Nested Pattern Unification

András Kovács and Rafaël Bocquet

Eötvös Loránd University

Miller's pattern unification algorithm [Mil91] is widely used in implementations of dependently typed languages. In practice, the basic algorithm is usually extended in several ways, for example with some support for Σ-types with η-rule. Abel and Pientka described a way to handle Σ in unification by getting rid of it [AP11]: currying and the "type-theoretic axiom of choice"[1] can be repeadetly applied to eliminate Σ from unification problems. Agda uses this approach, although not to the full extent described by Abel and Pientka.

However, Σ-elimination can be inefficient and produce unnecessarily η-long solutions. Also, it does not handle the definitional isomorphism $((a : A)(b : B) \to C\,a\,b) \simeq ((b : B)(a : A) \to C\,a\,b)$. For example, assuming a metavariable α and a bound variable f, Agda cannot solve $\alpha\,(\mathsf{flip}\,f) =? f$ with $\alpha := \mathsf{flip}$.

We propose a new algorithm, called *nested pattern unification*, which handles Σ efficiently and directly, without Σ-elimination or η-reductions, and also handles permutations of record fields and function inputs. We conjecture that for a bound variable x and *any* definitional isomorphism g, the algorithm solves $\alpha\,(g\,x) =? x$ with $\alpha := g^{-1}$.

The algorithm. Miller's pattern unification problems are of the form $\alpha\,\sigma =? t$, where σ is an application spine consisting of distinct bound variables. In this case, there is a *partial inverse substitution* σ^{-1} such that the problem is solvable if $t[\sigma^{-1}]$ is defined and α does not occur in t. Here, σ^{-1} is always a partial map from variables to variables (i.e. a partial renaming). We generalize this to maps from variables to *partial terms* in a *negative linear fragment* of the type theory, consisting of linear λ, pairing, variables, projections, applications and undefined values. We build σ^{-1} by starting with a completely undefined map and extending it with the "inversion" of each term in σ. However, terms in σ can be themselves neutral spines, headed by bound variables that we want to map to something in σ^{-1}. At such points, we try to compute a mapping by recursive pattern unification. Hence the term "nested pattern unification". The precise specification is a bit complicated, so we trace a particular example below.

1. Assuming a bound variable f, we aim to solve $\alpha\,(\lambda\,x\,y.\,f\,y\,x) =? f$.
2. We try to invert $\lambda\,x\,y.\,f\,y\,x$ by mapping it to a fresh variable a. Goal: $(\lambda\,x\,y.\,f\,y\,x) \mapsto a$.
3. We decompose the inversion problem to $\forall\,x\,y.\,f\,y\,x \mapsto a\,x\,y$. Here $\forall$ is a metatheoretic quantifier.
4. $f\,y\,x \mapsto a\,x\,y$ is now a nested pattern.
5. We invert the spine $y\,x$ to $[x \mapsto b,\, y \mapsto c]$ for fresh b, c. We produce the solution $[f \mapsto \lambda\,b\,c.\,a\,c\,b]$.
6. Now $[f \mapsto \lambda\,b\,c.\,a\,c\,b]$ is the top-level partial inverse substitution, so we solve α to $\lambda\,a.\,f[f \mapsto \lambda\,b\,c.\,a\,c\,b]$, which is $\lambda\,a\,b\,c.\,a\,c\,b$.

[1] That is, $((a : A) \to (b : B\,a) \times C\,a\,b) \simeq ((b : (a : A) \to B\,a) \times ((a : A) \to C\,a\,(b\,a)))$.

For an example which involves a partial term, consider $\alpha\,(\mathsf{fst}\,x) =? (\mathsf{fst}\,x)$. Here we decompose $\mathsf{fst}\,x \mapsto a$ as $x \mapsto (a, \mathsf{UNDEF})$, yielding the solution $\alpha := \lambda\,a.\,a$. The UNDEF disappears after we substitute the right hand side; but $\mathsf{snd}\,x$ there would have resulted in an error. Thus, UNDEF represents "out of scope" errors.

We implemented two slightly different versions of the algorithm, one in OCaml [Boc22] and one in Haskell [KB22], in experimental implementations of observational type theories. The implementations are tightly integrated with normalization-by-evaluation and they do not use global fresh name generation or naive term substitution.

References

[AP11] Andreas Abel and Brigitte Pientka. Higher-order dynamic pattern unification for dependent types and records. In C.-H. Luke Ong, editor, *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*, volume 6690 of *Lecture Notes in Computer Science*, pages 10–26. Springer, 2011. doi:10.1007/978-3-642-21691-6_5.

[Boc22] Rafaël Bocquet, 2022. URL: https://gitlab.com/RafaelBocquet/obstt.

[KB22] András Kovács and Rafaël Bocquet, 2022. URL: https://github.com/AndrasKovacs/sett.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991. doi:10.1093/logcom/1.4.497.