Extending Pattern Unification in the Rocq proof assistant

Ouassim Ait-Moussa
Supervised by Hugo Herbelin, IRIF

September 5, 2025

Proof assistants

```
Axioms (ℕ : Set) (O : ℕ) (S : ℕ -> ℕ).
Axioms (plus mult: ℕ -> ℕ -> ℕ) (eq:ℕ -> ℕ -> Prop).
```

Proof assistants

```
Axioms (ℕ : Set) (O : ℕ) (S : ℕ -> ℕ).
Axioms (plus mult: ℕ -> ℕ -> ℕ) (eq:ℕ -> ℕ -> Prop).
Axioms (ax_eqsymm: ∀nm, n = m -> m = n) (ax_eqtrans: ···) ··· (* Peano arithmetic *)
Axiom (ax_ind: ∀ P : ℕ -> Prop,(∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
···
```

Proof assistants

```
Axioms (ℕ : Set) (O : ℕ) (S : ℕ -> ℕ).
Axioms (plus mult: ℕ -> ℕ -> ℕ) (eq:ℕ -> ℕ -> Prop).
Axioms (ax_eqsymm: ∀nm, n = m -> m = n) (ax_eqtrans: ···) ··· (* Peano arithmetic *)
Axiom (ax_ind: ∀ P : ℕ -> Prop,(∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
···

Lemma S_is_addOne : ∀ n k, n =k  -> S n = k + 1. Proof. ··· Qed.
Lemma SStwise_is_twiceS : ∀ n k, n = 2*k+1 -> S n = 2*(S k). Proof. ··· Qed.
```

Proof assistants

```
Axioms (ℕ : Set) (O : ℕ) (S : ℕ -> ℕ).
Axioms (plus mult: ℕ -> ℕ -> ℕ) (eq:ℕ -> ℕ -> Prop).
Axioms (ax_eqsymm: ∀nm, n = m -> m = n) (ax_eqtrans: ···) ··· (* Peano arithmetic *)
Axiom (ax_ind: ∀ P : ℕ -> Prop,(∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
···

Lemma S_is_addOne : ∀ n k, n =k  -> S n = k + 1. Proof. ··· Qed.
Lemma SStwise_is_twiceS : ∀ n k, n = 2*k+1 -> S n = 2*(S k). Proof. ··· Qed.
```

Proposition

For all n, either (i) $n = 2k$ or (ii) $n = 2k + 1$ for some k.

Proof.

We prove it by induction on n. .

□

Proof assistants

```
Axioms (ℕ : Set) (O : ℕ) (S : ℕ -> ℕ).
Axioms (plus mult: ℕ -> ℕ -> ℕ) (eq:ℕ -> ℕ -> Prop).
Axioms (ax_eqsymm: ∀nm, n = m -> m = n) (ax_eqtrans: ···) ··· (* Peano arithmetic *)
Axiom (ax_ind: ∀ P : ℕ -> Prop,(∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
···

Lemma S_is_addOne : ∀ n k, n =k  -> S n = k + 1. Proof. ··· Qed.
Lemma SStwise_is_twiceS : ∀ n k, n = 2*k+1 -> S n = 2*(S k). Proof. ··· Qed.

Definition div2 :=
ax_ind (fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))
```

Proposition

For all n, either (i) $n = 2k$ or (ii) $n = 2k + 1$ for some k.

Proof.

We prove it by induction on n. .

□

Proof assistants

```
Axioms (ℕ : Set) (O : ℕ) (S : ℕ -> ℕ).
Axioms (plus mult: ℕ -> ℕ -> ℕ) (eq:ℕ -> ℕ -> Prop).
Axioms (ax_eqsymm: ∀nm, n = m -> m = n) (ax_eqtrans: ···) ··· (* Peano arithmetic *)
Axiom (ax_ind: ∀ P : ℕ -> Prop,(∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
···

Lemma S_is_addOne : ∀ n k, n =k  -> S n = k + 1. Proof. ··· Qed.
Lemma SStwise_is_twiceS : ∀ n k, n = 2*k+1 -> S n = 2*(S k). Proof. ··· Qed.

Definition div2 :=
ax_ind (fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))
```

Proposition

For all n, either (i) $n = 2k$ or (ii) $n = 2k + 1$ for some k.

Proof.

We prove it by induction on n. Assume it is true for some n. . We reason by cases:

□

Proof assistants

```
Axioms (ℕ : Set) (O : ℕ) (S : ℕ -> ℕ).
Axioms (plus mult: ℕ -> ℕ -> ℕ) (eq:ℕ -> ℕ -> Prop).
Axioms (ax_eqsymm: ∀nm, n = m -> m = n) (ax_eqtrans: ···) ··· (* Peano arithmetic *)
Axiom (ax_ind: ∀ P : ℕ -> Prop,(∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
···

Lemma S_is_addOne : ∀ n k, n =k  -> S n = k + 1. Proof. ··· Qed.
Lemma SStwise_is_twiceS : ∀ n k, n = 2*k+1 -> S n = 2*(S k). Proof. ··· Qed.

Definition div2 :=
ax_ind (fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))
       (fun n h => match h with
```

Proposition

For all n, either (i) $n = 2k$ or (ii) $n = 2k + 1$ for some k.

Proof.

We prove it by induction on n. Assume it is true for some n. . We reason by cases:

□

Proof assistants

```
Axioms (ℕ : Set) (O : ℕ) (S : ℕ -> ℕ).
Axioms (plus mult: ℕ -> ℕ -> ℕ) (eq:ℕ -> ℕ -> Prop).
Axioms (ax_eqsymm: ∀nm, n = m -> m = n) (ax_eqtrans: ···) ··· (* Peano arithmetic *)
Axiom (ax_ind: ∀ P : ℕ -> Prop,(∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
...

Lemma S_is_addOne : ∀ n k, n =k  -> S n = k + 1. Proof. ··· Qed.
Lemma SStwise_is_twiceS : ∀ n k, n = 2*k+1 -> S n = 2*(S k). Proof. ··· Qed.

Definition div2 :=
ax_ind (fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))
       (fun n h => match h with
```

Proposition

For all n, either (i) $n = 2k$ or (ii) $n = 2k + 1$ for some k.

Proof.

We prove it by induction on n. Assume it is true for some n. . We reason by cases:

- *If $n = 2k$, we show (ii) by choosing k as a witness. Indeed $Sn = S(2k) = 2k + 1$.*
- *Otherwise $n = 2k + 1$, and we show (i) by choosing Sk as a witness. Indeed $Sn = S(2k + 1) = 2k + 2 = 2(Sk)$.*

□

Proof assistants

```
Axioms (ℕ : Set) (O : ℕ) (S : ℕ -> ℕ).
Axioms (plus mult: ℕ -> ℕ -> ℕ) (eq:ℕ -> ℕ -> Prop).
Axioms (ax_eqsymm: ∀nm, n = m -> m = n) (ax_eqtrans: ···) ··· (* Peano arithmetic *)
Axiom (ax_ind: ∀ P : ℕ -> Prop,(∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
···

Lemma S_is_addOne : ∀ n k, n =k  -> S n = k + 1. Proof. ··· Qed.
Lemma SStwise_is_twiceS : ∀ n k, n = 2*k+1 -> S n = 2*(S k). Proof. ··· Qed.

Definition div2 :=
ax_ind (fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))
       (fun n h => match h with
                   | ι1 ⟨k1, is_even⟩ => ι2 ⟨k1, S_is_addOne is_even⟩
                   | ι2 ⟨k2, is_odd⟩ => ι1 ⟨S k2, SStwice_is_twiceS is_odd⟩
```

Proposition

For all n, either (i) $n = 2k$ or (ii) $n = 2k + 1$ for some k.

Proof.

We prove it by induction on n. Assume it is true for some n. . We reason by cases:

- *If $n = 2k$, we show (ii) by choosing k as a witness. Indeed $Sn = S(2k) = 2k + 1$.*
- *Otherwise $n = 2k + 1$, and we show (i) by choosing Sk as a witness. Indeed $Sn = S(2k + 1) = 2k + 2 = 2(Sk)$.*

□

Proof assistants

```
Axioms (ℕ : Set) (O : ℕ) (S : ℕ -> ℕ).
Axioms (plus mult: ℕ -> ℕ -> ℕ) (eq:ℕ -> ℕ -> Prop).
Axioms (ax_eqsymm: ∀nm, n = m -> m = n) (ax_eqtrans: ···) ··· (* Peano arithmetic *)
Axiom (ax_ind: ∀ P : ℕ -> Prop,(∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
···

Lemma S_is_addOne : ∀ n k, n =k  -> S n = k + 1. Proof. ··· Qed.
Lemma SStwise_is_twiceS : ∀ n k, n = 2*k+1 -> S n = 2*(S k). Proof. ··· Qed.

Definition div2 :=
ax_ind (fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))
       (fun n h => match h with
                   | ι1 ⟨k1, is_even⟩ => ι2 ⟨k1, S_is_addOne is_even⟩
                   | ι2 ⟨k2, is_odd⟩ => ι1 ⟨S k2, SStwice_is_twiceS is_odd⟩
                   end
       )
       (ι1 ⟨O, ax_eqsymm (mult_O 2)⟩_{fun k => O=2*k}).
```

Proposition

For all n, either (i) $n = 2k$ or (ii) $n = 2k + 1$ for some k.

Proof.

We prove it by induction on n. Assume it is true for some n. . We reason by cases:

- *If $n = 2k$, we show (ii) by choosing k as a witness. Indeed $Sn = S(2k) = 2k + 1$.*
- *Otherwise $n = 2k + 1$, and we show (i) by choosing Sk as a witness. Indeed $Sn = S(2k + 1) = 2k + 2 = 2(Sk)$.*

For the base case, we show (i) by choosing O as a witness, as $O = 2 * O$. □

Proof assistants

```
Definition div2 :=
ax_ind (fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))
       (fun n h => match h with
                   | ι1 ⟨k1, is_even⟩ => ι2 ⟨k1, S_is_addOne is_even⟩
                   | ι2 ⟨k2, is_odd⟩ => ι1 ⟨S k2, SStwice_is_twiceS is_odd⟩
                   end
       )
       (ι1 ⟨O, ax_eqsymm (mult_O 2)⟩_{fun k => O=2*k}).
```

Proposition

For all n, either (i) $n = 2k$ or (ii) $n = 2k + 1$ for some k.

Proof.

We prove it by induction on n. Assume it is true for some n. . We reason by cases:

- *If $n = 2k$, we show (ii) by choosing k as a witness. Indeed $Sn = S(2k) = 2k + 1$.*
- *Otherwise $n = 2k + 1$, and we show (i) by choosing Sk as a witness. Indeed $Sn = S(2k + 1) = 2k + 2 = 2(Sk)$.*

For the base case, we show (i) by choosing O as a witness, as $O = 2 * O$. □

Proof assistants

```
Definition div2 :=
ax_ind (fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))
       (fun (n:ℕ) (h: ∃k, n=2*k \/ ∃k, n=2*k+1) => match h with
          | ι1 ⟨k1, is_even⟩ =>
            @or_intror (∃k, S n=2*k) (∃k, S n=2*k+1)
               (@ex_intro ℕ (fun k'=> S n=2*k'+1) k1 (S_is_addOne n (2*k1) is_even))
          | ι2 ⟨k2, is_odd⟩ =>
            @or_introl (∃k, S n=2*k) (∃k, S n=2*k+1)
               (@ex_intro ℕ (fun k'=> n=2*k') (S k2) (SStistS n k2 is_even))
          end
       )
       (ι1 ⟨O, ax_eqsymm (mult_O 2)⟩_{fun k => O=2*k}).
```

Proposition

For all n, either (i) $n = 2k$ or (ii) $n = 2k + 1$ for some k.

Proof.

We prove it by induction on n. for some $n \in \mathbb{N}$, $(\exists k \in \mathbb{N}, n = 2k) \vee (\exists k \in \mathbb{N}, n = 2k + 1)$. We reason by cases:

- *If $n = 2k$, we show the right part of $(\exists k' \in \mathbb{N}, Sn = 2k) \vee (\exists k' \in \mathbb{N}, Sn = 2k' + 1)$. To show $\exists k' \in \mathbb{N}, Sn = 2k' + 1$, we choose k as a witness $\cdots$*
- *Otherwise $n = 2k + 1$, and we show the left part of $(\exists k' \in \mathbb{N}, Sn = 2k) \vee (\exists k' \in \mathbb{N}, Sn = 2k' + 1)$. To show $\exists k' \in \mathbb{N}, Sn = 2k'$, we choose Sk as a witness $\cdots$*

For the base case, we show (i) by choosing O as a witness, as $O = 2 * O$. □

Elaboration and type inference

Elaboration and type inference

Generating Metavariables.

```
Axiom (ax_ind: ∀ P : ℕ -> Prop,(∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
Check (@ex_intro : ∀ (A:Type) (P: A -> Prop) (x : A), P x -> exists x, P x).

Definition div2 :=
ax_ind (fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))
       (fun (n:?T) (h: ?P[n]) => match h with
           | ι1 ⟨k1, is_even⟩ =>
             @or_intror ?A1[n,h,k1,is_even] ?B1[n,h,k1,is_even]
                (@ex_intro ?N1[...] ?P1[n,h,k1,is_even] k1
                      (S_is_addOne ?n1[...] ?k1[...] is_even))
           | ι2 ⟨k2, is_odd⟩ =>
             @or_introl ?A2[n,h,k2,is_odd] ?B2[n,h,k2,is_odd]
                (@ex_intro ?N2[...] ?P2[n,h,k2,is_odd] k2
                      (SStwice_is_twiceS ?n2[...] ?k2[...] is_even))
           end
       )
```

Elaboration and type inference

Generating Metavariables.

```
Axiom (ax_ind: ∀ P : ℕ -> Prop, (∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
Check (@ex_intro : ∀ (A:Type) (P: A -> Prop) (x : A), P x -> exists x, P x).

Definition div2 :=
ax_ind (fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))
       (fun (n:?T) (h: ?P[n]) => match h with
           | ι1 ⟨k1, is_even⟩ =>
              @or_intror ?A1[n,h,k1,is_even] ?B1[n,h,k1,is_even]
                 (@ex_intro ?N1[...] ?P1[n,h,k1,is_even] k1
                       (S_is_addOne ?n1[...] ?k1[...] is_even))
           | ι2 ⟨k2, is_odd⟩ =>
              @or_introl ?A2[n,h,k2,is_odd] ?B2[n,h,k2,is_odd]
                 (@ex_intro ?N2[...] ?P2[n,h,k2,is_odd] k2
                       (SStwice_is_twiceS ?n2[...] ?k2[...] is_even))
           end
       )
```

Generating constraints.

- $?T \leq_\beta \mathbb{N}$
- $?P[n] \leq_\beta$ `(fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))`
- $?P_2$[n,h,is_odd] `(S k2)` $\leq_\beta$ `(S n = 2*S k2)`

Elaboration and type inference

Generating Metavariables.

```
Axiom (ax_ind: ∀ P : ℕ -> Prop, (∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
Check (@ex_intro : ∀ (A:Type) (P: A -> Prop) (x : A), P x -> exists x, P x).

Definition div2 :=
ax_ind (fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))
       (fun (n:?T) (h: ?P[n]) => match h with
           | ι1 ⟨k1, is_even⟩ =>
              @or_intror ?A1[n,h,k1,is_even] ?B1[n,h,k1,is_even]
                 (@ex_intro ?N1[...] ?P1[n,h,k1,is_even] k1
                       (S_is_addOne ?n1[...] ?k1[...] is_even))
           | ι2 ⟨k2, is_odd⟩ =>
              @or_introl ?A2[n,h,k2,is_odd] ?B2[n,h,k2,is_odd]
                 (@ex_intro ?N2[...] ?P2[n,h,k2,is_odd] k2
                       (SStwice_is_twiceS ?n2[...] ?k2[...] is_even))
           end
       )
```

Generating constraints.

- $?T \leq_\beta \mathbb{N}$
- $?P[n] \leq_\beta$ `(fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))`
- $?P_2$[n,h,is_odd] `(S k2)` $\leq_\beta$ `(S n = 2*S k2)`

Solving constraints:

Elaboration and type inference

Generating Metavariables.

```
Axiom (ax_ind: ∀ P : ℕ -> Prop, (∀ n, P n -> P (S n)) -> P 0 -> (∀ n, P n)).
Check (@ex_intro : ∀ (A:Type) (P: A -> Prop) (x : A), P x -> exists x, P x).

Definition div2 :=
ax_ind (fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))
       (fun (n:?T) (h: ?P[n]) => match h with
           | ι1 ⟨k1, is_even⟩ =>
              @or_intror ?A1[n,h,k1,is_even] ?B1[n,h,k1,is_even]
                 (@ex_intro ?N1[...] ?P1[n,h,k1,is_even] k1
                       (S_is_addOne ?n1[...] ?k1[...] is_even))
           | ι2 ⟨k2, is_odd⟩ =>
              @or_introl ?A2[n,h,k2,is_odd] ?B2[n,h,k2,is_odd]
                 (@ex_intro ?N2[...] ?P2[n,h,k2,is_odd] k2
                       (SStwice_is_twiceS ?n2[...] ?k2[...] is_even))
           end
       )
```

Generating constraints.

- $?T \leq_\beta \mathbb{N}$
- $?P[n] \leq_\beta$ `(fun n => (∃ k, n = 2*k) \/ (∃ k, n = 2*k+1))`
- $?P_2$[n,h,is_odd] `(S k2)` $\leq_\beta$ `(S n = 2*S k2)`

Solving constraints: Unification

Higher-Order Unification

- **Unification:** Given t and u, find *substitution* θ s.t. $t[\theta] = u[\theta]$
- **Substitution:** map assigning a values to each free variable
- **Higher-order.** The variables can be arbitrary functions

Higher-Order Unification

- **Unification:** Given t and u, find *substitution* θ s.t. $t[\theta] = u[\theta]$
- **Substitution:** map assigning a values to each free variable
- **Higher-order.** The variables can be arbitrary functions

Example

$t = \forall x\ y \in \mathbb{N}, ?P(x, y, ?Q)$
$u = \forall x\ y \in ?T, ?n(x, y) = x^2 - y$

Higher-Order Unification

- **Unification:** Given t and u, find *substitution* θ s.t. $t[\theta] = u[\theta]$
- **Substitution:** map assigning a values to each free variable
- **Higher-order.** The variables can be arbitrary functions

Example

$t = \forall x\ y \in \mathbb{N}, ?P(x, y, ?Q)$
$u = \forall x\ y \in ?T, ?n(x, y) = x^2 - y$
$\theta = \{?P \mapsto (\lambda abf.a + b^2 = fab);\ \ ?Q \mapsto (\lambda ab.a^2 - b);$
$\quad ?T \mapsto \mathbb{N};\ \ ?n \mapsto (\lambda ab.a + b^2)\ \ \}$

Higher-Order Unification

- **Unification:** Given t and u, find *substitution* θ s.t. $t[\theta] = u[\theta]$
- **Substitution:** map assigning a values to each free variable
- **Higher-order.** The variables can be arbitrary functions

Example

$t = \forall x\ y \in \mathbb{N}, ?P(x, y, ?Q)$
$u = \forall x\ y \in ?T, ?n(x, y) = x^2 - y$
$\theta = \{?P \mapsto (\lambda abf.a + b^2 = fab);\ \ ?Q \mapsto (\lambda ab.a^2 - b);$
$\quad ?T \mapsto \mathbb{N};\ \ ?n \mapsto (\lambda ab.a + b^2)\ \}$
$t\{\!\{\theta\}\!\} = u\{\!\{\theta\}\!\} = \forall x\ y \in \mathbb{N}, x + y^2 = x^2 + y$

Higher-Order Unification

- **Unification:** Given t and u, find *substitution* θ s.t. $t[\theta] = u[\theta]$
- **Substitution:** map assigning a values to each free variable
- **Higher-order.** The variables can be arbitrary functions

Example

$t = \forall x\ y \in \mathbb{N}, ?P(x, y, ?Q)$
$u = \forall x\ y \in ?T, ?n(x, y) = x^2 - y$
$\theta = \{?P \mapsto (\lambda abf.a + b^2 = fab);\ \ ?Q \mapsto (\lambda ab.a^2 - b);$
$\quad ?T \mapsto \mathbb{N};\ \ ?n \mapsto (\lambda ab.a + b^2)\ \}$
$t\{\!\{\theta\}\!\} = u\{\!\{\theta\}\!\} = \forall x\ y \in \mathbb{N}, x + y^2 = x^2 + y$

Theorem (Robinson, 1965)

First-order unification is decidable

Theorem (Huet, 1973)

Higher-order unification is undecidable.

Unification algorithms and proof-assistants

Huet's pre-unification procedure (Huet, 1975)

- Semi-complete algorithm for enumerating unifiers
- Reference algorithm for general unification
- Inefficient in practice and too complex to implement

Unification algorithms and proof-assistants

Huet's pre-unification procedure (Huet, 1975)

- Semi-complete algorithm for enumerating unifiers
- Reference algorithm for general unification
- Inefficient in practice and too complex to implement

Higher-order Pattern Unification (Miller, 1989)

- Restricts metavariable arguments to distinct bound variables ($?X[x_1, \ldots, x_n]$)

Unification algorithms and proof-assistants

Huet's pre-unification procedure (Huet, 1975)

- Semi-complete algorithm for enumerating unifiers
- Reference algorithm for general unification
- Inefficient in practice and too complex to implement

Higher-order Pattern Unification (Miller, 1989)

- Restricts metavariable arguments to distinct bound variables ($?X[x_1, \ldots, x_n]$)
- Has most-general unifiers (is complete)

Unification algorithms and proof-assistants

Huet's pre-unification procedure (Huet, 1975)

- Semi-complete algorithm for enumerating unifiers
- Reference algorithm for general unification
- Inefficient in practice and too complex to implement

Higher-order Pattern Unification (Miller, 1989)

- Restricts metavariable arguments to distinct bound variables ($?X[x_1, \ldots, x_n]$)
- Has most-general unifiers (is complete)
- Efficient and simple implementation

Unification algorithms and proof-assistants

Huet's pre-unification procedure (Huet, 1975)

- Semi-complete algorithm for enumerating unifiers
- Reference algorithm for general unification
- Inefficient in practice and too complex to implement

Higher-order Pattern Unification (Miller, 1989)

- Restricts metavariable arguments to distinct bound variables ($?X[x_1, \ldots, x_n]$)
- Has most-general unifiers (is complete)
- Efficient and simple implementation
- Sometimes too restrictive

Unification algorithms and proof-assistants

Huet's pre-unification procedure (Huet, 1975)

- Semi-complete algorithm for enumerating unifiers
- Reference algorithm for general unification
- Inefficient in practice and too complex to implement

Higher-order Pattern Unification (Miller, 1989)

- Restricts metavariable arguments to distinct bound variables ($?X[x_1, \ldots, x_n]$)
- Has most-general unifiers (is complete)
- Efficient and simple implementation
- Sometimes too restrictive

Practice in proof-assistants.

Unification algorithms and proof-assistants

Huet's pre-unification procedure (Huet, 1975)

- Semi-complete algorithm for enumerating unifiers
- Reference algorithm for general unification
- Inefficient in practice and too complex to implement

Higher-order Pattern Unification (Miller, 1989)

- Restricts metavariable arguments to distinct bound variables ($?X[x_1, \ldots, x_n]$)
- Has most-general unifiers (is complete)
- Efficient and simple implementation
- Sometimes too restrictive

Practice in proof-assistants.

- Miller pattern unification at the core in most cases

Unification algorithms and proof-assistants

Huet's pre-unification procedure (Huet, 1975)

- Semi-complete algorithm for enumerating unifiers
- Reference algorithm for general unification
- Inefficient in practice and too complex to implement

Higher-order Pattern Unification (Miller, 1989)

- Restricts metavariable arguments to distinct bound variables ($?X[x_1, \ldots, x_n]$)
- Has most-general unifiers (is complete)
- Efficient and simple implementation
- Sometimes too restrictive

Practice in proof-assistants.

- Miller pattern unification at the core in most cases
- Adaptated to richer type theories (dependent types, inductives, etc.) (Pfenning, 1991; Abel and Pientka, 2011)

Unification algorithms and proof-assistants

Huet's pre-unification procedure (Huet, 1975)

- Semi-complete algorithm for enumerating unifiers
- Reference algorithm for general unification
- Inefficient in practice and too complex to implement

Higher-order Pattern Unification (Miller, 1989)

- Restricts metavariable arguments to distinct bound variables ($?X[x_1, \ldots, x_n]$)
- Has most-general unifiers (is complete)
- Efficient and simple implementation
- Sometimes too restrictive

Practice in proof-assistants.

- Miller pattern unification at the core in most cases
- Adaptated to richer type theories (dependent types, inductives, etc.) (Pfenning, 1991; Abel and Pientka, 2011)
- Heuristics and user hints for handling general cases (Ziliani and Sozeau, 2017; Saïbi, 1999)

In my internship

Functions-as-constructors Unification. (Libal and Miller, 2016)

- Generalization of pattern unification

In my internship

Functions-as-constructors Unification. (Libal and Miller, 2016)

- Generalization of pattern unification
- Preserves completeness

In my internship

Functions-as-constructors Unification. (Libal and Miller, 2016)

- Generalization of pattern unification
- Preserves completeness
- Relaxes the restrictions on metavariable arguments
  - For example, $?X[F\ x\ (G y), x\ y]$ is allowed

In my internship

Functions-as-constructors Unification. (Libal and Miller, 2016)

- Generalization of pattern unification
- Preserves completeness
- Relaxes the restrictions on metavariable arguments
  - For example, $?X[F\ x\ (G y), x\ y]$ is allowed
- Implemented by Hamana (2019)

In my internship

Functions-as-constructors Unification. (Libal and Miller, 2016)

- Generalization of pattern unification
- Preserves completeness
- Relaxes the restrictions on metavariable arguments
  - For example, $?X[F\ x\ (G y), x\ y]$ is allowed
- Implemented by Hamana (2019)
- No implementation in proof assistants

In my internship

Functions-as-constructors Unification. (Libal and Miller, 2016)

- Generalization of pattern unification
- Preserves completeness
- Relaxes the restrictions on metavariable arguments
  - For example, $?X[F\ x\ (G y), x\ y]$ is allowed
- Implemented by Hamana (2019)
- No implementation in proof assistants

Goals

- Formally specify FCU in the context of the Rocq proof-assistant's type theory
- Implement it for Rocq

In my internship

Functions-as-constructors Unification. (Libal and Miller, 2016)

- Generalization of pattern unification
- Preserves completeness
- Relaxes the restrictions on metavariable arguments
  - For example, $?X[F\ x\ (G y), x\ y]$ is allowed
- Implemented by Hamana (2019)
- No implementation in proof assistants

Goals

- Formally specify FCU in the context of the Rocq proof-assistant's type theory
- Implement it for Rocq

In my internship

Functions-as-constructors Unification. (Libal and Miller, 2016)

- Generalization of pattern unification
- Preserves completeness
- Relaxes the restrictions on metavariable arguments
  - For example, $?X[F\ x\ (G y), x\ y]$ is allowed
- Implemented by Hamana (2019)
- No implementation in proof assistants

Goals

- Formally specify FCU in the context of the Rocq proof-assistant's type theory
- Implement it for Rocq

Polymorphic, Cumulative Caclulus of Constructions (PCUC)

A fragment of Rocq's meta-theory (Sozeau et al., 2025)

$$
\begin{array}{rcll}
t, u, A, B, \ldots & ::= & s \mid x \mid c[\vec{i}] \mid \lambda x^A.T \mid \Pi x^A.B \mid tu \mid ?X[\sigma] & \textbf{Terms} \\
s & ::= & \textbf{Prop} \mid \textbf{Type}(\vec{p}) & \textit{Sorts} \\
\sigma, \tau, \rho, \ldots & ::= & \cdot \mid \sigma, t & \textit{Spines}
\end{array}
$$

Polymorphic, Cumulative Caclulus of Constructions (PCUC)

A fragment of Rocq's meta-theory (Sozeau et al., 2025)

$$
\begin{aligned}
t, u, A, B, \ldots &::= s \mid x \mid c[\vec{i}] \mid \lambda x^A.T \mid \Pi x^A.B \mid tu \mid ?X[\sigma] && \textbf{Terms} \\
s &::= \textbf{Prop} \mid \textbf{Type}(\vec{p}) && \textit{Sorts} \\
\sigma, \tau, \rho, \ldots &::= \cdot \mid \sigma, t && \textit{Spines} \\
\Gamma, \Delta, \Psi, \ldots &::= \langle\rangle \mid \Gamma, x : A && \textbf{Local contexts}
\end{aligned}
$$

Polymorphic, Cumulative Caclulus of Constructions (PCUC)

A fragment of Rocq's meta-theory (Sozeau et al., 2025)

$$
\begin{array}{rll}
t, u, A, B, \ldots ::= & s \mid x \mid c[\vec{i}] \mid \lambda x^A.T \mid \Pi x^A.B \mid tu \mid ?X[\sigma] & \textbf{Terms} \\
s ::= & \textbf{Prop} \mid \textbf{Type}(\vec{p}) & \textit{Sorts} \\
\sigma, \tau, \rho, \ldots ::= & \cdot \mid \sigma, t & \textit{Spines} \\
\\
\Gamma, \Delta, \Psi, \ldots ::= & \langle\rangle \mid \Gamma, x : A & \textbf{Local contexts} \\
\Sigma ::= & \langle\rangle \mid \Sigma, ?X : A[\Psi] & \textbf{Meta-Contexts}
\end{array}
$$

Polymorphic, Cumulative Caclulus of Constructions (PCUC)

A fragment of Rocq's meta-theory (Sozeau et al., 2025)

$$
\begin{array}{rll}
t, u, A, B, \ldots ::= & s \mid x \mid c[\vec{i}] \mid \lambda x^A.T \mid \Pi x^A.B \mid tu \mid ?X[\sigma] & \textbf{Terms} \\
s ::= & \textbf{Prop} \mid \textbf{Type}(\vec{p}) & \textit{Sorts} \\
\sigma, \tau, \rho, \ldots ::= & \cdot \mid \sigma, t & \textit{Spines} \\
\\
\Gamma, \Delta, \Psi, \ldots ::= & \langle\rangle \mid \Gamma, x : A & \textbf{Local contexts} \\
\Sigma ::= & \langle\rangle \mid \Sigma, ?X : A[\Psi] & \textbf{Meta-Contexts} \\
E ::= & \langle\rangle \mid E, c : \forall \Phi.A & \textbf{Global contexts}
\end{array}
$$

Polymorphic, Cumulative Caclulus of Constructions (PCUC)

A fragment of Rocq's meta-theory (Sozeau et al., 2025)

$$
\begin{array}{rll}
t, u, A, B, \ldots ::= & s \mid x \mid c[\vec{i}] \mid \lambda x^A.T \mid \Pi x^A.B \mid tu \mid ?X[\sigma] & \textbf{Terms} \\
s ::= & \textbf{Prop} \mid \textbf{Type}(\vec{p}) & \textit{Sorts} \\
\sigma, \tau, \rho, \ldots ::= & \cdot \mid \sigma, t & \textit{Spines}
\end{array}
$$

$$
\begin{array}{rll}
\Gamma, \Delta, \Psi, \ldots ::= & \langle\rangle \mid \Gamma, x : A & \textbf{Local contexts} \\
\Sigma ::= & \langle\rangle \mid \Sigma, ?X : A[\Psi] & \textbf{Meta-Contexts} \\
E ::= & \langle\rangle \mid E, c : \forall \Phi.A & \textbf{Global contexts}
\end{array}
$$

Example

$E = \mathbb{N} : \textbf{Type}(i),\ O : \mathbb{N},\ S : \Pi x^{\mathbb{N}}.\mathbb{N}, \ldots,$
$\quad \text{ax\_ind} : \Pi P^{\textbf{Prop} \to \mathbb{N}}.(\Pi n^{\mathbb{N}}.Pn \to P(Sn)) \to P0 \to \Pi n^{\mathbb{N}}.Pn$

Polymorphic, Cumulative Caclulus of Constructions (PCUC)

A fragment of Rocq's meta-theory (Sozeau et al., 2025)

$$
\begin{array}{rcll}
t, u, A, B, \ldots & ::= & s \mid x \mid c[\vec{i}] \mid \lambda x^A.T \mid \Pi x^A.B \mid tu \mid ?X[\sigma] & \textbf{Terms} \\
s & ::= & \textbf{Prop} \mid \textbf{Type}(\vec{p}) & \textit{Sorts} \\
\sigma, \tau, \rho, \ldots & ::= & \cdot \mid \sigma, t & \textit{Spines}
\end{array}
$$

$$
\begin{array}{rcll}
\Gamma, \Delta, \Psi, \ldots & ::= & \langle\rangle \mid \Gamma, x : A & \textbf{Local contexts} \\
\Sigma & ::= & \langle\rangle \mid \Sigma, ?X : A[\Psi] & \textbf{Meta-Contexts} \\
E & ::= & \langle\rangle \mid E, c : \forall \Phi.A & \textbf{Global contexts}
\end{array}
$$

Example

$E = \mathbb{N} : \textbf{Type}(i),\ O : \mathbb{N},\ S : \Pi x^{\mathbb{N}}.\mathbb{N}, \ldots,$
$\quad \text{ax\_ind} : \Pi P^{\textbf{Prop}\to\mathbb{N}}.(\Pi n^{\mathbb{N}}.Pn \to P(Sn)) \to P0 \to \Pi n^{\mathbb{N}}.Pn$
$\Sigma = ?P : \mathbb{N} \to \textbf{Prop}[n : \mathbb{N}, h : \ldots, is\_odd : \exists k, n = 2 * k]$

Polymorphic, Cumulative Caclulus of Constructions (PCUC)

A fragment of Rocq's meta-theory (Sozeau et al., 2025)

$$t, u, A, B, \ldots ::= s \mid x \mid c[\vec{i}] \mid \lambda x^A.T \mid \Pi x^A.B \mid tu \mid ?X[\sigma] \qquad \textbf{Terms}$$
$$s ::= \textbf{Prop} \mid \textbf{Type}(\vec{p}) \qquad \textit{Sorts}$$
$$\sigma, \tau, \rho, \ldots ::= \cdot \mid \sigma, t \qquad \textit{Spines}$$

$$\Gamma, \Delta, \Psi, \ldots ::= \langle\rangle \mid \Gamma, x : A \qquad \textbf{Local contexts}$$
$$\Sigma ::= \langle\rangle \mid \Sigma, ?X : A[\Psi] \qquad \textbf{Meta-Contexts}$$
$$E ::= \langle\rangle \mid E, c : \forall \Phi.A \qquad \textbf{Global contexts}$$

Example

$E = \mathbb{N} : \textbf{Type}(i),\ O : \mathbb{N},\ S : \Pi x^{\mathbb{N}}.\mathbb{N}, \ldots,$
$\quad \text{ax\_ind} : \Pi P^{\textbf{Prop} \to \mathbb{N}}.(\Pi n^{\mathbb{N}}.Pn \to P(Sn)) \to P0 \to \Pi n^{\mathbb{N}}.Pn$
$\Sigma = ?P : \mathbb{N} \to \textbf{Prop}[n : \mathbb{N}, h : \ldots, is\_odd : \exists k, n = 2 * k]$

$$p, q, r, \ldots ::= i + n \qquad (n \in \mathbb{N}) \qquad \textit{Level expressions}$$

Polymorphic, Cumulative Caclulus of Constructions (PCUC)

A fragment of Rocq's meta-theory (Sozeau et al., 2025)

$$t, u, A, B, \ldots ::= s \mid x \mid c[\vec{i}] \mid \lambda x^A.T \mid \Pi x^A.B \mid tu \mid ?X[\sigma] \qquad \textbf{Terms}$$
$$s ::= \textbf{Prop} \mid \textbf{Type}(\vec{p}) \qquad \textit{Sorts}$$
$$\sigma, \tau, \rho, \ldots ::= \cdot \mid \sigma, t \qquad \textit{Spines}$$

$$\Gamma, \Delta, \Psi, \ldots ::= \langle\rangle \mid \Gamma, x : A \qquad \textbf{Local contexts}$$
$$\Sigma ::= \langle\rangle \mid \Sigma, ?X : A[\Psi] \qquad \textbf{Meta-Contexts}$$
$$E ::= \langle\rangle \mid E, c : \forall \Phi.A \qquad \textbf{Global contexts}$$

Example

$E = \mathbb{N} : \textbf{Type}(i),\ O : \mathbb{N},\ S : \Pi x^{\mathbb{N}}.\mathbb{N}, \ldots,$
$\quad \text{ax\_ind} : \Pi P^{\textbf{Prop}\to\mathbb{N}}.(\Pi n^{\mathbb{N}}.Pn \to P(Sn)) \to P0 \to \Pi n^{\mathbb{N}}.Pn$
$\Sigma = ?P : \mathbb{N} \to \textbf{Prop}[n : \mathbb{N}, h : \ldots, is\_odd : \exists k, n = 2 * k]$

$$p, q, r, \ldots ::= i + n \quad (n \in \mathbb{N}) \qquad \textit{Level expressions}$$
$$\Phi ::= (\vec{i}; \mathscr{C}) \qquad \textbf{Universe contexts}$$
$$\mathscr{C} ::= \top \mid \mathscr{C} \wedge i \mathscr{R} j \quad (\mathscr{R} \in \{=, \leq, <\}) \qquad \textit{Constraint sets}$$

Polymorphic, Cumulative Caclulus of Constructions (PCUC)

A fragment of Rocq's meta-theory (Sozeau et al., 2025)

$$t, u, A, B, \ldots ::= s \mid x \mid c[\vec{i}] \mid \lambda x^A.T \mid \Pi x^A.B \mid tu \mid ?X[\sigma] \qquad \textbf{Terms}$$
$$s ::= \textbf{Prop} \mid \textbf{Type}(\vec{p}) \qquad \textit{Sorts}$$
$$\sigma, \tau, \rho, \ldots ::= \cdot \mid \sigma, t \qquad \textit{Spines}$$

$$\Gamma, \Delta, \Psi, \ldots ::= \langle\rangle \mid \Gamma, x : A \qquad \textbf{Local contexts}$$
$$\Sigma ::= \langle\rangle \mid \Sigma, ?X : A[\Psi] \qquad \textbf{Meta-Contexts}$$
$$E ::= \langle\rangle \mid E, c : \forall \Phi.A \qquad \textbf{Global contexts}$$

Example

$E = \mathbb{N} : \textbf{Type}(i),\ O : \mathbb{N},\ S : \Pi x^{\mathbb{N}}.\mathbb{N}, \ldots,$ $\qquad \Phi = (i; \emptyset)$
$\quad \text{ax\_ind} : \Pi P^{\textbf{Prop} \to \mathbb{N}}.(\Pi n^{\mathbb{N}}.Pn \to P(Sn)) \to P0 \to \Pi n^{\mathbb{N}}.Pn$
$\Sigma = ?P : \mathbb{N} \to \textbf{Prop}[n : \mathbb{N}, h : \ldots, is\_odd : \exists k, n = 2 * k]$

$$p, q, r, \ldots ::= i + n \qquad (n \in \mathbb{N}) \qquad \textit{Level expressions}$$
$$\Phi ::= (\vec{i}; \mathscr{C}) \qquad \textbf{Universe contexts}$$
$$\mathscr{C} ::= \top \mid \mathscr{C} \wedge i \mathscr{R} j \qquad (\mathscr{R} \in \{=, \leq, <\}) \qquad \textit{Constraint sets}$$

Semantics of PCUC

Typing. $\boxed{\Phi|\Sigma|\Gamma \vdash t : A}$

Semantics of PCUC

Typing. $\boxed{\Phi|\Sigma|\Gamma \vdash t : A}$

$$\frac{\Phi|\Sigma|\Gamma, x : A \vdash t : B \qquad \Phi|\Sigma|\Gamma \vdash \Pi x^A.B : s}{\Phi|\Sigma|\Gamma \vdash \lambda x^A.t : \Pi x^A.B}$$

$$\frac{\Phi|\Sigma|\Gamma \vdash t : \Pi x^A.B \qquad \Phi|\Sigma|\Gamma \vdash u : A}{\Phi|\Sigma|\Gamma \vdash tu : B\{x \mapsto u\}}$$

Semantics of PCUC

Typing. $\boxed{\Phi|\Sigma|\Gamma \vdash t : A}$

$$\frac{\Phi|\Sigma|\Gamma, x : A \vdash t : B \qquad \Phi|\Sigma|\Gamma \vdash \Pi x^A.B : s}{\Phi|\Sigma|\Gamma \vdash \lambda x^A.t : \Pi x^A.B} \qquad \frac{\Phi|\Sigma|\Gamma \vdash t : \Pi x^A.B \qquad \Phi|\Sigma|\Gamma \vdash u : A}{\Phi|\Sigma|\Gamma \vdash tu : B\{x \mapsto u\}}$$

$$\frac{\Phi|\Sigma|\Gamma \vdash \mathcal{WF} \qquad c : \forall(\vec{l}; \mathscr{C}).A \in E \qquad \Phi \vDash \mathscr{C}\{\vec{l} \mapsto \vec{i}\}}{\Phi|\Sigma|\Gamma \vdash c[\vec{i}] : A\{\vec{l} \mapsto \vec{i}\}}$$

Semantics of PCUC

Typing. $\boxed{\Phi|\Sigma|\Gamma \vdash t : A}$

$$\frac{\Phi|\Sigma|\Gamma, x : A \vdash t : B \qquad \Phi|\Sigma|\Gamma \vdash \Pi x^A.B : s}{\Phi|\Sigma|\Gamma \vdash \lambda x^A.t : \Pi x^A.B} \qquad \frac{\Phi|\Sigma|\Gamma \vdash t : \Pi x^A.B \qquad \Phi|\Sigma|\Gamma \vdash u : A}{\Phi|\Sigma|\Gamma \vdash tu : B\{x \mapsto u\}}$$

$$\frac{\Phi|\Sigma|\Gamma \vdash \mathcal{WF} \qquad c : \forall(\vec{l}; \mathscr{C}).A \in E \qquad \Phi \vDash \mathscr{C}\{\vec{l} \mapsto \vec{i}\}}{\Phi|\Sigma|\Gamma \vdash c[\vec{i}] : A\{\vec{l} \mapsto \vec{i}\}}$$

Convertibility. $\boxed{\Phi \vdash t =_\beta u}$ $\boxed{\Phi \vdash A \leq_\beta B}$

Semantics of PCUC

Typing. $\boxed{\Phi|\Sigma|\Gamma \vdash t : A}$

$$\frac{\Phi|\Sigma|\Gamma, x : A \vdash t : B \qquad \Phi|\Sigma|\Gamma \vdash \Pi x^A.B : s}{\Phi|\Sigma|\Gamma \vdash \lambda x^A.t : \Pi x^A.B} \qquad \frac{\Phi|\Sigma|\Gamma \vdash t : \Pi x^A.B \qquad \Phi|\Sigma|\Gamma \vdash u : A}{\Phi|\Sigma|\Gamma \vdash tu : B\{x \mapsto u\}}$$

$$\frac{\Phi|\Sigma|\Gamma \vdash \mathcal{WF} \qquad c : \forall(\vec{l}; \mathscr{C}).A \in E \qquad \Phi \vDash \mathscr{C}\{\vec{l} \mapsto \vec{i}\}}{\Phi|\Sigma|\Gamma \vdash c[\vec{i}] : A\{\vec{l} \mapsto \vec{i}\}}$$

Convertibility. $\boxed{\Phi \vdash t =_\beta u}$ $\boxed{\Phi \vdash A \leq_\beta B}$

- $=_\beta$ the congruence relation generated by $(\lambda x^A.t)u \to_\beta t\{x \mapsto u\}$

Semantics of PCUC

Typing. $\boxed{\Phi|\Sigma|\Gamma \vdash t : A}$

$$\frac{\Phi|\Sigma|\Gamma, x : A \vdash t : B \qquad \Phi|\Sigma|\Gamma \vdash \Pi x^A.B : s}{\Phi|\Sigma|\Gamma \vdash \lambda x^A.t : \Pi x^A.B} \qquad \frac{\Phi|\Sigma|\Gamma \vdash t : \Pi x^A.B \qquad \Phi|\Sigma|\Gamma \vdash u : A}{\Phi|\Sigma|\Gamma \vdash tu : B\{x \mapsto u\}}$$

$$\frac{\Phi|\Sigma|\Gamma \vdash \mathcal{WF} \qquad c : \forall(\vec{l}; \mathscr{C}).A \in E \qquad \Phi \vDash \mathscr{C}\{\vec{l} \mapsto \vec{i}\}}{\Phi|\Sigma|\Gamma \vdash c[\vec{i}] : A\{\vec{l} \mapsto \vec{i}\}}$$

Convertibility. $\boxed{\Phi \vdash t =_\beta u}$ $\boxed{\Phi \vdash A \leq_\beta B}$

- $=_\beta$ the congruence relation generated by $(\lambda x^A.t)u \to_\beta t\{x \mapsto u\}$
- $\leq_\beta$ a partial order extending $=_\beta$ with
  - $\mathbf{Type}(\vec{p}) \leq_\beta \mathbf{Type}(\vec{p}')$ whenever $\Phi \vDash \vec{p} \leq \vec{p}'$

Semantics of PCUC

Typing. $\boxed{\Phi|\Sigma|\Gamma \vdash t : A}$

$$\frac{\Phi|\Sigma|\Gamma, x : A \vdash t : B \qquad \Phi|\Sigma|\Gamma \vdash \Pi x^A.B : s}{\Phi|\Sigma|\Gamma \vdash \lambda x^A.t : \Pi x^A.B} \qquad \frac{\Phi|\Sigma|\Gamma \vdash t : \Pi x^A.B \qquad \Phi|\Sigma|\Gamma \vdash u : A}{\Phi|\Sigma|\Gamma \vdash tu : B\{x \mapsto u\}}$$

$$\frac{\Phi|\Sigma|\Gamma \vdash \mathcal{WF} \qquad c : \forall(\vec{l}; \mathscr{C}).A \in E \qquad \Phi \vDash \mathscr{C}\{\vec{l} \mapsto \vec{i}\}}{\Phi|\Sigma|\Gamma \vdash c[\vec{i}] : A\{\vec{l} \mapsto \vec{i}\}}$$

Convertibility. $\boxed{\Phi \vdash t =_\beta u}$ $\boxed{\Phi \vdash A \leq_\beta B}$

- $=_\beta$ the congruence relation generated by $(\lambda x^A.t)u \to_\beta t\{x \mapsto u\}$
- $\leq_\beta$ a partial order extending $=_\beta$ with
  - $\mathbf{Type}(\vec{p}) \leq_\beta \mathbf{Type}(\vec{p}')$ whenever $\Phi \vDash \vec{p} \leq \vec{p}'$
  - If $t \leq_\beta u$ and $A =_\beta B$ then $\Pi x^A.t \leq_\beta \Pi x^B.u$ and $tA \leq_\beta tB$

Unification in PCUC

$$\theta ::= \cdot \mid \theta, ?X \mapsto t[\Psi] \qquad \textbf{(Meta substitutions)}$$

$$\phi ::= \vec{l} \mapsto \vec{u} \qquad (|\vec{l}| = |\vec{u}|) \qquad \textbf{(Level substitutions)}$$

$$\langle \phi; \theta \rangle \qquad \textbf{(Substitution pairs)}$$

$$\boxed{\Phi' | \Sigma \vdash \langle \phi; \theta \rangle : \Phi | \Sigma'}$$

Unification in PCUC

$$\theta ::= \cdot \mid \theta, ?X \mapsto t[\Psi] \qquad \textbf{(Meta substitutions)}$$

$$\phi ::= \vec{l} \mapsto \vec{u} \quad (|\vec{l}| = |\vec{u}|) \qquad \textbf{(Level substitutions)}$$

$$\langle \phi; \theta \rangle \qquad \textbf{(Substitution pairs)}$$

$$\boxed{\Phi' | \Sigma \vdash \langle \phi; \theta \rangle : \Phi | \Sigma'}$$

Definition (Unification problem)

A unification problem between two terms, denoted $\Phi | \Sigma | \Gamma \vdash t^A \approx^?_{\mathscr{R}} u^A$ is said to be solved by a substitution pair $\Phi' | \Sigma \vdash \langle \phi; \theta \rangle : \Phi | \Sigma'$ if $t\{\!\{\langle \phi; \theta \rangle\}\!\} \mathscr{R} u\{\!\{\langle \phi; \theta \rangle\}\!\}$.

Unification in PCUC

$$\theta ::= \cdot \mid \theta, ?X \mapsto t[\Psi] \qquad \textbf{(Meta substitutions)}$$

$$\phi ::= \vec{l} \mapsto \vec{u} \qquad (|\vec{l}| = |\vec{u}|) \qquad \textbf{(Level substitutions)}$$

$$\langle \phi; \theta \rangle \qquad \textbf{(Substitution pairs)}$$

$$\boxed{\Phi' | \Sigma \vdash \langle \phi; \theta \rangle : \Phi | \Sigma'}$$

Definition (Unification problem)

A unification problem between two terms, denoted $\Phi|\Sigma|\Gamma \vdash t^A \approx^?_{\mathscr{R}} u^A$ is said to be solved by a substitution pair $\Phi' | \Sigma \vdash \langle \phi; \theta \rangle : \Phi | \Sigma'$ if $t\{\!\{\langle \phi; \theta \rangle\}\!\} \mathscr{R} u\{\!\{\langle \phi; \theta \rangle\}\!\}$.

Definition (Substitution subsumption)

We say that $\langle \theta_1; \phi_1 \rangle \leq \langle \theta_2; \phi_2 \rangle$ if there exists $\langle \theta^*; \phi^* \rangle$ such that $\langle \theta_2; \phi_2 \rangle = \langle \theta^*; \phi^* \rangle \circ \langle \theta_1; \phi_1 \rangle$

Unification in PCUC

$$\theta ::= \cdot \mid \theta, ?X \mapsto t[\Psi] \qquad \textbf{(Meta substitutions)}$$

$$\phi ::= \vec{l} \mapsto \vec{u} \quad (|\vec{l}| = |\vec{u}|) \qquad \textbf{(Level substitutions)}$$

$$\langle \phi; \theta \rangle \qquad \textbf{(Substitution pairs)}$$

$$\boxed{\Phi' | \Sigma \vdash \langle \phi; \theta \rangle : \Phi | \Sigma'}$$

Definition (Unification problem)

A unification problem between two terms, denoted $\Phi|\Sigma|\Gamma \vdash t^A \approx^?_{\mathscr{R}} u^A$ is said to be solved by a substitution pair $\Phi' | \Sigma \vdash \langle \phi; \theta \rangle : \Phi | \Sigma'$ if $t\{\!\{\langle \phi; \theta \rangle\}\!\} \mathscr{R} u\{\!\{\langle \phi; \theta \rangle\}\!\}$.

Definition (Substitution subsumption)

We say that $\langle \theta_1; \phi_1 \rangle \leq \langle \theta_2; \phi_2 \rangle$ if there exists $\langle \theta^*; \phi^* \rangle$ such that $\langle \theta_2; \phi_2 \rangle = \langle \theta^*; \phi^* \rangle \circ \langle \theta_1; \phi_1 \rangle$

Definition (Most general unifier)

A solution to a unification problem is a most general unifier if it is minimal for $\leq$.

The FCU class of problems

$$
\begin{array}{rcll}
P & ::= & N_e \mid \lambda x^P.P \mid \Pi x^P.P & \textbf{Patterns} \\
N_e & ::= & E[x] \mid E[c[\vec{i}]] \mid s \mid ?X[\sigma_r] & \textbf{Neutral patterns} \\
E & ::= & \square \mid EP & \textbf{Neutral contexts} \\
\sigma_r & ::= & \cdot \mid \sigma_r, A & \textbf{Argument spines} \\
E_r & ::= & \square A \mid E_r A & \textbf{Restricted contexts} \\
A & ::= & x \mid E_r[x] \mid s \mid E_r[c[\vec{i}]] & \textbf{Restricted arguments}
\end{array}
$$

The FCU class of problems

$$
\begin{array}{rll}
P & ::= N_e \mid \lambda x^P.P \mid \Pi x^P.P & \textbf{Patterns} \\
N_e & ::= E[x] \mid E[c[\vec{i}]] \mid s \mid ?X[\sigma_r] & \textbf{Neutral patterns} \\
E & ::= \Box \mid EP & \textbf{Neutral contexts} \\
\sigma_r & ::= \cdot \mid \sigma_r, A & \textbf{Argument spines} \\
E_r & ::= \Box A \mid E_r A & \textbf{Restricted contexts} \\
A & ::= x \mid E_r[x] \mid s \mid E_r[c[\vec{i}]] & \textbf{Restricted arguments}
\end{array}
$$

Subterm modulo-universe $\boxed{\Phi \vdash A \sqsubseteq A'}$ and $\boxed{\Phi \vdash A \equiv A'}$

$\Phi \vdash c[\vec{i}] \equiv c[\vec{j}]$ iff for some Φ' extending Φ, $\vec{i} = \vec{j}$

The FCU class of problems

$$
\begin{array}{rll}
P & ::= N_e \mid \lambda x^P.P \mid \Pi x^P.P & \textbf{Patterns} \\
N_e & ::= E[x] \mid E[c[\vec{i}]] \mid s \mid ?X[\sigma_r] & \textbf{Neutral patterns} \\
E & ::= \Box \mid EP & \textbf{Neutral contexts} \\
\sigma_r & ::= \cdot \mid \sigma_r, A & \textbf{Argument spines} \\
E_r & ::= \Box A \mid E_r A & \textbf{Restricted contexts} \\
A & ::= x \mid E_r[x] \mid s \mid E_r[c[\vec{i}]] & \textbf{Restricted arguments}
\end{array}
$$

Subterm modulo-universe $\boxed{\Phi \vdash A \sqsubseteq A'}$ and $\boxed{\Phi \vdash A \equiv A'}$

$\Phi \vdash c[\vec{i}] \equiv c[\vec{j}]$ iff for some Φ' extending Φ, $\vec{i} = \vec{j}$

Definition (FCU problems)

A unification problem is an FCU problem if

1. The unified terms are *patterns*

The FCU class of problems

$$
\begin{array}{rll}
P & ::= N_e \mid \lambda x^P.P \mid \Pi x^P.P & \textbf{Patterns} \\
N_e & ::= E[x] \mid E[c[\vec{i}]] \mid s \mid ?X[\sigma_r] & \textbf{Neutral patterns} \\
E & ::= \Box \mid EP & \textbf{Neutral contexts} \\
\sigma_r & ::= \cdot \mid \sigma_r, A & \textbf{Argument spines} \\
E_r & ::= \Box A \mid E_r A & \textbf{Restricted contexts} \\
A & ::= x \mid E_r[x] \mid s \mid E_r[c[\vec{i}]] & \textbf{Restricted arguments}
\end{array}
$$

Subterm modulo-universe $\boxed{\Phi \vdash A \sqsubseteq A'}$ and $\boxed{\Phi \vdash A \equiv A'}$

$\Phi \vdash c[\vec{i}] \equiv c[\vec{j}]$ iff for some Φ' extending Φ, $\vec{i} = \vec{j}$

Definition (FCU problems)

A unification problem is an FCU problem if

1. The unified terms are *patterns*
2. For every $?X[A_1, \ldots, A_n]$, there is no $i \neq j$ s.t. $\Phi \vdash A_i \sqsubseteq A_j$

The FCU class of problems

$$
\begin{array}{rll}
P & ::= N_e \mid \lambda x^P.P \mid \Pi x^P.P & \textbf{Patterns} \\
N_e & ::= E[x] \mid E[c[\vec{i}]] \mid s \mid ?X[\sigma_r] & \textbf{Neutral patterns} \\
E & ::= \Box \mid EP & \textbf{Neutral contexts} \\
\sigma_r & ::= \cdot \mid \sigma_r, A & \textbf{Argument spines} \\
E_r & ::= \Box A \mid E_r A & \textbf{Restricted contexts} \\
A & ::= x \mid E_r[x] \mid s \mid E_r[c[\vec{i}]] & \textbf{Restricted arguments}
\end{array}
$$

Subterm modulo-universe $\boxed{\Phi \vdash A \sqsubseteq A'}$ and $\boxed{\Phi \vdash A \equiv A'}$

$\Phi \vdash c[\vec{i}] \equiv c[\vec{j}]$ iff for some Φ' extending Φ, $\vec{i} = \vec{j}$

Definition (FCU problems)

A unification problem is an FCU problem if

1. The unified terms are *patterns*
2. For every $?X[A_1, \ldots, A_n]$, there is no $i \neq j$ s.t. $\Phi \vdash A_i \sqsubseteq A_j$
3. For every $?X[A_1, \ldots, A_n]$ and $?X'[B_1, \ldots, B_m]$, we don't have $\Phi \vdash A_i \sqsubset B_j$

FCU algorithm for PCUC (1/2)

Unifying types.

$$\underbrace{\Phi|\Sigma|\Gamma \vdash t^A \approx^?_{\mathscr{R}} u^B}_{\textit{input}} \dashv \underbrace{\Phi' \mid \Sigma'; \theta}_{\textit{output}}$$

$$\frac{\begin{array}{c}\Phi^*|\Sigma|\Gamma \vdash T \approx^?_{\mathscr{R}} U \dashv \Phi_1 \mid \Sigma_1; \theta_1 \\ \Phi_1|\Sigma_1|\Gamma\{\!\{\theta_1\}\!\} \vdash t\{\!\{\theta_1\}\!\} \approx^?_{\mathscr{R}} u\{\!\{\theta_1\}\!\} \dashv \Phi_2 \mid \Sigma_2; \theta_2\end{array}}{\Phi|\Sigma|\Gamma \vdash t^T \approx^?_{\mathscr{R}} u^U \dashv \Phi_2 \mid \Sigma_2; \theta_2 \circ \theta_1}\ \textsc{UnifTypes}$$

FCU algorithm for PCUC (1/2)

Unifying types.

$$\underbrace{\Phi|\Sigma|\Gamma \vdash t^A \approx^?_{\mathscr{R}} u^B}_{\textit{input}} \dashv \underbrace{\Phi' \mid \Sigma'; \theta}_{\textit{output}}$$

$$\frac{\begin{array}{c}\Phi^*|\Sigma|\Gamma \vdash T \approx^?_{\mathscr{R}} U \dashv \Phi_1 \mid \Sigma_1; \theta_1 \\ \Phi_1|\Sigma_1|\Gamma\{\!\{\theta_1\}\!\} \vdash t\{\!\{\theta_1\}\!\} \approx^?_{\mathscr{R}} u\{\!\{\theta_1\}\!\} \dashv \Phi_2 \mid \Sigma_2; \theta_2\end{array}}{\Phi|\Sigma|\Gamma \vdash t^T \approx^?_{\mathscr{R}} u^U \dashv \Phi_2 \mid \Sigma_2; \theta_2 \circ \theta_1}\ \textsc{UnifTypes}$$

Syntax-directed algorithm

$$\underbrace{\Phi|\Sigma|\Gamma \vdash t \approx^?_{\mathscr{R}} u}_{\textit{input}} \dashv \underbrace{\Phi' \mid \Sigma'; \theta}_{\textit{output}}$$

FCU algorithm for PCUC (1/2)

Unifying types.

$$\underbrace{\Phi|\Sigma|\Gamma \vdash t^A \approx^?_{\mathscr{R}} u^B}_{\textit{input}} \dashv \underbrace{\Phi' \mid \Sigma'; \theta}_{\textit{output}}$$

$$\frac{\Phi^*|\Sigma|\Gamma \vdash T \approx^?_{\mathscr{R}} U \dashv \Phi_1 \mid \Sigma_1; \theta_1 \qquad \Phi_1|\Sigma_1|\Gamma\{\!\{\theta_1\}\!\} \vdash t\{\!\{\theta_1\}\!\} \approx^?_{\mathscr{R}} u\{\!\{\theta_1\}\!\} \dashv \Phi_2 \mid \Sigma_2; \theta_2}{\Phi|\Sigma|\Gamma \vdash t^T \approx^?_{\mathscr{R}} u^U \dashv \Phi_2 \mid \Sigma_2; \theta_2 \circ \theta_1}\ \textsc{UnifTypes}$$

Syntax-directed algorithm

$$\underbrace{\Phi|\Sigma|\Gamma \vdash t \approx^?_{\mathscr{R}} u}_{\textit{input}} \dashv \underbrace{\Phi' \mid \Sigma'; \theta}_{\textit{output}}$$

$$\frac{?X : \_[\Psi] \qquad \Phi \mid \Sigma \vdash \mathsf{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} N = N' \dashv \Phi'; \Sigma'; \theta}{\Phi|\Sigma|\Gamma \vdash ?X[\sigma] \approx^?_{\mathscr{R}} N \dashv \Phi \mid \Sigma'\{\!\{(?X \backslash N')_\Psi\}\!\}; (?X \backslash N')_\Psi \circ \theta}\ \textsc{Meta-InstR}$$

FCU algorithm for PCUC (2/2)

Inversion. $\boxed{\Phi \,|\, \Sigma \vdash \mathsf{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} N = N' \dashv \Phi'; \Sigma'; \theta}$

- Finds instances of $A \in \sigma_r$ in N and inverts it.
- Applies pruning

FCU algorithm for PCUC (2/2)

Inversion. $\boxed{\Phi \,|\, \Sigma \vdash \mathsf{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} N = N' \dashv \Phi'; \Sigma'; \theta}$

- Finds instances of $A \in \sigma_r$ in N and inverts it.
- Applies pruning

Example

$$?X[fx] \approx^{?}_{=_\beta} g\ (?Y[x])\ (fx)$$

FCU algorithm for PCUC (2/2)

Inversion. $\boxed{\Phi \,|\, \Sigma \vdash \mathsf{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} N = N' \dashv \Phi'; \Sigma'; \theta}$

- Finds instances of $A \in \sigma_r$ in N and inverts it.
- Applies pruning

Example

$$?X[fx] \approx^{?}_{=_\beta} g\ (?Y[x])\ (fx)$$

$\Sigma = ?X : T[z : T], ?Y : T[z : T]$
$\sigma_r = f\ x$
$\Psi = z : \_$
$N = g\ (?Y[x])(fx)$

FCU algorithm for PCUC (2/2)

Inversion. $\boxed{\Phi \,|\, \Sigma \vdash \text{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} N = N' \dashv \Phi'; \Sigma'; \theta}$

- Finds instances of $A \in \sigma_r$ in N and inverts it.
- Applies pruning

Example

$$?X[fx] \approx^{?}_{=_\beta} g\ (?Y[x])\ (fx)$$

$\Sigma = ?X : T[z : T], ?Y : T[z : T]$
$\sigma_r = f\ x$
$\Psi = z : \_$
$N = g\ (?Y[x])(fx)$

$$\Phi \,|\, \Sigma \vdash \text{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} N = g\ (?Y'[])\ z \dashv \Phi'; \Sigma'; \theta$$

Implementation

- As an extension to the Unicoq plugin
- Reasonable implementation complexity, no efficiency drawback
- Need for more refined benchmarks to evaluate usefulness

Implementation

- As an extension to the Unicoq plugin
- Reasonable implementation complexity, no efficiency drawback
- Need for more refined benchmarks to evaluate usefulness

```
Set Unicoq FCU.
Definition div2 :=
  ax_ind (fun n => (exists k, n = 2*k) \/ (exists k, n = 2*k+1))
    (fun n h => match h with
                | ι1 ⟨k1, is_even⟩ => ι2 ⟨k1, aux is_even⟩
                | ι2 ⟨k2, is_odd⟩ => ι1 ⟨S k2, aux' is_odd⟩
                end
    )
    (ι1 ⟨O, ax_eqsymm (mult_O 2)⟩_{fun k => O=2*k}).

Unset Unicoq FCU.
Fail Definition div2' :=
  ax_ind (fun n => (exists k, n = 2*k) \/ (exists k, n = 2*k+1))
    (fun n h => match h with
                | ι1 ⟨k1, is_even⟩ => ι2 ⟨k1, aux is_even⟩
                | ι2 ⟨k2, is_odd⟩ => ι1 ⟨S k2, aux' is_odd⟩
                end
    )
    (ι1 ⟨O, ax_eqsymm (mult_O 2)⟩_{fun k => O=2*k}).
```

Conclusion

Synthesis.

- Formal specification of syntax directed FCU for PCUC à la Pfenning (1991) and Ziliani and Sozeau (2017)
- No proof of soundness and completeness yet
- Implementation in Unicoq

Conclusion

Synthesis.

- Formal specification of syntax directed FCU for PCUC à la Pfenning (1991) and Ziliani and Sozeau (2017)
- No proof of soundness and completeness yet
- Implementation in Unicoq

Future work.

- Prove soundness and completeness in Rocq
- Extend FCU unification with records/Σ-types (Abel and Pientka, 2011; Kovács and Bocquet, 2023)
- Handle definitions and not only axiomatic constants (Pfenning and Schürmann, 1998)

Conclusion

Synthesis.

- Formal specification of syntax directed FCU for PCUC à la Pfenning (1991) and Ziliani and Sozeau (2017)
- No proof of soundness and completeness yet
- Implementation in Unicoq

Future work.

- Prove soundness and completeness in Rocq
- Extend FCU unification with records/Σ-types (Abel and Pientka, 2011; Kovács and Bocquet, 2023)
- Handle definitions and not only axiomatic constants (Pfenning and Schürmann, 1998)

Long term goals.

- Develop a modular formal framework for studying unification heuristics and their compositions
- Specify the full elaboration and unification implementation of Rocq

Bibliography I

Abel, Andreas and Brigitte Pientka (2011). "Higher-Order Dynamic Pattern Unification for Dependent Types and Records". In: *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*. Ed. by C.-H. Luke Ong. Vol. 6690. Lecture Notes in Computer Science. Springer, pp. 10–26. DOI: 10.1007/978-3-642-21691-6_5. URL: https://doi.org/10.1007/978-3-642-21691-6%5C_5.

Hamana, Makoto (2019). "How to prove decidability of equational theories with second-order computation analyser SOL". In: *J. Funct. Program.* 29, e20. DOI: 10.1017/S0956796819000157. URL: https://doi.org/10.1017/S0956796819000157.

Huet, Gérard P. (1973). "The Undecidability of Unification in Third Order Logic". In: *Inf. Control.* 22.3, pp. 257–267. DOI: 10.1016/S0019-9958(73)90301-X. URL: https://doi.org/10.1016/S0019-9958(73)90301-X.

— (1975). "A Unification Algorithm for Typed lambda-Calculus". In: *Theor. Comput. Sci.* 1.1, pp. 27–57. DOI: 10.1016/0304-3975(75)90011-0. URL: https://doi.org/10.1016/0304-3975(75)90011-0.

Kovács, András and Rafaël Bocquet (2023). *Nested Pattern Unification*. URL: https://andraskovacs.github.io/pdfs/wits23abstract.pdf.

Libal, Tomer and Dale Miller (2016). "Functions-as-Constructors Higher-Order Unification". In: *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*. Ed. by Delia Kesner and Brigitte Pientka. Vol. 52. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 26:1–26:17. DOI: 10.4230/LIPICS.FSCD.2016.26. URL: https://doi.org/10.4230/LIPIcs.FSCD.2016.26.

Miller, Dale (1989). "A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification". In: *Extensions of Logic Programming, International Workshop, Tübingen, FRG, December 8-10, 1989, Proceedings*. Ed. by Peter Schroeder-Heister. Vol. 475. Lecture Notes in Computer Science. Springer, pp. 253–281. DOI: 10.1007/BFB0038698. URL: https://doi.org/10.1007/BFb0038698.

Bibliography II

Pfenning, Frank (1991). "Unification and Anti-Unification in the Calculus of Constructions". In: *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*. IEEE Computer Society, pp. 74–85. DOI: 10.1109/LICS.1991.151632. URL: https://doi.org/10.1109/LICS.1991.151632.

Pfenning, Frank and Carsten Schürmann (1998). "Algorithms for Equality and Unification in the Presence of Notational Definitions". In: *Types for Proofs and Programs, International Workshop TYPES '98, Kloster Irsee, Germany, March 27-31, 1998, Selected Papers*. Ed. by Thorsten Altenkirch, Wolfgang Naraschewski, and Bernhard Reus. Vol. 1657. Lecture Notes in Computer Science. Springer, pp. 179–193. DOI: 10.1007/3-540-48167-2_13. URL: https://doi.org/10.1007/3-540-48167-2%5C_13.

Robinson, J. A. (Jan. 1965). "A Machine-Oriented Logic Based on the Resolution Principle". In: *J. ACM* 12.1, pp. 23–41. ISSN: 0004-5411. DOI: 10.1145/321250.321253. URL: https://doi.org/10.1145/321250.321253.

Saïbi, Amokrane (1999). "Outils Génériques de Modélisation et de Démonstration pour la Formalisation des Mathématiques en Théorie des Types. Application à la Théorie des Catégories. (Formalization of Mathematics in Type Theory. Generic tools of Modelisation and Demonstration. Application to Category Theory)". PhD thesis. Pierre and Marie Curie University, Paris, France. URL: https://tel.archives-ouvertes.fr/tel-00523810.

Sozeau, Matthieu et al. (2025). "Correct and Complete Type Checking and Certified Erasure for Coq, in Coq". In: *J. ACM* 72.1, 8:1–8:74. DOI: 10.1145/3706056. URL: https://doi.org/10.1145/3706056.

Ziliani, Beta and Matthieu Sozeau (2017). "A comprehensible guide to a new unifier for CIC including universe polymorphism and overloading". In: *J. Funct. Program.* 27, e10. DOI: 10.1017/S0956796817000028. URL: https://doi.org/10.1017/S0956796817000028.