Pattern matching without K

Jesper Cockx, Dominique Devriese, and Frank Piessens

DistriNet – KU Leuven

Dependent pattern matching [Coquand, 1992] is a technique for writing functions in languages based on dependent type theory, such as Agda [Norell, 2007], Coq [Sozeau, 2010], and Idris [Brady, 2013]. It allows us to define functions in a style similar to functional programming languages such as Haskell. Additionally, dependent pattern matching can be used to write *proofs* in the form of dependently-typed functions. For example, we can prove the transitivity of the propositional equality $x \equiv y$ by pattern matching on its only constructor $\texttt{refl} : x \equiv x$:

$$\begin{aligned} &\texttt{trans} : (x\ y\ z : A) \to x \equiv y \to y \equiv z \to x \equiv z \\ &\texttt{trans}\ x\ \lfloor x \rfloor\ \lfloor x \rfloor\ \texttt{refl}\ \texttt{refl} = \texttt{refl} \end{aligned} \tag{1}$$

Inaccessible patterns, like $\lfloor x \rfloor$ in this example, witness the fact that only one type-correct argument can be in that position. Indeed, matching on a proof of $x \equiv y$ with $\texttt{refl} : x \equiv x$ forces x and y to be the same.

Proofs by dependent pattern matching are typically much shorter and more readable than ones that use the classical *datatype eliminators* associated to each inductive family. On the other hand, Goguen et al. [2006] showed that all definitions by dependent pattern matching can be translated to ones that only use eliminators. For this translation they depend on the so-called K axiom. Coquand [1992] already observed that pattern matching allows proving this K axiom:

$$\begin{aligned} \mathrm{K} : &(P : a \equiv a \to \mathit{Set}) \to \\ &(p : P\ \texttt{refl})(e : a \equiv a) \to P\ e \\ \mathrm{K}\quad &P\ p\ \texttt{refl} = p \end{aligned} \tag{2}$$

An emerging field within dependent type theory is *homotopy type theory* (HoTT) [The Univalent Foundations Program, 2013]. It gives a new interpretation of terms of type $x \equiv y$ as *paths* from x to y. Many basic constructions in HoTT can be written very elegantly using pattern matching, for example `trans` (1) corresponds to the composition of two paths.

One of the core elements of HoTT is the *univalence axiom*. Univalence captures the common mathematical practice of informal reasoning "up to isomorphism" in a nice and formalized way. It also has a number of useful consequences, such as *functional extensionality*. However, the univalence axiom is *incompatible* with dependent pattern matching. This has forced people working on HoTT to avoid using pattern matching or risk unsoundness.

The source of the incompatibility between univalence and dependent pattern matching is that pattern matching relies on the K axiom. In an attempt to fix this, an option called –without-K was added to Agda. In theory, this option should allow people to use pattern matching in a safe way when it is undesirable to assume K. However, the option has been criticized many times for being too restrictive, for having unclear semantics, and for containing errors. These errors allowed one to prove (weaker versions of) the K axiom. While they are typically fixed quickly, this really calls for a more in-depth investigation of dependent pattern matching without K.

We present a new criterion that describes what kind of definitions by pattern matching are still allowed if we do not assume K, which is strictly more general than previous attempts. Our criterion works by limiting the unification algorithm used for case splitting in two ways:

- It is not allowed to delete equations of the form $x = x$.

- When applying injectivity on the equation $c\ \bar{s} = c\ \bar{t}$ where $c\ \bar{s}, c\ \bar{t} : D\ \bar{u}$, the indices $\bar{u}$ (but not the parameters) should be *self-unifiable*, i.e. unification of $\bar{u}$ with itself should succeed positively (while still adhering to these two restrictions).

This criterion has been implemented as a patch to Agda. It allows the definition of `trans` (1), but it prohibits the definition of K (2) because case splitting on the argument of type $a \equiv a$ fails.

We give a formal proof that definitions by pattern matching satisfying this criterion are conservative over standard type theory by translating them to eliminators in the style of Goguen et al. [2006], *without* relying on the K axiom. Our proof follows the same general outline, but there are two important differences:

- We work with the homogeneous equality instead of the heterogeneous version, because the elimination rule for the heterogeneous equality is equivalent with K [McBride, 2000].
- Working with the homogeneous equality leads us naturally to upgraded versions of the unification transitions given by Goguen et al. [2006], where the return type is dependent on the equality proof.

We hope that this is enough to convince the HoTT community that pattern matching *can* be used safely without assuming K, and maybe even helps in the creation of a language based on HoTT. Our criterion makes it possible to do pattern matching on *regular* inductive families without assuming K. But HoTT also introduces the concept of *higher inductive types*, which can have nontrivial identity proofs between their constructors. This implies that in general they don't satisfy the injectivity, disjointness, or acyclicity properties. Luckily, our proof is entirely *parametric* in the actual unification transitions that are used. So in order to allow pattern matching in a context with higher inductive types, we should just limit the unification algorithm further.

References

Edwin Brady. Idris, a general purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming*, 23(5), 2013.

Thierry Coquand. Pattern matching with dependent types. In *Types for proofs and programs*, 1992.

Healfdene Goguen, Conor McBride, and James McKinna. Eliminating dependent pattern matching. In *Algebra, Meaning, and Computation*. 2006.

Conor McBride. *Dependently typed functional programs and their proofs*. PhD thesis, University of Edinburgh, 2000.

Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology, 2007.

Matthieu Sozeau. Equations: A dependent pattern-matching compiler. In *Interactive theorem proving*, 2010.

The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.