Proof Relevant Corecursive Resolution

Peng Fu, Ekaterina Komentdantskaya, Tom Schrijvers, Andrew Pond

Type Class Inference

```
data OddList a   =  OCons a (EvenList a)
data EvenList a  =  Nil | ECons a (OddList a)

class Eq x where
  eq :: x -> x -> Bool

instance (Eq a, Eq (EvenList a)) =>  Eq (OddList a) where
    eq (OCons x xs) (OCons y ys) =  eq x y && eq xs ys
    eq  _ _ =  False

instance (Eq a, Eq (OddList a)) => Eq (EvenList a) where
    eq Nil Nil =  True
    eq (ECons x xs) (ECons y ys)   =  eq x y && eq xs ys
    eq  _ _ =  False

test :: Eq (EvenList Int) => Bool
test = eq (ECons 1 (OCons 2 Nil)) (ECons 1 (OCons 2 Nil))
```

Type Class Inference

```
data OddList a   =  OCons a (EvenList a)
data EvenList a  =  Nil | ECons a (OddList a)

data Eq x = EqD {eq :: x -> x -> Bool}

kOdd :: Eq a -> Eq (EvenList a) ->  Eq (OddList a)
kOdd d1 d2 = EqD q
 where q (OCons x xs) (OCons y ys) = eq d1 x y && eq d2 xs ys
       q  _ _ =  False

kEven :: Eq a -> Eq (OddList a) -> Eq (EvenList a)
kEven d1 d2 = EqD q
 where q Nil Nil =  True
       q (ECons x xs) (ECons y ys) = eq d1 x y && eq d2 xs ys
       q  _ _ =  False

test :: Eq (EvenList Int) -> Bool
test d = eq d (ECons 1 (OCons 2 Nil)) (ECons 1 (OCons 2 Nil))
```

How to obtain the evidence `d` for `Eq (EvenList Int)`?

Cycling nontermination

Consider the following logic program Φ

$$\kappa_{Odd} : Eq\ x, Eq\ (EvenList\ x) \Rightarrow Eq\ (OddList\ x)$$

$$\kappa_{Even} : Eq\ x, Eq\ (OddList\ x) \Rightarrow Eq\ (EvenList\ x)$$

$$\kappa_{Int} : Eq\ Int$$

- For Query $Eq\ (EvenList\ Int)$:

$\Phi \vdash \underline{Eq\ (EvenList\ Int)} \rightarrow_{\kappa_{Even}} Eq\ Int, Eq\ (OddList\ Int) \rightarrow_{\kappa_{Int}}$
$Eq\ (OddList\ Int) \rightarrow_{\kappa_{Odd}} Eq\ Int, Eq\ (EvenList\ Int) \rightarrow_{\kappa_{Int}}$
$\underline{Eq\ (EvenList\ Int)} \rightarrow \ldots$

Cycling nontermination

Consider the following logic program Φ

$$\kappa_{Odd} : Eq\ x, Eq\ (EvenList\ x) \Rightarrow Eq\ (OddList\ x)$$

$$\kappa_{Even} : Eq\ x, Eq\ (OddList\ x) \Rightarrow Eq\ (EvenList\ x)$$

$$\kappa_{Int} : Eq\ Int$$

- For Query $Eq\ (EvenList\ Int)$:

  $\Phi \vdash \underline{Eq\ (EvenList\ Int)} \rightarrow_{\kappa_{Even}} Eq\ Int, Eq\ (OddList\ Int) \rightarrow_{\kappa_{Int}}$
  $Eq\ (OddList\ Int) \rightarrow_{\kappa_{Odd}} Eq\ Int, Eq\ (EvenList\ Int) \rightarrow_{\kappa_{Int}}$
  $\underline{Eq\ (EvenList\ Int)} \rightarrow ...$
- So what is the d such that $\Phi \vdash d : Eq\ (EvenList\ Int)$?

Typing Rule for Fixpoint

$$\frac{\Phi, \alpha : T \vdash e : T}{\Phi \vdash \mu\alpha.e : T}$$

- We can view $\mu\alpha.e$ as $\alpha = e$, where $\alpha \in \mathrm{FV}(e)$

Typing Rule for Fixpoint

$$\frac{\Phi, \alpha : T \vdash e : T}{\Phi \vdash \mu\alpha.e : T}$$

- We can view $\mu\alpha.e$ as $\alpha = e$, where $\alpha \in \mathrm{FV}(e)$
- Operational meaning: $\mu\alpha.e \rightsquigarrow [\mu\alpha.e/\alpha]e$

Typing Rule for Fixpoint

$$\frac{\Phi, \alpha : T \vdash e : T}{\Phi \vdash \mu\alpha.e : T}$$

- We can view $\mu\alpha.e$ as $\alpha = e$, where $\alpha \in \mathrm{FV}(e)$
- Operational meaning: $\mu\alpha.e \rightsquigarrow [\mu\alpha.e/\alpha]e$
- The typing derivation for $\Phi \vdash d : \mathit{Eq}(\mathit{EvenList}\ \mathit{Int})$:

$$\frac{\dfrac{\cdots}{\Phi, \alpha : \mathit{Eq}(\mathit{EvenList}\ \mathit{Int}) \vdash \kappa_{\mathit{Even}}\ \kappa_{\mathit{Int}}\ (\kappa_{\mathit{Odd}} \kappa_{\mathit{Int}}\ \alpha) : \mathit{Eq}(\mathit{EvenList}\ \mathit{Int})}}{\Phi \vdash \mu\alpha.\kappa_{\mathit{Even}}\ \kappa_{\mathit{Int}}\ (\kappa_{\mathit{Odd}} \kappa_{\mathit{Int}}\ \alpha) : \mathit{Eq}(\mathit{EvenList}\ \mathit{Int})}$$

Type Class Inference

```
data OddList a    =  OCons a (EvenList a)
data EvenList a   =  Nil | ECons a (OddList a)
data Eq x = EqD {eq :: x -> x -> Bool}

kOdd :: Eq a -> Eq (EvenList a)) ->  Eq (OddList a)
kOdd d1 d2 = EqD q
 where q (OCons x xs) (OCons y ys) = eq d1 x y && eq d2 xs ys
       q  _ _ =  False

kEven :: Eq a -> Eq (OddList a)) -> Eq (EvenList a)
kEven d1 d2 = EqD q
 where q Nil Nil =  True
       q (ECons x xs) (ECons y ys) = eq d1 x y && eq d2 xs ys
       q  _ _ =  False

test :: Eq (EvenList Int) -> Bool
test d = eq d (ECons 1 (OCons 2 Nil)) (ECons 1 (OCons 2 Nil))

h :: Eq (EvenList Int)
h = kEven kInt (kOdd kInt h)

{- eval: test h ==> True -}
```

What about looping nontermination?

Example from *Haskell Programming with Nested Types* by P. Johann and N. Ghani

```
data Mu h a = In (h (Mu h) a)

data HPTree f a = HPLeaf a | HPNode (f (a, a))

type PTree a = Mu HPTree a

instance Eq (h (Mu h) a) => Eq (Mu h a) where
  eq (In x) (In y) = eq x y

instance (Eq a, Eq (f (a, a))) => Eq (HPTree f a) where
  eq (HPLeaf x) (HPLeaf y) = eq x y
  eq (HPNode xs) (HPNode ys) = eq xs ys
  eq _ _ = False

tree :: Mu HPTree Int
tree = In (HPLeaf 42)

test :: Eq (Mu HPTree Int) => Bool
test = eq tree tree
```

Looping Notermination

The corresponding logic program Φ:

$$\kappa_{Mu} : Eq(h\ (Mu\ h)\ a) \Rightarrow Eq(Mu\ h\ a)$$
$$\kappa_{HPTree} : (Eq\ a, Eq(f\ (a,a))) \Rightarrow Eq(HPTree\ f\ a)$$
$$\kappa_{Pair} : (Eq\ x, Eq\ y) \Rightarrow Eq(x,y)$$
$$\kappa_{Int} : Eq\ Int$$

- For query $Eq\ (Mu\ HPTree\ Int)$:

$$\Phi \vdash \underline{Eq(Mu\ HPTree\ Int)} \rightarrow_{\kappa_{Mu}} Eq(HPTree\ (Mu\ HPTree)\ Int) \rightarrow_{\kappa_{HPTree}}$$
$$Eq\ Int, Eq\ (Mu\ HPTree)\ (Int,Int) \rightarrow_{\kappa_{Int}} \underline{Eq\ (Mu\ HPTree\ (Int,Int))} \rightarrow_{\kappa_{Mu}}$$
$$Eq(HPTree\ (Mu\ HPTree)\ (Int,Int)) \rightarrow_{\kappa_{HPTree}}$$
$$Eq\ (Int,Int), Eq\ (Mu\ HPTree)\ ((Int,Int),(Int,Int)) \rightarrow_{\kappa_{Pair},\kappa_{Int},\kappa_{Int}}$$
$$\underline{Eq\ (Mu\ HPTree\ ((Int,Int),(Int,Int)))} \rightarrow \ldots$$

Looping Notermination

The corresponding logic program Φ:

$$\kappa_{Mu} : Eq(h\ (Mu\ h)\ a) \Rightarrow Eq(Mu\ h\ a)$$
$$\kappa_{HPTree} : (Eq\ a, Eq(f\ (a,a))) \Rightarrow Eq(HPTree\ f\ a)$$
$$\kappa_{Pair} : (Eq\ x, Eq\ y) \Rightarrow Eq(x,y)$$
$$\kappa_{Int} : Eq\ Int$$

- For query $Eq\ (Mu\ HPTree\ Int)$:

$$\Phi \vdash \underline{Eq(Mu\ HPTree\ Int)} \rightarrow_{\kappa_{Mu}} Eq(HPTree\ (Mu\ HPTree)\ Int) \rightarrow_{\kappa_{HPTree}}$$
$$Eq\ Int, Eq\ (Mu\ HPTree)\ (Int, Int) \rightarrow_{\kappa_{Int}} \underline{Eq\ (Mu\ HPTree\ (Int, Int))} \rightarrow_{\kappa_{Mu}}$$
$$Eq(HPTree\ (Mu\ HPTree)\ (Int, Int)) \rightarrow_{\kappa_{HPTree}}$$
$$Eq\ (Int, Int), Eq\ (Mu\ HPTree)\ ((Int, Int), (Int, Int)) \rightarrow_{\kappa_{Pair}, \kappa_{Int}, \kappa_{Int}}$$
$$\underline{Eq\ (Mu\ HPTree\ ((Int, Int), (Int, Int)))} \rightarrow \ldots$$

- What is the d such that $\Phi \vdash d : Eq\ (Mu\ HPTree\ Int)$?

A Theorem-proving perspective

Assume we have axioms Φ:

$$\begin{aligned}
\kappa_{Mu} &: Eq(h\ (Mu\ h)\ a) \Rightarrow Eq(Mu\ h\ a) \\
\kappa_{HPTree} &: (Eq\ a, Eq(f\ (a,a))) \Rightarrow Eq(HPTree\ f\ a) \\
\kappa_{Pair} &: (Eq\ x, Eq\ y) \Rightarrow Eq(x,y) \\
\kappa_{Int} &: Eq\ Int
\end{aligned}$$

- Directly prove $Eq\ (Mu\ HPTree\ Int)$ seems impossible

A Theorem-proving perspective

Assume we have axioms Φ:

$$\begin{aligned}\kappa_{Mu} &: Eq(h\ (Mu\ h)\ a) \Rightarrow Eq(Mu\ h\ a)\\ \kappa_{HPTree} &: (Eq\ a, Eq(f\ (a,a))) \Rightarrow Eq(HPTree\ f\ a)\\ \kappa_{Pair} &: (Eq\ x, Eq\ y) \Rightarrow Eq(x,y)\\ \kappa_{Int} &: Eq\ Int\end{aligned}$$

- Directly prove $Eq\ (Mu\ HPTree\ Int)$ seems impossible
- Prove a lemma $e : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$ instead

A Theorem-proving perspective

Assume we have axioms Φ:

$$\kappa_{Mu} : Eq(h\ (Mu\ h)\ a) \Rightarrow Eq(Mu\ h\ a)$$
$$\kappa_{HPTree} : (Eq\ a, Eq(f\ (a,a))) \Rightarrow Eq(HPTree\ f\ a)$$
$$\kappa_{Pair} : (Eq\ x, Eq\ y) \Rightarrow Eq(x,y)$$
$$\kappa_{Int} : Eq\ Int$$

- Directly prove $Eq\ (Mu\ HPTree\ Int)$ seems impossible
- Prove a lemma $e : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$ instead
- $(e\ \kappa_{Int}) : Eq\ (Mu\ HPTree\ Int)$

A Theorem-proving perspective

$$\kappa_{Mu} : Eq(h\ (Mu\ h)\ a) \Rightarrow Eq(Mu\ h\ a)$$
$$\kappa_{HPTree} : (Eq\ a, Eq(f\ (a,a))) \Rightarrow Eq(HPTree\ f\ a)$$
$$\kappa_{Pair} : (Eq\ x, Eq\ y) \Rightarrow Eq(x,y)$$
$$\kappa_{Int} : Eq\ Int$$

Derive $e : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$ using fixpoint typing rule

1. Assumption $\alpha : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$

A Theorem-proving perspective

$$\kappa_{Mu} : Eq(h\ (Mu\ h)\ a) \Rightarrow Eq(Mu\ h\ a)$$
$$\kappa_{HPTree} : (Eq\ a, Eq(f\ (a,a))) \Rightarrow Eq(HPTree\ f\ a)$$
$$\kappa_{Pair} : (Eq\ x, Eq\ y) \Rightarrow Eq(x,y)$$
$$\kappa_{Int} : Eq\ Int$$

Derive $e : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$ using fixpoint typing rule

1. Assumption $\alpha : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$
2. Assume $\alpha_1 : Eq\ x$, to show $Eq\ (Mu\ HPTree\ x)$

A Theorem-proving perspective

$$\kappa_{Mu} : Eq(h\ (Mu\ h)\ a) \Rightarrow Eq(Mu\ h\ a)$$
$$\kappa_{HPTree} : (Eq\ a, Eq(f\ (a,a))) \Rightarrow Eq(HPTree\ f\ a)$$
$$\kappa_{Pair} : (Eq\ x, Eq\ y) \Rightarrow Eq(x,y)$$
$$\kappa_{Int} : Eq\ Int$$

Derive $e : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$ using fixpoint typing rule

1. Assumption $\alpha : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$
2. Assume $\alpha_1 : Eq\ x$, to show $Eq\ (Mu\ HPTree\ x)$
3. Apply κ_{Mu}, we get a new goal $Eq(HPTree\ (Mu\ HPTree)\ x)$

A Theorem-proving perspective

$$\kappa_{Mu} : Eq(h\ (Mu\ h)\ a) \Rightarrow Eq(Mu\ h\ a)$$
$$\kappa_{HPTree} : (Eq\ a, Eq(f\ (a,a))) \Rightarrow Eq(HPTree\ f\ a)$$
$$\kappa_{Pair} : (Eq\ x, Eq\ y) \Rightarrow Eq(x,y)$$
$$\kappa_{Int} : Eq\ Int$$

Derive $e : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$ using fixpoint typing rule

1. Assumption $\alpha : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$
2. Assume $\alpha_1 : Eq\ x$, to show $Eq\ (Mu\ HPTree\ x)$
3. Apply κ_{Mu}, we get a new goal $Eq(HPTree\ (Mu\ HPTree)\ x)$
4. Apply κ_{HPTree}, we get $Eq\ x, Eq\ (Mu\ HPTree\ (x,x))$

A Theorem-proving perspective

$$\kappa_{Mu} : Eq(h\ (Mu\ h)\ a) \Rightarrow Eq(Mu\ h\ a)$$
$$\kappa_{HPTree} : (Eq\ a, Eq(f\ (a,a))) \Rightarrow Eq(HPTree\ f\ a)$$
$$\kappa_{Pair} : (Eq\ x, Eq\ y) \Rightarrow Eq(x,y)$$
$$\kappa_{Int} : Eq\ Int$$

Derive $e : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$ using fixpoint typing rule

1. Assumption $\alpha : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$
2. Assume $\alpha_1 : Eq\ x$, to show $Eq\ (Mu\ HPTree\ x)$
3. Apply κ_{Mu}, we get a new goal $Eq(HPTree\ (Mu\ HPTree)\ x)$
4. Apply κ_{HPTree}, we get $Eq\ x, Eq\ (Mu\ HPTree\ (x,x))$
5. $Eq\ x$ is proved by α_1

A Theorem-proving perspective

$$\kappa_{Mu} : Eq(h\ (Mu\ h)\ a) \Rightarrow Eq(Mu\ h\ a)$$
$$\kappa_{HPTree} : (Eq\ a, Eq(f\ (a,a))) \Rightarrow Eq(HPTree\ f\ a)$$
$$\kappa_{Pair} : (Eq\ x, Eq\ y) \Rightarrow Eq(x,y)$$
$$\kappa_{Int} : Eq\ Int$$

Derive $e : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$ using fixpoint typing rule

1. Assumption $\alpha : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$
2. Assume $\alpha_1 : Eq\ x$, to show $Eq\ (Mu\ HPTree\ x)$
3. Apply κ_{Mu}, we get a new goal $Eq(HPTree\ (Mu\ HPTree)\ x)$
4. Apply κ_{HPTree}, we get $Eq\ x, Eq\ (Mu\ HPTree\ (x,x))$
5. $Eq\ x$ is proved by α_1
6. Apply α on $Eq\ (Mu\ HPTree\ (x,x))$, get $Eq\ (x,x)$

A Theorem-proving perspective

$$\kappa_{Mu} : Eq(h\ (Mu\ h)\ a) \Rightarrow Eq(Mu\ h\ a)$$
$$\kappa_{HPTree} : (Eq\ a, Eq(f\ (a,a))) \Rightarrow Eq(HPTree\ f\ a)$$
$$\kappa_{Pair} : (Eq\ x, Eq\ y) \Rightarrow Eq(x,y)$$
$$\kappa_{Int} : Eq\ Int$$

Derive $e : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$ using fixpoint typing rule

1. Assumption $\alpha : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$
2. Assume $\alpha_1 : Eq\ x$, to show $Eq\ (Mu\ HPTree\ x)$
3. Apply κ_{Mu}, we get a new goal $Eq(HPTree\ (Mu\ HPTree)\ x)$
4. Apply κ_{HPTree}, we get $Eq\ x, Eq\ (Mu\ HPTree\ (x,x))$
5. $Eq\ x$ is proved by α_1
6. Apply α on $Eq\ (Mu\ HPTree\ (x,x))$, get $Eq\ (x,x)$
7. Apply κ_{Pair}, α_1 on $Eq\ (x,x)$, Q.E.D.
   $\mu\alpha.\lambda\alpha_1.\kappa_{Mu}\ (\kappa_{HPTree}\ \alpha_1\ (\alpha\ (\kappa_{Pair}\ \alpha_1\ \alpha_1))) : Eq\ x \Rightarrow Eq\ (Mu\ HPTree\ x)$

Looping Notermination

```
data Mu h a = In (h (Mu h) a)
data HPTree f a = HPLeaf a | HPNode (f (a, a))
type PTree a = (Mu HPTree) a
kMu :: Eq (h (Mu h) a) -> Eq (Mu h a)
kMu d = EqD q
 where q (In x) (In y) = eq d x y
kHPTree :: Eq a -> Eq (f (a, a))) -> Eq (HPTree f a)
kHPTree d1 d2 = EqD q
 where  q (HPLeaf x) (HPLeaf y) = eq d1 x y
        q (HPNode xs) (HPNode ys) = eq d2 xs ys
        q _ _ = False
tree :: (Mu HPTree) Int
tree = In (HPLeaf 42)
test :: Eq (Mu HPTree Int) -> Bool
test d = eq d tree tree

h :: Eq x -> Eq (Mu HPTree x)
h x = kMu (kHPTree x (h (kPair x x)))

g :: Eq (Mu HPTree Int)
g = h kInt
```

Summary

- Corecursive Resolution = Resolution + Mu + Lambda
- Discover lemma heuristically
- Construct dictionary automatically
- Please see the paper *Proof Relevant Corecursive Resolution* for more details. Thank you!