A sound unification algorithm based on telescope equivalences

Jesper Cockx

DistriNet – KU Leuven

20 April 2016
Pattern matching is awesome

Agda uses unification to:

- check which constructors are possible
- specialize the result type

```agda
data Vec (A : Set) : ℕ → Set where
  [] : Vec A 0
  cons : (n : ℕ) → A → Vec A n → Vec A (1 + n)

f : Vec A 1 → T
f (cons .0 x xs) = ...
```
Pattern matching is awesome

Agda uses unification to:

- check which constructors are possible
- specialize the result type

data Vec (A : Set) : ℕ → Set where

[] : Vec A 0
cons : (n : ℕ) → A → Vec A n → Vec A (1 + n)

f : Vec A 1 → T
f (cons .0 x xs) = ...
Pattern matching is awesome

Agda uses unification to:

- check which constructors are possible
- specialize the result type

```agda
data Vec (A : Set) : ℕ → Set where
  [] : Vec A 0
  cons : (n : ℕ) → A → Vec A n → Vec A (1 + n)

f : Vec A 1 → T
f (cons .0 x xs) = ...
Details of unification are important

Agda has pattern matching as a primitive, so results of unification determine Agda’s notion of equality

Example: deleting reflexive equations implies $K$
Details of unification are important

Agda has pattern matching as a primitive, so results of unification determine Agda’s notion of equality

Example: deleting reflexive equations implies $K$
Time for a quiz

Should the following code be accepted?

```haskell
{-# OPTIONS --without-K #-}
...
-- imports

f :: (Bool, true) <=< (Bool, false) -> ⊥
f ()
```
Time for a quiz

Should the following code be accepted?

```haskell
{-# OPTIONS --without-K #-}
...

f: (Bool, true) ⊔ (Bool, false) → ⊥
f()
```

Answer: depends on the type of the equation!
Postponing equations causes problems

If we postpone an equation, following equations can be heterogeneous

Naively continuing unification is bad

- Equality of second projections
- Injectivity of type constructors
- ...

It’s hard to distinguish good and bad situations!
Postponing equations causes problems

If we postpone an equation, following equations can be heterogeneous

Naively continuing unification is bad

- Equality of second projections
- Injectivity of type constructors
- ...

It’s hard to distinguish good and bad situations!
Postponing equations causes problems

If we postpone an equation, following equations can be heterogeneous

Naively continuing unification is bad

- Equality of second projections
- Injectivity of type constructors
- ...

It’s hard to distinguish good and bad situations!
We need a general way to think about unification

It’s not sufficient to “make things equal”

Core idea:

*Unification rules are equivalences between telescopes of equations*

This is the basis of the new unification algorithm in Agda 2.5.1
We need a general way to think about unification

It’s not sufficient to “make things equal”

Core idea:

Unification rules are equivalences between telescopes of equations

This is the basis of the new unification algorithm in Agda 2.5.1
We need a general way to think about unification

It’s not sufficient to “make things equal”

Core idea:

*Unification rules are equivalences between telescopes of equations*

This is the basis of the new unification algorithm in Agda 2.5.1
A sound unification algorithm based on telescope equivalences

1. Unifiers as equivalences
2. Unification rules
3. Higher-dimensional unification
A sound unification algorithm based on telescope equivalences

1. Unifiers as equivalences

2. Unification rules

3. Higher-dimensional unification
What do we want from unification?

It has to be possible to translate pattern matching to eliminators

The core tool we need is specialization by unification

Build a function $m : \Gamma \rightarrow \bar{u} \equiv_\Delta \bar{v} \rightarrow T$
from a function $m' : \Gamma' \rightarrow T\sigma$
where $\sigma : \Gamma' \rightarrow \Gamma$ is computed by unification
What do we want from unification?

It has to be possible to translate pattern matching to eliminators

The core tool we need is specialization by unification

Build a function $m : \Gamma \rightarrow \bar{u} \equiv_{\Delta} \bar{v} \rightarrow T$ from a function $m' : \Gamma' \rightarrow T\sigma$ where $\sigma : \Gamma' \rightarrow \Gamma$ is computed by unification
Intermezzo: telescopic equality

Type of an equation may depend on solution of previous equations

Heterogeneous equality doesn’t keep enough information:

- Safe to consider equation homogeneous?
- Does equation depend on other equation?
- How do equations depend on each other?
Intermezzo: telescopic equality

Type of an equation may depend on solution of previous equations.

Heterogeneous equality doesn’t keep enough information:

- Safe to consider equation homogeneous?
- Does equation depend on other equation?
- How do equations depend on each other?
Intermezzo: telescopic equality

Solution: use “path over” construction to keep track of dependencies

For example:

\((e_1 : m \equiv_N n)(e_2 : u \equiv^{e_1}_{\text{Vec} A} v)\)

Cubical (abuse of) notation:

\((e_1 : m \equiv_N n)(e_2 : u \equiv_{\text{Vec} A} e_1 v)\)
Intermezzo: telescopic equality

Solution: use “path over” construction to keep track of dependencies

For example:

\[(e_1 : m \equiv_N n)(e_2 : u \equiv_{\text{Vec} A}^{e_1} v)\]

Cubical (abuse of) notation:

\[(e_1 : m \equiv_N n)(e_2 : u \equiv_{\text{Vec} A e_1} v)\]
Specialization by unification

The goal is to construct \( m : \Gamma \rightarrow \bar{u} \equiv_\Delta \bar{v} \rightarrow T \)

Input:
- Telescope \( \Gamma \) of flexible variables
- Telescope \( \bar{u} \equiv_\Delta \bar{v} \) of equations

Output:
- New telescope \( \Gamma' \)
- Substitution \( \sigma : \Gamma' \rightarrow \Gamma \)
- Evidence of unification \( \bar{e} : \Gamma' \rightarrow \bar{u}\sigma \equiv_\Delta \sigma \bar{v}\sigma \)
Specialization by unification

The goal is to construct $m : \Gamma \rightarrow \bar{u} \equiv_{\Delta} \bar{v} \rightarrow T$

Input:

- Telescope $\Gamma$ of flexible variables
- Telescope $\bar{u} \equiv_{\Delta} \bar{v}$ of equations

Output:

- New telescope $\Gamma'$
- Substitution $\sigma : \Gamma' \rightarrow \Gamma$
- Evidence of unification $\bar{e} : \Gamma' \rightarrow \bar{u}\sigma \equiv_{\Delta\sigma} \bar{v}\sigma$
Specialization by unification

The goal is to construct $m : \Gamma \rightarrow \bar{u} \equiv_{\Delta} \bar{v} \rightarrow T$

Input:
- Telescope $\Gamma$ of flexible variables
- Telescope $\bar{u} \equiv_{\Delta} \bar{v}$ of equations

Output:
- New telescope $\Gamma'$
- Substitution $\sigma : \Gamma' \rightarrow \Gamma$
- Evidence of unification $\bar{e} : \Gamma' \rightarrow \bar{u}\sigma \equiv_{\Delta\sigma} \bar{v}\sigma$
Specialization by unification

The goal is to construct $m : \Gamma \rightarrow \bar{u} \equiv_\Delta \bar{v} \rightarrow T$

Input:
- Telescope $\Gamma$ of flexible variables
- Telescope $\bar{u} \equiv_\Delta \bar{v}$ of equations

Output:
- New telescope $\Gamma'$
- Telescope mapping $f : \Gamma' \rightarrow \Gamma(\bar{u} \equiv_\Delta \bar{v})$
Two more requirements

Let $f : \Gamma' \rightarrow \Gamma(\bar{u} \equiv_{\Delta} \bar{v})$ be a unifier

- $f$ should be most general
  $\Rightarrow f$ needs a right inverse $g_1$

- $\Gamma'$ should be minimal
  $\Rightarrow f$ needs a left inverse $g_2$
Two more requirements

Let $f : \Gamma' \rightarrow \Gamma(\tilde{u} \equiv_{\Delta} \tilde{v})$ be a unifier

- $f$ should be most general
  \[ \Rightarrow f \text{ needs a right inverse } g_1 \]

- $\Gamma'$ should be minimal
  \[ \Rightarrow f \text{ needs a left inverse } g_2 \]
Two more requirements

Let $f : \Gamma' \rightarrow \Gamma(\bar{u} \equiv_{\Delta} \bar{v})$ be a unifier

- $f$ should be most general
  $\Rightarrow f$ needs a right inverse $g_1$

- $\Gamma'$ should be minimal
  $\Rightarrow f$ needs a left inverse $g_2$
Most general unifiers as equivalences

A most general unifier of \( \bar{u} \) and \( \bar{v} \) is an equivalence \( f : \Gamma (\bar{u} \equiv_\Delta \bar{v}) \sim \Gamma' \) for some \( \Gamma' \)

Specialization by unification:

\[
m : \Gamma \rightarrow \bar{u} \equiv_\Delta \bar{v} \rightarrow T \\
m \bar{x} \bar{e} = \text{subst} (\lambda \bar{x} \bar{e} . T) (\text{isLinv} \ f \bar{x} \bar{e}) \\
(m' \ (f \bar{x} \bar{e}))
\]
Most general unifiers as equivalences

A most general unifier of $\bar{u}$ and $\bar{v}$ is an equivalence $f : \Gamma(\bar{u} \equiv \Delta \bar{v}) \simeq \Gamma'$ for some $\Gamma'$

Specialization by unification:

$m : \Gamma \rightarrow \bar{u} \equiv \Delta \bar{v} \rightarrow T$

$m \bar{x} \bar{e} = \text{subst} \ (\lambda \bar{x} \bar{e}. T) \ (\text{isLinv} \ f \ \bar{x} \ \bar{e})$

($m' \ (f \ \bar{x} \ \bar{e})$)
Disunifiers

A disunifier of $\bar{u}$ and $\bar{v}$ is an equivalence

$$f : \Gamma(\bar{u} \equiv_\Delta \bar{v}) \simeq \bot$$

Specialization by unification:

$$m : \Gamma \to \bar{u} \equiv_\Delta \bar{v} \to T$$

$$m \bar{x} \bar{e} = \text{elim}_\bot T (f \bar{x} \bar{e})$$
Disunifiers

A disunifier of $\bar{u}$ and $\bar{v}$ is an equivalence

$$f : \Gamma(\bar{u} \equiv_{\Delta} \bar{v}) \simeq \bot$$

Specialization by unification:

$$m : \Gamma \rightarrow \bar{u} \equiv_{\Delta} \bar{v} \rightarrow T$$

$$m \bar{x} \bar{e} = \text{elim}_{\bot} T (f \bar{x} \bar{e})$$
A sound unification algorithm based on telescope equivalences

1. Unifiers as equivalences

2. Unification rules

3. Higher-dimensional unification
Basic unification rules

MGU is constructed by chaining together equivalences given by unification rules

\[(k \ l : \mathbb{N})(e : \text{suc } k \equiv_{\mathbb{N}} \text{suc } l)\]  
\[\simeq (k \ l : \mathbb{N})(e : k \equiv_{\mathbb{N}} l)\]  
\[\simeq (k : \mathbb{N})\]

\[f^{-1} : (k : \mathbb{N}) \to (k \ l : \mathbb{N})(e : \text{suc } k \equiv_{\mathbb{N}} \text{suc } l)\]  
\[f^{-1} \ k = k; k; \text{refl}\]
Basic unification rules

MGU is constructed by chaining together equivalences given by unification rules

\[(k \ l : \mathbb{N})(e : \text{suc } k \equiv_{\mathbb{N}} \text{suc } l)\]
\[\leadsto (k \ l : \mathbb{N})(e : k \equiv_{\mathbb{N}} l)\]
\[\leadsto (k : \mathbb{N})\]

\[f^{-1} : (k : \mathbb{N}) \rightarrow (k \ l : \mathbb{N})(e : \text{suc } k \equiv_{\mathbb{N}} \text{suc } l)\]
\[f^{-1} k = k; k; \text{refl}\]
Basic unification rules

MGU is constructed by chaining together equivalences given by unification rules

\[
(k \mid l : \mathbb{N})(e : \text{suc } k \equiv_{\mathbb{N}} \text{suc } l)
\]

\[
\sim (k \mid l : \mathbb{N})(e : k \equiv_{\mathbb{N}} l)
\]

\[
\sim (k : \mathbb{N})
\]

\[
f^{-1} : (k : \mathbb{N}) \rightarrow (k \mid l : \mathbb{N})(e : \text{suc } k \equiv_{\mathbb{N}} \text{suc } l)
\]

\[
f^{-1} k = k; k; \text{refl}
\]
Basic unification rules

MGU is constructed by chaining together equivalences given by unification rules

\[ (k \mid l : \mathbb{N})(e : \text{suc } k \equiv_{\mathbb{N}} \text{suc } l) \]
\[ \simeq (k \mid l : \mathbb{N})(e : k \equiv_{\mathbb{N}} l) \]
\[ \simeq (k : \mathbb{N}) \]

\[ f^{-1} : (k : \mathbb{N}) \rightarrow (k \mid l : \mathbb{N})(e : \text{suc } k \equiv_{\mathbb{N}} \text{suc } l) \]
\[ f^{-1} k = k; k; \text{refl} \]
Basic unification rules

MGU is constructed by chaining together equivalences given by unification rules

\[(k \mid l : \mathbb{N})(e : \text{suc } k \equiv_{\mathbb{N}} \text{suc } l)\]
\[\sim (k \mid l : \mathbb{N})(e : k \equiv_{\mathbb{N}} l)\]
\[\sim (k : \mathbb{N})\]

\[f^{-1} : (k : \mathbb{N}) \rightarrow (k \mid l : \mathbb{N})(e : \text{suc } k \equiv_{\mathbb{N}} \text{suc } l)\]
\[f^{-1} k = k; k; \text{refl}\]
Basic unification rules

Solution: \((x : A)(e : x \equiv_A t) \sim ()\)

Deletion: \((f x \equiv_N f x) \sim ()\)

Injectivity: \((\text{suc } x \equiv_N \text{suc } y) \sim (x \equiv_N y)\)

Conflict: \((\text{inj}_1 x \equiv_{A \cup B} \text{inj}_2 y) \sim \bot\)

Cycle: \((n \equiv_N \text{suc } n) \sim \bot\)

+ auxiliary rules for weakening and reordering
Rules for $\eta$-equality of records

$\eta$-expansion of a flexible variable:

$$(p : \mathbb{N} \times \mathbb{N})(e : \text{fst } p \equiv_\mathbb{N} \text{ zero})$$

$\simeq (x : \mathbb{N})(y : \mathbb{N})(e : x \equiv_\mathbb{N} \text{ zero})$

$\simeq (y : \mathbb{N})$

$\eta$-expansion of an equation:

$$(e : x, y \equiv_{\mathbb{N} \times \mathbb{N}} f z)$$

$\simeq (e_1 : x \equiv_\mathbb{N} \text{ fst } (f z))$

$\simeq (e_2 : y \equiv_\mathbb{N} \text{ snd } (f z))$$
Rules for $\eta$-equality of records

$\eta$-expansion of a flexible variable:

$$(p : \mathbb{N} \times \mathbb{N})(e : \text{fst } p \equiv_\mathbb{N} \text{ zero})$$

$$\simeq (x : \mathbb{N})(y : \mathbb{N})(e : x \equiv_\mathbb{N} \text{ zero})$$

$$\simeq (y : \mathbb{N})$$

$\eta$-expansion of an equation:

$$(e : x, y \equiv_{\mathbb{N} \times \mathbb{N}} f \ z)$$

$$\simeq (e_1 : x \equiv_\mathbb{N} \text{ fst } (f \ z))$$

$$\simeq (e_2 : y \equiv_\mathbb{N} \text{ snd } (f \ z))$$
Rules for indexed data types

Idea: rules solve equations between indices together with equations between constructors

Example:

\[(e_1 : \text{suc } m \equiv_N \text{suc } n)\]
\[(e_2 : \text{cons } m \times x \times s \equiv_{\text{Vec } A e_1} \text{cons } n \times y \times y)\]
\(~\]
\[(e_1 : m \equiv_N n)(e_2 : x \equiv_A y)\]
\[(e_3 : s \equiv_{\text{Vec } A e_1} y)\]
Rules for indexed data types

Idea: rules solve equations between indices together with equations between constructors

Example:

\[
\begin{align*}
(e_1 : \text{\texttt{suc}} & \; m \equiv_{\mathbb{N}} \text{\texttt{suc}} \; n) \\
(e_2 : \text{\texttt{cons}} & \; m \times \text{\texttt{xs}} \equiv_{\text{\texttt{Vec}} \; A \; e_1} \text{\texttt{cons}} \; n \; y \; y\text{\texttt{s}}) \\
\sim & \quad (e_1 : m \equiv_{\mathbb{N}} n)(e_2 : x \equiv_{A} y) \\
& \quad (e_3 : \text{\texttt{xs}} \equiv_{\text{\texttt{Vec}} \; A \; e_1} y\text{\texttt{s}})
\end{align*}
\]
Rules for indexed data types

This can give a real boost to power:

data \( \text{Im} (f : A \to B) : B \to \text{Set} \) where

\[
\text{image} : (x : A) \to \text{Im} f (f x)
\]

\[
(x y : A)(e_1 : f x \cong_B f y) \\
(e_2 : \text{image} x \cong_{\text{Im} f} e_1 \text{ image} y) \\
\cong (x y : A)(e : x \cong_A y) \\
\cong (x : A)
\]
From this point, there be dragons

Any questions so far?
A sound unification algorithm based on telescope equivalences

1. Unifiers as equivalences
2. Unification rules
3. Higher-dimensional unification
Indexed rules are too restrictive

Rules for indexed datatypes require indices to be fully general

This is too restrictive:

\[
\left( e_1 : \text{cons } n \times xs \equiv_{\text{Vec } A} (\text{suc } n) \text{ cons } n \ y \ ys \right) \\
\not\equiv \left( e_1 : x \equiv_A y \right) \left( e_2 : xs \equiv_{\text{Vec } A} n \ ys \right)
\]
Indexed rules are too restrictive

Rules for indexed datatypes require indices to be fully general

This is too restrictive:

\[(e_1 : \text{cons } n \times xs \equiv_{\text{Vec } A} (\text{suc } n) \text{ cons } n \ y \ ys) \not\leadsto (e_1 : x \equiv_A y)(e_2 : xs \equiv_{\text{Vec } A} n \ ys)\]
Generalized rules for indexed data

The following rules can be generalized to arbitrary indices:

- Conflict
- Cycle
- Injectivity: only if index types satisfy $K!$
Reverse unification rules

Idea: we can generalize the indices by applying unification rules in reverse
Reverse unification rules: example

$\vdash (n : \mathbb{N})(x y : A)(xs ys : \text{Vec } A n) \quad (e : \text{cons } n x xs \equiv_{\text{Vec } A} (\text{suc } n) \text{ cons } n y ys)$

\[\vdash (m n : \mathbb{N})(x y : A)(xs : \text{Vec } A m)(ys : \text{Vec } A n)\]
\[\vdash (e_1 : m \equiv_{\mathbb{N}} n)\]
\[\vdash (e_2 : \text{cons } m x xs \equiv_{\text{Vec } A} (\text{suc } e_1) \text{ cons } n y ys)\]

\[\vdash (m n : \mathbb{N})(x y : A)(xs : \text{Vec } A m)(ys : \text{Vec } A n)\]
\[\vdash (e_1 : \text{suc } m \equiv_{\mathbb{N}} \text{suc } n)\]
\[\vdash (e_2 : \text{cons } m x xs \equiv_{\text{Vec } A} e_1 \text{ cons } n y ys)\]

\[\vdash (m n : \mathbb{N})(x y : A)(xs : \text{Vec } A m)(ys : \text{Vec } A n)\]
\[\vdash (e_1 : m \equiv_{\mathbb{N}} n)(e_2 : x \equiv_{A} y)(e_3 : xs \equiv_{\text{Vec } A} e_1 ys)\]
\[\vdash (n : \mathbb{N})(x : A)(xs : \text{Vec } A n)\]
Reverse unification rules: example

\((n : \mathbb{N})(x \ y : A)(xs \ ys : \text{Vec} \ A \ n)\)
\((e : \text{cons} \ n \ x \ xs \equiv_{\text{Vec} \ A} \text{suc} \ n \ \text{cons} \ n \ y \ ys)\)
\((m \ n : \mathbb{N})(x \ y : A)(xs : \text{Vec} \ A \ m)(ys : \text{Vec} \ A \ n)\)
\(\sim\)
\((e_1 : m \equiv_{\mathbb{N}} n)\)
\((e_2 : \text{cons} \ m \ x \ xs \equiv_{\text{Vec} \ A} \text{suc} \ e_1 \ \text{cons} \ n \ y \ ys)\)

\(\sim\)
\((m \ n : \mathbb{N})(x \ y : A)(xs : \text{Vec} \ A \ m)(ys : \text{Vec} \ A \ n)\)
\((e_1 : \text{suc} \ m \equiv_{\mathbb{N}} \text{suc} \ n)\)
\((e_2 : \text{cons} \ m \ x \ xs \equiv_{\text{Vec} \ A} e_1 \ \text{cons} \ n \ y \ ys)\)

\(\sim\)
\((m \ n : \mathbb{N})(x \ y : A)(xs : \text{Vec} \ A \ m)(ys : \text{Vec} \ A \ n)\)
\((e_1 : m \equiv_{\mathbb{N}} n)(e_2 : x \equiv_A y)(e_3 : xs \equiv_{\text{Vec} \ A} e_1 \ ys)\)
\(\sim\)
\((n : \mathbb{N})(x : A)(xs : \text{Vec} \ A \ n)\)
Reverse unification rules: example

\((n : \mathbb{N})(x \ y : A)(xs \ ys : \text{Vec} \ A \ n)\)
\((e : \text{cons} \ n \ x \ xs \ \equiv_{\text{Vec} A} (\text{suc} \ n) \ \text{cons} \ n \ y \ ys)\)

\((m \ n : \mathbb{N})(x \ y : A)(xs : \text{Vec} A \ m)(ys : \text{Vec} A \ n)\)
\(\models (e_1 : m \ \equiv_{\mathbb{N}} \ n)\)
\((e_2 : \text{cons} \ m \ x \ xs \ \equiv_{\text{Vec} A} (\text{suc} \ e_1) \ \text{cons} \ n \ y \ ys)\)

\((m \ n : \mathbb{N})(x \ y : A)(xs : \text{Vec} A \ m)(ys : \text{Vec} A \ n)\)
\(\models (e_1 : \text{suc} \ m \ \equiv_{\mathbb{N}} \ \text{suc} \ n)\)
\((e_2 : \text{cons} \ m \ x \ xs \ \equiv_{\text{Vec} A} e_1 \ \text{cons} \ n \ y \ ys)\)

\((m \ n : \mathbb{N})(x \ y : A)(xs : \text{Vec} A \ m)(ys : \text{Vec} A \ n)\)
\(\models (e_1 : m \ \equiv_{\mathbb{N}} \ n)(e_2 : x \ \equiv_{A} y)(e_3 : xs \ \equiv_{\text{Vec} A} e_1 \ ys)\)
\(\models (n : \mathbb{N})(x : A)(xs : \text{Vec} A \ n)\)
Reverse unification rules: example

\[
\begin{align*}
(n : \mathbb{N})(x \ y : A)(xs \ ys : \text{Vec } A \ n) \\
(e : \text{cons } n \ x \ xs \equiv_{\text{Vec } A} (\text{suc } n) \ \text{cons } n \ y \ ys)
\end{align*}
\]

\[
\begin{align*}
(m \ n : \mathbb{N})(x \ y : A)(xs : \text{Vec } A \ m)(ys : \text{Vec } A \ n) \\
\simeq (e_1 : m \equiv_{\mathbb{N}} n) \\
(e_2 : \text{cons } m \ x \ xs \equiv_{\text{Vec } A} (\text{suc } e_1) \ \text{cons } n \ y \ ys)
\end{align*}
\]

\[
\begin{align*}
(m \ n : \mathbb{N})(x \ y : A)(xs : \text{Vec } A \ m)(ys : \text{Vec } A \ n) \\
\simeq (e_1 : \text{suc } m \equiv_{\mathbb{N}} \text{suc } n) \\
(e_2 : \text{cons } m \ x \ xs \equiv_{\text{Vec } A} e_1 \ \text{cons } n \ y \ ys)
\end{align*}
\]

\[
\begin{align*}
(m \ n : \mathbb{N})(x \ y : A)(xs : \text{Vec } A \ m)(ys : \text{Vec } A \ n) \\
\simeq (e_1 : m \equiv_{\mathbb{N}} n)(e_2 : x \equiv_{A} y)(e_3 : xs \equiv_{\text{Vec } A} e_1 \ ys)
\end{align*}
\]

\[
\simeq (n : \mathbb{N})(x : A)(xs : \text{Vec } A \ n)
\]
Reverse unification rules: example

\[
(n : \mathbb{N})(x \ y : A)(xs \ ys : \text{Vec} \ A \ n)
\]

\[
(e : \text{cons} \ n \ x \ xs \ \cong_{\text{Vec} \ A} \text{(suc} \ n) \ \text{cons} \ n \ y \ ys)
\]

\[
(m \ n : \mathbb{N})(x \ y : A)(xs : \text{Vec} \ A \ m)(ys : \text{Vec} \ A \ n)
\]

\[
(\overset{\sim}{e_1} : m \cong_{\mathbb{N}} n)
\]

\[
(\overset{\sim}{e_2} : \text{cons} \ m \ x \ xs \ \cong_{\text{Vec} \ A} \text{(suc} \ e_1) \ \text{cons} \ n \ y \ ys)
\]

\[
(m \ n : \mathbb{N})(x \ y : A)(xs : \text{Vec} \ A \ m)(ys : \text{Vec} \ A \ n)
\]

\[
(\overset{\sim}{e_1} : \text{suc} \ m \cong_{\mathbb{N}} \text{suc} \ n)
\]

\[
(\overset{\sim}{e_2} : \text{cons} \ m \ x \ xs \ \cong_{\text{Vec} \ A} e_1 \ \text{cons} \ n \ y \ ys)
\]

\[
(m \ n : \mathbb{N})(x \ y : A)(xs : \text{Vec} \ A \ m)(xs : \text{Vec} \ A \ n)
\]

\[
(\overset{\sim}{e_1} : m \cong_{\mathbb{N}} n)(\overset{\sim}{e_2} : x \cong_{A} y)(\overset{\sim}{e_3} : xs \cong_{\text{Vec} \ A} e_1 \ ys)
\]

\[
(\overset{\sim}{e_1} : m \cong_{\mathbb{N}} n)(x : A)(xs : \text{Vec} \ A \ n)
\]
Reverse unification rules: problems

- Applicability is limited:
  indices need to be linear patterns
- Hard to implement
- Not clear how to apply injectivity for indexed data in reverse
Going beyond the first level

Realization: same problem as for case splitting, only for equations instead of variables

We can solve it in the same way as well: by specialization by unification
Going beyond the first level

Realization: same problem as for case splitting, only for equations instead of variables

We can solve it in the same way as well: by specialization by unification
Going beyond the first level

Realization: same problem as for case splitting, only for equations instead of variables

We can solve it in the same way as well: by specialization by unification
Higher-dimensional unification: example

\[(e : \text{cons } n \times xs \equiv_{\text{Vec } A (\text{suc } n)} \text{cons } n y ys)\]
\[(e_1 : \text{suc } n \equiv_{\text{N}} \text{suc } n)\]
\[\simeq (e_2 : \text{cons } n \times xs \equiv_{\text{Vec } A e_1} \text{cons } n y ys)\]
\[(f : e_1 \equiv_{\text{suc } n \equiv_{\text{N}} \text{suc } n \text{ refl}})\]
\[\simeq (e_1 : n \equiv_{\text{N}} n)(e_2 : x \equiv_{A} y)(e_3 : xs \equiv_{\text{Vec } A e_1} ys)\]
\[(f : \text{cong suc } e_1 \equiv_{\text{suc } n \equiv_{\text{N}} \text{suc } n \text{ refl}})\]
\[\simeq (e_1 : n \equiv_{\text{N}} n)(e_2 : x \equiv_{A} y)(e_3 : xs \equiv_{\text{Vec } A e_1} ys)\]
\[(f : e_1 \equiv_{n \equiv_{\text{N}} n \text{ refl}})\]
\[\simeq (e_2 : x \equiv_{A} y)(e_3 : xs \equiv_{\text{Vec } A n} ys)\]
Higher-dimensional unification: example

\[(e : \text{cons } n \times xs \equiv_{\text{Vec } A} \text{(suc } n) \text{ cons } n \ y \ ys)\]

\[\sim\]

\[(e_1 : \text{suc } n \equiv_N \text{suc } n)\]

\[\sim\]

\[(e_2 : \text{cons } n \times xs \equiv_{\text{Vec } A} e_1 \text{ cons } n \ y \ ys)\]

\[(f : e_1 \equiv_{\text{suc } n \equiv_N \text{suc } n} \text{refl})\]

\[\sim\]

\[(e_1 : n \equiv_N n)(e_2 : x \equiv_A y)(e_3 : xs \equiv_{\text{Vec } A} e_1 \ ys)\]

\[(f : \text{cong suc } e_1 \equiv_{\text{suc } n \equiv_N \text{suc } n} \text{refl})\]

\[\sim\]

\[(e_1 : n \equiv_N n)(e_2 : x \equiv_A y)(e_3 : xs \equiv_{\text{Vec } A} e_1 \ ys)\]

\[(f : e_1 \equiv_{n \equiv_N n} \text{refl})\]

\[\sim\]

\[(e_2 : x \equiv_A y)(e_3 : xs \equiv_{\text{Vec } A} n \ ys)\]
Higher-dimensional unification: example

\[(e : \text{cons } n \times xs \equiv_{\text{Vec } A(e_1 \cdot n)} \text{cons } n \cdot y \cdot ys)\]

\[\leadsto (e_1 : \text{suc } n \equiv_{\mathbb{N}} \text{suc } n)\]

\[\leadsto (e_2 : \text{cons } n \times xs \equiv_{\text{Vec } A e_1} \text{cons } n \cdot y \cdot ys)\]

\[f : e_1 \equiv_{\text{suc } n \equiv_{\mathbb{N}} \text{suc } n} \text{refl}\]

\[\leadsto (e_1 : n \equiv_{\mathbb{N}} n)(e_2 : x \equiv_{A} y)(e_3 : xs \equiv_{\text{Vec } A e_1} y)\]

\[f : \text{cong suc } e_1 \equiv_{\text{suc } n \equiv_{\mathbb{N}} \text{suc } n} \text{refl}\]

\[\leadsto (e_1 : n \equiv_{\mathbb{N}} n)(e_2 : x \equiv_{A} y)(e_3 : xs \equiv_{\text{Vec } A e_1} y)\]

\[f : e_1 \equiv_{n \equiv_{\mathbb{N}} n} \text{refl}\]

\[\leadsto (e_2 : x \equiv_{A} y)(e_3 : xs \equiv_{\text{Vec } A} n \cdot y)\]
Higher-dimensional unification: example

\[(e : \text{cons } n \times xs \equiv_{\text{Vec } A (\text{suc } n)} \text{cons } n \ y \ ys)\]

\[\subseteq\]

\[(e_1 : \text{suc } n \equiv_{\mathbb{N}} \text{suc } n)\]

\[\subseteq\]

\[(e_2 : \text{cons } n \times xs \equiv_{\text{Vec } A e_1} \text{cons } n \ y \ ys)\]

\[(f : e_1 \equiv_{\text{suc } n \equiv_{\mathbb{N}} \text{suc } n} \text{refl})\]

\[\subseteq\]

\[(e_1 : n \equiv_{\mathbb{N}} n)(e_2 : x \equiv_{A} y)(e_3 : xs \equiv_{\text{Vec } A e_1} ys)\]

\[(f : \text{cong suc } e_1 \equiv_{\text{suc } n \equiv_{\mathbb{N}} \text{suc } n} \text{refl})\]

\[\subseteq\]

\[(e_1 : n \equiv_{\mathbb{N}} n)(e_2 : x \equiv_{A} y)(e_3 : xs \equiv_{\text{Vec } A e_1} ys)\]

\[(f : e_1 \equiv_{n \equiv_{\mathbb{N}} n} \text{refl})\]

\[\subseteq\]

\[(e_2 : x \equiv_{A} y)(e_3 : xs \equiv_{\text{Vec } A n} ys)\]
Representing higher-order problems using first-order syntax

An $n$-dimensional unification problem consists of

- a telescope $\Gamma$ of flexible variables
- equation telescopes $\Delta_1, \ldots, \Delta_n$
  such that $\vdash \Gamma \Delta_1 \ldots \Delta_n$
- left- and right-hand sides $\bar{u}_1, \bar{v}_1, \ldots \bar{u}_n, \bar{v}_n$
  such that $\Gamma \Delta_1 \ldots \Delta_{i-1} \vdash \bar{u}_i, \bar{v}_1 : \Delta_i$
Higher-dimensional unification seems easier to implement than reverse rules

But maybe it goes too far?

Alternative: use reflection to implement a case splitting tactic based on unification
Discussion

Higher-dimensional unification seems easier to implement than reverse rules.

But maybe it goes too far?

Alternative: use reflection to implement a case splitting tactic based on unification.