Modular Confluence for Rewrite Rules in MetaCoq

Jesper Cockx[1], Nicolas Tabareau[2], and Théo Winterhalter[2]

[1] TU Delft, Delft, Netherlands
[2] Gallinette Project-Team, Inria, Nantes, France

Dependently typed languages provide strong guarantees of correctness for our programs and proofs, but they can be hard to use and extend. To increase their practicability and expressivity, they can be extended with user-defined *rewrite rules* [2]. For example, rewrite rules allow us to define 'parallel' plus that reduces on both sides. It is defined by one symbol `pplus` and the following rules:

$$\begin{array}{ll} m : \mathbb{N} \vdash \texttt{pplus}\ 0\ m \hookrightarrow m & n, m : \mathbb{N} \vdash \texttt{pplus}\ (\texttt{S}\ n)\ m \hookrightarrow \texttt{S}\ (\texttt{pplus}\ n\ m) \\ n : \mathbb{N} \vdash \texttt{pplus}\ n\ 0 \hookrightarrow n & n, m : \mathbb{N} \vdash \texttt{pplus}\ n\ (\texttt{S}\ m) \hookrightarrow \texttt{S}\ (\texttt{pplus}\ n\ m) \end{array}$$

Since rewrite rules are applied to expressions appearing at the type level, they interact directly with the type system. Hence they can very easily break expected good properties of these systems, *e.g.,* termination, confluence, canonicity or subject reduction [3]. Among those, confluence is a key ingredient to retain subject reduction and as such is essential.

We present a new criterion to ensure confluence of rewrite rules – and thus subject reduction of the system – for the predicative calculus of cumulative inductive constructions (PCUIC) as formalised in MetaCoq [6]. This criterion is *modular*: adding new rewrite rules that satisfy the criterion again yields a confluent system without having to check new properties of pre-existing rewrite rules. We have formalized this criterion in MetaCoq and extended the existing proof of confluence. The proof however is not particular to the system and should adapt easily to other settings such as Agda, allowing us to extend the Agda implementation [2] with a confluence checker.

Rewrite rules in our setting. We extend PCUIC with blocks consisting of symbol declarations and rewrite rules where the head of each rule is one of the locally defined symbols. A rewrite rule is given as $\Delta \vdash l \hookrightarrow r$ where:

- Δ is the telescope of pattern variables, which should be the only free variables in l and r, all pattern variables appear *linearly* in l (*i.e.,* exactly one time);
- l is the elimination of some symbol declared in the same block, where eliminations include application to a pattern, projection from a record type, and pattern-matching where the return predicate and branches are also patterns;
- patterns can be pattern variables applied to locally bound variables, bound variables, or λ-abstractions where the type and body are both patterns;
- r is unconstrained (except its free variables).

In order to preserve subject reduction, we also ask that there exists a type A such that $\Delta \vdash l : A$ and $\Delta \vdash r : A$. Rewrite rules are interpreted as follows (σ instantiates the pattern variables):

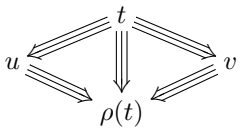
$$\frac{\sigma : \Gamma \to \Delta}{l\sigma \longrightarrow r\sigma}$$

The Tait–Martin-Löf criterion. This method to prove confluence [5] has been used in MetaCoq to show the confluence of PCUIC [7]. Basically, it requires to introduce—beside the standard reduction ($\longrightarrow$)—a notion of parallel reduction ($\Rightarrow$) which *may* do one-step reduction in all its subterms and such that:

$$\longrightarrow \subseteq \Rightarrow \subseteq \longrightarrow^\star$$

so that confluence of parallel reduction is sufficient to get confluence of reduction. In particular, parallel reduction is reflexive.

Confluence of $\Rightarrow$ relies on the existence of an auxiliary function ρ such that $\rho(t)$ is the best parallel reduct of t, *i.e.*, whenever $t \Rightarrow u$ we also have $u \Rightarrow \rho(t)$. This means that $t \Rightarrow \rho(t)$ holds as well. The existence of such a function ρ is called the *triangle property* and allows us to derive confluence by glueing two triangles as illustrated below:



Modular confluence for rewrite rules. For a given set S of rewrite rules to be confluent with the rest of the system, we ask that it satisfies the triangle property *locally* in the following sense. Assuming that the set S of rewrite rules is ordered (for instance, by their order of appearance), we define the function ρ_S by applying the first rule in S that matches and applying ρ_S recursively on the pattern variables. For instance, for the rule $x, y : A \vdash F\ (G\ x)\ y \twoheadrightarrow H\ x\ y$, the definition of ρ_S is $\rho_S(F\ (G\ x)\ y) = H\ (\rho_S\ x)\ (\rho_S\ y)$. Then, the local triangle property for S asks that ρ_S satisfies the triangle property for the rules in S.

Assuming this local triangle property for S and that the theory satisfies the triangle property (with function ρ), we can show that the theory extended with the new set S of rewrite rules still satisfies the triangle property, and thus is confluent. The new auxiliary function ρ is just obtained merging ρ_S into ρ; replacing every recursive call to ρ_S by a recursive call to ρ.

One key ingredient in the proof is the fact that if a term matches one of the rules in S, it still matches this rule after applying any of the 'old' rules to (subterms of) this term. This is very important for the modular reasoning to go through, and it is not satisfied by *non-linear* rewrite rules. Indeed, when a non-linear rule matches, this match is easily broken by rewriting only one of the two occurrences of the non-linear pattern, whatever this reduction is. Thus, the confluence of non-linear rewrite rules cannot be analyzed this way.

Coming back to the example of parallel plus, the local triangle property is satisfied after adding the following (admissible) reduction rule, with high priority:

$$n, m : \mathbb{N} \vdash \texttt{pplus}\ (\texttt{S}\ n)\ (\texttt{S}\ m) \Rightarrow \texttt{S}\ (\texttt{S}\ (\texttt{pplus}\ n\ m))$$

Extended this way, our criterion shows that adding `pplus` to the theory does not break confluence.

Compared to the literature on higher-order rewriting [4, 1, 8], our criterion may look very restrictive. The reason is that we focus on modularity and formal provability, and we work with PCUIC rather than a simpler Pure Type System as is usually the case in the literature.

Formalisation. The formalisation of this modular approach to confluence can be found at https://github.com/TheoWinterhalter/template-coq/tree/rewrite-rules. It is a fork of the MetaCoq repository where we add rewrite rules to the global environment. The confluence proof is in the process of being updated accordingly although there are still remaining assumptions at the time of submission.

References

[1] Frédéric Blanqui, Claude Kirchner, and Colin Riba. On the confluence of lambda-calculus with conditional rewriting. *Theor. Comput. Sci.*, 411(37):3301–3327, 2010.

[2] Jesper Cockx and Andreas Abel. Sprinkles of extensionality for your vanilla type theory. In *Types for Proofs and Programs*, TYPES, 2016.

[3] Jesper Cockx, Nicolas Tabareau, and Théo Winterhalter. How to tame your rewrite rules. In *Types for Proofs and Programs*, TYPES, 2019.

[4] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical computer science*, 192(1):3–29, 1998.

[5] Gert Smolka. Confluence and normalization in reduction systems. Lecture Notes, 2015.

[6] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The MetaCoq Project. To appear in Journal of Automated Reasoning, 2020.

[7] Matthieu Sozeau, Simon Boulier, Yannick Forster, Nicolas Tabareau, and Théo Winterhalter. Coq Coq correct! verification of type checking and erasure for Coq, in Coq. *PACMPL*, 4(POPL), 2020.

[8] Vincent van Oostrom. Confluence by decreasing diagrams. In *Rewriting Techniques and Applications, RTA 2008*, volume 5117 of *Lecture Notes in Computer Science*, 2008.