Unification with a heterogeneous equality

Víctor López Juan

2019-09-20, Aspenäs

Notation

$$
\begin{array}{rcll}
t, u, v, r, & ::= & \mathrm{Bool} & \text{boolean type} \\
T, U, A, B & \mid & \Pi A B & \text{function type} \\
 & \mid & \Sigma A B & \text{product type} \\
 & \mid & \mathrm{Set} & \text{universe} \\
 & \mid & \mathrm{true} \mid \mathrm{false} & \text{booleans} \\
 & \mid & \lambda.t & \lambda\text{-abstraction} \\
 & \mid & \langle t, u \rangle & \text{pair} \\
 & \mid & h\,\vec{e} & \text{neutral terms}
\end{array}
$$

$$
\begin{array}{rcll}
x, y, z & ::= & 0, 1, 2, \ldots & \\
h & ::= & x, y, z, \ldots & \text{variable} \\
 & \mid & \alpha, \beta, \ldots & \text{metavariable} \\
 & \mid & \mathbb{a}, \mathbb{b}, \mathbb{c}, \ldots & \text{atom} \\
 & \mid & \mathrm{if} & \text{boolean recursor} \\
e & ::= & & \textit{eliminators} \\
 & \mid & t & \text{term application} \\
 & \mid & .\pi_1 \mid .\pi_2 & \text{projections}
\end{array}
$$

- Signatures: $\Sigma ::= \cdot \mid \Sigma, \alpha : A \mid \Sigma, \alpha := t : A \mid \Sigma, \mathbb{a} : A \ (\mathrm{FV}(A) = \mathrm{FV}(t) = \emptyset)$
- Closed signatures: $\Theta ::= \cdot \mid \Theta, \alpha := u : B \mid \Theta, \mathbb{a} : B$, with $\mathrm{METAS}(u) = \mathrm{METAS}(B) = \emptyset$.
- Contexts: $\Gamma ::= \cdot \mid \Gamma, A$ Typing and equality judgments: $\Sigma; \Gamma \vdash t : A$, $\Sigma; \Gamma \vdash t \equiv u : A$.
- Hereditary substitution: $t[u] \Downarrow v$. Reduction: $\Sigma; \Gamma \vdash \mathrm{if}\,(\lambda.A)\,\mathrm{true}\,t\,u \longrightarrow t : A[\mathrm{true}]$, ….
- (We use $\Sigma; \Gamma \vdash \mathrm{Set} : \mathrm{Set}$).

Type checking by unification

- Signatures: $\Sigma ::= \cdot \mid \Sigma, \alpha : A \mid \Sigma, \alpha := t : A \mid \Sigma, \mathbb{a} : A \; (\textsc{fv}(A) = \textsc{fv}(t) = \emptyset)$
- Closed signatures: $\Theta ::= \cdot \mid \Theta, \alpha := u : B \mid \Theta, \mathbb{a} : B$, with $\textsc{metas}(u) = \textsc{metas}(B) = \emptyset$.

Definition (Type-checking problem*)

Given a 4-tuple $\Sigma; \Gamma \vdash^? t : A$ with $\Sigma; \Gamma \vdash A : \mathrm{Set}$ (a type-checking problem),
find a "unique" "instantiation" Θ of Σ such that $\Theta; \Gamma \vdash t : A$.

- According to Mazzoli and Abel [1], type-checking reduces to dependently-typed higher-order unification:

Definition (Higher-order unification problem*)

Given for each $i \in \{1, ..., m\}$, well-typed terms $\Sigma; \Gamma_i \vdash t_i : A_i$ and $\Sigma; \Gamma_i \vdash u_i : B_i$
(i.e. a unification problem $\Sigma; \Gamma_i \vdash t_i : A_i \equiv^? u_i : B_i$),
find a "unique" "instantiation" Θ of Σ such that,
$\forall i \in \{1, ..., m\}$, $\Theta; \Gamma_i \vdash A_i \equiv B_i : \mathrm{Set}$ and $\Theta; \Gamma_i \vdash t_i \equiv u_i : A_i$.

- Undecidable in general $\Longrightarrow$ many different approaches.

Motivation (1/2): Heterogeneous unification

Unifying terms before their types

- Problems like the following arise in TT-in-TT examples (more later).

```
data MaybeBool : Set where
  None :          MaybeBool
  Some : Bool → MaybeBool
```

```
get : MaybeBool → Bool
get None      = true
get (Some x) = x
```

$\mathbb{F} : \mathrm{Bool} \to \mathrm{Set}$,
$\alpha : \mathrm{Bool} \to \texttt{MaybeBool}$

$$\frac{}{x : \mathrm{Bool} \vdash (\lambda y.\texttt{None}) : \mathbb{F}\,(\texttt{get}\,(\alpha\,x)) \to \texttt{MaybeBool} \equiv^{?} (\lambda y.(\alpha\,x)) : \mathbb{F}\,\mathrm{true} \to \texttt{MaybeBool}}\ ;$$

Results

Coq, Matita, Idris, Lean, Tog (meh...)	Agda (yay!)
$\mathbb{F}\,(\texttt{get}\,(\alpha\,x)) \to \texttt{MaybeBool}$ $\neq \mathbb{F}\,\mathrm{true} \to \texttt{MaybeBool}$	$[\alpha := \lambda.\texttt{None} : \mathrm{Bool} \to \texttt{MaybeBool}]$

Motivation (2/2): Issue #3027

```
F : Bool → Set
F false = Bool
F true  = Nat
```

```
f : (b : Bool) → F b → Nat
f false false = 0
f false true  = 1
f true  x     = 2
```

$\mathbb{D} : \mathsf{Nat} \to \mathrm{Set},$
$\alpha : \mathsf{Nat} \to \mathrm{Set},$
$\beta : \mathsf{Nat} \to \mathrm{Bool},$

$$\cdot \vdash (x : \mathsf{Nat}) \to \alpha\, x : \mathrm{Set} \equiv^? (x : \mathsf{F}\,(\beta\,0)) \to \mathbb{D}\,(f\,(\beta\,0)\,x) : \mathrm{Set}$$

$$\cdot \vdash \beta : \mathsf{Nat} \to \mathrm{Bool} \equiv^? \lambda.\mathrm{false} : \mathsf{Nat} \to \mathrm{Bool}$$

Results

Coq, Matita, Idris, Lean	Agda (oj då…)
$[\beta := \lambda.\mathrm{false},], \ x : \mathsf{Nat} \neq \mathsf{F}\,(\beta\,0)$	$[\beta := \lambda.\mathrm{false}, \alpha := \lambda x.\mathbb{D}\,(f\ \mathrm{false}\ x) : \mathsf{Nat} \to \mathrm{Set}]$

- Internal errors in well-typed programs using instance search (e.g. #1467, #2709, #3870).

Approach

Goals

- Simplicity: Use existing theory and term syntax.
- Strength: Terms can unify before their types do.
- Correctness: Solutions are well-typed and unique.
- Performance: Comparable resource usage to Agda.

How

1. Unification algorithm based on Gundry and McBride's [2] twin types.
2. Implementation prototype based on Mazzoli and Abel's [1] Tog.
3. Evaluation on TT-in-TT examples inspired by McBride [3].

Unification with twin types: Heterogeneous equality

Definition

Let t and u be terms s.t. $\Sigma; \Gamma_1 \vdash t : A$ and $\Sigma; \Gamma_2 \vdash u : B$. If there exists v such that:

i) $\Sigma; \Gamma_1 \vdash t \equiv v : A$,

ii) $\Sigma; \Gamma_2 \vdash u \equiv v : B$,

iii) and $\text{FV}(v) \subseteq \text{FV}(t) \cap \text{FV}(u)$

... then we say that t and u are heterogeneously equal, and write $\Sigma; \Gamma_1 \ddagger \Gamma_2 \vdash t \equiv\{v\}\equiv u : A \ddagger B$.

Examples

$\cdot \; ; x : \text{Bool} \ddagger \mathtt{Nat} \vdash x \equiv\{x\}\equiv x : \text{Bool} \ddagger \mathtt{Nat}$

$\alpha : \text{Bool} \to \text{Set} \; ; x : (\text{Bool} \to \mathtt{Nat}) \ddagger (\alpha\, \text{true}) \vdash \lambda y.x\, y \equiv\{x\}\equiv x : (\text{Bool} \to \mathtt{Nat}) \ddagger (\alpha\, \text{true})$

- $\Theta; \Gamma \vdash A \equiv B : \text{Set} \;\wedge\; \Theta; \Gamma \vdash t \equiv u : A$
  $\Leftrightarrow \Theta; \Gamma \ddagger \Gamma \vdash A \equiv B : \text{Set} \ddagger \text{Set} \;\wedge\; \Theta; \Gamma \ddagger \Gamma \vdash t \equiv u : A \ddagger B$
- $\Longrightarrow \Sigma; \Gamma \vdash t : A \equiv^? u : B \rightsquigarrow \Sigma; \Gamma \ddagger \Gamma \vdash A \approx B : \text{Set} \ddagger \text{Set} \;\wedge\; \Sigma; \Gamma \ddagger \Gamma \vdash t \approx u : A \ddagger B$

Unification with twin types: Rules (1/2)

Rule (Definitional equality)

$$\Sigma; \Gamma \ddagger \Gamma' \vdash t \approx u : A \ddagger A' \rightsquigarrow \Sigma; \square \quad \textbf{where} \quad \Sigma; \Gamma \ddagger \Gamma' \vdash t \equiv u : A \ddagger A'$$

Rule (Strengthening)

$$\Sigma; \Gamma \ddagger \Gamma', x : A \ddagger A', \Delta \ddagger \Delta' \vdash t \approx u : B \ddagger B' \rightsquigarrow \Sigma; \Gamma \ddagger \Gamma', \Delta \ddagger \Delta' \vdash t \approx u : B \ddagger B'$$

$$\textbf{where} \quad x \notin \mathrm{FV}(\Delta \ddagger \Delta' \vdash t \approx u : B \ddagger B')$$

Rule (Metavariable instantiation, simplified*)

$$\Sigma, \alpha : A; \Gamma \ddagger \Gamma \vdash \alpha\, \vec{x}^n \approx t : B \ddagger B \rightsquigarrow \Sigma, \alpha := \lambda \vec{y}^n . t[\vec{x} \mapsto \vec{y}] : A; \square$$

where *all* $x \in \vec{x}$ *are pair-wise distinct* **and** $\mathrm{FV}(t) \subseteq \vec{x}$

Unification with twin types: Rules (2/2)

Rule (Π-types)

$$\Sigma; \Gamma \ddagger \Gamma' \vdash \Pi A B \approx \Pi A' B' : \mathrm{Set} \ddagger \mathrm{Set} \ \rightsquigarrow$$
$$\Sigma; \Gamma \ddagger \Gamma' \vdash A \approx A' : \mathrm{Set} \ddagger \mathrm{Set} \quad \wedge \quad \Gamma \ddagger \Gamma', x : A \ddagger A' \vdash B \approx B' : \mathrm{Set} \ddagger \mathrm{Set}$$

Rule (λ-abstractions)

$$\Sigma; \Gamma \ddagger \Gamma' \vdash \lambda.t \approx \lambda.u : \Pi A B \ddagger \Pi A' B' \ \rightsquigarrow \ \Sigma; \Gamma \ddagger \Gamma', A \ddagger A' \vdash t \approx u : B \ddagger B'$$

Rule (Strongly neutral terms)

$$\Sigma; \Gamma \ddagger \Gamma' \vdash f\, t \approx g\, u : T \ddagger T' \ \rightsquigarrow \ \Sigma\, ;\, f \approx g : \Pi A B \ddagger \Pi A' B' \quad \wedge \quad \Gamma \ddagger \Gamma' \vdash t \approx u : A \ddagger A'$$

where $f\, t$ *and* $g\, u$ *are strongly neutral (e.g.* $f = g = \mathbb{a}$*, or* $f = x\, v_1$ *and* $g = x\, v_2$*)*

and $\Sigma; \Gamma \vdash f : \Pi A B$ **and** $\Sigma; \Gamma' \vdash g : \Pi A' B'$

... and more

Unification with twin types: Example

```
data MaybeBool : Set where
  None :        MaybeBool
  Some : Bool → MaybeBool
```

```
get : MaybeBool → Bool
get None     = true
get (Some x) = x
```

$\mathbb{F} : \mathrm{Bool} \to \mathrm{Set},$

$\alpha : \mathrm{Bool} \to \mathtt{MaybeBool}$

$$\overline{x : \mathrm{Bool} \vdash (\lambda y.\mathtt{None}) : \mathbb{F}\,(\mathtt{get}\,(\alpha\,x)) \to \mathtt{MaybeBool} \equiv^{?} (\lambda y.(\alpha\,x)) : \mathbb{F}\,\mathrm{true} \to \mathtt{MaybeBool}}\;;$$

1. - $x : \mathrm{Bool} \ddagger \mathrm{Bool} \vdash \mathbb{F}\,(\mathtt{get}\,(\alpha\,x)) \to \mathtt{MaybeBool} \approx \mathbb{F}\,\mathrm{true} \to \mathtt{MaybeBool} : \mathrm{Set} \ddagger \mathrm{Set}$
   - $x : \mathrm{Bool} \vdash (\lambda y.\mathtt{None}) \approx (\lambda y.(\alpha\,x)) : \mathbb{F}\,(\mathtt{get}\,(\alpha\,x)) \to \mathtt{MaybeBool} \ddagger \mathbb{F}\,\mathrm{true} \to \mathtt{MaybeBool}$
2. - $x : \mathrm{Bool} \ddagger \mathrm{Bool} \vdash \mathbb{F}\,(\mathtt{get}\,(\alpha\,x)) \approx \mathbb{F}\,\mathrm{true} : \mathrm{Set} \ddagger \mathrm{Set}$
   - $x : \mathrm{Bool} \ddagger \mathrm{Bool}, \_ : \mathbb{F}\,(\mathtt{get}\,(\alpha\,x)) \ddagger \mathbb{F}\,\mathrm{true} \vdash \mathtt{MaybeBool} \approx \mathtt{MaybeBool} : \mathrm{Set} \ddagger \mathrm{Set}$
   - $x : \mathrm{Bool}, y : \mathbb{F}\,(\mathtt{get}\,(\alpha\,x)) \ddagger \mathbb{F}\,\mathrm{true} \vdash \mathtt{None} \approx \alpha\,x : \mathtt{MaybeBool} \ddagger \mathtt{MaybeBool}$
3. - $x : \mathrm{Bool} \ddagger \mathrm{Bool} \vdash \mathtt{get}\,(\alpha\,x) \approx \mathrm{true} : \mathrm{Bool} \ddagger \mathrm{Bool}$
   - $\square$
   - $[\alpha := \lambda x.\mathtt{None}]$
4. - $x : \mathrm{Bool} \ddagger \mathrm{Bool} \vdash \mathrm{true} \approx \mathrm{true} : \mathrm{Bool} \ddagger \mathrm{Bool}$

Unification with twin types: Example

$\texttt{data MaybeBool : Set}$, $\texttt{get : MaybeBool} \to \texttt{\textbackslash Bool}$, $\mathbb{F} : \mathrm{Bool} \to \mathrm{Set}$, $\alpha : \mathrm{Bool} \to \texttt{MaybeBool}$

$$\overline{\qquad\qquad\qquad} ; \overline{\qquad\qquad\qquad}$$

$$x : \mathrm{Bool} \vdash (\lambda y.\texttt{None}) : \mathbb{F}\,(\texttt{get}\,(\alpha\,x)) \to \texttt{MaybeBool} \equiv^{?} (\lambda y.(\alpha\,x)) : \mathbb{F}\,\mathrm{true} \to \texttt{MaybeBool}$$

1.
   - $x : \mathrm{Bool} \ddagger \mathrm{Bool} \vdash \mathbb{F}\,(\texttt{get}\,(\alpha\,x)) \to \texttt{MaybeBool} \approx \mathbb{F}\,\mathrm{true} \to \texttt{MaybeBool} : \mathrm{Set} \ddagger \mathrm{Set}$
   - $x : \mathrm{Bool} \vdash (\lambda y.\texttt{None}) \approx (\lambda y.(\alpha\,x)) : \mathbb{F}\,(\texttt{get}\,(\alpha\,x)) \to \texttt{MaybeBool} \ddagger \mathbb{F}\,\mathrm{true} \to \texttt{MaybeBool}$
2.
   - $x : \mathrm{Bool} \ddagger \mathrm{Bool} \vdash \mathbb{F}\,(\texttt{get}\,(\alpha\,x)) \approx \mathbb{F}\,\mathrm{true} : \mathrm{Set} \ddagger \mathrm{Set}$
   - $x : \mathrm{Bool} \ddagger \mathrm{Bool}, \_ : \mathbb{F}\,(\texttt{get}\,(\alpha\,x)) \ddagger \mathbb{F}\,\mathrm{true} \vdash \texttt{MaybeBool} \approx \texttt{MaybeBool} : \mathrm{Set} \ddagger \mathrm{Set}$
   - $x : \mathrm{Bool}, y : \mathbb{F}\,(\texttt{get}\,(\alpha\,x)) \ddagger \mathbb{F}\,\mathrm{true} \vdash \texttt{None} \approx \alpha\,x : \texttt{MaybeBool} \ddagger \texttt{MaybeBool}$
3.
   - $x : \mathrm{Bool} \ddagger \mathrm{Bool} \vdash \texttt{get}\,(\alpha\,x) \approx \mathrm{true} : \mathrm{Bool} \ddagger \mathrm{Bool}$
   - $\square$
   - $[\alpha := \lambda x.\texttt{None}]$
4.
   - $x : \mathrm{Bool} \ddagger \mathrm{Bool} \vdash \mathrm{true} \approx \mathrm{true} : \mathrm{Bool} \ddagger \mathrm{Bool}$
5.
   - $\square$

Solution: $[\mathbb{F} : \mathrm{Bool} \to \mathrm{Set}, \alpha := \lambda x.\texttt{None} : \mathrm{Bool} \to \texttt{MaybeBool}]$

Unification with twin types: Correctness theorem

- A unification problem is a pair $\Sigma; \vec{\mathcal{C}}^n$, s.t. $\forall i \in \{1, ..., n\}$, $\mathcal{C}_i = \Gamma_{i,1} \ddagger \Gamma_{i,2} \vdash t_i \approx u_i : A_i \ddagger B_i$.
- A closed signature Θ solves a problem $(\Theta \vDash \Sigma; \mathcal{C})$ iff, $\forall i \in \{1, ..., n\}$, $\Theta; \Gamma_{i,1} \ddagger \Gamma_{i,2} \vdash t_i \equiv u_i : A_i \ddagger B_i$.

Theorem (Correctness of unification)

Let $\Sigma; \vec{\mathcal{C}}$ be an essentially homogeneous, well-formed problem such that:

1. $\Sigma; \vec{\mathcal{C}} \rightsquigarrow^\star \Sigma'; \square$.
2. Σ' *is closed.*

Then, (under some reasonable assumptions about the theory):

1. *The signature Σ' is well-formed.*
2. *Let Θ be the closing signature of Σ'. Then $\Theta \vDash \Sigma; \vec{\mathcal{C}}$.*
3. *For every $\tilde{\Theta}$ such that $\tilde{\Theta} \vDash \Sigma; \vec{\mathcal{C}}$, we have $\Theta \equiv \tilde{\Theta}$ (relative to Σ).*

Evaluation: TT-in-TT, inspired by McBride [3]

```
data U : Set
El : U -> Set

data U where
  set   : U
  el    : Set -> U
  pi    : (a : U) → (El a → U) → U
  sigma : (a : U) → (El a → U) → U

El set          = Set
El (el A)       = A
El (pi a b)     = (x : El a) → El (b x)
El (sigma a b)  = Sigma (El a)
                    (λx → El (b x))
[...]
```

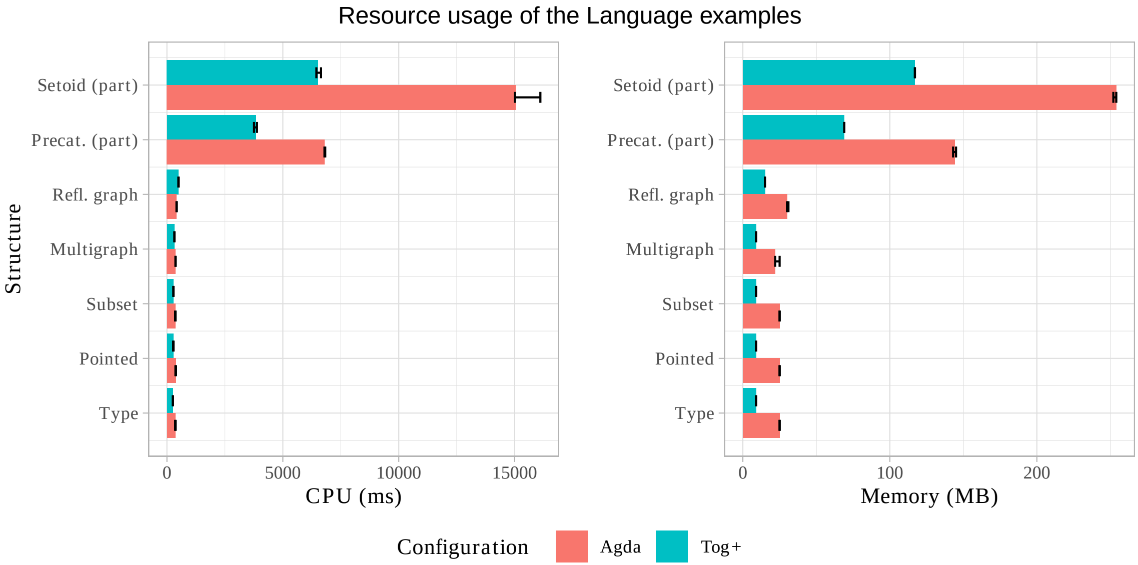
Example: Multigraph

```
graphU : U
graphU =
  sigma set (λ obj →        -- < Vertices
  (pi (el obj) (λ _ →       -- < Arrows
     (pi (el obj) (λ _ → set)))

graph : Type empty (λ _ → graphU)
graph =
  sigma' set'              -- < Vertices
  (pi' (el' (var zero)) -- < Arrows
    (pi' (el' (var (suc zero))) set'))
```

- Large (implicit) terms.
- Term-before-type unification problems.

Performance



Conclusion

Unification à la Gundry, with some modifications,

- can solve a wide range of heterogeneous problems …
- … without producing ill-typed terms
- … and using the same term syntax, typing rules and equality rules.

Our implementation:

- Can type check some complex examples (TT-in-TT).
- Does so with a resource usage comparable to Agda's.

Bibliography

[1] Francesco Mazzoli and Andreas Abel. Type checking through unification. Preprint Arxiv 1609.09709v1, 2016.

[2] Adam Gundry and Conor McBride. A tutorial implementation of dynamic pattern unification. Unpublished, 2012. URL `http://adam.gundry.co.uk/pub/pattern-unify/`.

[3] Conor McBride. Outrageous but meaningful coincidences: Dependent type-safe syntax and evaluation. In *WGP'10*, 2010. doi: 10.1145/1863495.1863497.