerf f a) where ...


Show (Perfect Int)
-> Show (Mu HPerf Int)
-> Show Int, forall f x. (Show x, forall y. Show y => Show (f y))
     => Show (HPerf f x)
```

Motivation:
Terminating (Co)recursive Resolution

```
instance (Show a,
forall f x. (Show x, forall y. Show y => Show (f y)) => Show (h f x))
        => Show (Mu h a) where ...

instance (Show a, forall x. Show x => Show (f x))
        => Show (HPerf f a) where ...
```

```
Show (Perfect Int)
-> Show (Mu HPerf Int)
-> Show Int, forall f x. (Show x, forall y. Show y => Show (f y))
    => Show (HPerf f x)
```

Motivation: Faster Coroutine Pipelines [8]

[8] Michael Spivey. 2017. Faster Coroutine Pipelines

Motivation: Faster Coroutine Pipelines [8]

```
class forall i o. Monad (pipe i o) => PipeKit pipe where
  input  :: pipe i o i
  output :: o -> pipe i o ()
  (||)   :: pipe i n () -> pipe n o () -> pipe i o a
  effect :: IO a -> pipe i o a
  exit   :: pipe i o a
```

[8] Michael Spivey. 2017. Faster Coroutine Pipelines

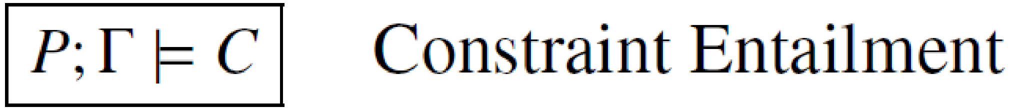
Motivation: Faster Coroutine Pipelines[8]

```
class forall i o. Monad (pipe i o) => PipeKit pipe where
  input  :: pipe i o i
  output :: o -> pipe i o ()
  (||)   :: pipe i n () -> pipe n o () -> pipe i o a
  effect :: IO a -> pipe i o a
  exit   :: pipe i o a
```

[8] Michael Spivey. 2017. Faster Coroutine Pipelines

Typing

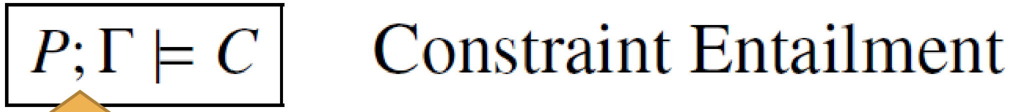
Constraint Entailment [9]



$\boxed{P; \Gamma \models C}$ Constraint Entailment

[9] Ralf Hinze and Simon Peyton Jones. 2001. Derivable Type Classes

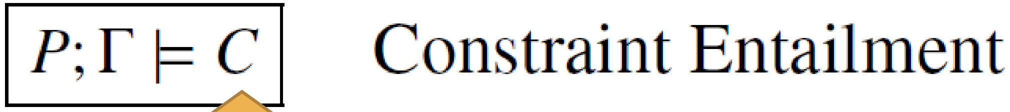
Constraint Entailment [9]



$\boxed{P; \Gamma \models C}$ Constraint Entailment

[9] Ralf Hinze and Simon Peyton Jones. 2001. Derivable Type Classes

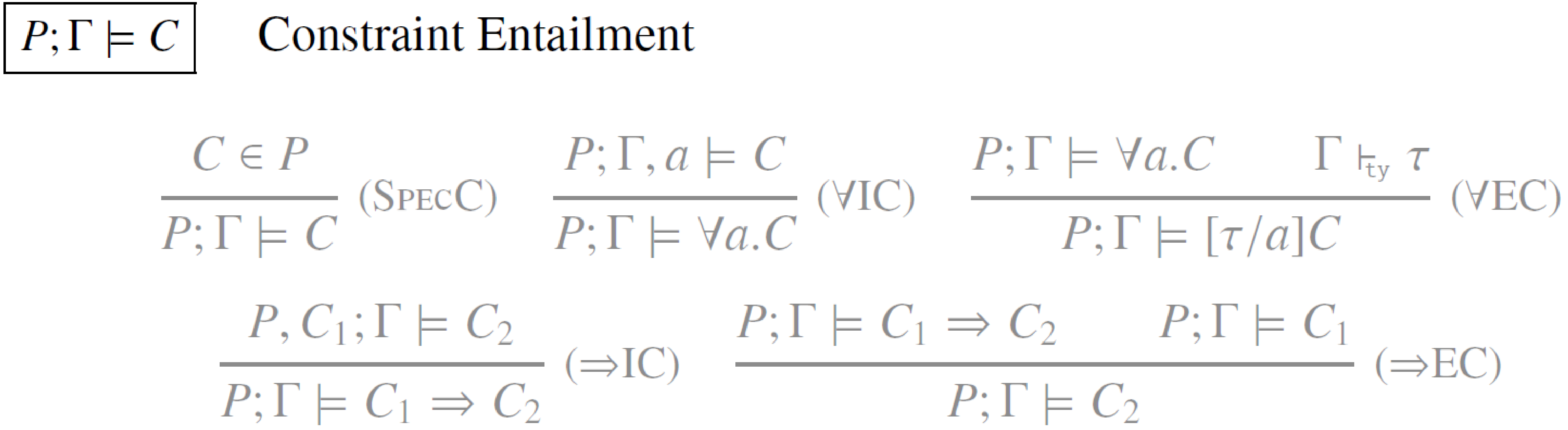
Constraint Entailment [9]



$\boxed{P;\Gamma \models C}$ Constraint Entailment

[9] Ralf Hinze and Simon Peyton Jones. 2001. Derivable Type Classes

Constraint Entailment [9]



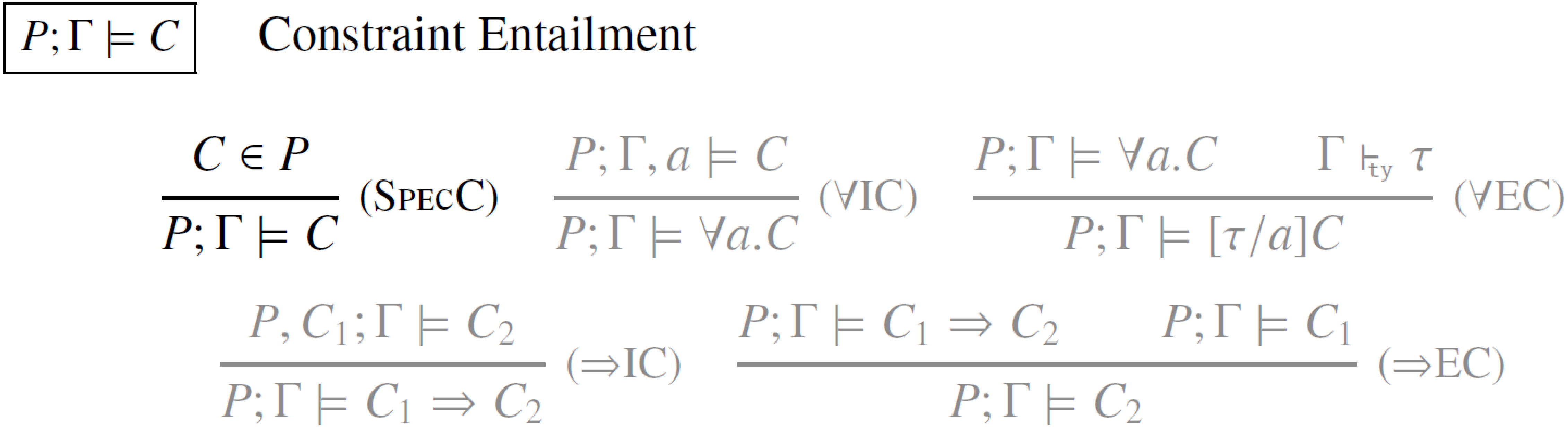
$\boxed{P; \Gamma \models C}$ Constraint Entailment

$$\frac{C \in P}{P; \Gamma \models C}\ (\textsc{SpecC}) \qquad \frac{P; \Gamma, a \models C}{P; \Gamma \models \forall a.C}\ (\forall\mathrm{IC}) \qquad \frac{P; \Gamma \models \forall a.C \qquad \Gamma \vdash_{\mathtt{ty}} \tau}{P; \Gamma \models [\tau / a]C}\ (\forall\mathrm{EC})$$

$$\frac{P, C_1; \Gamma \models C_2}{P; \Gamma \models C_1 \Rightarrow C_2}\ (\Rightarrow\mathrm{IC}) \qquad \frac{P; \Gamma \models C_1 \Rightarrow C_2 \qquad P; \Gamma \models C_1}{P; \Gamma \models C_2}\ (\Rightarrow\mathrm{EC})$$

[9] Ralf Hinze and Simon Peyton Jones. 2001. Derivable Type Classes

Constraint Entailment [9]



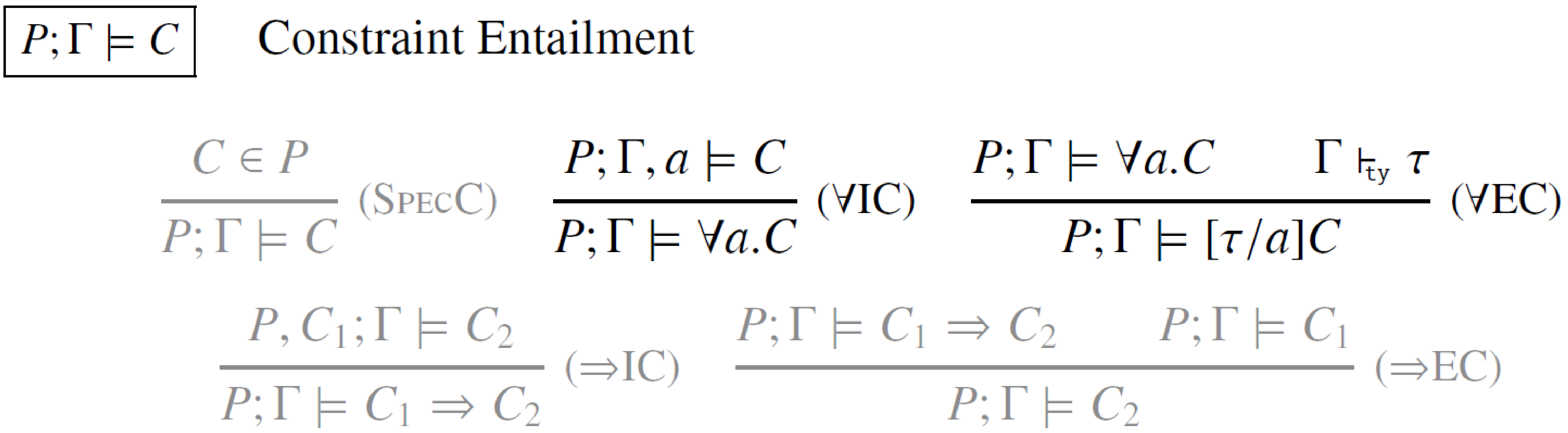
$\boxed{P; \Gamma \models C}$ Constraint Entailment

$$\frac{C \in P}{P; \Gamma \models C}\ (\textsc{SpecC}) \qquad \frac{P; \Gamma, a \models C}{P; \Gamma \models \forall a.C}\ (\forall\text{IC}) \qquad \frac{P; \Gamma \models \forall a.C \qquad \Gamma \vdash_{\mathtt{ty}} \tau}{P; \Gamma \models [\tau/a]C}\ (\forall\text{EC})$$

$$\frac{P, C_1; \Gamma \models C_2}{P; \Gamma \models C_1 \Rightarrow C_2}\ (\Rightarrow\text{IC}) \qquad \frac{P; \Gamma \models C_1 \Rightarrow C_2 \qquad P; \Gamma \models C_1}{P; \Gamma \models C_2}\ (\Rightarrow\text{EC})$$

[9] Ralf Hinze and Simon Peyton Jones. 2001. Derivable Type Classes

Constraint Entailment [9]



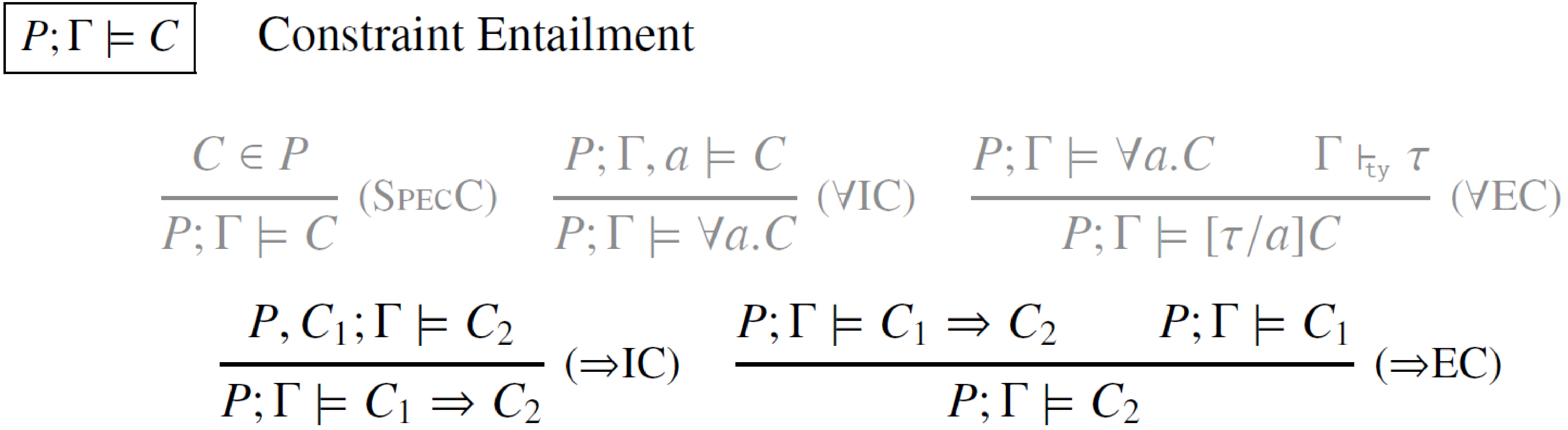
$\boxed{P; \Gamma \models C}$ Constraint Entailment

$$\frac{C \in P}{P; \Gamma \models C}\ (\textsc{SpecC}) \qquad \frac{P; \Gamma, a \models C}{P; \Gamma \models \forall a.C}\ (\forall \mathrm{IC}) \qquad \frac{P; \Gamma \models \forall a.C \qquad \Gamma \vdash_{\mathtt{ty}} \tau}{P; \Gamma \models [\tau / a]C}\ (\forall \mathrm{EC})$$

$$\frac{P, C_1; \Gamma \models C_2}{P; \Gamma \models C_1 \Rightarrow C_2}\ (\Rightarrow \mathrm{IC}) \qquad \frac{P; \Gamma \models C_1 \Rightarrow C_2 \qquad P; \Gamma \models C_1}{P; \Gamma \models C_2}\ (\Rightarrow \mathrm{EC})$$

[9] Ralf Hinze and Simon Peyton Jones. 2001. Derivable Type Classes

Constraint Entailment [9]



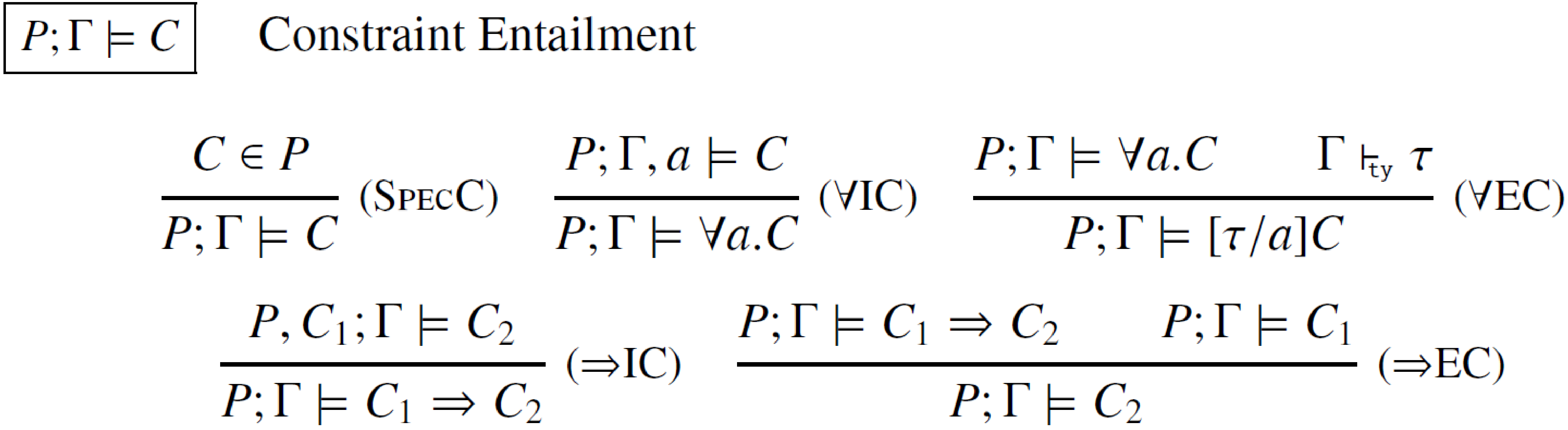
$\boxed{P;\Gamma \models C}$ Constraint Entailment

$$\frac{C \in P}{P;\Gamma \models C}\ (\textsc{SpecC}) \qquad \frac{P;\Gamma, a \models C}{P;\Gamma \models \forall a.C}\ (\forall\mathrm{IC}) \qquad \frac{P;\Gamma \models \forall a.C \qquad \Gamma \vdash_{\mathtt{ty}} \tau}{P;\Gamma \models [\tau/a]C}\ (\forall\mathrm{EC})$$

$$\frac{P, C_1;\Gamma \models C_2}{P;\Gamma \models C_1 \Rightarrow C_2}\ (\Rightarrow\mathrm{IC}) \qquad \frac{P;\Gamma \models C_1 \Rightarrow C_2 \qquad P;\Gamma \models C_1}{P;\Gamma \models C_2}\ (\Rightarrow\mathrm{EC})$$

[9] Ralf Hinze and Simon Peyton Jones. 2001. Derivable Type Classes

Constraint Entailment [9]



$$\boxed{P;\Gamma \models C} \qquad \text{Constraint Entailment}$$

$$\frac{C \in P}{P;\Gamma \models C}\ (\textsc{SpecC}) \qquad \frac{P;\Gamma, a \models C}{P;\Gamma \models \forall a.C}\ (\forall\text{IC}) \qquad \frac{P;\Gamma \models \forall a.C \qquad \Gamma \vdash_{\mathtt{ty}} \tau}{P;\Gamma \models [\tau/a]C}\ (\forall\text{EC})$$

$$\frac{P, C_1;\Gamma \models C_2}{P;\Gamma \models C_1 \Rightarrow C_2}\ (\Rightarrow\text{IC}) \qquad \frac{P;\Gamma \models C_1 \Rightarrow C_2 \qquad P;\Gamma \models C_1}{P;\Gamma \models C_2}\ (\Rightarrow\text{EC})$$

[9] Ralf Hinze and Simon Peyton Jones. 2001. Derivable Type Classes

Typing: Ambiguous

Typing: Ambiguous

$$\frac{\mathit{Show}\ a \in [\mathit{Eq}\ a, \mathit{Show}\ a]}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Show}\ a}\ (\textsc{SpecC})$$

Typing: Ambiguous

$$\frac{\mathit{Show}\ a \in [\mathit{Eq}\ a, \mathit{Show}\ a]}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Show}\ a}\ (\textsc{SpecC})$$

$$\frac{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Eq}\ a \Rightarrow \mathit{Show}\ a \qquad\qquad [\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Eq}\ a}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Show}\ a}\ (\Rightarrow\mathrm{EC})$$

Typing: Ambiguous

$$\frac{\mathit{Show}\ a \in [\mathit{Eq}\ a, \mathit{Show}\ a]}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Show}\ a}\ (\textsc{SpecC})$$

$$\frac{\dfrac{[\mathit{Eq}\ a, \mathit{Show}\ a, \mathit{Eq}\ a]; \Gamma \models \mathit{Show}\ a}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Eq}\ a \Rightarrow \mathit{Show}\ a}\ (\Rightarrow\mathrm{IC}) \qquad [\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Eq}\ a}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Show}\ a}\ (\Rightarrow\mathrm{EC})$$

Typing: Ambiguous

$$\frac{\mathit{Show}\ a \in [\mathit{Eq}\ a, \mathit{Show}\ a]}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Show}\ a}\ (\textsc{SpecC})$$

$$\dfrac{\dfrac{\dfrac{\mathit{Show}\ a \in [\mathit{Eq}\ a, \mathit{Show}\ a, \mathit{Eq}\ a]}{[\mathit{Eq}\ a, \mathit{Show}\ a, \mathit{Eq}\ a]; \Gamma \models \mathit{Show}\ a}\ (\textsc{SpecC})}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Eq}\ a \Rightarrow \mathit{Show}\ a}\ (\Rightarrow\mathrm{IC}) \qquad [\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Eq}\ a}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Show}\ a}\ (\Rightarrow\mathrm{EC})$$

Typing: Ambiguous

$$\dfrac{\mathit{Show}\ a \in [\mathit{Eq}\ a, \mathit{Show}\ a]}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Show}\ a}\ (\textsc{SpecC})$$

$$\dfrac{\dfrac{\dfrac{\mathit{Show}\ a \in [\mathit{Eq}\ a, \mathit{Show}\ a, \mathit{Eq}\ a]}{[\mathit{Eq}\ a, \mathit{Show}\ a, \mathit{Eq}\ a]; \Gamma \models \mathit{Show}\ a}\ (\textsc{SpecC})}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Eq}\ a \Rightarrow \mathit{Show}\ a}\ ({\Rightarrow}\mathrm{IC}) \qquad \dfrac{\mathit{Eq}\ a \in [\mathit{Eq}\ a, \mathit{Show}\ a]}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Eq}\ a}\ (\textsc{SpecC})}{[\mathit{Eq}\ a, \mathit{Show}\ a]; \Gamma \models \mathit{Show}\ a}\ ({\Rightarrow}\mathrm{EC})$$

Typing: Focusing [10]

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P; \Gamma \models [C]}$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P ; \Gamma \models [C]}$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P;\Gamma \models [C]}$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P; \Gamma \models [C]}$$

$$\boxed{\Gamma; [C] \models Q \leadsto A}$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P; \Gamma \models [C]}$$

$$\boxed{\Gamma; [C] \models Q \leadsto A}$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P; \Gamma \models [C]}$$

$$\boxed{\Gamma; [C] \models Q \leadsto A}$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P; \Gamma \models [C]}$$

$$\boxed{\Gamma; [C] \models Q \rightsquigarrow A}$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P; \Gamma \models [C]}$$

$$\boxed{\Gamma; [C] \models Q \rightsquigarrow A}$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P;\Gamma \models [C]}$$

$$\frac{P, C_1; \Gamma \models [C_2]}{P; \Gamma \models [C_1 \Rightarrow C_2]}\ (\Rightarrow\mathrm{R}) \qquad \frac{P; \Gamma, b \models [C]}{P; \Gamma \models [\forall b.C]}\ (\forall\mathrm{R})$$

$$\frac{C \in P :\ \ \Gamma; [C] \models Q \rightsquigarrow A \qquad \forall C_i \in A :\ \ P; \Gamma \models [C_i]}{P; \Gamma \models [Q]}\ (Q\mathrm{R})$$

$$\boxed{\Gamma; [C] \models Q \rightsquigarrow A}$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$\boxed{P; \Gamma \models [C]}$

$$\frac{P, C_1; \Gamma \models [C_2]}{P; \Gamma \models [C_1 \Rightarrow C_2]}\ (\Rightarrow\mathrm{R}) \qquad \frac{P; \Gamma, b \models [C]}{P; \Gamma \models [\forall b. C]}\ (\forall\mathrm{R})$$

$$\frac{C \in P :\ \ \Gamma; [C] \models Q \rightsquigarrow A \qquad \forall C_i \in A :\ \ P; \Gamma \models [C_i]}{P; \Gamma \models [Q]}\ (Q\mathrm{R})$$

$\boxed{\Gamma; [C] \models Q \rightsquigarrow A}$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P; \Gamma \models [C]}$$

$$\frac{P, C_1; \Gamma \models [C_2]}{P; \Gamma \models [C_1 \Rightarrow C_2]}\ (\Rightarrow\mathrm{R}) \qquad \frac{P; \Gamma, b \models [C]}{P; \Gamma \models [\forall b.C]}\ (\forall\mathrm{R})$$

$$\frac{C \in P : \ \Gamma; [C] \models Q \rightsquigarrow A \qquad \forall C_i \in A : \ \ P; \Gamma \models [C_i]}{P; \Gamma \models [Q]}\ (Q\mathrm{R})$$

$$\boxed{\Gamma; [C] \models Q \rightsquigarrow A}$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P; \Gamma \models [C]}$$

$$\frac{P, C_1; \Gamma \models [C_2]}{P; \Gamma \models [C_1 \Rightarrow C_2]}\ (\Rightarrow\mathrm{R}) \qquad \frac{P; \Gamma, b \models [C]}{P; \Gamma \models [\forall b.C]}\ (\forall\mathrm{R})$$

$$\frac{C \in P :\ \Gamma; [C] \models Q \rightsquigarrow A \qquad \forall C_i \in A :\ \ P; \Gamma \models [C_i]}{P; \Gamma \models [Q]}\ (Q\mathrm{R})$$

$$\boxed{\Gamma; [C] \models Q \rightsquigarrow A}$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P;\Gamma \models [C]}$$

$$\frac{P, C_1; \Gamma \models [C_2]}{P; \Gamma \models [C_1 \Rightarrow C_2]}\ (\Rightarrow\mathrm{R}) \qquad \frac{P; \Gamma, b \models [C]}{P; \Gamma \models [\forall b.C]}\ (\forall\mathrm{R})$$

$$\frac{C \in P : \ \Gamma; [C] \models Q \rightsquigarrow A \qquad \forall C_i \in A : \ \ P; \Gamma \models [C_i]}{P; \Gamma \models [Q]}\ (Q\mathrm{R})$$

$$\boxed{\Gamma; [C] \models Q \rightsquigarrow A}$$

$$\frac{\Gamma; [C_2] \models Q \rightsquigarrow A}{\Gamma; [C_1 \Rightarrow C_2] \models Q \rightsquigarrow A, C_1}\ (\Rightarrow\mathrm{L})$$

$$\frac{\Gamma; [[\tau/b]C] \models Q \rightsquigarrow A \qquad \Gamma \vdash_{\mathrm{ty}} \tau}{\Gamma; [\forall b.C] \models Q \rightsquigarrow A}\ (\forall\mathrm{L}) \qquad \frac{}{\Gamma; [Q] \models Q \rightsquigarrow \bullet}\ (Q\mathrm{L})$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P; \Gamma \models [C]}$$

$$\frac{P, C_1; \Gamma \models [C_2]}{P; \Gamma \models [C_1 \Rightarrow C_2]}\ (\Rightarrow\mathrm{R}) \qquad \frac{P; \Gamma, b \models [C]}{P; \Gamma \models [\forall b.C]}\ (\forall\mathrm{R})$$

$$\frac{C \in P : \ \Gamma; [C] \models Q \rightsquigarrow A \qquad \forall C_i \in A : \ \ P; \Gamma \models [C_i]}{P; \Gamma \models [Q]}\ (Q\mathrm{R})$$

$$\boxed{\Gamma; [C] \models Q \rightsquigarrow A}$$

$$\frac{\Gamma; [C_2] \models Q \rightsquigarrow A}{\Gamma; [C_1 \Rightarrow C_2] \models Q \rightsquigarrow A, C_1}\ (\Rightarrow\mathrm{L})$$

$$\frac{\Gamma; [[\tau/b]C] \models Q \rightsquigarrow A \qquad \Gamma \vdash_{\mathrm{ty}} \tau}{\Gamma; [\forall b.C] \models Q \rightsquigarrow A}\ (\forall\mathrm{L}) \qquad \frac{}{\Gamma; [Q] \models Q \rightsquigarrow \bullet}\ (Q\mathrm{L})$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P; \Gamma \models [C]}$$

$$\frac{P, C_1; \Gamma \models [C_2]}{P; \Gamma \models [C_1 \Rightarrow C_2]}\ (\Rightarrow\mathrm{R}) \qquad \frac{P; \Gamma, b \models [C]}{P; \Gamma \models [\forall b.C]}\ (\forall\mathrm{R})$$

$$\frac{C \in P :\ \ \Gamma; [C] \models Q \rightsquigarrow A \qquad \forall C_i \in A :\ \ P; \Gamma \models [C_i]}{P; \Gamma \models [Q]}\ (Q\mathrm{R})$$

$$\boxed{\Gamma; [C] \models Q \rightsquigarrow A}$$

$$\frac{\Gamma; [C_2] \models Q \rightsquigarrow A}{\Gamma; [C_1 \Rightarrow C_2] \models Q \rightsquigarrow A, C_1}\ (\Rightarrow\mathrm{L})$$

$$\frac{\Gamma; [[\tau/b]C] \models Q \rightsquigarrow A \qquad \Gamma \vdash_{\mathsf{ty}} \tau}{\Gamma; [\forall b.C] \models Q \rightsquigarrow A}\ (\forall\mathrm{L}) \qquad \frac{}{\Gamma; [Q] \models Q \rightsquigarrow \bullet}\ (Q\mathrm{L})$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P; \Gamma \models [C]}$$

$$\frac{P, C_1; \Gamma \models [C_2]}{P; \Gamma \models [C_1 \Rightarrow C_2]}\ (\Rightarrow\mathrm{R}) \qquad \frac{P; \Gamma, b \models [C]}{P; \Gamma \models [\forall b. C]}\ (\forall\mathrm{R})$$

$$\frac{C \in P : \ \Gamma; [C] \models Q \rightsquigarrow A \qquad \forall C_i \in A : \ \ P; \Gamma \models [C_i]}{P; \Gamma \models [Q]}\ (Q\mathrm{R})$$

$$\boxed{\Gamma; [C] \models Q \rightsquigarrow A}$$

$$\frac{\Gamma; [C_2] \models Q \rightsquigarrow A}{\Gamma; [C_1 \Rightarrow C_2] \models Q \rightsquigarrow A, C_1}\ (\Rightarrow\mathrm{L})$$

$$\frac{\Gamma; [[\tau/b]C] \models Q \rightsquigarrow A \qquad \Gamma \vdash_{\mathrm{ty}} \tau}{\Gamma; [\forall b. C] \models Q \rightsquigarrow A}\ (\forall\mathrm{L}) \qquad \frac{}{\Gamma; [Q] \models Q \rightsquigarrow \bullet}\ (Q\mathrm{L})$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P;\Gamma \models C}$$

$$\frac{P;\Gamma \models [C]}{P;\Gamma \models C}$$

$$\boxed{P;\Gamma \models [C]}$$

$$\frac{P, C_1;\Gamma \models [C_2]}{P;\Gamma \models [C_1 \Rightarrow C_2]}\ (\Rightarrow\mathrm{R}) \qquad \frac{P;\Gamma, b \models [C]}{P;\Gamma \models [\forall b.C]}\ (\forall\mathrm{R})$$

$$\frac{C \in P :\ \Gamma;[C] \models Q \rightsquigarrow A \qquad \forall C_i \in A :\ P;\Gamma \models [C_i]}{P;\Gamma \models [Q]}\ (Q\mathrm{R})$$

$$\boxed{\Gamma;[C] \models Q \rightsquigarrow A}$$

$$\frac{\Gamma;[C_2] \models Q \rightsquigarrow A}{\Gamma;[C_1 \Rightarrow C_2] \models Q \rightsquigarrow A, C_1}\ (\Rightarrow\mathrm{L})$$

$$\frac{\Gamma;[[\tau/b]C] \models Q \rightsquigarrow A \qquad \Gamma \vdash_{\mathtt{ty}} \tau}{\Gamma;[\forall b.C] \models Q \rightsquigarrow A}\ (\forall\mathrm{L}) \qquad \frac{}{\Gamma;[Q] \models Q \rightsquigarrow \bullet}\ (Q\mathrm{L})$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P; \Gamma \models C}$$

$$\frac{P; \Gamma \models [C]}{P; \Gamma \models C}$$

$$\boxed{P; \Gamma \models [C]}$$

$$\frac{P, C_1; \Gamma \models [C_2]}{P; \Gamma \models [C_1 \Rightarrow C_2]}\ (\Rightarrow\mathrm{R}) \qquad \frac{P; \Gamma, b \models [C]}{P; \Gamma \models [\forall b.C]}\ (\forall\mathrm{R})$$

$$\frac{C \in P :\ \Gamma; [C] \models Q \leadsto A \qquad \forall C_i \in A :\ \ P; \Gamma \models [C_i]}{P; \Gamma \models [Q]}\ (Q\mathrm{R})$$

$$\boxed{\Gamma; [C] \models Q \leadsto A}$$

$$\frac{\Gamma; [C_2] \models Q \leadsto A}{\Gamma; [C_1 \Rightarrow C_2] \models Q \leadsto A, C_1}\ (\Rightarrow\mathrm{L})$$

$$\frac{\Gamma; [[\tau/b]C] \models Q \leadsto A \qquad \Gamma \vdash_{\mathsf{ty}} \tau}{\Gamma; [\forall b.C] \models Q \leadsto A}\ (\forall\mathrm{L}) \qquad \frac{}{\Gamma; [Q] \models Q \leadsto \bullet}\ (Q\mathrm{L})$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$\boxed{P;\Gamma \models C}$

$$\frac{P;\Gamma \models [C]}{P;\Gamma \models C}$$

$\boxed{P;\Gamma \models [C]}$

$$\frac{P, C_1;\Gamma \models [C_2]}{P;\Gamma \models [C_1 \Rightarrow C_2]}\ (\Rightarrow\mathrm{R}) \qquad \frac{P;\Gamma, b \models [C]}{P;\Gamma \models [\forall b.C]}\ (\forall\mathrm{R})$$

$$\frac{C \in P :\ \Gamma;[C] \models Q \rightsquigarrow A \qquad \forall C_i \in A :\ P;\Gamma \models [C_i]}{P;\Gamma \models [Q]}\ (Q\mathrm{R})$$

$\boxed{\Gamma;[C] \models Q \rightsquigarrow A}$

$$\frac{\Gamma;[C_2] \models Q \rightsquigarrow A}{\Gamma;[C_1 \Rightarrow C_2] \models Q \rightsquigarrow A, C_1}\ (\Rightarrow\mathrm{L})$$

$$\frac{\Gamma;[[\tau/b]C] \models Q \rightsquigarrow A \qquad \Gamma \vdash_{\mathtt{ty}} \tau}{\Gamma;[\forall b.C] \models Q \rightsquigarrow A}\ (\forall\mathrm{L}) \qquad \frac{}{\Gamma;[Q] \models Q \rightsquigarrow \bullet}\ (Q\mathrm{L})$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Typing: Focusing [10]

$$\boxed{P;\Gamma \models C}$$

$$\frac{P;\Gamma \models [C]}{P;\Gamma \models C}$$

$$\boxed{P;\Gamma \models [C]}$$

$$\frac{P, C_1;\Gamma \models [C_2]}{P;\Gamma \models [C_1 \Rightarrow C_2]}\ (\Rightarrow\mathrm{R}) \qquad \frac{P;\Gamma, b \models [C]}{P;\Gamma \models [\forall b.C]}\ (\forall\mathrm{R})$$

$$\frac{C \in P :\ \Gamma;[C] \models Q \rightsquigarrow A \qquad \forall C_i \in A :\ \ P;\Gamma \models [C_i]}{P;\Gamma \models [Q]}\ (Q\mathrm{R})$$

$$\boxed{\Gamma;[C] \models Q \rightsquigarrow A}$$

$$\frac{\Gamma;[C_2] \models Q \rightsquigarrow A}{\Gamma;[C_1 \Rightarrow C_2] \models Q \rightsquigarrow A, C_1}\ (\Rightarrow\mathrm{L})$$

$$\frac{\Gamma;[[\tau/b]C] \models Q \rightsquigarrow A \qquad \Gamma \vdash_{\mathrm{ty}} \tau}{\Gamma;[\forall b.C] \models Q \rightsquigarrow A}\ (\forall\mathrm{L}) \qquad \frac{}{\Gamma;[Q] \models Q \rightsquigarrow \bullet}\ (Q\mathrm{L})$$

[10] Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits

Type Inference

Type Inference

- Backtracking

Type Inference

- Backtracking
- Unification

Type Inference

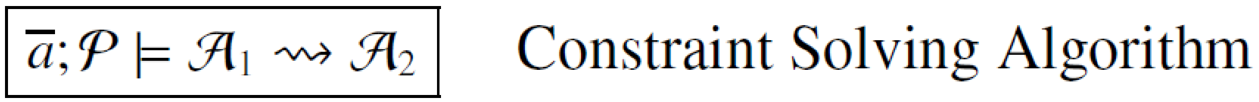
- Backtracking
- Unification
- Incremental

Type Inference

- Backtracking
- Unification
- Incremental
- Elaborate into System F

Type Inference

- Backtracking
- Unification
- Incremental
- Elaborate into System F



$\boxed{\overline{a};\mathcal{P} \models \mathcal{A}_1 \rightsquigarrow \mathcal{A}_2}$ Constraint Solving Algorithm

Type Inference

- Backtracking
- Unification
- Incremental
- Elaborate into System F

$\boxed{\overline{a}; \mathcal{P} \models \mathcal{A}_1 \rightsquigarrow \mathcal{A}_2}$ Constraint Solving Algorithm

$\boxed{\overline{a}; \mathcal{P} \models [C] \rightsquigarrow \mathcal{A}}$ Constraint Simplification

Type Inference

- Backtracking
- Unification
- Incremental
- Elaborate into System F

$\boxed{\overline{a}; \mathcal{P} \models \mathcal{A}_1 \rightsquigarrow \mathcal{A}_2}$ Constraint Solving Algorithm

$\boxed{\overline{a}; \mathcal{P} \models [C] \rightsquigarrow \mathcal{A}}$ Constraint Simplification

$\boxed{\overline{a}; [C] \models Q \rightsquigarrow \mathcal{A}; \theta}$ Constraint Matching

Type Inference

$$\bullet; \mathcal{P} \models [\forall b. Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (\forall b. \qquad ? \qquad)$$

Type Inference

$$\frac{\underline{b};\mathcal{P} \models [Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (Eq\ b \Rightarrow \quad ? \quad)}{\bullet;\mathcal{P} \models [\underline{\forall b}.Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (\forall b. \qquad ? \qquad)}\ (\forall \mathrm{R})$$

Type Inference

$$\dfrac{\dfrac{b;\mathcal{P}, \underline{Eq\ b} \models [Eq\ [b]] \rightsquigarrow\ ?}{b;\mathcal{P} \models [\underline{Eq\ b} \Rightarrow Eq\ [b]] \rightsquigarrow (Eq\ b \Rightarrow\ ?\ )}\,(\Rightarrow\mathrm{R})}{\bullet;\mathcal{P} \models [\forall b. Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (\forall b.\qquad ?\qquad)}\,(\forall\mathrm{R})$$

Type Inference

$$\dfrac{\dfrac{b; [\underline{\forall a. Eq\ a \Rightarrow Eq\ [a]}] \models Eq\ [b] \rightsquigarrow\ ?\ \ ;\ ?}{b; \mathcal{P}, Eq\ b \models [Eq\ [b]] \rightsquigarrow\ ?}\ (QR)}{\dfrac{b; \mathcal{P} \models [Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (Eq\ b \Rightarrow\ ?\ )}{\bullet; \mathcal{P} \models [\forall b. Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (\forall b.\ \ ?\ \ )}\ (\forall R)}\ (\Rightarrow R)$$

Type Inference

$$\cfrac{\cfrac{\cfrac{\cfrac{b; [Eq\ a \Rightarrow Eq\ [a]] \models Eq\ [b] \rightsquigarrow\ ?\quad ;\ ?}{b; [\underline{\forall a}.Eq\ a \Rightarrow Eq\ [a]] \models Eq\ [b] \rightsquigarrow\ ?\quad ;\ ?}\ (\forall \mathrm{L})}{b; \mathcal{P}, Eq\ b \models [Eq\ [b]] \rightsquigarrow\ ?}\ (\mathrm{QR})}{b; \mathcal{P} \models [Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (Eq\ b \Rightarrow\ ?\ )}\ (\Rightarrow\mathrm{R})}{\bullet; \mathcal{P} \models [\forall b.Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (\forall b.\quad ?\quad)}\ (\forall \mathrm{R})$$

Type Inference

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{b; [Eq\ [a]] \models Eq\ [b] \rightsquigarrow \bullet;\ ?}{b; [\underline{Eq\ a \Rightarrow}\ Eq\ [a]] \models Eq\ [b] \rightsquigarrow\ ?\ \ ;\ ?}\ (\Rightarrow\text{L})}{b; [\forall a. Eq\ a \Rightarrow Eq\ [a]] \models Eq\ [b] \rightsquigarrow\ ?\ \ ;\ ?}\ (\forall\text{L})}{b; \mathcal{P}, Eq\ b \models [Eq\ [b]] \rightsquigarrow\ ?}\ (Q\text{R})}{b; \mathcal{P} \models [Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (Eq\ b \Rightarrow\ ?\ )}\ (\Rightarrow\text{R})}{\bullet; \mathcal{P} \models [\forall b. Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (\forall b.\ \ \ ?\ \ \ )}\ (\forall\text{R})$$

Type Inference

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{unify(b; a \sim b) = \theta = [b/a]}{b; [Eq\,[a]] \models Eq\,[b] \rightsquigarrow \bullet;\ ?}\,(\mathrm{QL})}{b; [Eq\,a \Rightarrow Eq\,[a]] \models Eq\,[b] \rightsquigarrow\ ?\ ;\ ?}\,(\Rightarrow\mathrm{L})}{b; [\forall a.Eq\,a \Rightarrow Eq\,[a]] \models Eq\,[b] \rightsquigarrow\ ?\ ;\ ?}\,(\forall\mathrm{L})}{b; \mathcal{P}, Eq\,b \models [Eq\,[b]] \rightsquigarrow\ ?}\,(\mathrm{QR})}{b; \mathcal{P} \models [Eq\,b \Rightarrow Eq\,[b]] \rightsquigarrow (Eq\,b \Rightarrow\ ?\ )}\,(\Rightarrow\mathrm{R})}{\bullet; \mathcal{P} \models [\forall b.Eq\,b \Rightarrow Eq\,[b]] \rightsquigarrow (\forall b.\ \ ?\ \ )}\,(\forall\mathrm{R})$$

Type Inference

$$
\dfrac{
 \dfrac{
  \dfrac{
   \dfrac{
    \dfrac{
     \dfrac{unify(b; a \sim b) = \theta = [b/a]}{b; [Eq\ [a]] \models Eq\ [b] \leadsto \underline{\bullet; \theta}}\ \mathrm{(QL)}
    }{b; [Eq\ a \Rightarrow Eq\ [a]] \models Eq\ [b] \leadsto \ ?\ \ ; ?}\ (\Rightarrow\mathrm{L})
   }{b; [\forall a. Eq\ a \Rightarrow Eq\ [a]] \models Eq\ [b] \leadsto \ ?\ \ ; ?}\ (\forall\mathrm{L})
  }{b; \mathcal{P}, Eq\ b \models [Eq\ [b]] \leadsto \ ?}\ \mathrm{(QR)}
 }{b; \mathcal{P} \models [Eq\ b \Rightarrow Eq\ [b]] \leadsto (Eq\ b \Rightarrow \ ?\ )}\ (\Rightarrow\mathrm{R})
}{\bullet; \mathcal{P} \models [\forall b. Eq\ b \Rightarrow Eq\ [b]] \leadsto (\forall b. \qquad ? \qquad)}\ (\forall\mathrm{R})
$$

Type Inference

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{unify(b; a \sim b) = \theta = [b/a]}{b; [Eq\ [a]] \models Eq\ [b] \rightsquigarrow \bullet; \theta}\ (\text{QL})}{b; [Eq\ a \Rightarrow Eq\ [a]] \models Eq\ [b] \rightsquigarrow \underline{Eq\ b; \theta}}\ (\Rightarrow\text{L})}{b; [\forall a. Eq\ a \Rightarrow Eq\ [a]] \models Eq\ [b] \rightsquigarrow\ ?\ ;\ ?}\ (\forall\text{L})}{b; \mathcal{P}, Eq\ b \models [Eq\ [b]] \rightsquigarrow\ ?}\ (\text{QR})}{b; \mathcal{P} \models [Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (Eq\ b \Rightarrow\ ?\ )}\ (\Rightarrow\text{R})}{\bullet; \mathcal{P} \models [\forall b. Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (\forall b.\ \ ?\ \ )}\ (\forall\text{R})$$

Type Inference

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{unify(b; a \sim b) = \theta = [b/a]}{b; [Eq\ [a]] \models Eq\ [b] \rightsquigarrow \bullet; \theta}\ (QL)}{b; [Eq\ a \Rightarrow Eq\ [a]] \models Eq\ [b] \rightsquigarrow Eq\ b; \theta}\ (\Rightarrow L)}{b; [\forall a. Eq\ a \Rightarrow Eq\ [a]] \models Eq\ [b] \rightsquigarrow \underline{Eq\ b; \theta}}\ (\forall L)}{b; \mathcal{P}, Eq\ b \models [Eq\ [b]] \rightsquigarrow\ ?}\ (QR)}{b; \mathcal{P} \models [Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (Eq\ b \Rightarrow\ ?\ )}\ (\Rightarrow R)}{\bullet; \mathcal{P} \models [\forall b. Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (\forall b.\ \ ?\ \ )}\ (\forall R)$$

Type Inference

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathit{unify}(b; a \sim b) = \theta = [b/a]}{b; [\mathit{Eq}\ [a]] \models \mathit{Eq}\ [b] \rightsquigarrow \bullet; \theta}\ (\mathrm{QL})}{b; [\mathit{Eq}\ a \Rightarrow \mathit{Eq}\ [a]] \models \mathit{Eq}\ [b] \rightsquigarrow \mathit{Eq}\ b; \theta}\ (\Rightarrow\mathrm{L})}{b; [\forall a. \mathit{Eq}\ a \Rightarrow \mathit{Eq}\ [a]] \models \mathit{Eq}\ [b] \rightsquigarrow \mathit{Eq}\ b; \theta}\ (\forall\mathrm{L})}{b; \mathcal{P}, \mathit{Eq}\ b \models [\mathit{Eq}\ [b]] \rightsquigarrow \underline{\mathit{Eq}\ b}}\ (\mathrm{QR})}{b; \mathcal{P} \models [\mathit{Eq}\ b \Rightarrow \mathit{Eq}\ [b]] \rightsquigarrow (\mathit{Eq}\ b \Rightarrow\ ?\ )}\ (\Rightarrow\mathrm{R})}{\bullet; \mathcal{P} \models [\forall b. \mathit{Eq}\ b \Rightarrow \mathit{Eq}\ [b]] \rightsquigarrow (\forall b.\ \ ?\ \ )}\ (\forall\mathrm{R})$$

Type Inference

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{unify(b; a \sim b) = \theta = [b/a]}{b; [Eq\;[a]] \models Eq\;[b] \leadsto \bullet; \theta}\;(\text{QL})}{b; [Eq\;a \Rightarrow Eq\;[a]] \models Eq\;[b] \leadsto Eq\;b; \theta}\;(\Rightarrow\text{L})}{b; [\forall a.Eq\;a \Rightarrow Eq\;[a]] \models Eq\;[b] \leadsto Eq\;b; \theta}\;(\forall\text{L})}{b; \mathcal{P}, Eq\;b \models [Eq\;[b]] \leadsto Eq\;b}\;(\text{QR})}{b; \mathcal{P} \models [Eq\;b \Rightarrow Eq\;[b]] \leadsto \underline{(Eq\;b \Rightarrow Eq\;b)}}\;(\Rightarrow\text{R})}{\bullet; \mathcal{P} \models [\forall b.Eq\;b \Rightarrow Eq\;[b]] \leadsto (\forall b. \qquad ? \qquad)}\;(\forall\text{R})
$$

Type Inference

$$
\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{unify(b; a \sim b) = \theta = [b/a]}{b; [Eq\ [a]] \models Eq\ [b] \rightsquigarrow \bullet; \theta}\ (\text{QL})}{b; [Eq\ a \Rightarrow Eq\ [a]] \models Eq\ [b] \rightsquigarrow Eq\ b; \theta}\ (\Rightarrow\text{L})}{b; [\forall a. Eq\ a \Rightarrow Eq\ [a]] \models Eq\ [b] \rightsquigarrow Eq\ b; \theta}\ (\forall\text{L})}{b; \mathcal{P}, Eq\ b \models [Eq\ [b]] \rightsquigarrow Eq\ b}\ (\text{QR})}{b; \mathcal{P} \models [Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow (Eq\ b \Rightarrow Eq\ b)}\ (\Rightarrow\text{R})}{\bullet; \mathcal{P} \models [\forall b. Eq\ b \Rightarrow Eq\ [b]] \rightsquigarrow \underline{(\forall b. Eq\ b \Rightarrow Eq\ b)}}\ (\forall\text{R})
$$

Type Inference

$$\bullet; P \models [\forall b. Eq\ b \Rightarrow Eq\ b] \rightsquigarrow ?$$

Type Inference

$$\frac{\underline{b}; P \models [Eq\ b \Rightarrow Eq\ b] \rightsquigarrow\ ?}{\bullet; P \models [\underline{\forall b}.Eq\ b \Rightarrow Eq\ b] \rightsquigarrow\ ?}\ (\forall \mathrm{R})$$

Type Inference

$$\dfrac{\dfrac{b; P, \underline{Eq\ b} \models [Eq\ b] \rightsquigarrow\ ?}{b; P \models [\underline{Eq\ b \Rightarrow}\ Eq\ b] \rightsquigarrow\ ?}\ (\Rightarrow\mathrm{R})}{\bullet; P \models [\forall b. Eq\ b \Rightarrow Eq\ b] \rightsquigarrow\ ?}\ (\forall\mathrm{R})$$

Type Inference

$$\dfrac{\dfrac{\dfrac{b; \underline{[Eq\ b]} \models Eq\ b \rightsquigarrow ?;\ ?}{b; P, Eq\ b \models [Eq\ b] \rightsquigarrow\ ?}\ (QR)}{b; P \models [Eq\ b \Rightarrow Eq\ b] \rightsquigarrow\ ?}\ (\Rightarrow\mathrm{R})}{\bullet; P \models [\forall b. Eq\ b \Rightarrow Eq\ b] \rightsquigarrow\ ?}\ (\forall\mathrm{R})$$

Type Inference

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathit{unify}(b; b \sim b) = \theta = \;?}{b; [Eq\ b] \models Eq\ b \rightsquigarrow \;?;\;?}\,(QL)}{b; P, Eq\ b \models [Eq\ b] \rightsquigarrow \;?}\,(QR)}{b; P \models [Eq\ b \Rightarrow Eq\ b] \rightsquigarrow \;?}\,(\Rightarrow\mathrm{R})}{\bullet; P \models [\forall b. Eq\ b \Rightarrow Eq\ b] \rightsquigarrow \;?}\,(\forall\mathrm{R})$$

Type Inference

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathit{unify}(b; b \sim b) = \theta = \underline{\bullet}}{b; [\mathit{Eq}\ b] \models \mathit{Eq}\ b \rightsquigarrow ?;\ ?}\ (\mathcal{Q}\mathrm{L})}{b; P, \mathit{Eq}\ b \models [\mathit{Eq}\ b] \rightsquigarrow\ ?}\ (\mathcal{Q}\mathrm{R})}{b; P \models [\mathit{Eq}\ b \Rightarrow \mathit{Eq}\ b] \rightsquigarrow\ ?}\ (\Rightarrow\mathrm{R})}{\bullet; P \models [\forall b.\mathit{Eq}\ b \Rightarrow \mathit{Eq}\ b] \rightsquigarrow\ ?}\ (\forall\mathrm{R})$$

Type Inference

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathit{unify}(b; b \sim b) = \theta = \bullet}{b; [\mathit{Eq}\ b] \models \mathit{Eq}\ b \leadsto \underline{\bullet; \bullet}}\ (\mathit{Q}\mathrm{L})}{b; P, \mathit{Eq}\ b \models [\mathit{Eq}\ b] \leadsto\ ?}\ (\mathit{Q}\mathrm{R})}{b; P \models [\mathit{Eq}\ b \Rightarrow \mathit{Eq}\ b] \leadsto\ ?}\ (\Rightarrow\mathrm{R})}{\bullet; P \models [\forall b.\mathit{Eq}\ b \Rightarrow \mathit{Eq}\ b] \leadsto\ ?}\ (\forall\mathrm{R})$$

Type Inference

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathit{unify}(b; b \sim b) = \theta = \bullet}{b; [\mathit{Eq}\ b] \models \mathit{Eq}\ b \rightsquigarrow \bullet; \bullet}\ (\mathit{Q}\mathrm{L})}{b; P, \mathit{Eq}\ b \models [\mathit{Eq}\ b] \rightsquigarrow \underline{\bullet}}\ (\mathit{Q}\mathrm{R})}{b; P \models [\mathit{Eq}\ b \Rightarrow \mathit{Eq}\ b] \rightsquigarrow ?}\ (\Rightarrow\mathrm{R})}{\bullet; P \models [\forall b.\mathit{Eq}\ b \Rightarrow \mathit{Eq}\ b] \rightsquigarrow ?}\ (\forall\mathrm{R})$$

Type Inference

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathit{unify}(b; b \sim b) = \theta = \bullet}{b; [Eq\ b] \models Eq\ b \rightsquigarrow \bullet; \bullet}\ (QL)}{b; P, Eq\ b \models [Eq\ b] \rightsquigarrow \bullet}\ (QR)}{b; P \models [Eq\ b \Rightarrow Eq\ b] \rightsquigarrow \underline{\bullet}}\ (\Rightarrow\mathrm{R})}{\bullet; P \models [\forall b. Eq\ b \Rightarrow Eq\ b] \rightsquigarrow ?}\ (\forall\mathrm{R})$$

Type Inference

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathit{unify}(b; b \sim b) = \theta = \bullet}{b; [\mathit{Eq}\ b] \models \mathit{Eq}\ b \leadsto \bullet; \bullet}\ (Q\mathrm{L})}{b; P, \mathit{Eq}\ b \models [\mathit{Eq}\ b] \leadsto \bullet}\ (Q\mathrm{R})}{b; P \models [\mathit{Eq}\ b \Rightarrow \mathit{Eq}\ b] \leadsto \bullet}\ (\Rightarrow\mathrm{R})}{\bullet; P \models [\forall b. \mathit{Eq}\ b \Rightarrow \mathit{Eq}\ b] \leadsto \underline{\bullet}}\ (\forall\mathrm{R})$$

Metatheory

Metatheory: Termination

Metatheory: Termination

- Resolution tree
  - Node = goal
  - Edge = applying axiom

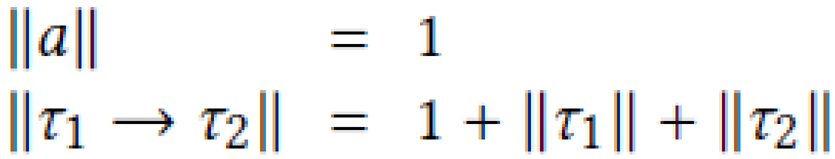
Metatheory: Termination

- Resolution tree
  - Node = goal
  - Edge = applying axiom
- Norm

$$
\begin{aligned}
\|a\| &= 1 \\
\|\tau_1 \longrightarrow \tau_2\| &= 1 + \|\tau_1\| + \|\tau_2\|
\end{aligned}
$$

Metatheory: Termination

- Resolution tree
  - Node = goal
  - Edge = applying axiom
- Norm
- Strictly decreasing -> no infinite paths



$$\begin{aligned} \|a\| &= 1 \\ \|\tau_1 \longrightarrow \tau_2\| &= 1 + \|\tau_1\| + \|\tau_2\| \end{aligned}$$

Metatheory: Termination

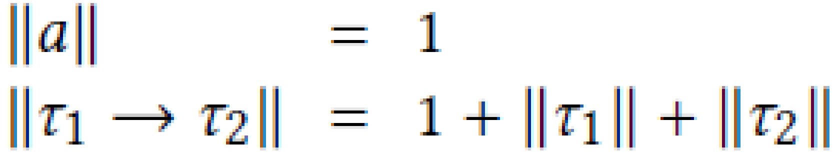
- Resolution tree
  - Node = goal
  - Edge = applying axiom
- Norm
- Strictly decreasing -> no infinite paths

- Superclass axiom: non increasing

$$\begin{aligned} \|a\| &= 1 \\ \|\tau_1 \longrightarrow \tau_2\| &= 1 + \|\tau_1\| + \|\tau_2\| \end{aligned}$$

Metatheory: Termination

- Resolution tree
  - Node = goal
  - Edge = applying axiom
- Norm
- Strictly decreasing -> no infinite paths
- Superclass axiom: non increasing
- DAG



$$\begin{aligned} \|a\| &= 1 \\ \|\tau_1 \longrightarrow \tau_2\| &= 1 + \|\tau_1\| + \|\tau_2\| \end{aligned}$$

Metatheory: Termination

- Resolution tree
  - Node = goal
  - Edge = applying axiom
- Norm
- Strictly decreasing -> no infinite paths

$$\begin{aligned} \|a\| &= 1 \\ \|\tau_1 \longrightarrow \tau_2\| &= 1 + \|\tau_1\| + \|\tau_2\| \end{aligned}$$

- Superclass axiom: non increasing
- DAG
- Bounded number of superclass applications

Metatheory: Coherence

Metatheory: Coherence

- Non determinism

Metatheory: Coherence

- Non determinism
- Computational content <= instances

Metatheory: Coherence

- Non determinism
- Computational content <= instances
- Non overlapping instances

Metatheory: Ambiguity

Metatheory: Ambiguity

```
instance C a => D Int where ...
```

Metatheory: Ambiguity

```
instance C a => D Int where ...
```

```
forall a. C a => D Int
```

Metatheory: Ambiguity

```
instance C a => D Int where ...
```

```
forall a. C a => D Int
```

- Haskell ‘98: All quantified variables should appear in the head

Metatheory: Ambiguity

```
instance C a => D Int where ...     ⇨     forall a. C a => D Int
```

- Haskell ‘98: All quantified variables should appear in the head
- QCC’s:

Metatheory: Ambiguity

```
instance C a => D Int where ...
```

➡

```
forall a. C a => D Int
```

- Haskell ‘98: All quantified variables should appear in the head
- QCC’s: $\boxed{\mathit{unamb}(C)}$ Unambiguity

$$\frac{\bullet \vdash_{\text{unamb}} C}{\mathit{unamb}(C)}\ \textsc{Unamb}$$

Metatheory: Ambiguity

```
instance C a => D Int where ...
```

```
forall a. C a => D Int
```

- Haskell ‘98: All quantified variables should appear in the head
- QCC’s:

$$\boxed{unamb(C)} \quad \text{Unambiguity}$$

$$\frac{\bullet \vdash_{\text{unamb}} C}{unamb(C)}\ \textsc{Unamb}$$

$$\boxed{\overline{a} \vdash_{\text{unamb}} C} \quad \text{Unambiguity}$$

$$\frac{\overline{a} \subseteq fv(Q)}{\overline{a} \vdash_{\text{unamb}} Q}\ (\text{QU}) \qquad \frac{\overline{a}, a \vdash_{\text{unamb}} C}{\overline{a} \vdash_{\text{unamb}} \forall a.C}\ (\forall\text{U}) \qquad \frac{\begin{array}{c} unamb(C_1) \\ \overline{a} \vdash_{\text{unamb}} C_2 \end{array}}{\overline{a} \vdash_{\text{unamb}} C_1 \Rightarrow C_2}\ (\Rightarrow\text{U})$$

Future Work

Future Work

- Metatheory

Future Work

- Metatheory
- Quantification over Predicates

Future Work

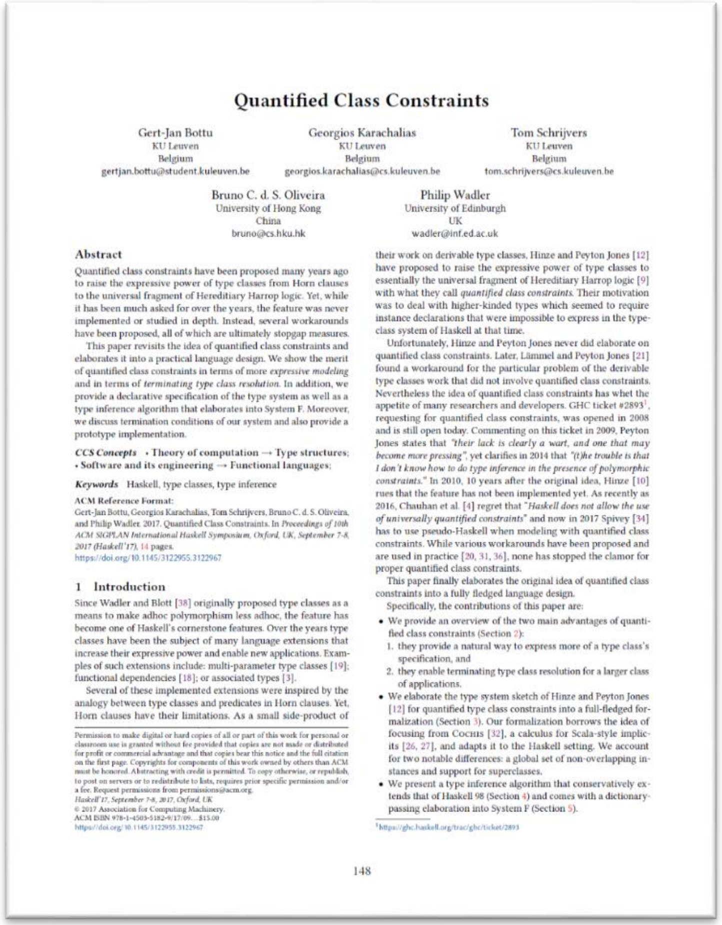
- Metatheory
- Quantification over Predicates
- Interaction with mainstream GHC features

Future Work

- Metatheory
- Quantification over Predicates
- Interaction with mainstream GHC features
- Coercions problem [11]

[11] https://ghc.haskell.org/trac/ghc/ticket/9123

Quantified Class Constraints



Quantified Class Constraints

Gert-Jan Bottu
KU Leuven
Belgium
gertjan.bottu@student.kuleuven.be

Georgios Karachalias
KU Leuven
Belgium
georgios.karachalias@cs.kuleuven.be

Tom Schrijvers
KU Leuven
Belgium
tom.schrijvers@cs.kuleuven.be

Bruno C. d. S. Oliveira
University of Hong Kong
China
bruno@cs.hku.hk

Philip Wadler
University of Edinburgh
UK
wadler@inf.ed.ac.uk

Abstract

Quantified class constraints have been proposed many years ago to raise the expressive power of type classes from Horn clauses to the universal fragment of Hereditiary Harrop logic. Yet, while it has been much asked for over the years, the feature was never implemented or studied in depth. Instead, several workarounds have been proposed, all of which are ultimately stopgap measures.

This paper revisits the idea of quantified class constraints and elaborates it into a practical language design. We show the merit of quantified class constraints in terms of more *expressive modeling* and in terms of *terminating type class resolution*. In addition, we provide a declarative specification of the type system as well as a type inference algorithm that elaborates into System F. Moreover, we discuss termination conditions of our system and also provide a prototype implementation.

CCS Concepts **• Theory of computation → Type structures; • Software and its engineering → Functional languages;**

Keywords Haskell, type classes, type inference

ACM Reference Format:
Gert-Jan Bottu, Georgios Karachalias, Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Quantified Class Constraints. In *Proceedings of 10th ACM SIGPLAN International Haskell Symposium, Oxford, UK, September 7-8, 2017 (Haskell'17),* 14 pages.
https://doi.org/10.1145/3122955.3122967

1 Introduction

Since Wadler and Blott [38] originally proposed type classes as a means to make adhoc polymorphism less adhoc, the feature has become one of Haskell's cornerstone features. Over the years type classes have been the subject of many language extensions that increase their expressive power and enable new applications. Examples of such extensions include: multi-parameter type classes [19]; functional dependencies [18]; or associated types [3].

Several of these implemented extensions were inspired by the analogy between type classes and predicates in Horn clauses. Yet, Horn clauses have their limitations. As a small side-product of their work on derivable type classes, Hinze and Peyton Jones [12] have proposed to raise the expressive power of type classes to essentially the universal fragment of Hereditiary Harrop logic [9] with what they call *quantified class constraints.* Their motivation was to deal with higher-kinded types which seemed to require instance declarations that were impossible to express in the type-class system of Haskell at that time.

Unfortunately, Hinze and Peyton Jones never did elaborate on quantified class constraints. Later, Lämmel and Peyton Jones [21] found a workaround for the particular problem of the derivable type classes work that did not involve quantified class constraints. Nevertheless the idea of quantified class constraints has whet the appetite of many researchers and developers. GHC ticket #2893[1], requesting for quantified class constraints, was opened in 2008 and is still open today. Commenting on this ticket in 2009, Peyton Jones states that *"Their lack is clearly a wart, and one that may become more pressing"*, yet clarifies in 2014 that *"(t)he trouble is that I don't know how to do type inference in the presence of polymorphic constraints."* In 2010, 10 years after the original idea, Hinze [10] rues that the feature has not been implemented yet. As recently as 2016, Chauhan et al. [4] regret that *"Haskell does not allow the use of universally quantified constraints"* and now in 2017 Spivey [34] has to use pseudo-Haskell when modeling with quantified class constraints. While various workarounds have been proposed and are used in practice [20, 31, 36], none has stopped the clamor for proper quantified class constraints.

This paper finally elaborates the original idea of quantified class constraints into a fully fledged language design.

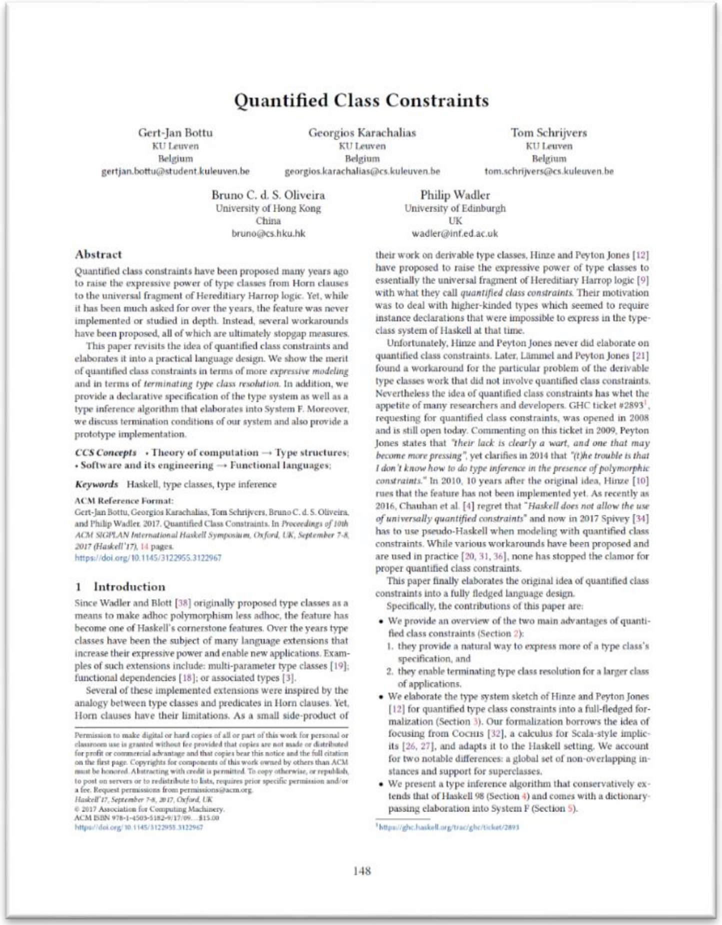
Specifically, the contributions of this paper are:

- We provide an overview of the two main advantages of quantified class constraints (Section 2):
  1. they provide a natural way to express more of a type class's specification, and
  2. they enable terminating type class resolution for a larger class of applications.
- We elaborate the type system sketch of Hinze and Peyton Jones [12] for quantified type class constraints into a full-fledged formalization (Section 3). Our formalization borrows the idea of focusing from COCHIS [32], a calculus for Scala-style implicits [26, 27], and adapts it to the Haskell setting. We account for two notable differences: a global set of non-overlapping instances and support for superclasses.
- We present a type inference algorithm that conservatively extends that of Haskell 98 (Section 4) and comes with a dictionary-passing elaboration into System F (Section 5).

Haskell'17, September 7-8, 2017, Oxford, UK
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5182-9/17/09...$15.00
https://doi.org/10.1145/3122955.3122967

[1] https://ghc.haskell.org/trac/ghc/ticket/2893

Quantified Class Constraints



Quantified Class Constraints

Gert-Jan Bottu
KU Leuven
Belgium
gertjan.bottu@student.kuleuven.be

Georgios Karachalias
KU Leuven
Belgium
georgios.karachalias@cs.kuleuven.be

Tom Schrijvers
KU Leuven
Belgium
tom.schrijvers@cs.kuleuven.be

Bruno C. d. S. Oliveira
University of Hong Kong
China
bruno@cs.hku.hk

Philip Wadler
University of Edinburgh
UK
wadler@inf.ed.ac.uk

Abstract

Quantified class constraints have been proposed many years ago to raise the expressive power of type classes from Horn clauses to the universal fragment of Hereditiary Harrop logic. Yet, while it has been much asked for over the years, the feature was never implemented or studied in depth. Instead, several workarounds have been proposed, all of which are ultimately stopgap measures.

This paper revisits the idea of quantified class constraints and elaborates it into a practical language design. We show the merit of quantified class constraints in terms of more *expressive modeling* and in terms of *terminating type class resolution*. In addition, we provide a declarative specification of the type system as well as a type inference algorithm that elaborates into System F. Moreover, we discuss termination conditions of our system and also provide a prototype implementation.

CCS Concepts • **Theory of computation → Type structures;** • **Software and its engineering → Functional languages;**

Keywords Haskell, type classes, type inference

ACM Reference Format:
Gert-Jan Bottu, Georgios Karachalias, Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Quantified Class Constraints. In *Proceedings of 10th ACM SIGPLAN International Haskell Symposium, Oxford, UK, September 7-8, 2017 (Haskell'17)*, 14 pages.
https://doi.org/10.1145/3122955.3122967

1 Introduction

Since Wadler and Blott [38] originally proposed type classes as a means to make adhoc polymorphism less adhoc, the feature has become one of Haskell's cornerstone features. Over the years type classes have been the subject of many language extensions that increase their expressive power and enable new applications. Examples of such extensions include: multi-parameter type classes [19]; functional dependencies [18]; or associated types [3].

Several of these implemented extensions were inspired by the analogy between type classes and predicates in Horn clauses. Yet, Horn clauses have their limitations. As a small side-product of their work on derivable type classes, Hinze and Peyton Jones [12] have proposed to raise the expressive power of type classes to essentially the universal fragment of Hereditiary Harrop logic [9] with what they call *quantified class constraints*. Their motivation was to deal with higher-kinded types which seemed to require instance declarations that were impossible to express in the type-class system of Haskell at that time.

Unfortunately, Hinze and Peyton Jones never did elaborate on quantified class constraints. Later, Lämmel and Peyton Jones [21] found a workaround for the particular problem of the derivable type classes work that did not involve quantified class constraints. Nevertheless the idea of quantified class constraints has whet the appetite of many researchers and developers. GHC ticket #2893[1], requesting for quantified class constraints, was opened in 2008 and is still open today. Commenting on this ticket in 2009, Peyton Jones states that *"their lack is clearly a wart, and one that may become more pressing"*, yet clarifies in 2014 that *"(t)he trouble is that I don't know how to do type inference in the presence of polymorphic constraints."* In 2010, 10 years after the original idea, Hinze [10] rues that the feature has not been implemented yet. As recently as 2016, Chauhan et al. [4] regret that *"Haskell does not allow the use of universally quantified constraints"* and now in 2017 Spivey [34] has to use pseudo-Haskell when modeling with quantified class constraints. While various workarounds have been proposed and are used in practice [20, 31, 36], none has stopped the clamor for proper quantified class constraints.

This paper finally elaborates the original idea of quantified class constraints into a fully fledged language design.

Specifically, the contributions of this paper are:

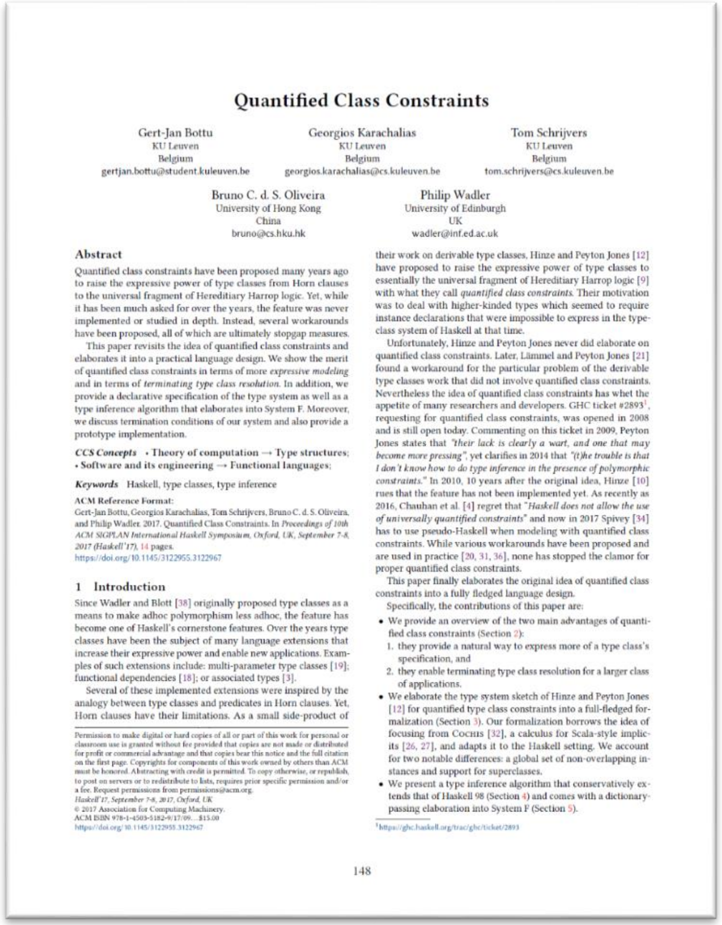
- We provide an overview of the two main advantages of quantified class constraints (Section 2):
  1. they provide a natural way to express more of a type class's specification, and
  2. they enable terminating type class resolution for a larger class of applications.
- We elaborate the type system sketch of Hinze and Peyton Jones [12] for quantified type class constraints into a full-fledged formalization (Section 3). Our formalization borrows the idea of focusing from Cochis [32], a calculus for Scala-style implicits [26, 27], and adapts it to the Haskell setting. We account for two notable differences: a global set of non-overlapping instances and support for superclasses.
- We present a type inference algorithm that conservatively extends that of Haskell 98 (Section 4) and comes with a dictionary-passing elaboration into System F (Section 5).

Haskell'17, September 7-8, 2017, Oxford, UK
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5182-9/17/09...$15.00
https://doi.org/10.1145/3122955.3122967

[1] https://ghc.haskell.org/trac/ghc/ticket/2893

- Additional examples
- Inference algorithm
- Elaboration

Quantified Class Constraints



Quantified Class Constraints

Gert-Jan Bottu
KU Leuven
Belgium
gertjan.bottu@student.kuleuven.be

Georgios Karachalias
KU Leuven
Belgium
georgios.karachalias@cs.kuleuven.be

Tom Schrijvers
KU Leuven
Belgium
tom.schrijvers@cs.kuleuven.be

Bruno C. d. S. Oliveira
University of Hong Kong
China
bruno@cs.hku.hk

Philip Wadler
University of Edinburgh
UK
wadler@inf.ed.ac.uk

Abstract

Quantified class constraints have been proposed many years ago to raise the expressive power of type classes from Horn clauses to the universal fragment of Hereditiary Harrop logic. Yet, while it has been much asked for over the years, the feature was never implemented or studied in depth. Instead, several workarounds have been proposed, all of which are ultimately stopgap measures.

This paper revisits the idea of quantified class constraints and elaborates it into a practical language design. We show the merit of quantified class constraints in terms of more *expressive modeling* and in terms of *terminating type class resolution*. In addition, we provide a declarative specification of the type system as well as a type inference algorithm that elaborates into System F. Moreover, we discuss termination conditions of our system and also provide a prototype implementation.

CCS Concepts • **Theory of computation → Type structures;** • **Software and its engineering → Functional languages;**

Keywords Haskell, type classes, type inference

ACM Reference Format:
Gert-Jan Bottu, Georgios Karachalias, Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Quantified Class Constraints. In *Proceedings of 10th ACM SIGPLAN International Haskell Symposium, Oxford, UK, September 7-8, 2017 (Haskell'17),* 14 pages.
https://doi.org/10.1145/3122955.3122967

1 Introduction

Since Wadler and Blott [38] originally proposed type classes as a means to make adhoc polymorphism less adhoc, the feature has become one of Haskell's cornerstone features. Over the years type classes have been the subject of many language extensions that increase their expressive power and enable new applications. Examples of such extensions include: multi-parameter type classes [19]; functional dependencies [18]; or associated types [3].

Several of these implemented extensions were inspired by the analogy between type classes and predicates in Horn clauses. Yet, Horn clauses have their limitations. As a small side-product of their work on derivable type classes, Hinze and Peyton Jones [12] have proposed to raise the expressive power of type classes to essentially the universal fragment of Hereditiary Harrop logic [9] with what they call *quantified class constraints*. Their motivation was to deal with higher-kinded types which seemed to require instance declarations that were impossible to express in the type-class system of Haskell at that time.

Unfortunately, Hinze and Peyton Jones never did elaborate on quantified class constraints. Later, Lämmel and Peyton Jones [21] found a workaround for the particular problem of the derivable type classes work that did not involve quantified class constraints. Nevertheless the idea of quantified class constraints has whet the appetite of many researchers and developers. GHC ticket #2893[1], requesting for quantified class constraints, was opened in 2008 and is still open today. Commenting on this ticket in 2009, Peyton Jones states that *"Their lack is clearly a wart, and one that may become more pressing"*, yet clarifies in 2014 that *"(t)he trouble is that I don't know how to do type inference in the presence of polymorphic constraints."* In 2010, 10 years after the original idea, Hinze [10] rues that the feature has not been implemented yet. As recently as 2016, Chauhan et al. [4] regret that *"Haskell does not allow the use of universally quantified constraints"* and now in 2017 Spivey [34] has to use pseudo-Haskell when modeling with quantified class constraints. While various workarounds have been proposed and are used in practice [20, 31, 36], none has stopped the clamor for proper quantified class constraints.

This paper finally elaborates the original idea of quantified class constraints into a fully fledged language design.

Specifically, the contributions of this paper are:

- We provide an overview of the two main advantages of quantified class constraints (Section 2):
  1. they provide a natural way to express more of a type class's specification, and
  2. they enable terminating type class resolution for a larger class of applications.
- We elaborate the type system sketch of Hinze and Peyton Jones [12] for quantified type class constraints into a full-fledged formalization (Section 3). Our formalization borrows the idea of focusing from Cochis [32], a calculus for Scala-style implicits [26, 27], and adapts it to the Haskell setting. We account for two notable differences: a global set of non-overlapping instances and support for superclasses.
- We present a type inference algorithm that conservatively extends that of Haskell 98 (Section 4) and comes with a dictionary-passing elaboration into System F (Section 5).

Haskell'17, September 7-8, 2017, Oxford, UK
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5182-9/17/09...$15.00
https://doi.org/10.1145/3122955.3122967

[1] https://ghc.haskell.org/trac/ghc/ticket/2893

- Additional examples
- Inference algorithm
- Elaboration

- https://github.com/gkaracha/quantcs-impl

Thanks!

Backtracking

```
class (E a => C a) => D a
class (G a => C a) => F a
```

Backtracking

```
class (E a => C a) => D a
class (G a => C a) => F a
```

```
D a => (E a => C a)
F a => (G a => C a)
```

Backtracking

```
class (E a => C a) => D a
class (G a => C a) => F a
```

```
D a => (E a => C a)
F a => (G a => C a)
```

```
Local : D a, F a, G a
Goal  : C a
```

Backtracking

```
class (E a => C a) => D a
class (G a => C a) => F a
```

```
D a => (E a => C a)
F a => (G a => C a)
```

```
Local : D a, F a, G a
Goal  : C a
```

Backtracking

```
class (E a => C a) => D a
class (G a => C a) => F a
```

```
D a => (E a => C a)
F a => (G a => C a)
```

```
Local : D a, F a, G a
Goal  : C a
```

Backtracking

```
class (E a => C a) => D a
class (G a => C a) => F a
```

```
D a => (E a => C a)
F a => (G a => C a)
```

```
Local : D a, F a, G a
Goal  : C a
```

Backtracking

```
class (E a => C a) => D a
class (G a => C a) => F a
```

```
D a => (E a => C a)
F a => (G a => C a)
```

```
Local : D a, F a, G a
Goal  : C a
```

Backtracking

Backtracking

- Order

Backtracking

- Order
- Order definition
  - Superclasses
  - Instances
  - Signatures
  - GADT pattern matching

Backtracking

- Order
- Order definition
  - Superclasses
  - Instances
  - Signatures
  - GADT pattern matching
- Prediction

Backtracking

- Order
- Order definition
  - Superclasses
  - Instances
  - Signatures
  - GADT pattern matching
- Prediction
- Reject overlap

Backtracking - Monotonicity

$$P \models C_1$$

$$P, C_2 \models C_1$$

Intermezzo: Simulating Quantified Class Constraints [7]

[7] Valery Trifonov. 2003. Simulating Quantified Class Constraints

Intermezzo: Simulating Quantified Class Constraints[7]

- Longer & more complex code

[7] Valery Trifonov. 2003. Simulating Quantified Class Constraints

Intermezzo: Simulating Quantified Class Constraints [7]

- Longer & more complex code
- Not generally applicable

[7] Valery Trifonov. 2003. Simulating Quantified Class Constraints

Elaboration

Elaboration

- System F

Elaboration

- System F
- Dictionary passing style

Elaboration

$$\boxed{\vdash_{\mathtt{ct}} C \rightsquigarrow \upsilon} \quad \text{Constraint Elaboration}$$

- System F
- Dictionary passing style

$$\frac{\vdash_{\mathtt{ty}} \tau \rightsquigarrow \upsilon}{\vdash_{\mathtt{ct}} TC\ \tau \rightsquigarrow T_{TC}\ \upsilon}\ (\mathrm{C}Q) \qquad \frac{\vdash_{\mathtt{ct}} C \rightsquigarrow \upsilon}{\vdash_{\mathtt{ct}} \forall a.C \rightsquigarrow \forall a.\upsilon}\ (\mathrm{C}\forall)$$

$$\frac{\vdash_{\mathtt{ct}} C_1 \rightsquigarrow \upsilon_1 \qquad \vdash_{\mathtt{ct}} C_2 \rightsquigarrow \upsilon_2}{\vdash_{\mathtt{ct}} C_1 \Rightarrow C_2 \rightsquigarrow \upsilon_1 \rightarrow \upsilon_2}\ (\mathrm{C}{\Rightarrow})$$

Metatheory

Metatheory

- Type preservation

Metatheory

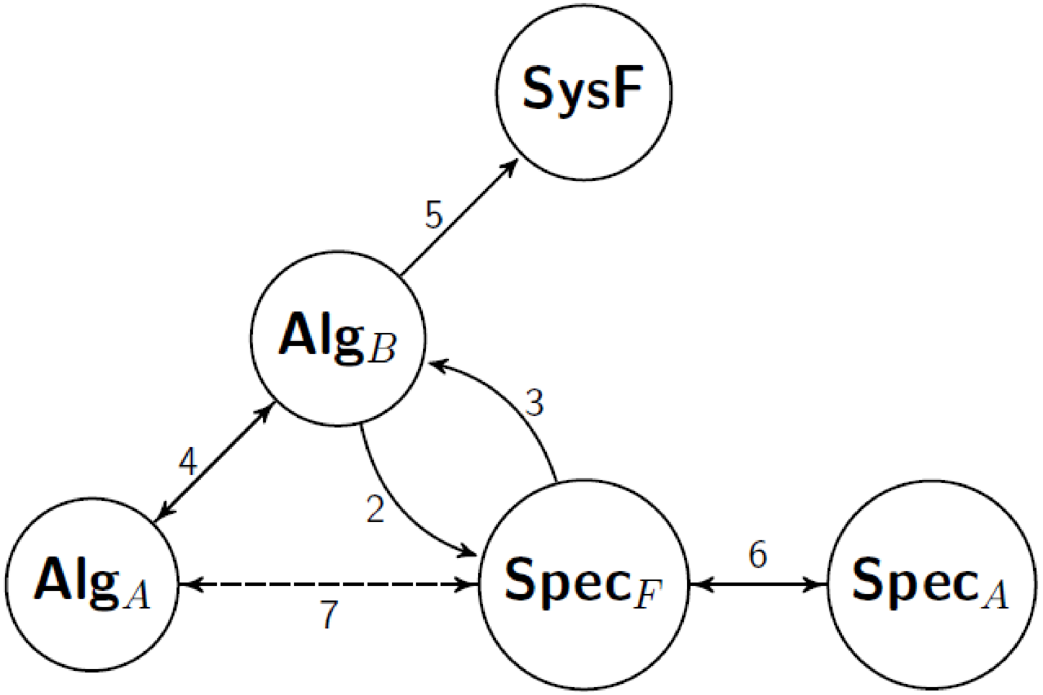
- Type preservation
- Equivalence Specification

Metatheory

- Type preservation
- Equivalence Specification
- Equivalence Specification & Algorithm

Metatheory

- Type preservation
- Equivalence Specification
- Equivalence Specification & Algorithm



Related Work

Related Work

- Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

Related Work

- Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes
- Valery Trifonov. 2003. Simulating Quantified Class Constraints

Related Work

- Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes
- Valery Trifonov. 2003. Simulating Quantified Class Constraints
- Ralf Lämmel and Simon Peyton Jones. 2005. Scrap Your Boilerplate with Class: Extensible Generic Functions

Related Work

- Ralf Hinze and Simon Peyton Jones. 2000. Derivable Type Classes

- Valery Trifonov. 2003. Simulating Quantified Class Constraints
- Ralf Lämmel and Simon Peyton Jones. 2005. Scrap Your Boilerplate with Class: Extensible Generic Functions

- Tom Schrijvers, Bruno C. d. S. Oliveira, and Philip Wadler. 2017. Cochis: Deterministic and Coherent Implicits