M2-MPRI - Internship Report

Extending Pattern Unification in the Rocq proof assistant

Wass Ait-Moussa

Supervised by Hugo Herbelin, IRIF

July 1, 2026

29/06/2026. This is the version of the report as it was submitted, which is incomplete and contains some serious mistakes. I intend to keep working on the goals presented here, and I would like to first complete and fix the current document as a start.

General context

Unification is the problem of solving equations between syntactic objects, such as logical formulas or programs' abstract syntax trees. In particular, *higher-order unification* (HOU), the general form of unification in which the unknown variables of the equations might be higher-order functions, despite being undecidable, has proven to be an essential tool in proof-assistants and logic programming. It is an important part of automatic proof-search techniques, and a core part of elaboration and type inference which are implemented in proof assistants as a way to drastically improve their usability by allowing them to infer implicit parts of a statement or a proof.

To deal with the highly untractable nature of HOU, Miller (1989) introduced a decidable class of unification problems he called *patterns* for which there is at most one solution, and an efficient algorithm to find it. This class and its algorithm, commonly refered to as *Higher-Order Pattern Unification* (HOPU), is currently still at the center of most proof-assistants' unification algorithms.

Since HOPU imposes quite severe restrictions on the class of accepted problems, proof assistants generally implement more sophisticated unification systems using various methods to handle cases that fall outside the pattern class. These systems are complex and hard to study, and the scientific focus in the last few decades has been to try to specify them faithfully with respect to their actual implementations in practice.

Research problem

I focused on an extension of HOPU introduced by Libal and Miller (2016) called Functions-as-Constructors Unification (FCU). This extension allows for a broader class of equations, but has more involved restrictions which are more difficult to implement in a mixed setting of equations falling both in and out of the FCU class.

Its theoretical formulation by Libal and Miller was for simply-typed lambda-calculus, and has not been adapted yet to a dependently-typed calculus, which is the family of logical frameworks of multiple proof assistants, and of Rocq in particular. Furthermore, there is, to my knowledge, no documented use of it in proof-assistants; and the only work implementing it in the literature (Hamana, 2017; Hamana, 2019) is in a setting where all equations fall by construction in the FCU class.

Your contribution

On the theoretical side, I formulated the FCU algorithm in a fragment[1] of PCUIC[2], the formal calculus behind the Rocq proof-assistant (Sozeau, Forster, et al., 2025). I restricted myself to a pure setting where all the terms are FCU-patterns in normal-form.

On the other hand, I implemented FCU as a replacement of pattern unification in Unicoq[3], a refined and well-specified implementation of Rocq's unification algorithm. I designed and ran experiments to measure gains in the behavior of unification in existing libraries, and any potential efficiency drawbacks.

An explanation is due for the choice of the restricted setting not corresponding to the implementation for the theoretical study. One aim of this presentation is to follow the footsteps of Pfenning (1991) and Abel and Pientka (2011) with their polished accounts of HOPU in dependently-typed settings[4], while extending the setting to FCU and and to a type-theory with cumulative universes. Another aim is to serve as a modest first stone for having a partial proof of correctness and a precise analysis of the non-completeness of the specification given by Ziliani and Sozeau (2017) for Unicoq, by extending a strict subset of their setting to FCU.

Arguments supporting its validity

For the formalization in PCUC, I formulated a general notion of correctness and completeness for the convertibility and subtyping unifications, and have rules structurally maintaining important invariants of soundness, and the algorithm is fully syntax directed and only unifies the types in the begining, allowing it to correspond closely to the presentation of Libal and Miller (2016) and to share its completeness proof. However, I have not written the full proofs of soundness and completenes yet and I intend to do so as the very next step to my work by formalizing them in MetaRocq[5].

For the implementation in Unicoq, there are three pertinent validity criteria: (a) added complexity to the code, (b) efficiency changes, (c) added "power". The implementation did not increase the complexity of the code or the execution time, but it did not show substential improvement in the solved problems in its current state, and would require further investigation and other extensions to justify replacing simple pattern unification.

Summary and future work

I provided a formal presentation of Functions-as-Constructors Unification (and by extension, of pattern unification) in the setting of a restricted PCUIC ; it was formulated with the objective of showing its soundness and completeness proofs as invariants of its rules, although those proofs are not made yet.

I also experimented with implementing FCU in Rocq's unification algorithm, and studied its adequacy and usefulness.

There are multiple directions I would like to explore next:

- A very promising extension would be to adapt the work of Pfenning and Schürmann (1998), which gives a syntactic criterion to be able to consider definitions as axiomatic constants that do not need to be expanded for equality checking. This might allow to broaden even further the class of FCU-patterns.
- On a practical side, I would like to implement my presentation in MetaRocq, prove its soundness and completeness, and try to extend the setting to handle non-necessarily FCU-pattern or normalized equations while preserving correction and a relative completeness.
- Another direction would be to support records and projections as valid patterns, which has multiple approaches (Abel and Pientka, 2011; Kovács and Bocquet, 2023).
- I would also like to explore ways to extend the pattern fragment to account for inductive types (or even non-recursive sum-types), by searching for appropriate restrictions limiting the combinatory explosion caused by disjunctions.

[1] Without (Co-)Inductives, and without definitions

[2] Polymorphic Cumulative Calculus of Inductive Constructions

[3] `https://github.com/blume0/unicoq/tree/fcu_trial_1`

[4] respectively the Calculus of Constructions and $\lambda^{\Pi\Sigma}$ calculus

[5] An implementation of the meta-theory of Rocq in Rocq itself

Outline

In Section 1, I present the PCUC formal system, and I state some of its known meta-theoretical properties. In the setting of PCUC, I present the appropriate notions of unification and substitutions in a context of cumulative universes (Section 2). In Section 3, I introduce the Functions-as-Constructors restriction, and I present a unification algorithm for it. Finally, in Section 4, I and discuss my implementation of FCU in the Unicoq plugin.

1. The Polymorphic, Cumulative Calculus of Constructions

In this section, we formally present PCUC, the fragment of the Polymorphic Cumulative Calculus of Inductive Constructions for which we formalize FCU. This part closely follows the standard presentations of PCUIC (Sozeau, Forster, et al., 2025; Winterhalter, 2020; Ziliani and Sozeau, 2017; Sozeau and Tabareau, 2014), and differs only in defining typing rules for metavariable instances and their contexts and substitutions.

The missing constructs are (Co-)Inductive types and definitions. Adding Inductive types would not meaningfully change the presentation, as they are not involved in the part of unification we are interested in[6]. Definitions are discussed in Section 3.3.

Section 1.1 defines the different syntactic objects of the language and its typying rules, and Section 1.2 defines its semantics. Some useful properties are then stated in Section 1.3.

1.1. The language

Definition 1 (Syntax)**.** The sets of terms and various environments are defined by the following grammar.

$$
\begin{array}{rll}
x, y, z, \dots \in \mathbb{V} & & \textit{Variables} \\
a, b, c, \dots \in \mathbb{C} & & \textit{Constants} \\
?X, ?Y, ?Z, \dots \in \mathbb{M} & & \textit{Metavariables} \\
i, j, k, l, \dots \in \mathbb{L} & & \textit{Universe levels} \\
t, u, A, B, \dots ::= s \mid x \mid c[\vec{i}] \mid \lambda x^A.T \mid \Pi x^A.B \mid tu \mid ?X[\sigma] & & \textbf{Terms} \\
s ::= \textbf{Prop} \mid \textbf{Type}(\vec{p}) & & \textit{Sorts} \\
p, q, r, \dots ::= i + n & (n \in \mathbb{N}) & \textit{Level expressions} \\
\sigma, \tau, \rho, \dots ::= \cdot \mid \sigma, t & & \textit{Spines} \\
\Gamma, \Delta, \Psi, \dots ::= \langle\rangle \mid \Gamma, x : A & & \textbf{Local contexts} \\
\Sigma ::= \langle\rangle \mid \Sigma, ?X : A[\Psi] & & \textbf{Meta-Contexts} \\
E ::= \langle\rangle \mid E, c : \forall \Phi.A & & \textbf{Global contexts} \\
\Phi ::= (\vec{l}; \mathscr{C}) & & \textbf{Universe contexts} \\
\mathscr{C} ::= \top \mid \mathscr{C} \wedge i \mathscr{R} j & (\mathscr{R} \in \{=, \leq, <\}) & \textit{Level constraint sets} \\
\mathcal{C} ::= \top \mid \mathcal{C} \wedge \vec{p} \mathscr{R} \vec{q} & (\mathscr{R} \in \{=, \leq, <\}) & \textit{General constraint sets}
\end{array}
$$

□

Identifiers. The first block shows the notations for the four kinds of identifiers of the language, and we assume $\mathbb{V}, \mathbb{C}, \mathbb{M}$ and $\mathbb{L}$ are countably infinite and mutually disjoint sets. In expressions of the form $\lambda x^A.t$, $\Pi x^A.t$ or $\Sigma, ?X : t[\Gamma_1, x : A, \Gamma_2]$, the variable x is bound in t. In global contexts of the form $E, c : \forall(\vec{l}; \mathscr{C}).t$, the levels $\vec{l}$ are bound in t, where the vector notation in the grammar denotes a list of elements. For both binding kinds, we identify expressions up to renaming of bound variables.

Metavariables are morally the unknowns of a given term, which the unification algorithm will try to replace with appropriate values. When $?X : A[\Psi]$ appears in a meta-context, it denotes a term t that has its free variables described by the context Ψ, and in that case $?X[\sigma]$ represents t where we replace the variables of Ψ by the terms in σ.

Environments. There are four kinds of contexts, forming the *environment* in which a terms have meaning. The global context declares axioms ; we assume it is fixed and always implicitly present, and well-formed as in Figure 1e. Local contexts specify the types of free variables, and grow when traversing binders. A meta-context declares the metavariables, and is refined during unification. Universe contexts are expanded upon in Section 1.2.

Given a local context $\Gamma = x_1 : T_1, \dots, x_n : T_n$ and a variable y, we note $\widehat{\Gamma}$ for the list $(x_1, \dots, x_n)$, and $y \in \Gamma$ or $y : T_i \in \Gamma$ whenever $y = x_i$ for some i. We also use the notation $?X \in \Sigma$ in the expected way. We call *assignment* a pair $(\vec{x}, \vec{t})$ of a list of distinct variables and a list of terms of the same size, and we note it as $\vec{x} \mapsto \vec{t}$. We also note $y \in (\vec{x} \mapsto \vec{t})$ or $y \mapsto u \in (\vec{x} \mapsto \vec{t})$ whenever $(y, u) = (x_i, t_i)$ for some i. Given a syntactic object o, we denote the set of its free variables with respect to the different binders with $FV(o)$.

[6] In Rocq's algorithm, they are only treated by structural rules and reduction heuristics. See (TODOCITE:appendix A) for a detailed discussion.

Definition 2 (Parallel substitution)**.** Given an assignment κ, we define parallel substitution of κ on terms, spines and contexts, denoted by $t\{\kappa\}$, as follows.

$$
\begin{array}{rlrlrl}
x\{\kappa\} &\coloneqq x & \text{when } x \notin \kappa & & a\{\kappa\} &\coloneqq a \qquad a \in \{\mathbf{Prop}, \mathbf{Type}(\vec{p}), c[\vec{i}]\} \\
x\{\kappa\} &\coloneqq t & \text{when } (x \mapsto t) \in \kappa & & tu\{\kappa\} &\coloneqq t\{\kappa\}u\{\kappa\} \\
(\lambda x^A.t)\{\kappa\} &\coloneqq \lambda x^{A\{\kappa\}}.t\{\kappa\} & \big(\text{assuming w.l.o.g } x, x_i \notin \kappa & & ?X[\sigma]\{\kappa\} &\coloneqq ?X[\sigma\{\kappa\}] \\
(\Pi x^A.B)\{\kappa\} &\coloneqq \Pi x^{A\{\kappa\}}.B\{\kappa\} & \text{and} & & \cdot\{\kappa\} &\coloneqq \cdot \\
(\Gamma = \overrightarrow{x_i : t_i})\{\kappa\} &\coloneqq \overrightarrow{x_i : t_i\{\kappa\}} & x, x_i \notin FV(\kappa)\big) & & \sigma, t\{\kappa\} &\coloneqq \sigma\{\kappa\}, t\{\kappa\}
\end{array}
$$

□

We define substitution of universe levels $\cdot\{\vec{l} \mapsto \vec{p}\}$ in an analogous way, where there are no bound variable issues. In addition to computation rules, substitution is also used in in typing rules for instantiating products and typing constants.

We now proceed to define the typing rules of the calculus. They depend on the convertibility relation, discussed in Section 1.2, but we present them together with the syntax to convey the meaning of some constructs of the language first.

Definition 3 (Typing)**.** We define the type system of PCUC in Figure 1.

In the premises of the Consistent and Valid rules, $[\![\mathcal{C}]\!]_f$ denotes the interpretation of the constraint set $\mathcal{C}$ with the valuation f, interpreting i as $f(i)$ and the $<, \leq, =, +$ symbols as the corresponding relations on natural numbers, and $[\![\vec{p}]\!]_f \coloneqq \max_i([\![p_i]\!]_f)$. The notation $\Psi \vdash \vec{p}$ (resp. $\vec{l} \vdash \mathscr{C}$) means that Ψ (resp. $\vec{l}$) declares every universe level occurring in $\vec{p}$ (resp. $\mathscr{C}$). □

Pure type system. The rules in Figures 1a, 1b and 1e are the usual ones for pure dependently-typed systems, except that sorts are not rigid but are made of abstract level variables, and constants are instantiated with those same level variables.

Metavariables. Rules MVar, SpineCons and MetaCons show that a metavariable $?X : A[\Psi]$ is an object of type A when abstracted over the context Ψ, and appears in terms fully applied to a spine of appropriately typed terms.

Universe context. The rule Consistent reveals the meaning of universe levels and contexts. Levels can be seen as unknown positive numbers, with contexts specifying arithmetic constraints on those numbers. A context is then consistent if we can choose a concrete value for each level such that they satisfy all the constraints. This part of typing which allows one to know there exist some appropriate concrete universes for a given well-typed term without manipulating those is called *typical ambiguity* (G. Huet, 1987; Herbelin, 2005; Sozeau and Tabareau, 2014) The definition on consistency and validity in fig. 1c is declarative, but there are algorithms deciding them for some restriction of constraints that is sufficient for type-checking (Herbelin, 2005; Sozeau, Boulier, et al., 2019). These questions are above the scope of the present report, and we assume for its remaining that we have such an algorithm.

Universe polymorphism. The rules Cst and GCtxCons show the polymorphic part of the calculus. A constant is parameterized over a set of levels $\vec{l}$ required to satisfy constraints $\mathscr{C}$, and to be instantiated in a term with levels $\vec{i}$, its instance must satisfy $\mathscr{C}\{\vec{l} \mapsto \vec{i}\}$. Universe polymorphism is most useful when used with defined constants having a body and nested definitions, but in this presentation we support only axioms as our unification algorithm is limited to normal-forms where any definition is expanded.

1.2. The convertibility relations

Unification aims at making two terms equal *modulo some equivalence*. In this subsection, we define the two notions of equivalence we try to satisfy during unification: *β-convertibility* and *cumulativity*.

Definition 4 (Beta-reduction)**.** Figure 2a defines beta-reduction, the sole computation rule of our language. We note $\rightsquigarrow_\beta^*$ its reflexive transitive closure. We say a term t is in normal-form if there is no t' such that $t \rightsquigarrow_\beta t'$ □

Definition 5 (Convertibility and cumulativity)**.** Given $\mathscr{R} \in \{=_\beta, \leq_\beta\}$, Figure 2b defines the relation $\Phi \vdash t \mathrel{\mathscr{R}} u$. We say that t and u are convertible if $\Phi \vdash t =_\beta u$, and that t is a subtype of u under Φ if $\Phi \vdash t \leq_\beta u$. □

$$\frac{\Phi|\Sigma|\Gamma \vdash \mathcal{WF} \qquad \Phi \vdash \vec{p}}{\Phi|\Sigma|\Gamma \vdash \mathbf{Prop} : \mathbf{Type}(\vec{p})}\ \textsc{Prop} \qquad \frac{\Phi|\Sigma|\Gamma \vdash \mathcal{WF} \qquad \Phi \vdash \vec{p}}{\Phi|\Sigma|\Gamma \vdash \mathbf{Type}(\vec{p}) : \mathbf{Type}(\overrightarrow{p+1})}\ \textsc{Type} \qquad \frac{\Phi|\Sigma|\Gamma \vdash \mathcal{WF} \qquad x : A \in \Gamma}{\Phi|\Sigma|\Gamma \vdash x : A}\ \textsc{Var}$$

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

$$\frac{\Phi|\Sigma|\Gamma \vdash \mathcal{WF} \qquad ?X : A[\Psi] \in \Sigma \qquad \Phi|\Sigma|\Gamma \vdash \sigma : \Psi}{\Phi|\Sigma|\Gamma \vdash ?X[\sigma] : A\{\widehat{\Psi} \mapsto \sigma\}}\ \textsc{MVar} \qquad \frac{\Phi|\Sigma|\Gamma, x : A \vdash t : B \qquad \Phi|\Sigma|\Gamma \vdash \Pi x^A.B : s}{\Phi|\Sigma|\Gamma \vdash \lambda x^A.t : \Pi x^A.B}\ \textsc{Lam}$$

$$\frac{\Phi|\Sigma|\Gamma \vdash t : \Pi x^A.B \qquad \Phi|\Sigma|\Gamma \vdash u : A}{\Phi|\Sigma|\Gamma \vdash tu : B\{x \mapsto u\}}\ \textsc{App} \qquad \frac{\Phi|\Sigma|\Gamma \vdash A : s_1 \qquad \Phi|\Sigma|\Gamma, x : A \vdash B : s_2 \qquad \mathscr{R}(s_1, s_2, s_3)}{\Phi|\Sigma|\Gamma \vdash \Pi x^A.B : s_3}\ \textsc{Prod}$$

$$\frac{\Phi|\Sigma|\Gamma \vdash t : A \qquad A \leq_\beta B}{\Phi|\Sigma|\Gamma \vdash t : B}\ \textsc{Cumul}$$

(a) Typing judgement for terms $\boxed{\Phi|\Sigma|\Gamma \vdash t : A}$

$$\frac{}{\mathscr{R}(\mathbf{Prop}, s, s)}\ \textsc{Prop-s} \qquad \frac{}{\mathscr{R}(\mathbf{Type}(\vec{p}), \mathbf{Type}(\vec{q}), \mathbf{Type}(\overrightarrow{p, q}))}\ \textsc{Type-Type} \qquad \frac{}{\mathscr{R}(s, \mathbf{Prop}, \mathbf{Prop})}\ \textsc{Impr}$$

(b) Product formation rules $\boxed{\mathscr{R}(s_1, s_2, s_3)}$

$$\frac{\exists f : \{l_i \mid i\} \to \mathbb{N}, [\![\mathscr{C}]\!]_f \text{ is true}}{(\vec{l}; \mathscr{C}) \vDash}\ \textsc{Consistent} \qquad \frac{\vec{l} \vdash \mathcal{C}' \qquad \forall f, [\![\mathscr{C}]\!]_f \implies [\![\mathcal{C}']\!]_f}{(\vec{l}; \mathscr{C}) \vDash \mathcal{C}'}\ \textsc{Valid}$$

(c) Universe consistency $\boxed{\Phi \vDash}$ and Constraint validity $\boxed{\Phi \vDash \mathscr{C}'}$

$$\frac{\Phi|\Sigma|\Gamma \vdash \mathcal{WF}}{\Phi|\Sigma|\Gamma \vdash \cdot : \langle\rangle}\ \textsc{SpineEmpty} \qquad \frac{\Phi|\Sigma|\Gamma \vdash \sigma : \Psi \qquad \Phi|\Sigma|\Gamma \vdash t : T\{\Psi \mapsto \sigma\}}{\Phi|\Sigma|\Gamma \vdash \sigma, t : \Psi, x : T}\ \textsc{SpineCons}$$

(d) Typing judgement for spines $\boxed{\Phi|\Sigma|\Gamma \vdash \sigma : \Psi}$

$$\frac{\Phi|\Sigma \vdash \mathcal{WF}_{\text{META}}}{\Phi|\Sigma|\langle\rangle \vdash \mathcal{WF}}\ \textsc{CtxEmpty} \qquad \frac{\Phi|\Sigma|\Gamma \vdash \mathcal{WF} \qquad \Phi|\Sigma|\Gamma \vdash x : T \qquad x \notin \Gamma}{\Phi|\Sigma|\Gamma, x : T \vdash \mathcal{WF}}\ \textsc{CtxCons}$$

$$\frac{\Phi \vDash \top}{\Phi|\langle\rangle \vdash \mathcal{WF}_{\text{META}}}\ \textsc{MetaEmpty} \qquad \frac{\Phi|\Sigma \vdash \mathcal{WF}_{\text{META}} \qquad \Phi|\Sigma|\Psi \vdash A : s \qquad ?X \notin \Sigma}{\Phi|\Sigma, ?X : A[\Psi] \vdash \mathcal{WF}_{\text{META}}}\ \textsc{MetaCons}$$

$$\frac{}{\langle\rangle \vdash \mathcal{WF}_{\text{GLOB}}}\ \textsc{GCtxEmpty} \qquad \frac{E \vdash \mathcal{WF}_{\text{GLOB}} \qquad \Phi \vDash \qquad \Phi|\langle\rangle|\langle\rangle \vdash_E A : s}{E, c : \forall\Phi.A \vdash \mathcal{WF}_{\text{GLOB}}}\ \textsc{GCtxCons}$$

(e) Context well formation $\boxed{\Phi|\Sigma|\Gamma \vdash \mathcal{WF}}$ and $\boxed{\Phi|\Sigma \vdash \mathcal{WF}_{\text{META}}}$ and $\boxed{E \vdash \mathcal{WF}_{\text{GLOB}}}$

Figure 1: Typing rules of PCUC

$$\frac{}{(\lambda x^A.t)u \rightsquigarrow_\beta t\{x \mapsto u\}}\ \textsc{Beta} \qquad \frac{C[\Box] \text{ any context}^7 \qquad t \rightsquigarrow_\beta u}{C[t] \rightsquigarrow_\beta C[u]}\ \textsc{Closure}$$

(a) Beta reduction $\boxed{t \rightsquigarrow_\beta u}$

$$\frac{}{\Phi \vdash (\lambda x^A.t)u \mathrel{\mathscr{R}} t\{x \mapsto u\}}\ \textsc{Beta} \qquad \frac{\Phi \vDash \vec{p} \mathrel{\mathscr{R}} \vec{q}}{\Phi \vdash \textbf{Type(}\vec{p}\textbf{)} \mathrel{\mathscr{R}} \textbf{Type(}\vec{q}\textbf{)}}\ \textsc{CumulT} \qquad \frac{}{\Phi \vdash \textbf{Prop} \leq_\beta s}\ \textsc{CumulP}$$

$$\frac{}{\Phi \vdash t \mathrel{\mathscr{R}} t}\ \textsc{Refl} \qquad \frac{\Phi \vdash u =_\beta t}{\Phi \vdash t \mathrel{\mathscr{R}} u}\ \textsc{Symm} \qquad \frac{\Phi \vdash t \mathrel{\mathscr{R}} t' \qquad \Phi \vdash t' \mathrel{\mathscr{R}} t''}{\Phi \vdash t \mathrel{\mathscr{R}} t''}\ \textsc{Trans}$$

$$\frac{\Phi \vdash A =_\beta B \qquad \Phi \vdash t \mathrel{\mathscr{R}} u}{\Phi \vdash \lambda x^A.t \mathrel{\mathscr{R}} \lambda x^B.u}\ \textsc{ContcLambda} \qquad \frac{\Phi \vdash A =_\beta B \qquad \Phi \vdash T \mathrel{\mathscr{R}} U}{\Phi \vdash \Pi x^A.T \mathrel{\mathscr{R}} \Pi x^B.U}\ \textsc{ContcPi} \qquad \frac{\Phi \vDash \vec{i} = \vec{j}}{\Phi \vdash c[\vec{i}] \mathrel{\mathscr{R}} c[\vec{j}]}\ \textsc{ConvConst}$$

$$\frac{\Phi \vdash t \mathrel{\mathscr{R}} t' \qquad \Phi \vdash u =_\beta u'}{\Phi \vdash tu \mathrel{\mathscr{R}} t'u'}\ \textsc{ContcApp} \qquad \frac{\left(\Phi \vdash t_i \mathrel{\mathscr{R}} u_i\right)_{1 \leq i \leq n}}{\Phi \vdash ?X[t_1, \ldots, t_n] \mathrel{\mathscr{R}} ?X[u_1, \ldots, u_n]}\ \textsc{ContcMeta}$$

(b) Conversion relations $\boxed{\Phi \vdash t \mathrel{\mathscr{R}} u}$ $(\mathscr{R} \in \{=_\beta, \leq_\beta\})$

Figure 2: Reduction and conversion

The convertibility relation is exactly the equivalence relation generated by beta-reduction, plus the CumulT $=_\beta$ rule identifying two sorts if they have provably equal universes with respect to the constraints in context.

The cumulativity relation extends convertibility with the elements **Type(**a**)** $\leq_\beta$ **Type(**b**)** on subterms not appearing in the premise of an abstraction or product whenever we have $\Phi \vDash a \leq b$. This is the relation used in the Cumul typing rule, which is why this relation is also called *subtyping*.

1.3. Some meta-theoretical properties

In this subsection, we state some known properties of the system, which will be useful to us later on when discussing what a good specification for unification problems should be.

These are standard properties for typed lambda-calculi[8], and have been formally proven in Rocq[9] as part of the MetaRocq project (Sozeau, Forster, et al., 2025).

Theorem 6 (Principality)**.** *Let $\Phi|\Sigma|\Gamma \vdash t : A$ be a well-typed term. There exists a unique term $A^\perp$ modulo convertibility such that*

1. $\Phi|\Sigma|\Gamma \vdash t : A^\perp$,
2. *For all A', $\Phi|\Sigma|\Gamma \vdash t : A' \implies \Phi \vdash A^\perp \leq_\beta A$.*

We call $A^\perp$ the principal type of t and note it $\mathcal{P}r_{\Phi|\Sigma|\Gamma}(t)$.

Theorem 7 (Confluence)**.** *If $t \rightsquigarrow_\beta^* u$ and $t \rightsquigarrow_\beta^* u'$ then there exists v such that $u \rightsquigarrow_\beta^* v$ and $u' \rightsquigarrow_\beta^* v$.*

Theorem 8 (Subject reduction)**.** *If $\Phi|\Sigma|\Gamma \vdash t : A$ and $t \rightsquigarrow_\beta^* u$, then $\Phi|\Sigma|\Gamma \vdash u : A$.*

Theorem 9 (Validity)**.** *If $\Phi|\Sigma|\Gamma \vdash t : A$, then $\Phi|\Sigma|\Gamma \vdash A : s$ for some sort s.*

[7] $C ::= \Box \mid tC \mid Ct \mid \lambda x^C.t \mid \lambda x^t.C \mid \Pi x^C.t \mid \Pi x^t.C \mid ?X[t, \ldots, t, C, t, \ldots, t]$

[8] with some subtelty here due to universe cumulativity

[9] With the assumption of soundness, and without metavariables. The latter can be safely adapted by seeing metavariables as non-polyhmorphic constants.

With some combination of these properties, we can derive the following lemma.

WRONG ‼ Take $t \triangleq (\lambda x^{\mathbf{Type}(i+1)}.x)\mathbf{Type}(i)$: then we have $t =_\beta \mathbf{Type}(i)$ but not $\mathbf{Type}(i+2) =_\beta \mathbf{Type}(i+1)$. This was intended as a justification for the completeness of the UnifTypes rule.

Lemma 10. *If t and u are well-typed and $\Phi \vdash t \mathrel{\mathscr{R}} u$, then we have $\mathcal{P}r_{\Phi|\Sigma|\Gamma}(t) \mathrel{\mathscr{R}} \mathcal{P}r_{\Phi|\Sigma|\Gamma}(u)$*

Proof.
- For equality: by confluence there exists a common reduct v of t and u. By subject reduction v shares the types of both u and v. We conclude by principality.
- For cumulativity, we proceed by induction on the derivation of the judgement, and for the non-trivial rules we use the fact that the typing rule of each of the two terms is necessarily a succession of Cumul rules followed by a rule structurally destructing the term and allowing to use the IH.

□

2. Unification

Now that we have defined the relevant parts of the language, we can specify what a unification problem is.

We start by defining what it means to choose solutions for metavariables and universe levels, and how to compare those solutions and universe levels (Section 2.1). We then give the formal definition of a unification problem and the notion of a complete solution (Section 2.2), and conclude the section by discussing the different possible approaches for a unification algorithm (Section 2.3).

2.1. Substitution and subsumption

Definition 11 (Substitution pairs)**.** (i.) Meta substitutions and level substitutions are defined by the following grammar.

$$\begin{array}{lll} \theta ::= \cdot \mid \theta, ?X \mapsto t[\Psi] & & \textbf{(Meta substitutions)} \\ \phi ::= \vec{l} \mapsto \vec{\vec{p}} & (|\vec{l}| = |\vec{\vec{p}}|) & \textbf{(Level substitutions)} \\ \langle \phi ; \theta \rangle & & \textbf{(Substitution pairs)} \end{array}$$

(ii.) A pair $\langle \phi ; \theta \rangle$ is a well-formed substitution pair from $\Phi \,|\, \Sigma$ to $\Phi' \,|\, \Sigma'$ if it satisfies the judgement defined in Figure 3b.

(iii.) Given a pair $\langle \phi ; \theta \rangle$ and a term t, we define the operation $t\{\!\{\langle \phi ; \theta \rangle\}\!\}$ by induction on terms.

$$\begin{array}{rlrl} ?X[\sigma]\{\!\{\langle \phi ; \theta \rangle\}\!\} &:= t\{\Psi \mapsto \sigma\{\!\{\langle \phi ; \theta \rangle\}\!\}\} \quad \text{when } X \mapsto t[\Psi] \in \theta & \lambda x^A.t\{\!\{\langle \phi ; \theta \rangle\}\!\} &:= \lambda x^{A\{\!\{\langle \phi ; \theta \rangle\}\!\}}.t\{\!\{\langle \phi ; \theta \rangle\}\!\} \\ ?X[\sigma]\{\!\{\langle \phi ; \theta \rangle\}\!\} &:= ?X[\sigma\{\!\{\langle \phi ; \theta \rangle\}\!\}] \quad \text{when } ?X \notin \theta & \Pi x^A.B\{\!\{\langle \phi ; \theta \rangle\}\!\} &:= \Pi x^{A\{\!\{\langle \phi ; \theta \rangle\}\!\}}.B\{\!\{\langle \phi ; \theta \rangle\}\!\} \\ \mathbf{Type}(\vec{l})\{\!\{\langle \phi ; \theta \rangle\}\!\} &:= \mathbf{Type}(\vec{l}\{\phi\}) & tu\{\!\{\theta\}\!\} &:= t\{\!\{\langle \phi ; \theta \rangle\}\!\} u\{\!\{\langle \phi ; \theta \rangle\}\!\} \\ c[\vec{l}]\{\!\{\langle \phi ; \theta \rangle\}\!\} &:= c[\vec{l}\{\phi\}] & \cdot\{\!\{\langle \phi ; \theta \rangle\}\!\} &:= \cdot \\ a\{\!\{\langle \phi ; \theta \rangle\}\!\} &:= a \quad a \in \{\mathbf{Prop}, x\} & \sigma, t\{\!\{\langle \phi ; \theta \rangle\}\!\} &:= \sigma\{\!\{\langle \phi ; \theta \rangle\}\!\}, t\{\!\{\langle \phi ; \theta \rangle\}\!\} \end{array}$$

(iv.) Furthermore, we define composition of meta-substitutions as follows.

$$\begin{aligned} \langle \phi_1 ; \theta \rangle \circ \langle \phi_2 ; \psi \rangle &:= \langle \phi_1 \circ \phi_2 ; \theta \circ \psi \rangle \\ \phi_1 \circ (\vec{l} \mapsto \vec{i}) &:= \vec{l} \mapsto \vec{i}\{\phi_1\} \\ \psi \circ \cdot &:= \cdot \\ \psi \circ (\theta, ?X \mapsto t[\Psi]) &:= (\psi \circ \theta), ?X \mapsto t\{\!\{\psi\}\!\}[\Psi\{\!\{\psi\}\!\}] \end{aligned}$$

□

$$\frac{\vec{j} \vdash \vec{k} \qquad (\vec{j}; \mathscr{C}') \vDash \mathscr{C}\{\vec{i} \mapsto \vec{k}\}}{(\vec{j}; \mathscr{C}') \vdash (\vec{i} \mapsto \vec{k}) : (\vec{i}; \mathscr{C})}\ \textsc{LSubst}$$

(a) Level substitution well-formation $\boxed{\Phi' \vdash \phi : \Phi}$

$$\frac{\Phi'|\Sigma' \vdash \mathcal{WF}_{\textsc{meta}} \qquad \Phi' \vdash \phi : \Phi}{\Phi'|\Sigma' \vdash \langle \phi; \cdot \rangle : \Phi|\langle\rangle}\ \textsc{MSubstEmpty}$$

$$\frac{\Phi'|\Sigma' \vdash \langle \phi; \theta \rangle : \Phi|\Sigma \qquad \Phi|\Sigma|\Psi \vdash T : s \qquad \Phi|\Sigma'|\Psi\{\!\{\theta\}\!\} \vdash t : T\{\!\{\theta\}\!\}}{\Phi'|\Sigma' \vdash \langle \phi; \theta, ?X \mapsto t[\Psi] \rangle : \Phi|\Sigma, ?X : T[\Psi]}\ \textsc{MSubstCons}$$

(b) Substitution pair well-formation $\boxed{\Phi'|\Sigma' \vdash \langle \phi; \theta \rangle : \Phi|\Sigma}$

Figure 3: Typing rules for valid substitution pairs

It might seem like this definition of level substitutions is too general, as in practice we merely extend universe contexts with new levels, and never rename old ones; but it actually allows to formulate a well-behaved subsumption relation between substitutions.

Similarly to Pfenning (1991), and unfaithfully to practical specification (Ziliani and Sozeau, 2017), the rule MSubstCons restricts well-formed meta-substitutions to have a full domain, and to be sorted with no front-dependencies[10] ; nevertheless, this simplification is equivalent to the unsorted version.

Definition 12 (Notable substitution pairs)**.** Let $\Phi|\Sigma \vdash \mathcal{WF}_{\textsc{meta}}$ where $\Phi = (\vec{l}; \mathscr{C})$ and $\Sigma = \overrightarrow{?X_i : A_i[\Psi_i]}$. We define

(i.) $\mathrm{Id}_{\langle \Phi;\Sigma \rangle} \coloneqq \langle \mathrm{Id}_\Phi; \mathrm{Id}_\Sigma \rangle \coloneqq \langle \vec{l} \mapsto \vec{l}; \overrightarrow{?X_i \mapsto ?X_i[\widehat{\Psi_i}][\Psi_i]} \rangle$

(ii.) $(?X_j \backslash t)_\Sigma \coloneqq \langle \vec{l} \mapsto \vec{l}; \mathrm{Id}_{\Sigma_{i \leq j}}, (?X_j \mapsto t[\Psi_j]), \overrightarrow{?X_i \mapsto ?X_i[\widehat{\Psi_i}][\Psi_i\{\!\{\mathrm{Id}_{\Sigma_{i \leq j}}, ?X_j \mapsto A_j[\Psi_j]\}\!\}]}^{i>j} \rangle$

□

Item (i.) defines the identity substitution leaving contexts unchanged. Item (ii.) defines the substitution instantiating only one metavariable ; this is the only base form that will be used by the unification algorithm. In item (ii.), $\Sigma_{i \leq j}$ means the metacontext Σ stripped to its $j-1$ first declarations.

To state that a unification procedure is *complete*, we need an order relation between substitutions.

Definition 13 (Subsumption relation)**.** Given well-typed substitutions $\Phi_1|\Sigma_1 \vdash \langle \phi_1; \theta_1 \rangle : \Phi|\Sigma$ and $\Phi_2|\Sigma_2 \vdash \langle \phi_2; \theta_2 \rangle : \Phi|\Sigma$, $\langle \phi_1; \theta_1 \rangle$ is said to be *more general than* $\langle \phi_2; \theta_2 \rangle$ with respect to $\Phi \mid \Sigma$, noted $\langle \phi_1; \theta_1 \rangle \preceq_{\Phi|\Sigma} \langle \phi_2; \theta_2 \rangle$ whenever there exists $\Phi_2|\Sigma_2 \vdash \langle \phi^*; \theta^* \rangle : \Phi_1|\Sigma_1$ such that $\langle \phi_2; \theta_2 \rangle = \langle \phi^*; \theta^* \rangle \circ \langle \phi_1; \theta_1 \rangle$. □

For our notion of substitution and subsumption to be an appropriate one, it is necessary to verify the following properties.

Lemma 14 (Identity substitution)**.** *If* $\Phi|\Sigma \vdash \mathcal{WF}_{\textsc{meta}}$*, then* $\Phi|\Sigma \vdash Id_{\langle \Phi;\Sigma \rangle} : \Phi|\Sigma$

Lemma 15 (One-substitution)**.** *The following rule, where* $\Sigma = \Sigma_1, ?X : A[\Psi], \Sigma_2$ *and* $\Phi = (\vec{l}; \mathscr{C})$*, is admissible.*

$$\frac{\begin{array}{cc} \Sigma \cap \Sigma^* = \emptyset & \Phi|\Sigma \vdash \mathcal{WF}_{\textsc{meta}} \\ \Phi' \vdash l & \Phi' \vDash \mathscr{C} \qquad \Phi'|\Sigma_1, \Sigma^*|\Psi \vdash t : A \end{array}}{\Phi'|\Sigma_1, \Sigma^*, \Sigma_2\{\!\{(?X \backslash t)_\Sigma\}\!\} \vdash (?X \backslash t)_\Sigma : (\vec{l}; \mathscr{C})|\Sigma}\ \textsc{OneSubst}$$

Substitutions behave as expected.

[10] For example, we can't have $?X \mapsto ?Z[], ?Z \mapsto \mathbf{Prop}$ as $?Z$ would be declared after $?X$.

Lemma 16 (Properties of meta-substitution). *If* $\Phi|\Sigma|\Gamma \vdash t : A$, $\Phi'|\Sigma' \vdash \langle \phi; \theta \rangle : \Phi|\Sigma$ *and* $\Phi''|\Sigma'' \vdash \langle \phi'; \theta' \rangle : \Phi'|\Sigma'$, *then*

1. $\Phi|\Sigma'|\Gamma \vdash t\{\!\{\langle \phi; \theta \rangle\}\!\} : A\{\!\{\langle \phi; \theta \rangle\}\!\}$
2. $\Phi''|\Sigma'' \vdash \langle \phi'; \theta' \rangle \circ \langle \phi; \theta \rangle : \Phi|\Sigma$
3. $t\{\!\{\langle \phi'; \theta' \rangle \circ \langle \phi; \theta \rangle\}\!\} = t\{\!\{\langle \phi; \theta \rangle\}\!\}\{\!\{\langle \phi'; \theta' \rangle\}\!\}$
4. $\langle \phi; \theta \rangle \circ Id = Id \circ \langle \phi; \theta \rangle = \langle \phi; \theta \rangle$
5. $(\langle \phi_1; \theta_1 \rangle \circ \langle \phi_2; \theta_2 \rangle) \circ \langle \phi_3; \theta_3 \rangle = \langle \phi_1; \theta_1 \rangle \circ (\langle \phi_2; \theta_2 \rangle \circ \langle \phi_3; \theta_3 \rangle)$

Lemma 17. $\preceq_{\Phi|\Sigma}$ *is a partial order.*

2.2. Unification problem

Definition 18 (Unification problem)**.** A unification problem is a triple ($\Phi|\Sigma|\Gamma \vdash t : A$,$\mathscr{R}$,$\Phi|\Sigma|\Gamma \vdash u : B$) of two well-typed terms under a common environment and a symbol $\mathscr{R} \in \{=_\beta, \leq_\beta\}$. We denote the triple by $\Phi|\Sigma|\Gamma \vdash t^A \approx^?_{\mathscr{R}} u^B$. □

Now we need to define what a solution to a unification problem is. It is simply a well-formed substitution pair making the two terms equal.

Definition 19 (Unifying substitution)**.** Given a problem $\Phi \,|\, \Sigma \,|\, \Gamma \vdash t^A \approx^?_{\mathscr{R}} u^B$ and a well-formed meta-substitution $\Phi' \,|\, \Sigma' \vdash \langle \phi; \theta \rangle : \Phi \,|\, \Sigma$, $\langle \phi; \theta \rangle$ is a *unifying substitution* for (or *solution* to) the problem if $t\{\!\{\langle \phi; \theta \rangle\}\!\} \mathscr{R} u\{\!\{\langle \phi; \theta \rangle\}\!\}$ and we say in that case that the two terms are $\mathscr{R}$*-unifiable.*

We call $\mathcal{S}^{\mathscr{R}}_{\Phi|\Sigma|\Gamma}(t, u)$ the set of unifiers for a given problem. □

One might worry that the unification process succeeds at unifying two terms that do not share the same type. Fortunately, we have the following property.

Lemma 20. *Let* $\Phi|\Sigma|\Gamma \vdash t^A \approx^?_{\mathscr{R}} u^B$ *be a problem, and* $\langle \phi; \theta \rangle \in \mathcal{S}^{\mathscr{R}}_{\Phi|\Sigma|\Gamma}(t, u)$.
We have $\mathcal{P}r_{\Phi|\Sigma|\Gamma}(t\{\!\{\langle \phi; \theta \rangle\}\!\})\ \mathscr{R}\ \mathcal{P}r_{\Phi|\Sigma|\Gamma}(u\{\!\{\langle \phi; \theta \rangle\}\!\})$.

In our case, we don't want just any unifier, but one that is unambiguous and that does not unnecessarily commit to any superfluous metavariable instance.

Definition 21 (Most general unifier)**.** A unifying substitution $\Phi'|\Sigma' \vdash \langle \phi; \theta \rangle : \Phi|\Sigma$ for a problem $\Phi|\Sigma|\Gamma \vdash t^A \approx^?_{\mathscr{R}} u^B$ is a *most general unifier* if it is a lower bound of $\left(\mathcal{S}^{\mathscr{R}}_{\Phi|\Sigma|\Gamma}(t, u), \preceq_{\Phi|\Sigma}\right)$ □

It is well-known that a most general unifier does not necessarily exist upon the presence of a solution.

Example 22 (Some elementary unification problems)**.** Assume $E = \langle\rangle, A : \forall\emptyset.\mathbf{Prop}, a : A, f : A \to A$, $\Sigma = \langle\rangle, ?X : P[x_1 : A, x_2 : A]$ and $\Gamma = \langle\rangle, z_1 : A, z_2 : A$.

- $\Phi|\Sigma|\Gamma \vdash ?X[z_2, z_1] \approx^?_{=_\beta} z_1$ has a unique solution $\langle \mathrm{Id}; ?X \mapsto x_2[\cdots] \rangle$
- $\Phi \,|\, \Sigma \,|\, \Gamma \vdash ?X[z_1, z_1] \approx^?_{=_\beta} z_1$ has the two incompatible solutions $\langle \mathrm{Id}; ?X \mapsto x_1[\cdots] \rangle$ and $\langle \mathrm{Id}; ?X \mapsto x_2[\cdots] \rangle$
- $\Phi|\Sigma|\Gamma \vdash ?X[f z_1, z_2] \approx^?_{=_\beta} f(?X[z_1, z_2])$ has at least the infinitely many incompatible unifiers $\langle \mathrm{Id}; ?X \mapsto f^n x_1[\cdots] \rangle$

In fact, it is undecidable to find even one unifier.

Theorem (G. P. Huet, 1973)**.** *The problem of finding if a unifier exists is undecidable.*

2.3. Unification algorithms

There are multiple kinds of unification procedures, depending on the class of the treated problems and on the intent: we may want to always enumerate all possible solutions without failing, we may need to always return at least one solution event if it is not the best, or maybe we want to make the most progress while staying complete without necessarily finding a solution, etc...

This subsection is an attempt at classifying such variances, with the objective of establishing a blueprint of a framework to study different unification methods and their composability. It is not essential for the main development about FCU in Section 3, but it will hopefully serve as a useful tool when extending the algorithm.

Definition 23 (Classifying by result)**.** We say a unification algorithm is *final* if it either returns solutions to the problem, or fails. Otherwise, we say it is *non-final* or *postponing*. □

Definition 24 (Notions of soundness and completeness)**.**

- A final unification algorithm is sound if its returned value is a unifying substitution. It is complete if it returns most general unifiers whenever a solution exists. It is relatively complete if it can fail even when solutions exist, but always returns most general unifiers when it does not fail.
- A postponing algorithm is sound if the set of solutions to its result is a subset of that of its input. It is relatively complete if those sets are equal, and it is complete if furthermore no progress can be made.

□

These properties are to be taken with respect to a class of problems. We can for example restrict the number, the occurrences and the order of metavariables, or restrict the syntax of terms.

Definition 25 (Classifying by input)**.** We say a unification algorithm is

1. *Type directed* if it takes as input two terms and their types,
2. *Syntax directed/Type informed* if it takes as input two terms with the precondition that they are of convertible types,
3. *Purely syntax directed* if it takes two terms as input with the only precondition that they are well-typed.

□

Thanks to Lemma 20, we know that these three kinds of algorithms answer equivalent problems. In practice, syntax directed algorithms are generally preferred for efficiency reasons, although type-directed ones can be more expressive and handle more cases. In the following, we review and classify the different algorithms cited in the present work.

Patterns. Miller's[11] original patterns, as well as Libal and Miller's[12] function-as-constructor patterns are purely syntax directed. Both can be expressed as final algorithms which are complete for their respective classes.

Pattern unification in CC. Pfenning's[13] adaptation of pattern unification for the calculus of constructions is also final and complete. It is syntax directed but type informed, as it first unifies the types of its input.

Dynamic pattern unification in $\lambda^{\Pi\Sigma}$. Abel and Pientka's[14] dynamic unification is a postponing algorithm: it operates on an unrestricted class of problems, and solves all possible pattern constraints while staying relatively complete. It is also type-directed and uses types to handle dependent pairs.

Rocq's algorithm and Unicoq The unification algorithm in Rocq's implementation is a combination of multiple sets of rules, including heuristics. It is not complete, not relatively complete and has no proof of soundness. Some rules are also postponing. It is syntax directed in principle, but has rules checking the types of their conclusion, due to heuristics potentially rendering the problem ill-typed.
On the other hand, Ziliani and Sozeau's[15] Unicoq, while sharing most of these properties with the above, does not incorporate postponing rules, and its authors argue these are not crucial for the power of the algorithm.

[11] Miller, 1989.
[12] Libal and Miller, 2016.
[13] Pfenning, 1991.
[14] Abel and Pientka, 2011.
[15] Ziliani and Sozeau, 2017.

3. Functions-as-Constructors Unification

In this section, we present the functions-as-constructors class of terms and unification algorithm, with some adaptation for the polymorphic and cumulative setting.

3.1. FCU problems

The rationale behind the FCU restriction, similarly to that of its patterns predecessor, is to have a class of higher-order problems for which there exists a unification procedure that is complete, syntax directed and does not trigger additional computation[16]. Miller's patterns achieve this by restricting the arguments of metavariables to be distinct bound variables. The work of Libal and Miller (2016) relaxes this retriction to allow arbitrary terms composed of constants and variables, as long as all their leafs are variables.

Definition 26 (FC patterns)**.** Functions-as-constructors patterns[17] are a subclass of terms specified by the grammar

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

where $\square[t] \coloneqq t$ and $(Et)[a] \coloneqq E[a]t$. □

This class has two sides: it restricts the arguments of a metavariable to neutral contexts with variable leafs, and syntactically restricts the set of terms to β-normal forms. However, this is not sufficient to ensure that there is only one possible solution, and we need to furthermore add restrictions on the occurences of those arguments.

Definition 27 (Subterm modulo universe)**.** We define the relations $\boxed{\Phi \vdash A_1 \sqsubseteq A_2}$ and $\boxed{\Phi \vdash A_1 \equiv A_2}$ on restricted arguments as follows.

$$
\frac{(\vec{l};\mathscr{C}) \vdash E_r \equiv E'_r \qquad (\vec{l};\mathscr{C} \wedge \max(\vec{p}) = \max(\vec{q})) \vDash}{(\vec{l};) \vdash E_r[\textbf{Type}(\vec{p})] \equiv E'_r[\textbf{Type}(\vec{q})]}
\qquad
\frac{(\vec{l};\mathscr{C}) \vdash E_r \equiv E'_r \qquad (\vec{l};\mathscr{C} \wedge (i_k = j_k)_k) \vDash}{\Phi \vdash E_r[c[\vec{i}]] \equiv E'_r[c[\vec{j}]]}
$$

$$
\frac{}{\Phi \vdash x \equiv x}
\qquad
\frac{\Phi \vdash E_r \equiv E'_r \qquad a \in \{x, \textbf{Prop}\}}{\Phi \vdash E_r[a] \equiv E'_r[a]}
\qquad
\frac{\Phi \vdash E_r \equiv E'_r \qquad \Phi \vdash A \equiv A'}{\Phi \vdash E_r A \equiv E'_r A'}
$$

$$
\frac{\Phi \vdash A \equiv A'}{\Phi \vdash \square A \equiv \square A'}
$$

$$
\frac{\Phi \vdash A \equiv B}{\Phi \vdash A \sqsubseteq B}
\qquad
\frac{\Phi \vdash A \sqsubseteq E_r \qquad a \in \{x, \textbf{Prop}, c[\vec{i}]\}}{\Phi \vdash A \sqsubseteq E_r[a]}
\qquad
\frac{\Phi \vdash A \equiv B}{\Phi \vdash A \sqsubseteq E_r B}
\qquad
\frac{\Phi \vdash A \equiv B}{\Phi \vdash A \sqsubseteq \square B}
$$

We also define $\Phi \vdash A \sqsubset A'$ as $\Phi \vdash A \sqsubseteq A' \wedge \neg(\Phi \vdash A \equiv A')$. □

Definition 28 (FCU problem)**.** A unification problem $\Phi|\Sigma|\Gamma \vdash t^A \approx^?_{\mathscr{R}} u^B$ is an FCU problem if the following three properties, where $T, T' \in \{t, u, A, B\}$, are satisfied.

1. **Argument restriction.** T is a pattern.

[16] Meaning β-reductions

[17] Starting here, we will refer to FC patterns as just patterns, and will say "simple patterns" to refer to Miller's original patterns

2. **Local restriction.** For every occurence of the form $?X[A_1, \ldots, A_n]$ in T, there is no $i \neq j$ such that $\Phi \vdash A_i \sqsubseteq A_j$

3. **Global restriction.** For every two occurences of the form $?X[A_1, \ldots, A_n]$ and $?Y[B_1, \ldots, B_m]$ in T and T', there is no i, j such that $\Phi \vdash A_i \sqsubset B_j$

□

Before discussing the reasoning behind the subterm relation we have chosen, let us discuss the three restrictions from a syntactic point of view.

The local restricion. This is the simplest restriction, as it only prevents non-determinism. It generaizes the condition that variable arguments be distinct in simple patterns. If we violate this pattern, we won't have most general unifiers, but we will still have finite complete sets of unifiers that can be computed and described efficiently. A representative example violating this restriction is $?X[x, cx] \approx^{?}_{=_\beta} cx$: it has two incompatible solutions, $X \mapsto cz_1[z_1, z_2]$ and $X \mapsto z_2[z_1, z_2]$.

The argument restriction. The essence of the argument restriction is to prevent implicit computation that might appear after substituting a metavariable instance with its body, and to that end it forces metavariables arguments to be neutral: they can't contain abstractions nor metavariable instances. The requirement that all term leafs be variables, however, is more subtle, and is of the same nature of the global restriction.

The global restriction. This restriction is essential for the FCU's pruning operation ; consider the following example in the environment $E = T : \mathbf{Type(}i\mathbf{)}, f : T \to T$ and contexts $?X : T[z : T], ?Y : T[z : T] \mid x : T$.

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

In this situation, we would like to make a definitive choice for the value of $?X$. This is achieved by expressing the right-hand side with only constants, metavariable symbols and arguments of $?X$ (here, just fx). This would not be possible as is, because the argument x occuring in the $?Y$ variable cannot be expressed in terms of fx. Therefore, we restrict $?Y$ to not use its first argument, obtaining the updated meta-context $?Y : T[]$ and the solution $X \mapsto g(?Y[])(z)[z : T]$. Howerer, this is not complete: since $x \sqsubset fx$, we also have the solution $?X \mapsto gzz[z : T],\ ?Y \mapsto fz[z : T]$.

We call the operation expressing a term using only the arguments of the metavariable instance it has to be equal to *inversion*, and the operation of restricting the incompatible arguments of a metavariable *pruning*. The global restriction ensures that these operations are complete.

One of the goals we initialy had was to find a variant of the FCU algorithm which operates on terms without the global, local and leaf variable restrictions, and which finds finite descriptions of the sets of complete unifiers of such problems. Unfortunately, we realized that the global restriction and the variable restriction play a more essential role than just preventing non-determinism, as there is the following result.

Theorem (Levy and Veanes, 2000, Corollary 16)**.** *If we relax the FCU-problems class by either removing the global restriction or by allowing constants in the leafs of arguments of metavariables, then the unification problem becomes undecidable.*

Universe restriction. To conclude, we discuss the main adaptation of FC patterns to the universe polymorphic setting. As stated earlier, the main idea behind patterns is to have rigit arguments to metavariables. This is imposed syntactically by only allowing variables and constants. But in our setting, constants are instantiated with universes, which are not rigid and may be instantiated during unification. To account for this, we use a notion of approximate term equality modulo universes, which considers two constants as equal whenever their universe arguments are *not* provably distinct in the universe context. This way, when the local and global restrictions are respected, there is no possible refinement of the universe context which would violate the restrictions later.

3.2. An Algorithm for FCU

We now present a syntax directed unification algorithm for FCU-problems. It shares most of its structural rules with those of Unicoq (Ziliani and Sozeau, 2017), and can be seen both as

- a generalization of Unicoq's simple patterns fragment to FC patterns,

$$\frac{\Phi \vDash \mathcal{P}r_{\Phi|\Sigma|\Gamma}(T) \mathscr{R} \mathcal{P}r_{\Phi|\Sigma|\Gamma}(U) \dashv \Phi^* \quad \Phi^*|\Sigma|\Gamma \vdash T \approx^?_{\mathscr{R}} U \dashv \Phi_1 \,|\, \Sigma_1;\theta_1 \quad \Phi_1|\Sigma_1|\Gamma\{\!\{\theta_1\}\!\} \vdash t\{\!\{\theta_1\}\!\} \approx^?_{\mathscr{R}} u\{\!\{\theta_1\}\!\} \dashv \Phi_2 \,|\, \Sigma_2;\theta_2}{\Phi|\Sigma|\Gamma \vdash t^T \approx^?_{\mathscr{R}} u^U \dashv \Phi_2 \,|\, \Sigma_2;\theta_2 \circ \theta_1} \textsc{UnifTypes}$$

(a) Unification of types $\boxed{\Phi|\Sigma|\Gamma \vdash t^T \approx^?_{\mathscr{R}} u^U \dashv \Phi' \,|\, \Sigma';\theta}$

$$\frac{\Phi \vDash \mathcal{P}r_{\Phi|\Sigma|\Gamma}(T) = \mathcal{P}r_{\Phi|\Sigma|\Gamma}(U) \dashv \Phi^* \quad \Phi^*|\Sigma|\Gamma \vdash T \approx^?_{=_\beta} U \dashv \Phi_1 \,|\, \Sigma_1;\theta_1 \quad \Phi_1|\Sigma_1|\Gamma\{\!\{\theta_1\}\!\} \vdash T'\{\!\{\theta_1\}\!\} \approx^?_{\mathscr{R}} U'\{\!\{\theta_1\}\!\} \dashv \Phi_2 \,|\, \Sigma_2;\theta_2}{\Phi|\Sigma|\Gamma \vdash \Pi x^T.T' \approx^?_{\mathscr{R}} \Pi x^U.U' \dashv \Phi_2 \,|\, \Sigma_2;\theta_2 \circ \theta_1} \text{ Π-Π}$$

$$\frac{\Phi|\Sigma|\Gamma, x : T \vdash t \approx^?_{\mathscr{R}} ux \dashv \Phi' \,|\, \Sigma';\theta}{\Phi|\Sigma|\Gamma \vdash \lambda x^T.t \approx^?_{\mathscr{R}} u \dashv \Phi' \,|\, \Sigma';\theta}\ \lambda\eta \qquad \frac{\Phi|\Sigma|\Gamma, x : T \vdash t \approx^?_{\mathscr{R}} u \dashv \Phi' \,|\, \Sigma';\theta}{\Phi|\Sigma|\Gamma \vdash \lambda x^T.t \approx^?_{\mathscr{R}} \lambda x^T.u \dashv \Phi' \,|\, \Sigma';\theta}\ \lambda\lambda$$

$$\frac{\Phi|\Sigma|\Gamma, x : U \vdash tx \approx^?_{\mathscr{R}} u \dashv \Phi' \,|\, \Sigma';\theta}{\Phi|\Sigma|\Gamma \vdash t \approx^?_{\mathscr{R}} \lambda x^U.u \dashv \Phi' \,|\, \Sigma';\theta}\ \eta\lambda \qquad \frac{\Phi|\Sigma|\Gamma \vdash E \approx^?_{=_\beta} E' \dashv \Phi' \,|\, \Sigma';\theta}{\Phi|\Sigma|\Gamma \vdash E[x] \approx^?_{\mathscr{R}} E'[x] \dashv \Phi' \,|\, \Sigma';\theta}\ \text{E[x]-E[x]}$$

$$\frac{}{\Phi|\Sigma|\Gamma \vdash \Box \approx^?_{=_\beta} \Box \dashv \Phi \,|\, \Sigma;\mathrm{Id}_\Sigma}\ \Box\text{-}\Box$$

$$\frac{\Phi|\Sigma|\Gamma \vdash E \approx^?_{=_\beta} E' \dashv \Phi' \,|\, \Sigma';\theta_1 \quad \Phi'|\Sigma'|\Gamma\{\!\{\theta_1\}\!\} \vdash N\{\!\{\theta_1\}\!\} \approx^?_{=_\beta} N'\{\!\{\theta_1\}\!\} \dashv \Phi'' \,|\, \Sigma'';\theta_2}{\Phi|\Sigma|\Gamma \vdash EN \approx^?_{=_\beta} E'N' \dashv \Phi'' \,|\, \Sigma'';\theta_2 \circ \theta_1} \textsc{EApp-EApp}$$

$$\frac{\Phi \vDash s \mathscr{R} s' \dashv \Phi'}{\Phi|\Sigma|\Gamma \vdash s \approx^?_{\mathscr{R}} s' \dashv \Phi' \,|\, \Sigma;\mathrm{Id}_\Sigma}\ \textsc{Sort-Sort} \qquad \frac{\Phi \vDash (i_k = j_k)_k \dashv \Phi^* \quad \Phi^*|\Sigma|\Gamma \vdash E \approx^?_{=_\beta} E' \dashv \Phi' \,|\, \Sigma';\theta}{\Phi|\Sigma|\Gamma \vdash E[c[\vec{i}]] \approx^?_{\mathscr{R}} E'[c[\vec{j}]] \dashv \Phi \,|\, \Sigma;\theta}\ \text{E[c]-E[c]}$$

$$\frac{?X : \Pi \overrightarrow{x_j^{B_j}}.T[\overrightarrow{y_i : A_i}] \in \Sigma \quad \theta^* := (?X \backslash \lambda \overrightarrow{x_j^{B_j}}.?X'[\overrightarrow{y_i}, \overrightarrow{x_i}]) \quad \Phi|\Sigma\{\!\{\theta^*\}\!\}|\Gamma \vdash ?X'[\sigma_r, \overrightarrow{z_j}] \approx^?_{\mathscr{R}} t\{\!\{\theta^*\}\!\} \dashv \Phi' \,|\, \Sigma';\theta \quad \mathscr{R} \in \{=_\beta, \leq_\beta, \geq_\beta\}^a}{\Phi|\Sigma|\Gamma \vdash ?X[\sigma_r]\ z_1\ \cdots\ z_n \approx^?_{\mathscr{R}} t \dashv \Phi' \,|\, \Sigma';\theta \circ \theta^*} \textsc{NormMeta}$$

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

$$\frac{?X : \_[\Psi] \quad \Phi \,|\, \Sigma \vdash \mathrm{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} N = \Phi' \dashv \Sigma';\theta;N' \quad \mathscr{R} := \mathscr{R}' \text{ reversed}^b}{\Phi|\Sigma|\Gamma \vdash N \approx^?_{\mathscr{R}'} ?X[\sigma] \dashv \Phi \,|\, \Sigma'\{\!\{(?X \backslash N')_\Psi\}\!\};(?X \backslash N')_\Psi \circ \theta}\ \textsc{Meta-InstL}$$

(b) Untyped unification $\boxed{\Phi|\Sigma|\Gamma \vdash t \approx^?_{\mathscr{R}} u \dashv \Phi' \,|\, \Sigma';\theta}$

[a] Abuse of notation to represent two rules as one
[b] If $\mathscr{R}'$ is $\leq$ then $\mathscr{R}$ is $\geq$, etc.

Figure 4: Unification algorithm

- a restriction of Unicoq's setting to the relevant fragment for pattern unification.

Definition 29 (FCU algorithm)**.** Figure 4 defines the unification algorithm.

$\boxed{\Phi|\Sigma|\Gamma \vdash t^T \approx^?_{\mathscr{R}} u^U \dashv \Phi' \mid \Sigma';\theta}$ unifies two well-typed terms and their types.

$\boxed{\Phi|\Sigma|\Gamma \vdash t \approx^?_{\mathscr{R}} u \dashv \Phi' \mid \Sigma';\theta}$ unifies two terms with the precondition that their types are convertible.

For both judgements, the inputs and outputs are respectively on the left and on the right of the $\dashv$ symbol. □

For the remaining of this subsection, we describe the algorithm on a high level by exhibiting its general structure and expanding on its pivot points, while pointing out where it differs from Unicoq, and when special care is needed compared to the simply-typed setting.

3.2.1. Unifying types first

The core algorithm, defined in Figure 4b, expects two terms of the convertible types and relies on that fact in its rules. To achieve that, we wrap it by the global procedure with the unique rule UnifTypes, which first unifies the two types of the terms, and only then unifies the terms. Pfenning (1991) mentioned this already in his presentation, and here we chose to exhibit it explicitly as a rule. This way of proceeding is correct and complete thanks to Lemma 20.

The types themselves are unified using the untyped and syntax directed algorithm, and this is possible because of Theorem 9, which ensures that the types of types are sorts, and we can unify those with the universe constraint checking procedure (Figure 1c[18].

3.2.2. The "algorithm" is an algorithm

We are reffering to the inference system of Figure 4 as an algorithm. Let us stress here that it does in fact precisely specify a deterministic recursive procedure: in each rule, the inputs of a premise depend either on the inputs of the conclusion or on the outputs of a preceding premise.

3.2.3. Structural rules

The sturctural rules, namely Π-Π, $\lambda\lambda$, $\lambda\eta$, $\eta\lambda$, E[x]-E[x], EApp-EApp and □-□ are almost identical to those of Unicoq. We traverse the binders and the neutral terms, and we unifiy sequentially their subterms. When unifying the premises of products or the arguments of a neutral application, we downcast the unification kind to convertibililty, which corresponds to the contextual rules of subtyping (Figure 2b).

The only difference with Unicoq's corresponding rules is that here we have the precondition that the two sides of the equation are of the same type, and thus when applying the rules $\lambda\lambda$, $\lambda\eta$ and $\eta\lambda$, we can safely assume that the premise types of the product are equal, whereas Unicoq first unifies the premise types before proceeding.

3.2.4. Sorts and polymorphic constants

The rules E[c]-E[c] and Sort-Sort update the universe constraints according to equations between sorts and between constant instances. Again, these correspond exactly to their Unicoq counterparts.

Apart from these two rules, we will see later that there is another rule that can refine universe constraints when comparing a metavariable with a sort.

[18] Remember we mentioned in Section 1.1 that such a procedure is decidable and can be efficiently implemented when certain invariants are maintained, and that we do not concern ourselves with that here as it is outside of our scope

3.2.5. Instantiating metavariables

We are left with the META-INSTR rule[19]. When we have an equation of the form $?X[\sigma] \approx N$, we have to find a term $N[\sigma]^{-1}$ as the body of $?X$. More precisely, we try to express N as a term only composed of constants, metavariables and terms appearing in σ. This is the operation "$\Phi \mid \Sigma \vdash \text{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} N = N[\sigma]^{-1} \dashv \Phi'; \Sigma'; \theta$" defined in Figure 5a.

Inversion may update the meta contexts (and produce the corresponding substitution) when doing *pruning*, an operation discarding arguments of metavariables occuring in N which would make the inversion fail otherwise. It can also update the universe context when inverting a sort occurring in the conclusion of a pattern in a cumulativity situation (see rule INV-SORT).

Thanks to the global and local restrictions, there is always at most a single way to invert the term.

Because of the rules $\lambda\eta$ and $\eta\lambda$, there can be problems of the form $?X[\sigma]\ z_1 \cdots z_n \approx N$. Unicoq handles these cases directly in its META-INSTR rule, but here we use the NORMMETA rule to normalize the representation of arguments of metavariables.

3.2.6. Pruning

When we have an inversion of the form $?X[\sigma] \approx c_0(c_1 x)(?Y[c_2 x])$, if we can't invert $c_1 x$ the inversion fails and the equation has no solution. However, if we can't invert $c_2 x$, all is not lost, and we can invert the term with the condition that $?Y$ does not use its first argument. The operation $\Phi \vdash \text{prune\_invert}_{\sigma_r \mapsto \Psi}(\Phi_Y \mapsto \sigma_Y) = \Phi'_Y \mapsto \sigma'_Y$ inverts the spine of a metavariable, and discards the arguments that cannot be inverted on the fly, producing an inverted spine in a subcontext of the initial one. The rule INV-META then uses the result to replace $?Y$ by applying the substituion $(?Y \backslash ?Y'[\widehat{\Psi'_Y}])_{\Psi_Y}$.

3.3. Synthesis, weaknesses and desired extensions

We presented a functions-as-constructors unification algorithm with restrictions that are adapted to settings with cumulativity and polymorphism by introducing the notion of subterms modulo universes.

Weaknesses. While this the restrictions of this presentation allow us to make use of the well-behaved normal forms to study the adaptability of FCU with PCUC, it does not reflect practical use as in any realistic implementation, it would be too costly to systematically normalize terms. We would like to exdend our presentation by allowing non-normal terms, definitions and inductives, and try to preserve soundness and completeness while applying reductions conservatively.

The main point of that formalization is to prove soundness and completeness, unfortunately I was not able do that on time, and as such the next step is to try to implement this presentation in MetaRocq and prove it correct and complete.

Extensions. Apart from having less impractical restrictions as mentioned above, we would like to adapt two ideas in particular in our setting.

- *Adding definitions together with syntactic criteria for injectivity:* Currently, if we added defined constants to our system, it would not change anything in the unification algorithm, as any occurence of a constant (as a head of a neutral term, or inside a restricted argument of a metavariable) has to be expanded if we want to ensure completeness. However, Pfenning and Schürmann (1998) presented a method for approximating syntactically the notion of injectivity of definitions, allowing to view them as rigid constants. If we succeed in that adaptation, it would considerably broaden the class of problems handled by FCU. One challenge in this approach is that in order to be complete, we also need to fold any expanded occurence of a constant before inverting a term during variable instantiation. This looks conceptually interesting as folding rigid definitions can be seen as an operation of the same nature as the *inversion* operation in the context of FCU.

- *Handling Σ-types (i.e records) inside patterns:* Abel and Pientka (2011) managed to generalize simple pattern unification to handle records, by elimitating those into prooducts using several type isomorphisms. However, their work is not easily adaptable to ours since they have a type directed algorithm and furthermore do not have first class sorts (variables can't be instantiated with types, so in particular all

[19] And its symmetric counterpart META-INSTL

$$\frac{\Phi \mid \Sigma \vdash \mathrm{invert}^{=_\beta}_{\sigma_r \mapsto \Psi} P_2 = P_2' \dashv \Phi'; \Sigma'; \theta_1 \qquad \Phi' \mid \Sigma' \vdash \mathrm{invert}^{\mathscr{R}}_{\sigma_r, x \mapsto \Psi, x} P_2 \{\!\{\theta_1\}\!\} = P_2' \dashv \Phi''; \Sigma''; \theta_2 \qquad \mathscr{B} \in \{\lambda, \Pi\}}{\Phi \mid \Sigma \vdash \mathrm{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} \mathscr{B} x^{P_2}.P = \mathscr{B} x^{P_2'}.P' \dashv \Phi''; \Sigma''; \theta_2 \circ \theta_1} \textsc{Inv-Bind}$$

$$\frac{\Phi \vdash (E[a] \mapsto x) \in \sigma_r \mapsto \Phi}{\Phi \mid \Sigma \vdash \mathrm{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} E[a] = x \dashv \Phi; \Sigma; \mathrm{Id}_\Sigma} \textsc{Inv-E[a]}_1$$

$$\frac{\Phi \nvdash (E[a] \mapsto \_) \in \sigma_r \mapsto \Phi \qquad \Phi \mid \Sigma \vdash \mathrm{invert}^{=_\beta}_{\sigma_r \mapsto \Psi} E = E' \dashv \Phi'; \Sigma'; \theta}{\Phi \mid \Sigma \vdash \mathrm{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} E[a] = E'[a] \dashv \Phi'; \Sigma'; \theta} \textsc{Inv-E[a]}_2$$

$$\frac{\Phi \mid \Sigma \vdash \mathrm{invert}^{=_\beta}_{\sigma_r \mapsto \Psi} E = E' \dashv \Phi_1; \Sigma_1; \theta_1 \qquad \Phi_1 \mid \Sigma_1 \vdash \mathrm{invert}^{=_\beta}_{\sigma_r \mapsto \Psi} N \{\!\{\theta_1\}\!\} = N' \{\!\{\theta_1\}\!\} \dashv \Phi_2; \Sigma_2; \theta_2}{\Phi \mid \Sigma \vdash \mathrm{invert}^{=_\beta}_{\sigma_r \mapsto \Psi} EN = E'N' \dashv \Phi_2; \Sigma_2; \theta_2 \circ \theta_1} \textsc{Inv-App}$$

$$\frac{}{\Phi \mid \Sigma \vdash \mathrm{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} \square = \square \dashv \Phi; \Sigma; \mathrm{Id}_\Sigma} \textsc{Inv-Square}$$

$$\frac{(\vec{l}; \mathscr{C}) \coloneqq \Phi \qquad i \text{ fresh in } \vec{l} \qquad \Phi' \coloneqq (\vec{l}, i; \mathscr{C} \wedge i \mathscr{R} \mathrm{max}(\vec{p})) \qquad \Phi' \vDash}{\Phi \mid \Sigma \vdash \mathrm{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} \mathbf{Type}(\vec{p}) = \mathbf{Type}(i) \dashv \Phi'; \Sigma; \mathrm{Id}_\Sigma} \textsc{Inv-Sort}$$

$$\frac{?Y : T[\Psi] \in \Sigma \qquad \Phi \vdash \mathrm{prune\_invert}_{\sigma_r \mapsto \Psi}(\Psi_Y \mapsto \sigma_Y) = \Psi_Y' \mapsto \sigma_Y' \qquad \Phi | \Sigma | \Psi_Y' \vdash T : s \qquad \theta \coloneqq (?Y \setminus ?\widehat{Y[\Psi_Y']})_{\Psi_Y}}{\Phi \mid \Sigma \vdash \mathrm{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} ?Y[\sigma_Y] = ?Y[\sigma_Y'] \dashv \Phi; \Sigma\{\!\{\theta\}\!\}; \theta} \textsc{Inv-meta}$$

(a) Inversion of patterns $\boxed{\Phi \mid \Sigma \vdash \mathrm{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} P = P' \dashv \Phi'; \Sigma'; \theta}$ and $\boxed{\Phi \mid \Sigma \vdash \mathrm{invert}^{\mathscr{R}}_{\sigma_r \mapsto \Psi} E = E' \dashv \Phi'; \Sigma'; \theta}$

$$\frac{}{\Phi \vdash \mathrm{prune\_invert}_{\sigma_r \mapsto \Psi}(\langle\rangle \mapsto \cdot) = \langle\rangle \mapsto \cdot} \textsc{Prune-Base}$$

$$\frac{\Phi \vdash \mathrm{prune\_invert}_{\sigma_r \mapsto \Psi}(\Psi_Y \mapsto \sigma_Y) = \Psi_Y' \mapsto \sigma_Y' \qquad \Phi \mid \Sigma \vdash \mathrm{invert}^{=_\beta}_{\sigma_r \mapsto \Psi} P = P' \dashv \Psi; \Sigma; \mathrm{Id}_\Sigma}{\Phi \vdash \mathrm{prune\_invert}_{\sigma_r \mapsto \Psi}(\Psi_Y, x : T \mapsto \sigma_Y, P) = \Psi_Y', x : T \mapsto \sigma_Y', P'} \textsc{Prune-Keep}$$

$$\frac{\Phi \vdash \mathrm{prune\_invert}_{\sigma_r \mapsto \Psi}(\Psi_Y \mapsto \sigma_Y) = \Psi_Y' \mapsto \sigma_Y' \qquad \Phi \mid \Sigma \nvdash \mathrm{invert}^{=_\beta}_{\sigma_r \mapsto \Psi} P = P' \dashv \Psi; \Sigma; \mathrm{Id}_\Sigma}{\Phi \vdash \mathrm{prune\_invert}_{\sigma_r \mapsto \Psi}(\Psi_Y, x : T \mapsto \sigma_Y, P) = \Psi_Y' \mapsto \sigma_Y'} \textsc{Prune-Discard}$$

(b) Pruning $\boxed{\Phi \vdash \mathrm{prune\_invert}_{\sigma_r \mapsto \Psi}(\Phi' \mapsto \sigma') = \Phi'' \mapsto \sigma''}$

$$\frac{\Phi \vdash N_e \equiv N_e'}{\Phi \vdash (N_e \mapsto x) \in \sigma_r, N_e' \mapsto \Phi, x} \qquad \frac{\Phi \vdash (N_e \mapsto x) \in \sigma_r \mapsto \Phi}{\Phi \vdash (N_e \mapsto x) \in \sigma_r, M \mapsto \Phi, y}$$

(c) Occurence in spine modulo universe $\boxed{\Phi \vdash (N_e \mapsto x) \in \sigma_r \mapsto \Phi}$

Figure 5: Inversion and pruning

record terms are known at the start of unification). We would like to exhibit some invariant or additional conditions which would allow us to state that if a metavariable has a solution as a record type (for example), our pattern algorithm can discover it and eliminate it without being stuck. Another approach, proposed by Kovács and Bocquet (2023), is a purely syntax directed and relies on a generalization of the substitution inversion operation.

4. Implementation in Unicoq

4.1. FCU in a dynamic setting

We implemented functions-as-constructors unification in place of simple pattern unification in the Unicoq plugin. Since the structural and computational rules of Unicoq are more general and permissive than the formalization we presented, we only replaced the rule responsible for pattern unification, namely Meta-InstR.

Our presentation of the Meta-InstR rule in Section 3.2 is accurate to our implementation, where we invert a term and collect metavariable offending arguments to be pruned on the fly.

An important difference though is the fact that Unicoq operates in a dynamic setting: an equation does not necessarily verify the local and global restrictions. Therefore before proceding with the inversion, we must check for each metavariable pair that there is no argument occuring as a subterm to another argument. To achieve this efficiently, it was necessary to be able to compare two restricted arguments in constant time, which we could achieve thanks to the hash consing infrastructure available in Rocq's source code.

Apart from this detail, the implementation did not require much structural change, and the total length of the inversion code was about 150 lines, while the old inversion code was about 100 lines.

4.2. A heuristics rule

We also tried to use FCU unification as a heuristic by allowing inductive constructors or defined constants to occur as arguments to metavariables, giving us a more fine grained alternative to the first-order heuristic implemented in Unicoq.

4.3. Results

To test our implementation, we measured and compared the execution time and the number of instances covered by pattern unification with and without the FCU extension, on various Rocq libraries and theories.

The addition of FCU did not cause a gain in execution time. However, in all the libraries we tested, it did not allow to solve more problems than Unicoq (even with the heuristics mentioned above turned on).

Thus, although adding functions-as-constructors unification does not have too big of a cost in code complexity and efficiency, we did not gather enough evidence to justify its implementation.

References

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 (2017). "A Functional Implementation of Function-as-Constructor Higher-Order Unification". In: *Proceedings of the 31s International Workshop on Unification*. URL: https://unif-workshop.github.io/UNIF2017/papers/UNIF_2017_paper_10.pdf.

– (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.

Herbelin, Hugo (2005). *Type Inference with Algebraic Universes in the Calculus of Inductive Constructions*. URL: https://pauillac.inria.fr//~herbelin/publis/univalgcci.pdf.

Huet, Gérard (1987). *Extending the Calculus of Constructions with Type:Type*. Unpublished draft. URL: https://pauillac.inria.fr/~huet/PUBLIC/typtyp.pdf.

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.

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

Levy, Jordi and Margus Veanes (2000). "On the Undecidability of Second-Order Unification". In: *Inf. Comput.* 159.1-2, pp. 125–150. DOI: 10.1006/INCO.2000.2877. URL: https://doi.org/10.1006/inco.2000.2877.

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.

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.

Sozeau, Matthieu, Simon Boulier, et al. (Dec. 2019). "Coq Coq correct! verification of type checking and erasure for Coq, in Coq". In: *Proc. ACM Program. Lang.* 4.POPL. DOI: 10.1145/3371076. URL: https://doi.org/10.1145/3371076.

Sozeau, Matthieu, Yannick Forster, 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.

Sozeau, Matthieu and Nicolas Tabareau (2014). "Universe Polymorphism in Coq". In: *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*. Ed. by Gerwin Klein and Ruben Gamboa. Vol. 8558. Lecture Notes in Computer Science. Springer, pp. 499–514. DOI: 10.1007/978-3-319-08970-6_32. URL: https://doi.org/10.1007/978-3-319-08970-6%5C_32.

Winterhalter, Théo (2020). "Formalisation and Meta-Theory of Type Theory". PhD thesis. URL: http://www.theses.fr/2020NANT4012/document.

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.

Appendix A Unification and Inductives

This appendix serves as an argument and elaboration for the claim that the inductive types play no essential part in current unification algorithms for dependent-type theories such as Agda's or Rocq's (TODO: reformuler en mieux) ; it is also a tentative presentation of the ideas I have for integrating inductives as "first-class citizens" in unification.

In this appendix, I present the rules one would add to the algorithm if we were to add dependent inductive types, and I try to elaborate on why it is unsatisfactory for me, trivial/treats inductives/match constructs as

rigid objects (?).

A.1 Syntax for inductives

A.2 Rules

A.3 Tentative pattern fragment (?)