Nontermination in Type Class Inference

Fu-Peng

University of Dundee
School of Computing

Introduction: Type Class Inference

Informally speaking

- Type Class Inference $=$
  Hindley-Milner $+$ Instance Resolution

Introduction: Type Class Inference

Informally speaking

- Type Class Inference $=$
  Hindley-Milner $+$ Instance Resolution
- Instance Resolution $=$
  Context Reduction $+$ Evidence Construction

Introduction: Example

```
class Eq a where
  eq :: Eq a => a -> a -> Bool

instance Eq a, Eq b => Eq (a, b) where
  eq (x1, y1) (x2, y2) = and (eq x1 x2) (eq y1 y2)

instance => Eq Char where
  eq = primtiveCharEq

-- test :: Eq (Char, Char) => Bool
test = eq ('a', 'b') ('c', 'd')
```

Introduction: One possible translation

```
data Eq a where
  CEq :: (a -> a -> Bool) -> Eq a

eq :: Eq a -> (a -> a -> Bool)
eq (CEq m) = m

f :: Eq a, Eq b -> Eq (a, b)
f d1 d2 = CEq q
 where q (x1, y1) (x2, y2) =
         and (eq d1 x1 x2) (eq d2 y1 y2)

g :: Eq Char
g = CEq primtiveCharEq

test = eq d ('a', 'b') ('c', 'd')
-- some d :: Eq (Char, Char)
```

Introduction: Context Reduction

- Given

  ```
  f :: Eq a, Eq b -> Eq (a, b)
  g :: Eq Char
  ```

  How to *automatically* construct

  `d ::  Eq (Char, Char)` ?

Introduction: Context Reduction

- Given

  ```
  f :: Eq a, Eq b -> Eq (a, b)
  g :: Eq Char
  ```

  How to *automatically* construct

  `d ::  Eq (Char, Char)` ?

- Rewrite Rules Φ on Multiset:

  {...`Eq (a, b)`...} $\rightarrow_f$ {...`Eq a`, `Eq b`...}

  {...`Eq Char`...} $\rightarrow_g$ {...}

Introduction: Context Reduction

- Given

  ```
  f :: Eq a, Eq b -> Eq (a, b)
  g :: Eq Char
  ```

  How to *automatically* construct

  `d ::  Eq (Char, Char)` ?

- Rewrite Rules Φ on Multiset:

  $\{\ldots\texttt{Eq (a, b)}\ldots\} \rightarrow_f \{\ldots\texttt{Eq a}, \texttt{Eq b}\ldots\}$

  $\{\ldots\texttt{Eq Char}\ldots\} \rightarrow_g \{\ldots\}$

- Context reduction:

  $\Phi \vdash \{\texttt{Eq (Char, Char)}\} \rightarrow_f \{\texttt{Eq Char}, \texttt{Eq Char}\} \rightarrow_g$
  $\{\texttt{Eq Char}\} \rightarrow_g \emptyset$

Introduction: Context Reduction

- Given

  ```
  f :: Eq a, Eq b -> Eq (a, b)
  g :: Eq Char
  ```

  How to *automatically* construct

  `d ::  Eq (Char, Char)` ?

- Rewrite Rules Φ on Multiset:

  $\{...\texttt{Eq (a, b)}...\} \rightarrow_f \{...\texttt{Eq a}, \texttt{Eq b}...\}$
  $\{...\texttt{Eq Char}...\} \rightarrow_g \{...\}$

- Context reduction:

  $\Phi \vdash \{\texttt{Eq (Char, Char)}\} \rightarrow_f \{\texttt{Eq Char}, \texttt{Eq Char}\} \rightarrow_g \{\texttt{Eq Char}\} \rightarrow_g \emptyset$

- So `d = f g g`

Context Reduction

$\Phi \vdash \{\texttt{Eq (Char, Char)}\} \rightarrow_f \{\texttt{Eq Char}, \texttt{Eq Char}\} \rightarrow_g$
$\{\texttt{Eq Char}\} \rightarrow_g \emptyset$
Thus `d = f g g`

- Evidence construction seems to rely on termination of context reduction
- What happen if we have a non-terminating reduction?

Context Reduction: Nontermination

- Example[1]:

```
instance Data SizeD t => Size t where ...
instance Sat (c Char) => Data c Char where ...
instance Size t => Sat (SizeD t) where ...
```

- Corresponding rules(Φ):

$\{...\texttt{Size t}...\} \rightarrow_a \{...\texttt{Data SizeD t}...\}$
$\{...\texttt{Data c Char}...\} \rightarrow_b \{...\texttt{Sat (c Char)}...\}$
$\{...\texttt{Sat (SizeD t)}...\} \rightarrow_c \{...\texttt{Size t}...\}$

[1] from R. Lämmel & S.P. Jones's Scrap your boilerplate with class

Context Reduction: Nontermination

- Example[1]:

```
instance Data SizeD t => Size t where ...
instance Sat (c Char) => Data c Char where ...
instance Size t => Sat (SizeD t) where ...
```

- Corresponding rules(Φ):

  $\{\ldots\texttt{Size t}\ldots\} \rightarrow_a \{\ldots\texttt{Data SizeD t}\ldots\}$
  $\{\ldots\texttt{Data c Char}\ldots\} \rightarrow_b \{\ldots\texttt{Sat (c Char)}\ldots\}$
  $\{\ldots\texttt{Sat (SizeD t)}\ldots\} \rightarrow_c \{\ldots\texttt{Size t}\ldots\}$

- How to construct an evidence `d :: Data SizeD Char`?

[1] from R. Lämmel & S.P. Jones's Scrap your boilerplate with class

Context Reduction: Nontermination

- Example[1]:

```
instance Data SizeD t => Size t where ...
instance Sat (c Char) => Data c Char where ...
instance Size t => Sat (SizeD t) where ...
```

- Corresponding rules(Φ):

  $\{\ldots\texttt{Size t}\ldots\} \rightarrow_a \{\ldots\texttt{Data SizeD t}\ldots\}$
  $\{\ldots\texttt{Data c Char}\ldots\} \rightarrow_b \{\ldots\texttt{Sat (c Char)}\ldots\}$
  $\{\ldots\texttt{Sat (SizeD t)}\ldots\} \rightarrow_c \{\ldots\texttt{Size t}\ldots\}$

- How to construct an evidence
  `d :: Data SizeD Char`?

- Let's try to reduce `Data SizeD Char`:

  $\Phi \vdash \{\texttt{Data SizeD Char}\} \rightarrow_b$
  $\{\texttt{Sat (SizeD Char)}\} \rightarrow_c \{\texttt{Size Char}\} \rightarrow_a$
  $\{\texttt{Data SizeD Char}\} \rightarrow_b \cdot \rightarrow_c \cdot \rightarrow_a \ldots$

[1] from R. Lämmel & S.P. Jones's Scrap your boilerplate with class

Nontermination

- What can we do when context reduction diverge?

Nontermination

- What can we do when context reduction diverge?
- Cycle detection(tie the knot)
  {`Data SizeD Char`} $\to_b$ {`Sat (SizeD Char)`} $\to_c$
  {`Size Char`} $\to_a$ {`Data SizeD Char`} $\to_b \cdot \to_c \cdot \to_a \ldots$
- Given:

```
a :: Data SizeD t -> Size t
b :: Sat (c Char) -> Data c Char
c :: Size t -> Sat (SizeD t)
```

  We have:

```
d :: Data SizeD Char
d = b (c (a d))
```

Nontermination

What if the context reduction is diverging without forming any cycle?

```
data Nested a where
 Nil :: Nested a
 Cons :: a -> Nested [a] -> Nested a

instance Eq a, Eq (Nested [a]) => Eq (Nested a) where
  eq Nil Nil = True
  eq (Cons a as) (Cons b bs) = eq a b && eq as bs
```

{...`Eq (Nested a)`...} → {...`Eq a, Eq (Nested [a])`...}
{`Eq (Nested Char)`} →
{`Eq Char,Eq (Nested [Char])`} →
{`Eq Char,Eq [Char],Eq (Nested [[Char]])`}...

Nontermination

What if the context reduction is diverging without forming any cycle?

- Context reduction seems too *earger*.
- How to have *lazy* context reduction?
- A change of perspective:
  Rewriting on multiset:
  $\{...\texttt{Eq (a, b)}...\} \rightarrow_f \{...\texttt{Eq a}, \texttt{Eq b}...\}$
  $\{...\texttt{Eq Char}...\} \rightarrow_g \{...\}$
  Rewriting on first order term:
  $\texttt{Eq (a, b)} \rightarrow f\ (\texttt{Eq a})\ (\texttt{Eq b})$
  $\texttt{Eq Char} \rightarrow g$
- Constructing `d :: Eq (Char, Char)`
  $\texttt{Eq (Char, Char)} \rightarrow f\ (\texttt{Eq Char})\ (\texttt{Eq Char}) \rightarrow$
  $f\ g\ (\texttt{Eq Char}) \rightarrow f\ g\ g$

Summary and Further Works

- Evidence construction process is a kind of rewriting process
- Knowing termination behavior in advance, we may be able to rewrite eagerly/lazily
- **Next Step** Extend the current cycle detection techniques to obtain evidence(statically) for non-obvious "looping" example
- **Long Term Goal** Explore the connection between the evidence that gives rise to infinite rewrite process and the notion of productivity in Katya's Structural Resolution