A Predictable Unification Algorithm for Coq

Beta Ziliani (MPI-SWS $\rightarrow$ Córdoba)
Matthieu Sozeau (Inria πr^2 & PPS, Université Paris Diderot)

MIT
April 29th 2015
Cambridge, MA, USA

Unification in Coq

Unification is a crucial tool:

- Type-checking/refinement.
  `Definition c : A := t.` $\Rightarrow$ unify the infered type T of t with A.
- Tactic applications.
  On goal $\Gamma \vdash A$, `apply (lemma : forall x1 .. xn, T)` unifies $T[?x_1 ... ?x_n]$ with A.

Currently two slightly different unification algorithms are used, one for each case, with different notions of metavariables...

Unification in Coq

- Not documented, not verified, but its results directly impact users (esp. tactic writers and library developers).
- Higher-order (pattern-unification + postponing + heuristics)
- Dependent on reductions: β, ι and δ
- Uses a backtracking first-order unification rule for fast success: $f\ u_1 \dots u_n \approx f\ v_1 \dots v_n \rightsquigarrow \overrightarrow{u_i \approx v_i}$
- Includes canonical structure resolution, an overloading mechanism relying on the precise behavior of the algorithm.
- Untyped: does not rely on the types of terms at hands, which might even not be directly convertible.
- Unsuitable for automation.

Ideally

Document and verify a predictable, efficient unification algorithm to be used both for type-checking and inside tactics.

Applications:

- Tactic programming: `eauto`, new tactic engine with existential variables.
- Programming type inference, e.g. How To Make Ad-Hoc Proof Automation Less Ad-Hoc (Gonthier *et al*, JFP'13).

Unification in Coq

The term language of Coq

$$
\begin{aligned}
t, u, T, U = \;& x \mid c[\bar{\ell}] \mid i[\bar{\ell}] \mid k[\bar{\ell}] \mid s \mid ?x[\sigma] \\
& \mid \forall x : T.\, U \mid \lambda x : T.\, t \mid t\ u \mid \textbf{let}\ x := t : T\ \textbf{in}\ u \\
& \mid \textbf{match}_T\ t\ \textbf{with}\ c_1\ \overline{x_1} \Rightarrow t_1 \mid \ldots \mid c_n\ \overline{x_n} \Rightarrow t_n\ \textbf{end} \\
& \mid \textbf{fix}_j\ \{x_1/n_1 : T_1 := t_1; \ldots; x_m/n_m : T_m := t_m\} \\
\sigma = \;& \bar{t} \\
\ell, \kappa \in \;& \mathcal{L} \cup 0^- \\
s = \;& \mathsf{Type}(\overline{\kappa(+1)?}^+)
\end{aligned}
$$

Reduction

$$(\lambda x : T.\, t)\, u \;\rightsquigarrow_\beta\; t\{u/x\} \qquad \textbf{let}\ x := u : T\ \textbf{in}\ t \;\rightsquigarrow_\zeta\; t\{u/x\}$$

$$\frac{(x := t : T) \in \Gamma}{x \;\rightsquigarrow_{\delta\Gamma}\; t} \qquad \frac{?x := t : T[\Psi] \in \Sigma}{?x[\sigma] \;\rightsquigarrow_{\delta\Sigma}\; t\{\sigma/\widehat{\Psi}\}}$$

$$\frac{(c[\overline{\ell} \vDash \mathcal{C}] := t : T) \in E}{c[\overline{\kappa}] \;\rightsquigarrow_{\delta E}\; t[\overline{\kappa/\ell}]}$$

$$\textbf{match}_T\ k_j\,[\overline{\kappa}]\ \overline{t}\ \textbf{with}\ \overline{k\ \overline{x} \Rightarrow u}\ \textbf{end} \;\rightsquigarrow_\iota\; u_j\{\overline{t/x_j}\}$$

$$\frac{F = \overline{x/n : T := t} \qquad a_n = k_j\,[\overline{\kappa}]\ \overline{t}}{\textbf{fix}_j\ \{F\}\ \overline{a} \;\rightsquigarrow_\iota\; t_j\{\overline{\textbf{fix}_m\ \{F\}/x_m}\}\ \overline{a}}$$

Unification

Judgment: $\Sigma; \Gamma \vdash t_1 \approx t_2 \rhd \Sigma'$

Example rule:

$$\frac{\Sigma; \Gamma \vdash \bar{t} \approx \overline{t'} \rhd \Sigma'}{\Sigma; \Gamma \vdash ?u[\xi]\ \bar{t} \approx ?u[\xi]\ \overline{t'} \rhd \Sigma'}\ \textsc{Meta-Same-Same}$$

Unification in Coq

Type dependency and postponment

```
Check (exist : ∀ (A : Type) (P : A → Prop) (x : A) (p : P x),
                                { x : A | P x}).

Definition f : { x : nat | x ≤ x } :=
  @exist _ _ 0 (le_n 0).
```

Uses *constraint postponment* to delay the instanciation of $?P$.
Solves $?P\ 0 = 0 \leq 0 \wedge\ ?P = \lambda\ x,\ x \leq x$.

Type dependency and postponment II

This has *unpredictable* behavior:

- Non-local effect: hard to reason about, error locations hard to guess
- Source of exponential complexity in presence of backtracking.

$\Rightarrow$ Switch to a best-effort local type inference based on bidirectional type-checking, without postponment.

In this example, this would set $?P = \lambda\ x,\ x \leq x$ before typechecking the arguments.

First-Order unification

$$\frac{\Sigma_0; \Gamma \vdash u \approx u' \rhd \Sigma_1}{\Sigma_0; \Gamma \vdash t\ u \approx t\ u' \rhd \Sigma_1}\ \textsc{App-FO}$$

- Applies even when t is an unfoldable constant: lose most general unifiers, but fast success in general and gives "natural" unifiers.
- Requires backtracking when the argument unification fails $\Rightarrow$ the reduced terms might be unifiable (slow failures if repeateadly applied).
- Complicated by universe polymorphism.

First-Order unification

Solutions:

- Parameterize by a set of constants on which the rule applies without backtracking, e.g. abbreviations only (Pfenning and Schürmann, TYPES'98). Does not play well with reduction.
- Experiment with caching mechanisms to be less penalized by failures.

Canonical Structure Resolution

Overloading mechanism based on records.

```
Structure monoid :=
  { carrier : Set;
    unit : carrier;
    mult : carrier → carrier → carrier;
    mult_unit_left : ∀ x, mult unit x = x
  }.

Canonical Structure nat_monoid :=
  { carrier := nat;
      unit := 0;
      mult := plus;
      mult_unit_left := fun x ⇒ eq_refl }.
```

Structure resolution

```
Check (mult :
         ∀ m : monoid, carrier m → carrier m → carrier m).
Lemma on_nat_monoid (n : nat) : mult _ 0 n = n.
Proof.
  change (mult nat_monoid 0 n = n).
  apply mult_unit_left.
Qed.
```

The unification solved the problem carrier ?m = nat by instantiating ?m with nat_monoid.

Canonical Structures

- Relies on unfolding behavior (step-by-step unfolding of constants)
- Hard to reason about if constraints can be postponed.
- Powerful mechanism to program type inference (Ziliani *et al*, ICFP'11).

The new algorithm

A pencil and paper presentation (3 pages of rules plus pruning) and an implementation.

- No constraint postponment
- Clean definition of pruning and higher-order pattern unification.
- Sound: (weakly) type-safe (mechanical proof in progress).
- Not complete: an *algorithmic* presentation.
- Three heuristics: unfolding oracle, first-order approximation and deletion of dependencies.

Rules I/III

TYPE-SAME

$$\frac{\mathcal{C}' = \mathcal{C} \wedge \overline{u}\ \mathcal{R}\ \kappa \qquad \mathcal{C}' \vDash}{\ell \vDash \mathcal{C}; \Sigma; \Gamma \vdash \mathsf{Type}(\overline{u}) \approx_{\mathcal{R}} \mathsf{Type}(\kappa) \rhd \ell \vDash \mathcal{C}'; \Sigma}$$

PROD-SAME, LAM-SAME

$$\frac{\Pi \in \{\lambda, \forall\} \qquad \Sigma_0; \Gamma \vdash T_1 \approx_{\equiv} U_1 \rhd \Sigma_1 \qquad \Sigma_1; \Gamma, x : T_1 \vdash T_2 \approx_{\mathcal{R}} U_2 \rhd \Sigma_2}{\Sigma_0; \Gamma \vdash \Pi x : T_1.\ T_2 \approx_{\mathcal{R}} \Pi x : U_1.\ U_2 \rhd \Sigma_2}$$

LET-SAME

$$\frac{\Sigma_0; \Gamma \vdash T \approx_{\equiv} U \rhd \Sigma_1 \qquad \Sigma_1; \Gamma \vdash t_2 \approx_{\equiv} u_2 \rhd \Sigma_2 \qquad \Sigma_2; \Gamma, x := t_2 \vdash t_1 \approx_{\mathcal{R}} u_1 \rhd \Sigma_3}{\Sigma_0; \Gamma \vdash_0 \textbf{let}\ x := t_2 : T\ \textbf{in}\ t_1 \approx_{\mathcal{R}} \textbf{let}\ x := u_2 : U\ \textbf{in}\ u_1 \rhd \Sigma_3}$$

RIGID-SAME

$$\frac{h \in \mathcal{V} \cup \mathcal{I} \cup \mathcal{K} \qquad \Phi_1 = \Phi_0 \wedge \overline{\ell = \kappa} \qquad \Phi_1 \models}{\Phi_0; \Sigma; \Gamma \vdash_0 h[\overline{\ell}] \approx_{\mathcal{R}} h[\overline{\kappa}] \rhd \Phi_1, \Sigma}$$

FLEXIBLE-SAME

$$\frac{h \in \mathcal{C} \qquad \Phi_0 \models \overline{\ell} = \overline{\kappa} \rhd \Phi_1}{\Phi_0; \Sigma; \Gamma \vdash_0 h[\overline{\ell}] \approx_{\mathcal{R}} h[\overline{\kappa}] \rhd \Phi_1, \Sigma}$$

UNIV-EQ

$$\frac{\Phi \models i = j}{\Phi \models i = j \rhd \Phi}$$

UNIV-FLEXIBLE

$$\frac{i_{\mathtt{f}} \vee j_{\mathtt{f}} \in \overline{\ell} \qquad \mathcal{C} \wedge i = j \models}{(\overline{\ell} \models \mathcal{C}) \models i = j \rhd \mathcal{C} \wedge i = j}$$

CASE-SAME

$$\frac{\Sigma_0; \Gamma \vdash T \approx_{\equiv} U \rhd \Sigma_1 \qquad \Sigma_1; \Gamma \vdash t \approx_{\equiv} u \rhd \Sigma_2 \qquad \Sigma_2; \Gamma \vdash \overline{b} \approx_{\equiv} \overline{b'} \rhd \Sigma_3}{\Sigma_0; \Gamma \vdash_0 \textbf{match}_T\ t\ \textbf{with}\ \overline{b}\ \textbf{end} \approx_{\mathcal{R}} \textbf{match}_U\ u\ \textbf{with}\ \overline{b'}\ \textbf{end} \rhd \Sigma_3}$$

FIX-SAME

$$\frac{\Sigma_0; \Gamma \vdash \overline{T} \approx_{\equiv} \overline{U} \rhd \Sigma_1 \qquad \Sigma_1; \Gamma \vdash \overline{t} \approx_{\equiv} \overline{u} \rhd \Sigma_2}{\Sigma_0; \Gamma \vdash_0 \textbf{fix}_j\ \{\overline{x/n : T := t}\} \approx_{\mathcal{R}} \textbf{fix}_j\ \{\overline{x/n : U := u}\} \rhd \Sigma_2}$$

APP-FO

$$\frac{\Sigma_0; \Gamma \vdash_0 t \approx_{\mathcal{R}} u \rhd \Sigma_1 \qquad n \geq 0 \qquad \Sigma_1; \Gamma \vdash \overline{t_n} \approx_{\equiv} \overline{u_n} \rhd \Sigma_2}{\Sigma_0; \Gamma \vdash t\ \overline{t_n} \approx_{\mathcal{R}} u\ \overline{u_n} \rhd \Sigma_2}$$

META-δR (META-δL)

$$\frac{\Sigma; \Gamma \vdash t \overset{\mathrm{w}}{\rightsquigarrow}_{\delta\Sigma} t' \qquad \Sigma; \Gamma \vdash u \approx_{\mathcal{R}} t' \rhd \Sigma'}{\Sigma; \Gamma \vdash u \approx_{\mathcal{R}} t \rhd \Sigma'}$$

LAM-βR (LAM-βL)

$$\frac{\Sigma; \Gamma \vdash t \overset{\mathrm{w}}{\rightsquigarrow}_{\beta} t' \qquad \Sigma; \Gamma \vdash u \approx_{\mathcal{R}} t' \rhd \Sigma'}{\Sigma; \Gamma \vdash u \approx_{\mathcal{R}} t \rhd \Sigma'}$$

LET-PARζ

$$\frac{\Sigma; \Gamma \vdash t \overset{\mathrm{w}}{\rightsquigarrow}_{\zeta} t' \qquad \Sigma; \Gamma \vdash u \overset{\mathrm{w}}{\rightsquigarrow}_{\zeta} u' \qquad \Sigma; \Gamma \vdash t' \approx_{\mathcal{R}} u' \rhd \Sigma'}{\Sigma; \Gamma \vdash t \approx_{\mathcal{R}} u \rhd \Sigma'}$$

LET-ζR (LET-ζL)

$$\frac{u\text{'s head not a } \textbf{let-in} \qquad \Sigma; \Gamma \vdash t \overset{\mathrm{w}}{\rightsquigarrow}_{\zeta} t' \qquad \Sigma; \Gamma \vdash u \approx_{\mathcal{R}} t' \rhd \Sigma'}{\Sigma; \Gamma \vdash u \approx_{\mathcal{R}} t \rhd \Sigma'}$$

CASE-ιR (CASE-ιL)

$$\frac{t \text{ is } \textbf{fix} \text{ or } \textbf{match} \qquad \Sigma; \Gamma \vdash t \downarrow^{\mathrm{w}}_{\beta\iota\theta} t' \qquad t \neq t' \qquad \Sigma; \Gamma \vdash u \approx_{\mathcal{R}} t' \rhd \Sigma'}{\Sigma; \Gamma \vdash u \approx_{\mathcal{R}} t \rhd \Sigma'}$$

CONS-δNOTSTUCKR

$$\frac{\text{not } \Sigma; \Gamma \vdash \text{is\_stuck } (c\ \overline{t_n}) \qquad c \overset{\mathrm{w}}{\rightsquigarrow}_{\delta} t \qquad \Sigma; \Gamma \vdash u \approx_{\mathcal{R}} t\ \overline{t_n} \rhd \Sigma'}{\Sigma; \Gamma \vdash u \approx_{\mathcal{R}} c\ \overline{t_n} \rhd \Sigma'}$$

CONS-δSTUCKL

$$\frac{\Sigma; \Gamma \vdash \text{is\_stuck } u \qquad c \overset{\mathrm{w}}{\rightsquigarrow}_{\delta} t \qquad \Sigma; \Gamma \vdash t\ \overline{t_n} \approx_{\mathcal{R}} u \rhd \Sigma'}{\Sigma; \Gamma \vdash c\ \overline{t_n} \approx_{\mathcal{R}} u \rhd \Sigma'}$$

CONS-δR (CONS-δL)

$$\frac{c \overset{\mathrm{w}}{\rightsquigarrow}_{\delta} t \qquad \Sigma; \Gamma \vdash u \approx_{\mathcal{R}} t\ \overline{t_n} \rhd \Sigma'}{\Sigma; \Gamma \vdash u \approx_{\mathcal{R}} c\ \overline{t_n} \rhd \Sigma'}$$

LAM-ηR (LAM-ηL)

$$\frac{\begin{array}{c}u\text{'s head is not an abstraction} \\ \Sigma_0; \Gamma \vdash u : U \qquad \phi_0 = \Phi_0 \cup i \vDash; \Sigma_0, ?v : \mathsf{Type}(i)[\Gamma, y : T] \\ \phi_0; \Gamma \vdash U \approx_{\equiv} \forall y : T.\ ?v[\widehat{\Gamma}, y] \rhd \phi_1 \\ \phi_1; \Gamma, x : T \vdash u\ x \approx_{\equiv} t \rhd \phi_2\end{array}}{\Phi_0; \Sigma_0; \Gamma \vdash u \approx_{\mathcal{R}} \lambda x : T.\ t \rhd \phi_2}$$

META-SAME-SAME

$$\frac{\Sigma; \Gamma \vdash \overline{t} \approx_{\equiv} \overline{u} \rhd \Sigma'}{\Sigma; \Gamma \vdash ?x[\sigma]\ \overline{t} \approx_{\mathcal{R}} ?x[\sigma]\ \overline{u} \rhd \Sigma'}$$

META-SAME

$$\frac{?x : T[\Psi_1] \in \Sigma \qquad \Psi_1 \vdash \sigma \cap \sigma' \rhd \Psi_2 \qquad \cdot \vdash \mathsf{sanitize}(\Psi_2) \rhd \Psi_3 \qquad \mathsf{FV}(T) \subseteq \Psi_3 \qquad \Sigma \cup \{?y : T[\Psi_3], ?x := ?y[\widehat{\Psi_3}]\}; \Gamma \vdash \overline{t} \approx_{\equiv} \overline{u} \rhd \Sigma'}{\Sigma; \Gamma \vdash ?x[\sigma]\ \overline{t} \approx_{\mathcal{R}} ?x[\sigma']\ \overline{u} \rhd \Sigma'}$$

META-INSTR

$$\frac{?x : T[\Psi] \in \Sigma_0 \qquad t', \xi_1 = \mathsf{remove\_tail}(t; \xi') \qquad t' \downarrow^{\mathrm{w}}_{\beta} t'' \qquad \Sigma_0 \vdash \mathsf{prune}(?x; \xi, \xi_1; t'') \rhd \Sigma_1 \qquad \Sigma_1; \Gamma \vdash \xi_1 : \overline{U} \qquad t''' = \lambda y : U\{\xi, \xi_1/\hat{\Psi}, \overline{y}\}^{-1}.\ \Sigma_1(t'')\{\xi, \xi_1/\hat{\Psi}, \overline{y}\}^{-1} \qquad \Sigma_1; \Psi \vdash t''' : T' \qquad \Sigma_1; \Psi \vdash T' \approx_{\leq} T \rhd \Sigma_2 \qquad ?x \notin \mathsf{FMV}(t''')}{\Sigma_0; \Gamma \vdash t \approx_{\mathcal{R}} ?x[\xi]\ \xi' \rhd \Sigma_2 \cup \{?x := t'''\}}$$

META-FOR

$$\frac{?x : T[\Psi] \in \Sigma_0 \qquad 0 < n \qquad \Sigma_0; \Gamma \vdash u\ \overline{u'_m} \approx_{\equiv} ?x[\sigma] \rhd \Sigma_1 \qquad \Sigma_1; \Gamma \vdash \overline{u''_n} \approx_{\equiv} \overline{t_n} \rhd \Sigma_2}{\Sigma_0; \Gamma \vdash u\ \overline{u'_m u''_n} \approx_{\mathcal{R}} ?x[\sigma]\ \overline{t_n} \rhd \Sigma_2}$$

META-DELDEPSR

$$\frac{?x : T[\Psi] \in \Sigma \qquad l = [i | \sigma_i \text{ is variable and } \nexists j > i.\ \sigma_i = (\sigma, \overline{u})_j] \qquad \cdot \vdash \mathsf{sanitize}(\Psi_l) \rhd \Psi' \qquad \mathsf{FV}(T) \subseteq \Psi' \qquad \Sigma \cup \{?y : T[\Psi'], ?x := ?y[\widehat{\Psi'}]\}; \Gamma \vdash t \approx_{\mathcal{R}} ?y[\sigma_l]\ \overline{u} \rhd \Sigma'}{\Sigma; \Gamma \vdash t \approx_{\mathcal{R}} ?x[\sigma]\ \overline{u} \rhd \Sigma'}$$

META-REDUCER

$$\frac{?u : T[\Psi] \in \Sigma_0 \qquad t \overset{\mathrm{w}\ 0..1}{\rightsquigarrow_{\delta}} t' \qquad t' \downarrow^{\mathrm{w}}_{\beta\iota\theta} t'' \qquad \Sigma_0; \Gamma \vdash t'' \approx_{\mathcal{R}} ?u[\sigma]\ \overline{t_n} \rhd \Sigma_1}{\Sigma_0; \Gamma \vdash t \approx_{\mathcal{R}} ?u[\sigma]\ \overline{t_n} \rhd \Sigma_1}$$

INTERSEC-NIL

$$\frac{}{\cdot \vdash \cdot \cap \cdot \rhd \cdot}$$

INTERSEC-KEEP

$$\frac{\Psi \vdash \sigma \cap \sigma' \rhd \Psi'}{\Psi, x := u : A \vdash \sigma, t \cap \sigma', t \rhd \Psi', x : A}$$

INTERSEC-REMOVE

$$\frac{\Psi \vdash \sigma \cap \sigma' \rhd \Psi' \qquad y \neq z}{\Psi, x := u : T \vdash \sigma, y \cap \sigma', z \rhd \Psi'}$$

SANITIZE-NIL

$$\frac{}{\xi \vdash \mathsf{sanitize}(\cdot) \rhd \cdot}$$

SANITIZE-KEEP

$$\frac{\mathsf{FV}(T) \subseteq \xi \qquad \mathsf{FV}(u) \subseteq \xi \qquad x, \xi \vdash \mathsf{sanitize}(\Psi) \rhd \Psi'}{\xi \vdash \mathsf{sanitize}(x := u : T, \Psi) \rhd x : T, \Psi'}$$

SANITIZE-REMOVE

$$\frac{\mathsf{FV}(T) \not\subseteq \xi \vee \mathsf{FV}(u) \not\subseteq \xi \qquad \xi \vdash \mathsf{sanitize}(\Psi) \rhd \Psi'}{\xi \vdash \mathsf{sanitize}(x := u : T, \Psi) \rhd \Psi'}$$

Figure 6. Meta-variable instantiation.

Rules III/III

LOOKUP-CS

$$\frac{\begin{array}{c}(p_j, h, c_\iota) \in \Delta_{\text{db}} \\ \Phi_1, \iota = \mathsf{fresh}(\Phi_0, c_\iota) \qquad \iota \rightsquigarrow_{\delta E} \overline{\lambda x : T.}\ k[\overline{\kappa'}]\ \overline{p'}\ \overline{v} \\ \Sigma_1 = \Sigma_0, \overline{?y : T} \qquad \Phi_1 \models \overline{\kappa} = \overline{\kappa'} \rhd \Phi_2 \\ \Phi_2; \Sigma_1; \Gamma \vdash \overline{p} \approx_{\equiv} \overline{p'\{\overline{?y}/\overline{x}\}} \rhd \Phi_3; \Sigma_2\end{array}}{\Phi_0; \Sigma_0 \vdash (p_j, \overline{\kappa}, \overline{p}, h) \in ?\ \Delta_{\text{db}} \rhd \Phi_3, \Sigma_2, \iota\ \overline{?y}, v_j\{\overline{?y}/\overline{x}\}}$$

CS-CONSTR

$$\frac{\begin{array}{c}\Phi_0; \Sigma_0 \vdash (p_j, \overline{\kappa}, \overline{p}, c) \in ?\ \Delta_{\text{db}} \rhd \Phi_1, \Sigma_1, \iota, c[\overline{\ell'}]\ \overline{u'} \\ \Phi_1 \models \overline{\ell} = \overline{\ell'} \rhd \Phi_2 \qquad \Phi_2; \Sigma_1; \Gamma \vdash \overline{u} \approx_{\equiv} \overline{u'} \rhd \Phi_3; \Sigma_2 \\ \Phi_3; \Sigma_2; \Gamma \vdash i \approx_{\equiv} \iota \rhd \Phi_4; \Sigma_3 \\ \Phi_4; \Sigma_4; \Gamma \vdash \overline{t'} \approx_{\equiv} \overline{t} \rhd \Phi_5; \Sigma_4\end{array}}{\Phi_0; \Sigma_0; \Gamma \vdash c[\overline{\ell}]\ \overline{u}\ \overline{t'} \approx_{\mathcal{R}} p_j[\overline{\kappa}]\ \overline{p}\ i\ \overline{t} \rhd \Phi_5; \Sigma_4}$$

CS-PRODR

$$\frac{\begin{array}{c}\Phi_0; \Sigma_0 \vdash (p_j, \overline{\kappa}, \overline{p}, \to) \in ?\ \Delta_{\text{db}} \rhd \Phi_1, \Sigma_1, \iota, u \to u' \\ \Phi_1; \Sigma_1; \Gamma \vdash t \approx_{\equiv} u \rhd \Phi_2; \Sigma_2 \\ \Phi_2; \Sigma_2; \Gamma \vdash t' \approx_{\mathcal{R}} u' \rhd \Phi_3; \Sigma_3 \\ \Phi_3; \Sigma_3; \Gamma \vdash i \approx_{\equiv} \iota \rhd \Phi_4; \Sigma_4\end{array}}{\Phi_0, \Sigma_0; \Gamma \vdash t \to t' \approx_{\mathcal{R}} p_j[\overline{\kappa}]\ \overline{p}\ i \rhd \Phi_4; \Sigma_4}$$

CS-SORTR

$$\frac{\begin{array}{c}\Phi_0; \Sigma_0 \vdash (p_j, \overline{\kappa}, \overline{p}, s) \in ?\ \Delta_{\text{db}} \rhd \Phi_1, \Sigma_1, \iota, v_j \\ \Phi_1; \Sigma_1; \Gamma \vdash s \approx_{\mathcal{R}} v_j \rhd \Phi_2; \Sigma_2 \\ \Phi_2; \Sigma_2; \Gamma \vdash i \approx_{\equiv} \iota \rhd \Phi_3; \Sigma_3\end{array}}{\Phi_0; \Sigma_0; \Gamma \vdash s \approx_{\mathcal{R}} p_j[\overline{\kappa}]\ \overline{p}\ i \rhd \Phi_3; \Sigma_3}$$

CS-DEFAULTR

$$\frac{\begin{array}{c}\Phi_0; \Sigma_0 \vdash (p_j, \overline{\kappa}, \overline{p}, \_) \in ?\ \Delta_{\text{db}} \rhd \Phi_1, \Sigma_1, \iota, v_j \\ \Phi_3; \Sigma_2; \Gamma \vdash t \approx_{\mathcal{R}} v_j \rhd \Phi_4; \Sigma_3 \\ \Phi_4; \Sigma_3; \Gamma \vdash i \approx_{\equiv} \iota \rhd \Phi_5; \Sigma_4\end{array}}{\Phi_0; \Sigma_0; \Gamma \vdash t \approx_{\mathcal{R}} p_j[\overline{\kappa}]\ \overline{p}\ i \rhd \Phi_5; \Sigma_4}$$

Rules

- – Still a mouthful.
- \+ Faithful to the code.
- \+ Modular, one could add or remove heuristics easily.

Dealing with first-order unification

Problem: $?t +\!\!+ ?u \approx [1;2] +\!\!+ [3;4]$.

- In general, many solutions, no m.g.u.
- Here we allow fist-order unification, so $?t := [1;2]$, $?u := [3;4]$ is found.
- A most natural unifier notion? It finds the m.g.u. assuming all constants are opaque, failing that it does one step of reduction (according to some oracle) and starts again.

Dealing with dependencies

A usual problem coming from dependent pattern-matching, e.g. Ssreflect's `if`:

$$?t[\mathsf{true}] \approx \mathsf{bool} \wedge ?t[\mathsf{false}] \approx \mathsf{bool}$$

No m.g.u. but one expects: $?t[x] := \mathsf{bool}$.
Achieved by brutal removing of dependencies of metavariables:

$$?t[x] :=?t'[]$$

Highly debatable, only necessary in cases the user or refiner made metavariables dependent when they should not be.

Implementation

`http://github.com/unicoq` plugin for Coq 8.5.

- "Easy" debugging with traces (interactive debugger in the works)
- Not fast (yet, about 20% slower than vanilla Coq).
- No control on reduction/expansion (yet).

Benchmarks

Tested on SSReflect + MathComp library:

- 82kLoC, 10M+ unification problems.
- 40 lines diff (with brutal deletion of dependencies).
- Bidirectional typechecking would reduce this.

Benchmarks

Tested on SSReflect + MathComp library:

- 82kLoC, 10M+ unification problems.
- 40 lines diff (with brutal deletion of dependencies).
- Bidirectional typechecking would reduce this.

Also tested on:

- CPDT: 14 type annotations needed out of 6200 LoC, 6 if we had bidir type-checking.
- stdlib: negligible changes but not strees testing the algorithm much
- Lemma Overloading: one type annotation needed.

Related work

- Higher-Order Dynamic Pattern Unification (Abel & Pientka): uses postponment.
- Same for J. Reed, U. Norrel's unifier for Agda and C.S.Coen's unifier for CIC (all idealized fragments, e.g. no δ reduction).

Conclusion and Future Work

- First specification of unification with Canonical Structures, Universe Polymorphism and all reductions.
- A simpler, documented and predictable unification algorithm for Coq.
- A realistic implementation.

What's left:

- Parameterization.
- Proofs.
- Further integration of the algorithm and bidirectional type-checking.

The End

That's all folks!