# Unification in a context of postponed equations

Jesper Cockx

DistriNet – KU Leuven

4 June 2015

# Postponed equations cause problems

- Issue 292: Heterogenous equality is crippled by the Bool $\neq$ Fin 2 fix
- Issue 1071: Regression in unifier, possibly related to modules and/or heterogeneous constraints
- Issue 1406: Injectivity of type constructors is partially back. Agda refutes excluded middle
- Issue 1408: Heterogeneous equality incompatible with univalence even –without-K
- Issue 1411: Order of patterns matters for checking left hand sides
- Issue 1427: Circumvention of forcing analysis brings back easy proof of Fin injectivity
- Issue 1435: Dependent pattern matching is broken

# The underlying problem

Current representation of heterogeneous equations lacks information:

Morally different equations have same representation.

I propose a better representation.

# Advantages of new representation

- Handles previous issues in a uniform way
- Also accepts some new examples, especially when –without-K is enabled
- Theoretically appealing
  $\Rightarrow$ possibility for correctness proof

# Unification in a context of postponed equations

1. Why do we need unification?
2. A context of postponed equations
3. Reverse unification rules

# Unification in a context of postponed equations

1. Why do we need unification?
2. A context of postponed equations
3. Reverse unification rules

# Dependent pattern matching

$$\textbf{data}\ \_\leq\_ : \mathbb{N} \to \mathbb{N} \to \texttt{Set}\ \textbf{where}$$
$$\quad \texttt{lz} : (n : \mathbb{N}) \to \texttt{z} \leq n$$
$$\quad \texttt{ls} : (m\ n : \mathbb{N}) \to m \leq n \to \texttt{s}\ m \leq \texttt{s}\ n$$

$$\texttt{antisym} : (x\ y : \mathbb{N}) \to x \leq y \to y \leq x \to x \equiv y$$
$$\texttt{antisym}\quad x \quad y \quad p \quad q \quad = \quad ?$$

- lz: $\begin{matrix} x \equiv_{\mathbb{N}} \texttt{z}, \\ y \equiv\ \ n \end{matrix} \xrightarrow{x:=\texttt{z}} y \equiv_{\mathbb{N}} n \xrightarrow{y:=n} ()$
- ls: $\begin{matrix} x \equiv_{\mathbb{N}} \texttt{s}\ m, \\ y \equiv_{\mathbb{N}} \texttt{s}\ n \end{matrix} \xrightarrow{x:=\texttt{s}\ m} y \equiv_{\mathbb{N}} \texttt{s}\ n \xrightarrow{y:=\texttt{s}\ n} ()$

# Dependent pattern matching

$$\textbf{data}\ \_ \leq \_ : \mathbb{N} \to \mathbb{N} \to \texttt{Set}\ \textbf{where}$$
$$\quad \texttt{lz} : (n : \mathbb{N}) \to \texttt{z} \leq n$$
$$\quad \texttt{ls} : (m\ n : \mathbb{N}) \to m \leq n \to \texttt{s}\ m \leq \texttt{s}\ n$$

$$\texttt{antisym} : (x\ y : \mathbb{N}) \to x \leq y \to y \leq x \to x \equiv y$$
$$\texttt{antisym}\quad x \quad y \quad p \quad q \quad = \quad ?$$

- $\texttt{lz}$: $\begin{matrix} x \equiv_{\mathbb{N}} \texttt{z}, \\ y \equiv \ n \end{matrix} \xrightarrow{x:=\texttt{z}} y \equiv_{\mathbb{N}} n \xrightarrow{y:=n} ()$
- $\texttt{ls}$: $\begin{matrix} x \equiv_{\mathbb{N}} \texttt{s}\ m, \\ y \equiv_{\mathbb{N}} \texttt{s}\ n \end{matrix} \xrightarrow{x:=\texttt{s}\ m} y \equiv_{\mathbb{N}} \texttt{s}\ n \xrightarrow{y:=\texttt{s}\ n} ()$

# Dependent pattern matching

$$\textbf{data}\ \_\leq\_ : \mathbb{N} \to \mathbb{N} \to \mathtt{Set}\ \textbf{where}$$
$$\mathtt{lz} : (n : \mathbb{N}) \to \mathtt{z} \leq n$$
$$\mathtt{ls} : (m\ n : \mathbb{N}) \to m \leq n \to \mathtt{s}\ m \leq \mathtt{s}\ n$$

$$\mathtt{antisym} : (x\ y : \mathbb{N}) \to x \leq y \to y \leq x \to x \equiv y$$
$$\mathtt{antisym}\quad x \quad y \quad p \quad q \quad = \quad ?$$

- $\mathtt{lz}$: $\begin{matrix} x \equiv_{\mathbb{N}} \mathtt{z}, \\ y \equiv n \end{matrix} \xRightarrow{x:=\mathtt{z}} y \equiv_{\mathbb{N}} n \xRightarrow{y:=n} ()$
- $\mathtt{ls}$: $\begin{matrix} x \equiv_{\mathbb{N}} \mathtt{s}\ m, \\ y \equiv_{\mathbb{N}} \mathtt{s}\ n \end{matrix} \xRightarrow{x:=\mathtt{s}\ m} y \equiv_{\mathbb{N}} \mathtt{s}\ n \xRightarrow{y:=\mathtt{s}\ n} ()$

# Dependent pattern matching

**data** $\_ \leq \_ : \mathbb{N} \to \mathbb{N} \to \texttt{Set}$ **where**

$\quad \texttt{lz} : (n : \mathbb{N}) \to \texttt{z} \leq n$

$\quad \texttt{ls} : (m\ n : \mathbb{N}) \to m \leq n \to \texttt{s}\ m \leq \texttt{s}\ n$

$$
\begin{array}{lllll}
\texttt{antisym} : (x\ y : \mathbb{N}) \to x \leq y \to & y \leq x \to x \equiv y \\
\texttt{antisym}\ \lfloor \texttt{z} \rfloor\ \lfloor y \rfloor\ (\texttt{lz}\ y) & q & = & ? \\
\texttt{antisym}\ \lfloor \texttt{s}\ x \rfloor\ \lfloor \texttt{s}\ y \rfloor\ (\texttt{ls}\ x\ y\ p) & q & = & ?
\end{array}
$$

- `lz`: $y \equiv_{\mathbb{N}} \texttt{z},\ \texttt{z} \equiv_{\mathbb{N}} n \xRightarrow{y := \texttt{z}} \texttt{z} \equiv_{\mathbb{N}} n \xRightarrow{n := \texttt{z}} ()$
- `ls`: $y \equiv_{\mathbb{N}} \texttt{s}\ m,\ \texttt{z} \equiv_{\mathbb{N}} \texttt{s}\ n \xRightarrow{y := \texttt{s}\ m} \texttt{z} \equiv_{\mathbb{N}} \texttt{s}\ n \xRightarrow{\text{conflict}} \bot$

# Dependent pattern matching

$$\textbf{data}\ \_ \leq \_ : \mathbb{N} \to \mathbb{N} \to \texttt{Set}\ \textbf{where}$$
$$\texttt{lz} : (n : \mathbb{N}) \to \texttt{z} \leq n$$
$$\texttt{ls} : (m\ n : \mathbb{N}) \to m \leq n \to \texttt{s}\ m \leq \texttt{s}\ n$$

$$\begin{array}{llllll} \texttt{antisym} : (x\ y : \mathbb{N}) \to & x \leq y \to & & y \leq x & \to & x \equiv y \\ \texttt{antisym}\ \lfloor \texttt{z} \rfloor & \lfloor y \rfloor & (\texttt{lz}\ y) & q & = & ? \\ \texttt{antisym}\ \lfloor \texttt{s}\ x \rfloor & \lfloor \texttt{s}\ y \rfloor & (\texttt{ls}\ x\ y\ p) & q & = & ? \end{array}$$

- $\texttt{lz}$: $\begin{array}{l} y \equiv_{\mathbb{N}} \texttt{z}, \\ \texttt{z} \equiv_{\mathbb{N}} n \end{array} \xRightarrow{y:=\texttt{z}} \texttt{z} \equiv_{\mathbb{N}} n \xRightarrow{n:=\texttt{z}} ()$

- $\texttt{ls}$: $\begin{array}{l} y \equiv_{\mathbb{N}} \texttt{s}\ m, \\ \texttt{z} \equiv_{\mathbb{N}} \texttt{s}\ n \end{array} \xRightarrow{y:=\texttt{s}\ m} \texttt{z} \equiv_{\mathbb{N}} \texttt{s}\ n \xRightarrow{\textbf{conflict}} \bot$

# Dependent pattern matching

$$\textbf{data}\ \_\leq\_ : \mathbb{N} \to \mathbb{N} \to \texttt{Set}\ \textbf{where}$$
$$\quad \texttt{lz} : (n : \mathbb{N}) \to \texttt{z} \leq n$$
$$\quad \texttt{ls} : (m\ n : \mathbb{N}) \to m \leq n \to \texttt{s}\ m \leq \texttt{s}\ n$$

$$\begin{array}{llllll}
\texttt{antisym} : (x\ y : \mathbb{N}) \to & x \leq y \to & & y \leq x & \to & x \equiv y \\
\texttt{antisym} & \lfloor \texttt{z} \rfloor & \lfloor \texttt{z} \rfloor\ (\texttt{lz}\ \lfloor \texttt{z} \rfloor) & (\texttt{lz}\ \lfloor \texttt{z} \rfloor) & = & \texttt{refl} \\
\texttt{antisym} & \lfloor \texttt{s}\ x \rfloor & \lfloor \texttt{s}\ y \rfloor\ (\texttt{ls}\ x\ y\ p) & q & = & ?
\end{array}$$

- lz: $\texttt{s}\ y \equiv_{\mathbb{N}} \texttt{z},\ \texttt{s}\ x \equiv_{\mathbb{N}} n \xRightarrow{\text{conflict}} \bot$
- ls: $\texttt{s}\ y \equiv_{\mathbb{N}} \texttt{s}\ m,\ \texttt{s}\ x \equiv_{\mathbb{N}} \texttt{s}\ n \xRightarrow{\text{injectivity}} y \equiv_{\mathbb{N}} m,\ \texttt{s}\ x \equiv_{\mathbb{N}} \texttt{s}\ n$
  $\xRightarrow{m:=y} \texttt{s}\ x \equiv_{\mathbb{N}} \texttt{s}\ n \xRightarrow{\text{injectivity}} x \equiv_{\mathbb{N}} n \xRightarrow{n:=x} ()$

# Dependent pattern matching

$$\textbf{data}\ \_ \leq \_ : \mathbb{N} \to \mathbb{N} \to \texttt{Set}\ \textbf{where}$$
$$\texttt{lz} : (n : \mathbb{N}) \to \texttt{z} \leq n$$
$$\texttt{ls} : (m\ n : \mathbb{N}) \to m \leq n \to \texttt{s}\ m \leq \texttt{s}\ n$$

$$\begin{array}{llllll}
\texttt{antisym} : (x\ y : \mathbb{N}) \to & x \leq y \to & & y \leq x & \to & x \equiv y \\
\texttt{antisym}\ \lfloor \texttt{z} \rfloor & \lfloor \texttt{z} \rfloor & (\texttt{lz}\ \lfloor \texttt{z} \rfloor) & (\texttt{lz}\ \lfloor \texttt{z} \rfloor) & = & \texttt{refl} \\
\texttt{antisym}\ \lfloor \texttt{s}\ x \rfloor & \lfloor \texttt{s}\ y \rfloor & (\texttt{ls}\ x\ y\ p) & q & = & ?
\end{array}$$

- lz: $\begin{array}{l} \texttt{s}\ y \equiv_{\mathbb{N}} \texttt{z}, \\ \texttt{s}\ x \equiv_{\mathbb{N}} n \end{array} \xrightarrow{\textbf{conflict}} \bot$
- ls: $\begin{array}{l} \texttt{s}\ y \equiv_{\mathbb{N}} \texttt{s}\ m, \\ \texttt{s}\ x \equiv_{\mathbb{N}} \texttt{s}\ n \end{array} \xrightarrow{\textbf{injectivity}} \begin{array}{l} y \equiv_{\mathbb{N}} m, \\ \texttt{s}\ x \equiv_{\mathbb{N}} \texttt{s}\ n \end{array}$
  $\xrightarrow{m := y} \texttt{s}\ x \equiv_{\mathbb{N}} \texttt{s}\ n \xrightarrow{\textbf{injectivity}} x \equiv_{\mathbb{N}} n \xrightarrow{n := x} ()$

# Dependent pattern matching

$$
\begin{array}{l}
\textbf{data}\ \_\leq\_ : \mathbb{N} \to \mathbb{N} \to \texttt{Set}\ \textbf{where} \\
\quad \texttt{lz} : (n : \mathbb{N}) \to \texttt{z} \leq n \\
\quad \texttt{ls} : (m\ n : \mathbb{N}) \to m \leq n \to \texttt{s}\ m \leq \texttt{s}\ n
\end{array}
$$

$$
\begin{array}{l}
\texttt{antisym} : (x\ y : \mathbb{N}) \to x \leq y \to y \leq x \to x \equiv y \\
\texttt{antisym}\ \lfloor \texttt{z} \rfloor\ \lfloor \texttt{z} \rfloor\ (\texttt{lz}\ \lfloor \texttt{z} \rfloor)\ (\texttt{lz}\ \lfloor \texttt{z} \rfloor) = \texttt{refl} \\
\texttt{antisym}\ \lfloor \texttt{s}\ x \rfloor\ \lfloor \texttt{s}\ y \rfloor\ (\texttt{ls}\ x\ y\ p)\ (\texttt{ls}\ \lfloor y \rfloor\ \lfloor x \rfloor\ q) \\
\quad = \texttt{cong}\ \texttt{s}\ (\texttt{antisym}\ x\ y\ p\ q)
\end{array}
$$

# Postponed equations

Some equations cannot be solved right away

$$f\ \mathtt{z} \equiv_{\mathbb{N}} \mathtt{s}\ \mathtt{z} \overset{?}{\Rightarrow}$$

but solving later equations can change this

$$\begin{aligned} f\ \mathtt{z} &\equiv_{\mathbb{N}} \mathtt{s}\ \mathtt{z}, \\ f &\equiv_{\mathbb{N}\to\mathbb{N}} \mathtt{s} \end{aligned}$$

$$\xRightarrow{f := \mathtt{s}} \mathtt{s}\ \mathtt{z} \equiv_{\mathbb{N}} \mathtt{s}\ \mathtt{z}$$

$$\xRightarrow{\textbf{injectivity}} \mathtt{z} \equiv_{\mathbb{N}} \mathtt{z}$$

$$\xRightarrow{\textbf{injectivity}} ()$$

# Heterogeneous types

$$\textbf{data}\ \texttt{Box} : A \to \texttt{Set}\ \textbf{where}$$
$$\texttt{box} : (x : A) \to \texttt{Box}\ x$$

Let $s, t : A$, then in

$$s \equiv_A t,$$
$$\texttt{box}\ s\ {}_{\texttt{Box}\ s}{\cong}_{\texttt{Box}\ t}\ \texttt{box}\ t$$

the second equation has a heterogeneous type.

Can we apply unification rules

on heterogeneous equations?

# Heterogeneous types

**data** Bool1 : Set **where**
  true1 : Bool1
  false1 : Bool1

**data** Bool2 : Set **where**
  true2 : Bool2
  false2 : Bool2

$$\begin{array}{lcl} \texttt{Bool1} & \equiv_{\texttt{Set}} & \texttt{Bool2}, \\ \texttt{true1} & {}_{\texttt{Bool1}}\cong_{\texttt{Bool2}} & \texttt{true2} \end{array} \xRightarrow{\textbf{conflict}} \bot\ ?$$

This allows us to prove that $\texttt{Bool1} \not\equiv \texttt{Bool2}$!

# Heterogeneous types

Solution (until now):
types must have the same *shape*

ok: $\texttt{box}\ s\ {}_{\texttt{Box}\ s}{\cong}_{\texttt{Box}\ t}\ \texttt{box}\ t \xRightarrow{\textbf{injectivity}} s \equiv_A t$
(types both have the shape $\texttt{Box}$ ...)

not ok: $\texttt{true1}\ {}_{\texttt{Bool1}}{\cong}_{\texttt{Bool2}}\ \texttt{true2} \xRightarrow{\textbf{conflict}} \bot$
(types are unrelated)

# Unification in a context of postponed equations

# Lack of information in current representation

$$\textbf{data}\ \mathtt{Box} : A \to \mathtt{Set}\ \textbf{where}$$
$$\mathtt{box} : (x : A) \to \mathtt{Box}\ x$$

What's different between second equation of

$$\begin{array}{ccc} x & \equiv_A & y, \\ \mathtt{box}\ x & {}_{\mathtt{Box}\ x}{\cong}_{\mathtt{Box}\ y} & \mathtt{box}\ y \end{array} \quad \text{and} \quad \begin{array}{ccc} \mathtt{Box}\ x & \equiv_{\mathtt{Set}} & \mathtt{Box}\ y, \\ \mathtt{box}\ x & {}_{\mathtt{Box}\ x}{\cong}_{\mathtt{Box}\ y} & \mathtt{box}\ y \end{array} \ ?$$

In current representation, nothing!

# Lack of information in current representation

$$\textbf{data}\ \texttt{Box} : A \to \texttt{Set}\ \textbf{where}$$
$$\quad \texttt{box} : (x : A) \to \texttt{Box}\ x$$

What's different between second equation of

$$\begin{array}{ccc} x & \equiv_A & y, \\ \texttt{box}\ x & {}_{\texttt{Box}\ x}{\cong}_{\texttt{Box}\ y} & \texttt{box}\ y \end{array} \quad \text{and} \quad \begin{array}{ccc} \texttt{Box}\ x & \equiv_{\texttt{Set}} & \texttt{Box}\ y, \\ \texttt{box}\ x & {}_{\texttt{Box}\ x}{\cong}_{\texttt{Box}\ y} & \texttt{box}\ y \end{array} \ ?$$

In current representation, nothing!

# Lack of information in current representation

**data** Box : $A \to$ Set **where**
  box : $(x : A) \to$ Box $x$

$$\text{Box } x \equiv \text{Box } y, \quad \text{box } x \cong \text{box } y \xrightarrow{\text{injectivity}} \text{Box } x \equiv \text{Box } y, \; x \cong y \xrightarrow{y := x} \text{Box } x \equiv \text{Box } x \xrightarrow{\text{deletion}} ()$$

- Ok to apply **injectivity** b/c types are equal
- Types are equal because we can apply **injectivity**

$\Rightarrow$ circular argument!

# Lack of information in current representation

$$\textbf{data}\ \texttt{Box} : A \to \texttt{Set}\ \textbf{where}$$
$$\quad \texttt{box} : (x : A) \to \texttt{Box}\ x$$

$$\texttt{Box}\ x \equiv \texttt{Box}\ y,$$
$$\texttt{box}\ x \cong \texttt{box}\ y$$
$$\xRightarrow{\textbf{injectivity}} \texttt{Box}\ x \equiv \texttt{Box}\ y,$$
$$x \cong y$$
$$\xRightarrow{y := x} \texttt{Box}\ x \equiv \texttt{Box}\ x$$
$$\xRightarrow{\textbf{deletion}} ()$$

- Ok to apply **injectivity** b/c types are equal
- Types are equal because we can apply **injectivity**

$\Rightarrow$ circular argument!

# Lack of information in current representation

$$\textbf{data}\ \texttt{Box} : A \to \texttt{Set}\ \textbf{where}$$
$$\quad \texttt{box} : (x : A) \to \texttt{Box}\ x$$

$$
\begin{aligned}
&\texttt{Box}\ x \equiv \texttt{Box}\ y, \\
&\texttt{box}\ x \cong \texttt{box}\ y \\
\xrightarrow{\textbf{injectivity}}\ &\texttt{Box}\ x \equiv \texttt{Box}\ y, \\
&x \cong y \\
\xrightarrow{y := x}\ &\texttt{Box}\ x \equiv \texttt{Box}\ x \\
\xrightarrow{\textbf{deletion}}\ &()
\end{aligned}
$$

- Ok to apply **injectivity** b/c types are equal
- Types are equal because we can apply **injectivity**

$\Rightarrow$ circular argument!

# Representing postponed equations as fresh variables

**data** $\texttt{Box} : A \to \texttt{Set}$ **where**
  $\texttt{box} : (x : A) \to \texttt{Box}\ x$

What's different between second equation of

$$\begin{array}{l} e_1 : x \equiv_A y, \\ e_2 : \texttt{box}\ x \equiv_{\texttt{Box}\ e_1} \texttt{box}\ y \end{array} \quad \text{and} \quad \begin{array}{l} e_1 : \texttt{Box}\ x \equiv_{\texttt{Set}} \texttt{Box}\ y, \\ e_2 : \texttt{box}\ x \equiv_{e_1} \texttt{box}\ y \end{array} \quad ?$$

It's obvious now!

# Representing postponed equations as fresh variables

$$\textbf{data}\ \texttt{Box} : A \to \texttt{Set}\ \textbf{where}$$
$$\texttt{box} : (x : A) \to \texttt{Box}\ x$$

What's different between second equation of

$$\begin{array}{l} e_1 : x \equiv_A y, \\ e_2 : \texttt{box}\ x \equiv_{\texttt{Box}\ e_1} \texttt{box}\ y \end{array} \quad \text{and} \quad \begin{array}{l} e_1 : \texttt{Box}\ x \equiv_{\texttt{Set}} \texttt{Box}\ y, \\ e_2 : \texttt{box}\ x \equiv_{e_1} \texttt{box}\ y \end{array} \quad ?$$

It's obvious now!

# Unification rules require fully general indices

In order to apply injectivity,

1. the type of the equation should be a datatype
2. the indices should be distinct equation variables

Injectivity solves the index equations as well!

# Examples

$$\begin{array}{l} e_1 : \quad x \equiv_A y, \\ e_2 : \mathtt{box}\ x \equiv_{\mathtt{Box}\ e_1} \mathtt{box}\ y \end{array} \xrightarrow{\textbf{injectivity}} x \equiv_A y \xrightarrow{y := x} ()$$

$$\begin{array}{l} e_1 : \mathtt{Box}\ x \equiv_{\mathtt{Set}} \mathtt{Box}\ y, \\ e_2 : \mathtt{box}\ x \equiv_{e_1} \mathtt{box}\ y \end{array} \not\xrightarrow{\textbf{injectivity}} \text{(not a datatype)}$$

$$e_1 : \mathtt{box}\ x \equiv_{\mathtt{Box}\ x} \mathtt{box}\ x \not\xrightarrow{\textbf{injectivity}} \text{(not an equation var)}$$

Uh oh...

# Examples

$$\begin{array}{l} e_1 : \quad x \equiv_A y, \\ e_2 : \mathtt{box}\ x \equiv_{\mathtt{Box}\ e_1} \mathtt{box}\ y \end{array} \xrightarrow{\textbf{injectivity}} x \equiv_A y \xrightarrow{y := x} ()$$

$$\begin{array}{l} e_1 : \mathtt{Box}\ x \equiv_{\mathtt{Set}} \mathtt{Box}\ y, \\ e_2 : \mathtt{box}\ x \equiv_{e_1} \mathtt{box}\ y \end{array} \not\xrightarrow{\textbf{injectivity}} \text{(not a datatype)}$$

$$e_1 : \mathtt{box}\ x \equiv_{\mathtt{Box}\ x} \mathtt{box}\ x \not\xrightarrow{\textbf{injectivity}} \text{(not an equation var)}$$

Uh oh...

# Examples

$$\begin{array}{l} e_1 : \quad x \equiv_A y, \\ e_2 : \texttt{box}\ x \equiv_{\texttt{Box}\ e_1} \texttt{box}\ y \end{array} \xRightarrow{\textbf{injectivity}} x \equiv_A y \xRightarrow{y := x} ()$$

$$\begin{array}{l} e_1 : \texttt{Box}\ x \equiv_{\texttt{Set}} \texttt{Box}\ y, \\ e_2 : \texttt{box}\ x \equiv_{e_1} \texttt{box}\ y \end{array} \not\xRightarrow{\textbf{injectivity}} \text{(not a datatype)}$$

$$e_1 : \texttt{box}\ x \equiv_{\texttt{Box}\ x} \texttt{box}\ x \not\xRightarrow{\textbf{injectivity}} \text{(not an equation var)}$$

Uh oh...

# Examples

$$\begin{array}{l} e_1 : x \equiv_A y, \\ e_2 : \texttt{box}\ x \equiv_{\texttt{Box}\ e_1} \texttt{box}\ y \end{array} \xRightarrow{\textbf{injectivity}} x \equiv_A y \xRightarrow{y:=x} ()$$

$$\begin{array}{l} e_1 : \texttt{Box}\ x \equiv_{\texttt{Set}} \texttt{Box}\ y, \\ e_2 : \texttt{box}\ x \equiv_{e_1} \texttt{box}\ y \end{array} \not\xRightarrow{\textbf{injectivity}} \text{(not a datatype)}$$

$$e_1 : \texttt{box}\ x \equiv_{\texttt{Box}\ x} \texttt{box}\ x \not\xRightarrow{\textbf{injectivity}} \text{(not an equation var)}$$

Uh oh...

# Examples

$$\begin{array}{l} e_1 : \quad x \equiv_A y, \\ e_2 : \mathtt{box}\ x \equiv_{\mathtt{Box}\ e_1} \mathtt{box}\ y \end{array} \xrightarrow{\textbf{injectivity}} x \equiv_A y \xrightarrow{y:=x} ()$$

$$\begin{array}{l} e_1 : \mathtt{Box}\ x \equiv_{\mathtt{Set}} \mathtt{Box}\ y, \\ e_2 : \mathtt{box}\ x \equiv_{e_1} \mathtt{box}\ y \end{array} \overset{\textbf{injectivity}}{\not\Longrightarrow} \text{(not a datatype)}$$

$$e_1 : \mathtt{box}\ x \equiv_{\mathtt{Box}\ x} \mathtt{box}\ x \overset{\textbf{injectivity}}{\not\Longrightarrow} \text{(not an equation var)}$$

Uh oh...

# Unification in a context of postponed equations

# Reverse solution

When indices are regular variables, we can fix that by introducing a new equation.

$$
\begin{aligned}
& e_1 : \texttt{box}\ x \equiv_{\texttt{Box}\ x} \texttt{box}\ x \\
\xrightarrow{\textbf{solution}^{-1}}\quad & e_1 : x \equiv_A y, \\
& e_2 : \texttt{box}\ x \equiv_{\texttt{Box}\ e_1} \texttt{box}\ y \\
\xrightarrow{\textbf{injectivity}}\quad & e_1 : x \equiv_A y \\
\xrightarrow{y := x}\quad & ()
\end{aligned}
$$

# Reverse injectivity

When indices are constructor forms, we can fix that by gathering the equations together.

$$e_1 : \texttt{box}\ (\texttt{s}\ \texttt{z}) \equiv_{\texttt{Box}\ (\texttt{s}\ \texttt{z})} \texttt{box}\ (\texttt{s}\ \texttt{z})$$

$$\xrightarrow{\textbf{injectivity}^{-1}} \begin{array}{l} e_1 : \texttt{z} \equiv_{\mathbb{N}} \texttt{z}, \\ e_2 : \texttt{box}\ (\texttt{s}\ \texttt{z}) \equiv_{\texttt{Box}\ (\texttt{s}\ e_1)} \texttt{box}\ (\texttt{s}\ \texttt{z}) \end{array}$$

$$\xrightarrow{\textbf{injectivity}^{-1}} \begin{array}{l} e_1 : \texttt{s}\ \texttt{z} \equiv_{\mathbb{N}} \texttt{s}\ \texttt{z}, \\ e_2 : \texttt{box}\ (\texttt{s}\ \texttt{z}) \equiv_{\texttt{Box}\ e_1} \texttt{box}\ (\texttt{s}\ \texttt{z}) \end{array}$$

$$\xrightarrow{\textbf{injectivity}} e_1 : \texttt{s}\ \texttt{z} \equiv_{\mathbb{N}} \texttt{s}\ \texttt{z}$$

$$\xrightarrow{\textbf{injectivity}} e_1 : \texttt{z} \equiv_{\mathbb{N}} \texttt{z}$$

$$\xrightarrow{\textbf{injectivity}} ()$$

# Exodus: implementation

I've tried implementing this in Agda

As usual, the code is *much* uglier than the theory

Or maybe I just haven't found the right abstraction yet...

Any ideas or insights are welcome

Thank you for your attention!

# Exodus: implementation

I've tried implementing this in Agda

As usual, the code is *much* uglier than the theory

Or maybe I just haven't found the right abstraction yet...

Any ideas or insights are welcome

Thank you for your attention!

# Exodus: implementation

I've tried implementing this in Agda

As usual, the code is *much* uglier than the theory

Or maybe I just haven't found the right abstraction yet...

Any ideas or insights are welcome

Thank you for your attention!

# Exodus: implementation

I've tried implementing this in Agda

As usual, the code is *much* uglier than the theory

Or maybe I just haven't found the right abstraction yet...

Any ideas or insights are welcome

Thank you for your attention!

# Exodus: implementation

I've tried implementing this in Agda

As usual, the code is *much* uglier than the theory

Or maybe I just haven't found the right abstraction yet...

Any ideas or insights are welcome

Thank you for your attention!