Nested Pattern Unification

András Kovács[1] j.w.w. Rafaël Bocquet[1]

[1]Eötvös Loránd University

28 Aug 2023, Braga, 2nd Workshop on the Implementation of Type Systems

Pattern Unification

$\alpha\, x_0\, x_1 \ldots x_n \stackrel{?}{=}$ rhs is solvable if

1. x_i are distinct bound vars.
2. rhs only depends on x_i bound vars.
3. α does not occur in rhs.

Then $\alpha := \lambda\, x_0\, x_1 \ldots x_n.$ rhs.

All major dependently typed languages use pattern unification with some extensions.

Reduction to patterns

Reduce non-pattern problems to pattern ones. Examples:

1. η-contraction:

$$\alpha\,(\lambda\,x.\,f\,x) \stackrel{?}{=} \mathsf{rhs} \quad \Rightarrow \quad \alpha\,f \stackrel{?}{=} \mathsf{rhs}$$

2. Σ-elimination:

$$\begin{array}{rl} & x : (a : A) \times B\,a \vdash \alpha\,(\mathsf{fst}\,x) \stackrel{?}{=} \mathsf{fst}\,x \\ \Rightarrow & a : A, b : B\,a \vdash \alpha\,a \stackrel{?}{=} a \end{array}$$

$$\begin{array}{rl} & \alpha : A \times B \to C \vdash \alpha\,(a,\ b) \stackrel{?}{=} a \\ \Rightarrow & \alpha' : A \to B \to C, \alpha := \mathsf{uncurry}\,\alpha' \vdash \alpha'\,a\,b \stackrel{?}{=} a \end{array}$$

Issues with reduction to patterns

η-contraction for Σ is expensive (needs conversion checks).

Σ-elimination is potentially expensive and unnecessarily η-long.

$\alpha\,(\lambda\, x\, y.\, f\, y\, x) \stackrel{?}{=}$ rhs is not reducible to a pattern problem.

Nested patterns

We directly solve a larger class of problems, called **nested pattern** problems.

Advantages:

- Conjecture: whenever f is a definitional isomorphism, $\alpha\,(f x) \stackrel{?}{=} x$ is solvable as $\alpha := f^{-1}$.
- A single pass on the spine and rhs is enough. No η-contraction, Σ-elimination or administrative metas are needed.

Implementations:

- `https://github.com/AndrasKovacs/sett`
- `https://gitlab.com/RafaelBocquet/obstt`

Algorithm

Basic idea:

- λ, pairing and projection is allowed in spines, recursively.
- In $\Gamma \vdash \alpha\,\text{spine} \stackrel{?}{=} \text{rhs}$, rhs lives in Γ but the eventual solution is *closed* w.r.t. bound vars. So we need a substitution to make rhs depend only on the λ-bound vars in the solution.
- Recursing on the spine, we generate λ-s and pairings in the solution and also build the mentioned substitution.

It's not realistic to fully explain the algorithm in this talk so I'll focus on examples. I include a more detailed spec on the slides though.

Scope notation

The actual implementation uses De Bruijn indices and levels.

Scope notation

The actual implementation uses De Bruijn indices and levels.

Informally, we use names and distinguish two scopes by naming:

- x, y, z, f, g, h live in the **problem scope**, which is Γ in $\Gamma \vdash \alpha\,\mathsf{spine} \stackrel{?}{=} \mathsf{rhs}$.
- a, b, c, d, e live in the **solution scope**.

Scope notation

The actual implementation uses De Bruijn indices and levels.

Informally, we use names and distinguish two scopes by naming:

- x, y, z, f, g, h live in the **problem scope**, which is Γ in $\Gamma \vdash \alpha\,\mathsf{spine} \overset{?}{=} \mathsf{rhs}$.
- a, b, c, d, e live in the **solution scope**.

Notation: $\Gamma \vdash \alpha \overset{a}{x} \overset{b}{y} \overset{?}{=} x$ marks the solution scope binders already in the problem statement.

Scope notation

The actual implementation uses De Bruijn indices and levels.

Informally, we use names and distinguish two scopes by naming:

- x, y, z, f, g, h live in the **problem scope**, which is Γ in $\Gamma \vdash \alpha\,\mathsf{spine} \stackrel{?}{=} \mathsf{rhs}$.
- a, b, c, d, e live in the **solution scope**.

Notation: $\Gamma \vdash \alpha\,\overset{a}{x}\,\overset{b}{y} \stackrel{?}{=} x$ marks the solution scope binders already in the problem statement.

Here, pattern inversion yields $[x \mapsto a,\ y \mapsto b]$, and the solution is

$$\alpha := \lambda\, a\, b.\, x[x \mapsto a,\ y \mapsto b]$$

which is $\alpha := \lambda\, a\, b.\, a$.

Partial substitutions

Simple pattern inversion: outputs a *partial* map from problem scope vars to solution scope vars.

Partial substitutions

Simple pattern inversion: outputs a *partial* map from problem scope vars to solution scope vars.

We generalize this to maps sending variables to certain *partial values*.

Partial values are generated by:

- Variables, λ, application, projection, pairing.
- A formal TOP and a BOT value, both inhabiting any type.

Partial substitutions

Simple pattern inversion: outputs a *partial* map from problem scope vars to solution scope vars.

We generalize this to maps sending variables to certain *partial values*.

Partial values are generated by:

- Variables, λ, application, projection, pairing.
- A formal TOP and a BOT value, both inhabiting any type.

Partial values have a partial ordering where TOP is top and BOT is bottom.

- If a meta solution contains TOP or BOT, that's a unification error.
- TOP signals an ambiguity from non-linearity, while BOT signals an out-of-scope dependency.

Informal examples (1)

In a α spine $\stackrel{?}{=}$ rhs problem, we recurse on spine while accumulating a partial substitution. The starting substitution maps all vars to BOT, and at each step it may get more defined.

Informal examples (1)

In a $\alpha\,\mathsf{spine} \stackrel{?}{=} \mathsf{rhs}$ problem, we recurse on spine while accumulating a partial substitution. The starting substitution maps all vars to BOT, and at each step it may get more defined.

- To solve: $\alpha\,(\mathsf{fst}^{a}\,x) \stackrel{?}{=} \mathsf{fst}\,x$.

Informal examples (1)

In a $\alpha\,\mathsf{spine} \stackrel{?}{=} \mathsf{rhs}$ problem, we recurse on spine while accumulating a partial substitution. The starting substitution maps all vars to BOT, and at each step it may get more defined.

- To solve: $\alpha\,(\overset{a}{\mathsf{fst}}\,x) \stackrel{?}{=} \mathsf{fst}\,x$.
- We start with $\sigma := [x \mapsto \mathsf{BOT}]$.

Informal examples (1)

In a $\alpha\,\mathsf{spine} \stackrel{?}{\doteq} \mathsf{rhs}$ problem, we recurse on spine while accumulating a partial substitution. The starting substitution maps all vars to BOT, and at each step it may get more defined.

- To solve: $\alpha\,(\overset{a}{\mathsf{fst}}\,x) \stackrel{?}{\doteq} \mathsf{fst}\,x$.
- We start with $\sigma := [x \mapsto \mathsf{BOT}]$.
- We try to extend σ with $[\mathsf{fst}\,x \mapsto a]$.

Informal examples (1)

In a $\alpha\,\text{spine} \stackrel{?}{=} \text{rhs}$ problem, we recurse on spine while accumulating a partial substitution. The starting substitution maps all vars to BOT, and at each step it may get more defined.

- To solve: $\alpha\,(\overset{a}{\text{fst}}\,x) \stackrel{?}{=} \text{fst}\,x$.
- We start with $\sigma := [x \mapsto \text{BOT}]$.
- We try to extend σ with $[\text{fst}\,x \mapsto a]$.
- This decomposes to $[x \mapsto (a, \text{BOT})]$.

Informal examples (1)

In a $\alpha\,\mathsf{spine} \stackrel{?}{=} \mathsf{rhs}$ problem, we recurse on spine while accumulating a partial substitution. The starting substitution maps all vars to BOT, and at each step it may get more defined.

- To solve: $\alpha\,(\overset{a}{\mathsf{fst}}\,x) \stackrel{?}{=} \mathsf{fst}\,x$.
- We start with $\sigma := [x \mapsto \mathsf{BOT}]$.
- We try to extend σ with $[\mathsf{fst}\,x \mapsto a]$.
- This decomposes to $[x \mapsto (a,\ \mathsf{BOT})]$.
- Now $\sigma \sqcup [x \mapsto (a,\ \mathsf{BOT})]$ is $[x \mapsto (a,\ \mathsf{BOT})]$.

Informal examples (1)

In a $\alpha\,\text{spine} \stackrel{?}{=} \text{rhs}$ problem, we recurse on spine while accumulating a partial substitution. The starting substitution maps all vars to BOT, and at each step it may get more defined.

- To solve: $\alpha\,(\overset{a}{\text{fst}}\,x) \stackrel{?}{=} \text{fst}\,x$.
- We start with $\sigma := [x \mapsto \text{BOT}]$.
- We try to extend σ with $[\text{fst}\,x \mapsto a]$.
- This decomposes to $[x \mapsto (a,\ \text{BOT})]$.
- Now $\sigma \sqcup [x \mapsto (a,\ \text{BOT})]$ is $[x \mapsto (a,\ \text{BOT})]$.
- We recurse on the spine, and the result will be under $\lambda\,a$.

Informal examples (1)

In a $\alpha\,\text{spine} \stackrel{?}{=} \text{rhs}$ problem, we recurse on spine while accumulating a partial substitution. The starting substitution maps all vars to BOT, and at each step it may get more defined.

- To solve: $\alpha\,(\overset{a}{\text{fst}}\,x) \stackrel{?}{=} \text{fst}\,x$.
- We start with $\sigma := [x \mapsto \text{BOT}]$.
- We try to extend σ with $[\text{fst}\,x \mapsto a]$.
- This decomposes to $[x \mapsto (a,\ \text{BOT})]$.
- Now $\sigma \sqcup [x \mapsto (a,\ \text{BOT})]$ is $[x \mapsto (a,\ \text{BOT})]$.
- We recurse on the spine, and the result will be under $\lambda\,a$.
- The rest of the spine is empty, so we return rhs substituted with σ.

Informal examples (1)

In a $\alpha\,\text{spine} \stackrel{?}{=} \text{rhs}$ problem, we recurse on spine while accumulating a partial substitution. The starting substitution maps all vars to BOT, and at each step it may get more defined.

- To solve: $\alpha\,(\overset{a}{\text{fst}}\,x) \stackrel{?}{=} \text{fst}\,x$.
- We start with $\sigma := [x \mapsto \text{BOT}]$.
- We try to extend σ with $[\text{fst}\,x \mapsto a]$.
- This decomposes to $[x \mapsto (a, \text{BOT})]$.
- Now $\sigma \sqcup [x \mapsto (a, \text{BOT})]$ is $[x \mapsto (a, \text{BOT})]$.
- We recurse on the spine, and the result will be under $\lambda\, a$.
- The rest of the spine is empty, so we return rhs substituted with σ.
- Hence, the solution is $\lambda\, a.\, (\text{fst}\,x)[\sigma]$, that is, $\lambda\, a.\, a$.

Informal examples (2)

Informal examples (2)

- To solve: $\alpha \overset{a}{x} \overset{b}{x} \overset{?}{=} x$.

Informal examples (2)

- To solve: $\alpha \overset{a}{x} \overset{b}{x} \overset{?}{=} x$.
- We start again with $\sigma := [x \mapsto \mathsf{BOT}]$.

Informal examples (2)

- To solve: $\alpha \overset{a}{x} \overset{b}{x} \overset{?}{=} x$.
- We start again with $\sigma := [x \mapsto \mathsf{BOT}]$.
- Processing the next spine entry, we get $[x \mapsto a]$.

Informal examples (2)

- To solve: $\alpha \overset{a}{x} \overset{b}{x} \overset{?}{=} x$.
- We start again with $\sigma := [x \mapsto \mathsf{BOT}]$.
- Processing the next spine entry, we get $[x \mapsto a]$.
- Next, we do $\sigma := \sigma \sqcup [x \mapsto b]$.

Informal examples (2)

- To solve: $\alpha \overset{a}{x} \overset{b}{x} \stackrel{?}{=} x$.
- We start again with $\sigma := [x \mapsto \mathsf{BOT}]$.
- Processing the next spine entry, we get $[x \mapsto a]$.
- Next, we do $\sigma := \sigma \sqcup [x \mapsto b]$.
- We get $[x \mapsto \mathsf{TOP}]$ because the lub of distinct variables is TOP.

Informal examples (2)

- To solve: $\alpha \overset{a}{x} \overset{b}{x} \stackrel{?}{=} x$.
- We start again with $\sigma := [x \mapsto \mathsf{BOT}]$.
- Processing the next spine entry, we get $[x \mapsto a]$.
- Next, we do $\sigma := \sigma \sqcup [x \mapsto b]$.
- We get $[x \mapsto \mathsf{TOP}]$ because the lub of distinct variables is TOP.
- Hence, the solution is $\lambda\, a\, b.\mathsf{TOP}$, i.e. a non-linearity error.

Informal examples (3)

Informal examples (3)

- To solve: $\alpha\,(\lambda\, x\, y. \overset{a}{}\, f\, y\, x) \stackrel{?}{=} f$.

Informal examples (3)

- To solve: $\alpha\,(\lambda\, x\, \overset{a}{y}.\, f\, y\, x) \overset{?}{=} f$.
- We try to decompose $[(\lambda\, x\, y.\, f\, y\, x) \mapsto a]$.

Informal examples (3)

- To solve: $\alpha\,(\lambda\, x\,\overset{a}{y}.\, f\,y\,x) \overset{?}{=} f$.
- We try to decompose $[(\lambda\, x\, y.\, f\,y\,x) \mapsto a]$.
- That gets us $x, y \vdash f\,\overset{b}{y}\,\overset{c}{x} \mapsto a\,x\,y$.

Informal examples (3)

- To solve: $\alpha\,(\lambda\,x\,\overset{a}{y}.\,f\,y\,x) \overset{?}{\doteq} f$.
- We try to decompose $[(\lambda\,x\,y.\,f\,y\,x) \mapsto a]$.
- That gets us $x, y \vdash f\,\overset{b}{y}\,\overset{c}{x} \mapsto a\,x\,y$.
- This is a nested pattern problem!

Informal examples (3)

- To solve: $\alpha\,(\overset{a}{\lambda\,x\,y.\,f\,y\,x}) \overset{?}{=} f$.
- We try to decompose $[(\lambda\,x\,y.\,f\,y\,x) \mapsto a]$.
- That gets us $x, y \vdash f\,\overset{b}{y}\,\overset{c}{x} \mapsto a\,x\,y$.
- This is a nested pattern problem!
- Note that f is OK here in head position, but e.g. x would not be!

Informal examples (3)

- To solve: $\alpha\,\overset{a}{(\lambda\, x\, y.\, f\, y\, x)} \overset{?}{=} f$.
- We try to decompose $[(\lambda\, x\, y.\, f\, y\, x) \mapsto a]$.
- That gets us $x, y \vdash f\,\overset{b}{y}\,\overset{c}{x} \mapsto a\, x\, y$.
- This is a nested pattern problem!
- Note that f is OK here in head position, but e.g. x would not be!
- We'll see in the details later that *solvable* and *parameter* vars need to be distinguished.

Informal examples (3)

- To solve: $\alpha\,(\lambda\, x\, \overset{a}{y}.\, f\, y\, x) \stackrel{?}{=} f$.
- We try to decompose $[(\lambda\, x\, y.\, f\, y\, x) \mapsto a]$.
- That gets us $x, y \vdash f\, \overset{b}{y}\, \overset{c}{x} \mapsto a\, x\, y$.
- This is a nested pattern problem!
- Note that f is OK here in head position, but e.g. x would not be!
- We'll see in the details later that *solvable* and *parameter* vars need to be distinguished.
- Recursive solving gets us $[f \mapsto \lambda\, b\, c.\, a\, c\, b]$.

Informal examples (3)

- To solve: $\alpha\,(\overset{a}{\lambda\, x\, y.\, f\, y\, x}) \stackrel{?}{=} f$.
- We try to decompose $[(\lambda\, x\, y.\, f\, y\, x) \mapsto a]$.
- That gets us $x, y \vdash f\, \overset{b}{y}\, \overset{c}{x} \mapsto a\, x\, y$.
- This is a nested pattern problem!
- Note that f is OK here in head position, but e.g. x would not be!
- We'll see in the details later that *solvable* and *parameter* vars need to be distinguished.
- Recursive solving gets us $[f \mapsto \lambda\, b\, c.\, a\, c\, b]$.
- This gets lub-ed to the top σ, so we get $\lambda\, a\, b\, c.\, a\, c\, b$ as overall solution.

Specification (1/4)

solve tries to produce a solution for $\Gamma \vdash \alpha \, \text{spine} \stackrel{?}{=} \text{rhs}$:

```
solve Γ α sp rhs := solveSp Γ • [xᵢ ↦ BOT] α sp rhs
```

solveSp is a worker function that iterates through the spine and accumulates the partial substitution σ. Δ is the solution scope.

```
solveSp Γ Δ σ α sp rhs
```

invertArg tries to extend σ with the mapping $[t \mapsto a \; \text{sp}']$. The problem scope Γ is split to three regions, "unsolvable" (`Γᵤ`), "solvable" (`Γₛ`), and "parameters" (`Γₚ`). A nested pattern is only solvable if headed by a solvable variable.

```
invertArg Γᵤ Γₛ Γₚ Δ σ t (a sp')
```

solveNestedSp produces a solution from a nested spine.

```
solveNestedSp Γᵤ Γₛ Γₚ Δ σ sp (a sp')
```

Specification (2/4)

```
solve Γ α sp rhs = solveSp Γ • [xᵢ ↦ BOT] sp rhs

solveSp Γ Δ σ α [] rhs =
  rhs[σ, α ↦ BOT]

solveSp Γ Δ σ α (app t :: sp) rhs =
  λ a. solveSp Γ (Δ, a) (invertArg • Γ • Δ σ t (a [])) α sp rhs

solveSp Γ Δ σ α (fst :: sp) rhs =
  (solveSp Γ Δ σ α sp rhs, freshMeta Δ)

solveSp Γ Δ σ α (snd :: sp) rhs =
  (freshMeta Δ, solveSp Γ Δ σ α sp rhs)

solveSp Γ Δ σ α _ rhs =
  fail
```

Specification (3/4)

```
invertArg Γᵤ Γₛ Γₚ Δ σ (x sp) (a sp')
  | x ∈ Γᵤ ∨ x ∈ Γₚ =
    fail
  | x ∈ Γₛ =
    σ ⊔ [x ↦ solveNestedSp (Γᵤ,Γₛ) Γₚ • Δ σ sp (a sp')]

invertArg Γᵤ Γₛ Γₚ Δ σ (t, u) (a sp) =
  let σ = invertArg Γᵤ Γₛ Γₚ Δ σ t (a (sp :: .fst)) in
  invertArg Γᵤ Γₛ Γₚ Δ σ u (a (sp :: .snd))

invertArg Γᵤ Γₛ Γₚ Δ σ (λ x. t) (a sp) =
  invertArg Γᵤ Γₛ (Γₚ, x) Δ t (a (sp :: app x))

invertArg Γᵤ Γₛ Γₚ Δ σ _ (a sp) =
  fail
```

Specification (4/4)

```
solveNestedSp Γᵤ Γₛ Γₚ Δ σ [] (a sp') =
  a (sp'[σ])

solveNestedSp Γᵤ Γₛ Γₚ σ (app t :: sp) (a sp') =
  λ a'. solveNestedSp Γᵤ Γₛ Γₚ (Δ, a')
                      (invertArg Γᵤ Γₛ Γₚ Δ σ t (a' []))
                      sp (a sp')

solveNestedSp Γᵤ Γₛ Γₚ σ (fst :: sp) (a sp') =
  (solveNestedSp Γᵤ Γₛ Γₚ σ sp (a sp'), BOT)

solveNestedSp Γᵤ Γₛ Γₚ σ (snd :: sp) (a sp') =
  (BOT, solveNestedSp Γᵤ Γₛ Γₚ σ sp (a sp'))

solveNestedSp Γᵤ Γₛ Γₚ σ _ (a sp') =
  fail
```

Not explained here

- Integration into NbE.
- Implementation of partial substitution.
- Support for unit η and typed inversion.
- Analogous generalization of *pruning* where we can prune dependencies from inside nested Π/Σ types.