# Inductive Consequences in the Calculus of Constructions*

Daria Walukiewicz-Chrząszcz and Jacek Chrząszcz

Institute of Informatics, Warsaw University
ul. Banacha 2, 02-097 Warsaw, Poland
{daria,chrzaszcz}@mimuw.edu.pl

**Abstract.** Extending the calculus of constructions with rewriting would greatly improve the efficiency of proof assistants such as Coq. In this paper we address the issue of the logical power of such an extension.
In our previous work we proposed a procedure to check completeness of user-defined rewrite systems. In many cases this procedure demonstrates that only a basic subset of the rules is sufficient for completeness. Now we investigate the question whether the remaining rules are inductive consequences of the basic subset.
We show that the answer is positive for most practical rewrite systems. It is negative for some systems whose critical pair diagrams require rewriting under a lambda. However the positive answer can be recovered when the notion of inductive consequences is modified by allowing a certain form of functional extensionality.

## 1 Introduction

Theorem provers based on type theory and Curry-Howard Isomorphism, such as Coq [9], are built on top of a powerful computing mechanism which is the lambda calculus. The typing rule which allows for integration of computation into the typing system is called the conversion rule.

Conversion is based on the terminating syntactic reduction relation, it is decidable and hence rather weak. A separate, stronger propositional equality used for reasoning is defined as an inductive predicate with no special treatment in the typing systems. The negative consequences of such approach are the dificulties in handling propositional equalities. For example, if addition is defined by induction on the first argument then even though $\forall x{:}nat,\ x+0=x$ can be proved by induction, the type `vector (n+0)` is not convertible to `vector n` while `vector (0+n)` is.

One solution is to look for new type theories where equality has better reasoning properties without compromising its computational properties [20,2,1]. Another approach is to try to put more power in the computation part, either by adding specific decision procedures to the existing conversion relation, e.g.

* This work was partly supported by Polish government grant N N206 355836.

congruence closure [6], or simply extending the reduction mechanism with user-defined rewriting rules. This paper is concerned with the latter possibility.

Of course adding arbitrary rewrite rules may easily break strong normalization or confluence and hence compromise good meta-theoretical properties of the formalism. Since these properties are undecidable for rewriting systems, there are a number of incomplete decidable criteria that ensure termination and that are flexible enough to be met by many useful rewriting systems [3,5,4,16,17]. Once termination is established an easy test of joinability of critical pairs can tell if the rewriting system is also confluent. A terminating and confluent rewriting system leads to the decidable conversion—it is sufficient to syntactically compare the unique normal forms of inspected terms.

There are two most important roles a rewriting system may play in a theorem prover. First, it can provide a means to decide e.g. a word problem in certain axiomatic theories, for example in group theory. Second, it can be used to define functions, similarly to definitions by pattern matching, but in a more straightforward way. Moreover, a definition by rewriting can contain rules which add more equations to conversion compared to a regular definition by pattern matching. Consider again the addition on unary natural numbers. Using rewriting one can define it in a symmetric way and even include associativity, therefore making the conversion richer and hence the proofs shorter and more automatic.

```
0 + y   ⟶  y                x + 0   ⟶  x
(S x) + y ⟶ S (x + y)       x + (S y) ⟶ S (x + y)
                            (x + y) + z ⟶ x + (y + z)
```

In order to trust a proof developed with the help of a proof assistant, one must make sure the development is logically consistent. One can show that it is the case for all developments in which there are no axioms and all definitions by rewriting are complete, i.e. the functions they define are total [4,18]. An automatic procedure to check completeness of definitions by rewriting is also provided in our paper [18].

It turns out that in many cases this procedure uses only a subset of the rules given by the user, demonstrating that this basic subset is already complete. The basic subset roughly corresponds to a definition by cases one can write for example in Coq or a definition using recursors. In this paper we analyse the logical power of the rules that are outside the basic subset to understand how strong can be the generated conversion relation. These additional rules seem to be quite arbitrary even though we know that the whole set of rules is terminating and confluent. Throughout this paper we investigate the question whether the additional rules are inductive consequences of the basic subset.

In the example given above, the set of five rules is strongly normalizing and confluent. The additional three rules (in the right column) are indeed inductive consequences of the basic complete system (in the left column) because all constructor instances of their respective left- and right-hand sides are joinable using the basic set of rules. This can be generalized to all first order rewriting systems (see e.g. Theorem 7.6.5 in [15]).

The standard notion of an inductive consequence [10], meaning an equation which is valid for all ground instances, must of course be adapted to the higher-order setting. Instead of ground instances, we use instances typeable in closed environments. Our proof technique works by analysing critical pairs formed by additional rules with rules from the basic complete subsystem. In order to transfer the critical pair diagrams to the typed context we impose some slight restrictions on the form of rules and critical pairs.

The first result of the paper concerns the majority of useful rewriting systems: if confluence of critical pairs can be proved without rewriting under a binder, additional rules are inductive consequences of the basic subset (Section 6, Theorem 1). This of course includes all first order rewriting systems, but also definitions over dependently typed symbols and higher-order functions as used in functional programming languages.

The situation is more difficult when rewriting under a binder is necessary to join the critical pair diagrams (Section 7). This is usually the case for functions defined over functional inductive types. To get the positive result in this case (Theorem 2), we must modify the notion of inductive consequences, allowing for a more lax comparison of functional terms, similar in spirit to functional extensionality. Otherwise there are examples where closed instances of left- and right-hand sides of additional rules are not joinable using the basic subset.

From the perspective of a proof assistant user, our results are reassuring about relevance of definitions by rewriting with additional rules. Indeed, for a given definition by rewriting, if the additional rules are shown to be inductive consequences, it means they are valid equations on closed terms, even if they are not necessarily provable as lemmas. Another motivation concerns extraction [12] in a system with rewriting. Although the extraction mechanism can only treat rules which resemble pattern matching, it turns out that the others do not affect the results of computation even for higher order and polymorphic functions.

This paper is the full version of [19]. In the appendix, there are all proofs and examples that have been removed from [19] due to space limitation.

## 2 Motivating examples

In a future version of Coq with rewriting, apart from definitions, axioms and inductive definitions, a user would be allowed to enter definitions by rewriting, even for higher-order and polymorphic functions:[1]

```
Inductive  list (A:Set) : Set  :=  nil  |  cons : A → list A → list A
Symbol map : forall A:Set, (A → A) → list A → list A
Rules
  map A f (nil A)  ⟶  nil A
  map A f (cons A a l)  ⟶  cons A (f a) (map A f l)
  map A (fun x ⇒ x) l  ⟶  l
Parameter l : list nat.
```

[1] The syntax of the definition by rewriting is inspired by the experimental "recriture" branch of Coq developed by Blanqui. For the sake of clarity we omit certain details, like environments of rule variables.

The above fragment can be interpreted as an environment consisting of the inductive definition of lists, a definition by rewriting of the `map` function and the declaration of a variable `l` of type `list nat`. In this environment all rules for `map` contribute to conversion. They form a terminating and confluent rewriting system in which the first two rules completely define `map`. The third rule is an inductive consequence of the first two (by Theorem 1) and, transformed into equation, can be proved by induction.

Even though we consider higher-order rewriting, we choose the simple matching modulo $\alpha$-conversion to match a rule to a term. Higher-order matching is useful for example to encode logical languages by higher-order abstract syntax, but it is seldom used in Coq where modeling relies rather on inductive types.

Let us consider another example, the inductive identity function on Brouwer's ordinals:

```
Inductive ord : Set  :=
  o : ord  |  s : ord → ord  |  lim : (nat → ord) → ord.

Symbol id : ord → ord
Rules
  id o ⟶ o
  id (s x) ⟶ s (id x)
  id (lim f) ⟶ lim (fun n ⇒ id (f n))
  id (id x) ⟶ id x
```

This set of rules is also terminating and confluent and the first three rules completely define `id`. The fourth rule says that `id` is an idempotent function. This time, because `ord` is a functional inductive type, the fourth rule cannot be proved to be an inductive consequence of the first three without weakening the notion of inductive consequences. This example will be discussed in details in Section 7. Note however that $\forall x : \texttt{ord},\ \texttt{id}\ (\texttt{id}\ x) = x$ can be proved in Coq assuming the functional extensionality axiom $\forall f\ g\colon A \to B, (\forall x\colon A, f\,x = g\,x) \to f = g$.

The last example in this section is the substitutivity property of equality:

```
Inductive  eq (A:Set)(a:A) : A → Prop  :=  refl : eq A a a.

Symbol subst : forall (A: Set)(P: A → Set)(a b: A)(p: eq A a b)(x: P a), P b
Rules
 subst A P a a (refl A a) x ⟶ x
 subst A P a a p x ⟶ x
```

This set of rules is also terminating and confluent and already the first rule completely defines `subst`. The second rule is an inductive consequence of the first one (by Theorem 1) but this time the corresponding equality lemma `subst_eq` $\forall(A : Set)(P : A \to Set)(a : A)(p : eq\,A\,a\,a)(x : P\,a),\ subst\,A\,P\,a\,a\,p\,x = x$, is unprovable in the environment where we have only the standard (like in Coq) elimination of equality. Indeed, one can show[2] that `subst_eq` implies Streicher's axiom $K$ which is not derivable from the standard elimination of equality [11].

[2] See for example `http://coq.inria.fr/stdlib/Coq.Logic.EqdepFacts.html`

## 3 Pure type systems with generative definitions

Even though most papers motivated by the development of Coq concentrate on the calculus of constructions, we present here a slightly more general formalization of a pure type system with inductive definitions and definitions by rewriting. The presentation, taken from [7,8,18], is quite close to the way these elements are and could possibly be implemented in Coq. The formalism is built upon a set of PTS sorts $\mathcal{S}$, a binary relation $\mathcal{A}$ and a ternary relation $\mathcal{R}$ over $\mathcal{S}$ governing the typing rules **(Term/Ax)** and **(Term/Prod)** respectively (Fig. 1). The syntactic class of pseudoterms is defined as follows:

$$t ::= v \mid \mathrm{s} \mid (t_1\ t_2) \mid (\lambda v{:}t_1.t_2) \mid (\forall v{:}t_1.t_2)$$

A pseudoterm can be a variable $v \in \mathit{Var}$, a sort $\mathrm{s} \in \mathcal{S}$, an application, an abstraction or a product. Pseudoterms are identified with finite labelled trees; a $\lambda v$ and a $\forall v$ are binary nodes with the first child corresponding to the type of the variable $v$ and the second to the body of the abstraction (product).

Positions are strings of positive integers. The subterm of $t$ at position $p$ is denoted by $t|_p$ while $t_1[t_2]_p$ stands for the result of replacing $t_1|_p$ with $t_2$ in $t_1$. We use $FV(t)$ to denote the set of free variables of a term $t$. For convenience we assume that all bound variables are different and are different from the free ones.

We use Greek letters $\gamma, \delta$ to denote substitutions which are finite partial maps from variables to pseudoterms. The postfix notation is used for the application of substitutions. We write $[t/x]$ for the substitution of $t$ to a variable $x$.

Inductive definitions and definitions by rewriting are *generative*, i.e. they are stored in the environment and are used in terms only through names they "generate". An environment is a sequence of declarations, each of them is a variable declaration $v : t$, an inductive definition $\mathsf{Ind}(\Gamma^I := \Gamma^C)$, where $\Gamma^I$ and $\Gamma^C$ are environments providing names and types of (possibly mutually defined) inductive types and their constructors, or a definition by rewriting $\mathsf{Rew}(\Gamma, R)$, where $\Gamma$ is an environment providing names and types of (possibly mutually defined) function symbols and $R$ is a set of rewrite rules defining them. A rewrite rule is a triple denoted by $G \vdash l \longrightarrow r$, where $l$ and $r$ are pseudoterms and $G$ is an environment, assigning types to variables occurring in the left- and right-hand sides $l$ and $r$. Each $l$ is of the form $f(l_1, \ldots, l_n)$ where $f \in \Gamma$.

Given an environment $E$, inductive types, constructors and function symbols declared in $E$ are called constants (even though syntactically they are variables). General environments are denoted by $E$ and the environment containing only variable declarations are denoted by $\Gamma$, $\Delta$, $G$, $D$. We assume that names of all declarations in environments are pairwise disjoint. The set of all variables declared in an environment $\Gamma$ is denoted by $\mathit{dom}(\Gamma)$.

Given a term $t$ and a position $p$, we write $\Gamma(t, p)$ to denote the environment of variables that are bound in $t$ on the path from the root to $p$.

**Definition 1.** *A* pure type system with generative definitions *is defined by the typing rules in Fig. 1, where:*

$$\text{Let}\quad \Gamma^I = I_1 : t_1^I \dots I_n : t_n^I \quad\text{and}\quad \Gamma^C = c_1 : t_1^C \dots c_m : t_m^C$$

$$\frac{E \vdash t_j^I : \mathrm{s}_j \qquad t_j^I = \forall(\overrightarrow{z : Z_j})\,\mathrm{s}'_j \qquad \text{for } j = 1 \dots n \\ E; \Gamma^I \vdash t_i^C : \hat{\mathrm{s}}_i \qquad t_i^C = \forall(\overrightarrow{z : Z'_i})\, I_{j_i}\vec{w} \qquad \text{for } i = 1 \dots m}{E \vdash \mathsf{Ind}(\Gamma^I := \Gamma^C) : \mathsf{correct}} \quad \text{if } \mathsf{POS}_E(\Gamma^I := \Gamma^C)$$

$$\text{Let}\quad \Gamma = f_1 : t_1 \dots f_n : t_n \quad\text{and}\quad R = \{G_i \vdash l_i \longrightarrow r_i\}_{i=1\dots m}, \text{ where } G_i = x_1^i : t_1^i; \dots; x_{n_i}^i : t_{n_i}^i$$

$$\frac{E \vdash t_k : \mathrm{s}_k \quad \text{for } k = 1 \dots n \\ E; G_i \vdash \mathtt{ok} \quad FV(l_i, r_i) \subseteq G_i \quad \text{for } i = 1 \dots m}{E \vdash \mathsf{Rew}(\Gamma, R) : \mathsf{correct}} \quad \text{if } \mathsf{ACC}_E(\Gamma, R)$$

$$\frac{}{\epsilon \vdash \mathtt{ok}} \qquad \frac{E \vdash \mathtt{ok} \quad E \vdash t : \mathrm{s}}{E; v : t \vdash \mathtt{ok}}$$

$$\frac{E \vdash \mathtt{ok} \quad E \vdash \mathsf{Ind}(\Gamma^I := \Gamma^C) : \mathsf{correct}}{E; \mathsf{Ind}(\Gamma^I := \Gamma^C) \vdash \mathtt{ok}} \qquad \frac{E \vdash \mathtt{ok} \quad E \vdash \mathsf{Rew}(\Gamma, R) : \mathsf{correct}}{E; \mathsf{Rew}(\Gamma, R) \vdash \mathtt{ok}}$$

$$\frac{E_1; v : t; E_2 \vdash \mathtt{ok}}{E_1; v : t; E_2 \vdash v : t}$$

$$\frac{E \vdash \mathtt{ok}}{E \vdash I_i : t_i^I} \qquad \frac{E \vdash \mathtt{ok}}{E \vdash c_i : t_i^C} \qquad \text{where } \begin{cases} E = E_1; \mathsf{Ind}(\Gamma^I := \Gamma^C); E_2 \\ \Gamma^I = I_1 : t_1^I \dots I_n : t_n^I \\ \Gamma^C = c_1 : t_1^C \dots c_m : t_m^C \end{cases}$$

$$\frac{E \vdash \mathtt{ok}}{E \vdash f_i : t_i} \qquad \frac{E \vdash \mathtt{ok} \quad \delta : G_i \to E}{E \vdash l_i\delta \longrightarrow_R r_i\delta} \qquad \text{where } \begin{cases} E = E_1; \mathsf{Rew}(\Gamma, R); E_2 \\ \Gamma = f_1 : t_1 \dots f_n : t_n \\ R = \{G_i \vdash l_i \longrightarrow r_i\}_{i=1\dots m} \end{cases}$$

**(Term/Prod)**
$$\frac{E \vdash t_1 : \mathrm{s}_1 \quad E; v : t_1 \vdash t_2 : \mathrm{s}_2}{E \vdash \forall v{:}t_1.t_2 : \mathrm{s}_3} \text{ where } (\mathrm{s}_1, \mathrm{s}_2, \mathrm{s}_3) \in \mathcal{R}$$

**(Term/Abs)**
$$\frac{E; v : t_1 \vdash e : t_2 \quad E \vdash \forall v{:}t_1.t_2 : \mathrm{s}}{E \vdash \lambda v{:}t_1.e : \forall v{:}t_1.t_2}$$

**(Term/Ax)**
$$\frac{E \vdash \mathtt{ok}}{E \vdash \mathrm{s}_1 : \mathrm{s}_2} \text{ where } (\mathrm{s}_1, \mathrm{s}_2) \in \mathcal{A}$$

**(Term/App)**
$$\frac{E \vdash e : \forall v{:}t_1.t_2 \quad E \vdash e' : t_1}{E \vdash e\ e' : t_2\{v \mapsto e'\}}$$

**(Term/Conv)**
$$\frac{E \vdash e : t \quad E \vdash t' : \mathrm{s} \quad E \vdash t \approx t'}{E \vdash e : t'}$$

**Fig. 1.** Definition correctness, environment correctness and lookup, PTS rules

- *The relation $\approx$ used in the rule* **(Term/Conv)** *is the smallest congruence on well typed terms, generated by $\longrightarrow$ which is the sum of beta and rewrite reductions, denoted by $\longrightarrow_\beta$ and $\longrightarrow_R$ respectively (for the exact definition see [8], Section 2.8).*
- *The notation $\delta : \Gamma \to E$ means that $\delta$ is a* well-typed substitution*, i.e. $E \vdash v\delta : t\delta$ for all $v : t \in \Gamma$.*

As in [17,4], recursors and their reduction rules have no special status and they are supposed to be expressed by rewriting.

**Assumptions.** We assume that we are given a positivity condition $\mathsf{POS}$ for inductive definitions and an acceptance condition $\mathsf{ACC}$ for definitions by rewriting. Together with the right choice of the PTS they must imply the following properties:

**P1** subject reduction, i.e. $E \vdash e : t$, $E \vdash e \longrightarrow e'$ implies $E \vdash e' : t$
**P2** uniqueness of types, i.e. $E \vdash e : t$, $E \vdash e : t'$ implies $E \vdash t \approx t'$.
**P3** strong normalization, i.e. $E \vdash \mathtt{ok}$ implies that reductions of all well-typed terms in $E$ are finite
**P4** confluence, i.e. $E \vdash e : t$, $E \vdash e \longrightarrow^* e'$, $E \vdash e \longrightarrow^* e''$ implies $E \vdash e' \longrightarrow^* \hat{e}$ and $E \vdash e'' \longrightarrow^* \hat{e}$ for some $\hat{e}$.

These properties are usually true in all well-behaved type theories. They are for example all proved for the calculus of algebraic constructions [4], an extension of the calculus of constructions with inductive types and rewriting, where $\mathsf{POS}$ is the strict positivity condition as defined in [14], and $\mathsf{ACC}$ is the General Schema.

From now on, we use the notation $t\downarrow$ for the unique normal form of $t$.

## 4 Completeness of definitions

The definitions given in this section correspond to the ones given in [18]. Here, for the sake of clarity, we unfold and hence eliminate several auxiliary definitions.

**Definition 2 (Canonical form and canonical substitution).** *Given a judgment $E \vdash e : t$ we say that the term $e$ is in* canonical *form if and only if:*

- *if $t\downarrow$ is an inductive type then $e = c(e_1, \ldots, e_n)$ for some constructor $c$ and terms $e_1, \ldots, e_n$ in canonical form*
- *otherwise $e$ is arbitrary*

*Let $\Delta$ be a variable environment and $E$ a correct environment. We call $\delta : \Delta \to E$* canonical *if for every variable $x \in \Delta$, the term $x\delta$ is canonical.*

**Definition 3 (Complete definition).** *Let $E$ be an environment and $\mathsf{Rew}(\Gamma, R)$ a rewrite definition such that $E \vdash \mathsf{Rew}(\Gamma, R) : \mathsf{correct}$. The definition is* complete*, which is denoted by $\mathsf{COMP}_E(\Gamma, R)$, if and only if for all function symbols $f : (x_1 : t_1) \ldots (x_n : t_n)\, t \in \Gamma$, all environments $E'$ and all canonical substitutions $\delta : (x_1 : t_1; \ldots; x_n : t_n) \to (E; \mathsf{Rew}(\Gamma, R); E')$, such that all terms $x_i\delta$ are in normal form, the term $f(x_1\delta, \ldots x_n\delta)$ is head-reducible by $R$.*

Below we recall Definition 4.5 and Lemma 4.6 from [18].

**Definition 4 (Closed environment).** *An environment $E$ is* closed *if and only if it contains only inductive definitions and complete definitions by rewriting, i.e. for each partition of $E$ into $E_1; \mathsf{Rew}(\Gamma, R); E_2$ the condition $\mathsf{COMP}_{E_1}(\Gamma, R)$ is satisfied.*

**Lemma 1 (Canonicity).** *Let $E$ be a closed environment. If $E \vdash e : t$ and $e$ is in normal form then $e$ is canonical.*

**Corollary 1.** *Let $E$ be a closed environment such that $E = E_1; \mathsf{Rew}(\Gamma, R); E_2$. Let $f \in \Gamma$ and let $t_1, \ldots, t_n$ be terms such that $E \vdash f(t_1, \ldots, t_n) : t$ for some $t$. Then $f(t_1, \ldots, t_n)$ is reducible.*

In [18] we also give a sound and terminating, but necessarily incomplete, algorithm that checks whether a rewrite definition is complete. In many cases this algorithm demonstrates that only a basic subset of the rules is sufficient to show completeness.

In the next sections we investigate the question whether the remaining rules are inductive consequences of the basic subset.

## 5 Towards inductive consequences

In equational logic one says that $s = t$ is an inductive consequence of a theory $E$ if for all closed substitutions $\sigma$ the judgement $E \vdash s\sigma = t\sigma$ holds (see for example [10]).

If equational theories are generated by rewriting systems then the problem of inductive consequences may be reformulated as follows: assuming $R' \supseteq R$, are the rules from $R' - R$ inductive consequences of the rules from $R$.

In our setting $R'$ is the set of rules given by the user, $R$ is a confluent and complete subsystem of $R'$ and the question is roughly whether for all rules $l \to r \in R' - R$ and all closed substitutions $\sigma$, we have $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma \approx r\sigma$.

In order to show it one must find a sequence of reductions in $E; \mathsf{Rew}(\Gamma, R)$ between $l\sigma$ and $r\sigma$. The proof is done by induction on the sum of the reduction ordering associated with $E; \mathsf{Rew}(\Gamma, R')$ plus the suitable subterm relation. The crucial reduction is the first one from $l\sigma$. It always exists because $\sigma$ is closed, $l$ starts with a function symbol and $E; \mathsf{Rew}(\Gamma, R)$ is closed. The reduction takes place either entirely in $\sigma$ or overlaps with $l$. In the first case one easily gets a new substitution $\sigma'$ such that $l\sigma$ rewrites to $l\sigma'$ and $r\sigma$ to $r\sigma'$, and one can use the induction hypothesis.

In the second case $l$ and the left-hand side of some rule from $E; \mathsf{Rew}(\Gamma, R)$ overlap and one gets an instance of a critical pair of $R'$. Since $R'$ is confluent and terminating, its critical pairs are joinable. Following the critical pair diagram one replaces each $R'$ step (which is smaller than $l\sigma$), with a sequence of $R$ steps obtained from the induction hypotheses. In the end one gets the complete $R$ sequence from $l\sigma$ to $r\sigma$.

The usual critical pairs are defined for untyped terms using syntactic unification. The latter has several good properties: it is decidable and it does not introduce new variables.

In order to transfer a critical pair diagram from untyped terms and untyped rewriting to typable terms and rewriting in a PTS, we must slightly restrict the form of rules, which must be left-algebraic (Definition 5), and critical pairs, which must be type compatible (Definition 8). In particular we must be able to compute the types of variables that appear in the left-hand sides of the rules and to check whether these types are compatible (Definition 7) with what is written in the local environments of the rules.

**Definition 5 (Algebraic terms, left-algebraic rules).** *A term is* algebraic *if every free variable that appears in it is an argument of a constant symbol. A rule $G \vdash l \longrightarrow r$ is* left-algebraic *if $l$ is algebraic.*

**Definition 6 (Computed type).** *Let $t$ be an algebraic term, $p$ a position and $c : \forall(\overrightarrow{z : Z}).t_c$ a constant such that $t|_p = c(\vec{a})$ for some $\vec{a}$. For a given $j$ the* computed type *for a term $t|_{p \cdot j}$, denoted by $CT(t|_{p \cdot j})$, is $Z_j[a_1/z_1, \ldots a_{j-1}/z_{j-1}]$.*

**Definition 7 (Type compatibility).** *An environment $G$ is* type compatible *with an algebraic term $t$ if for every $x : T \in G$ there is a position $q$ such that $t|_q = x$ and $T = CT(t|_q)$.*

An environment $G$ is type compatible with a term $t$ if computed types for free variables of $t$ agree with $G$. It can be understood as a weaker version of typability of $t$ in $G$.

A well-typed instance of an algebraic, possibly untypable term equipped with a type compatible environment defines a well-typed substitution.

**Lemma 2 (Well-typed substitution from well-typed term).** *Let $t$ be an algebraic term, $G$ an environment type compatible with $t$ and let $\rho$ be a substitution such that $G' \vdash t\rho : U$ for some $U$. Then $\rho : G \to G'$ is well-typed.*

For our needs we equip each critical pair with an environment of variables that appear in the pair and we impose the type compatibility assumption on this environment.

**Definition 8 (Critical pairs for $R'$ wrt $R$, type compatibility).** *Critical pairs for $R'$ wrt $R$ are critical pairs for every $G \vdash l \longrightarrow r \in R'$ and every $D \vdash g \longrightarrow d \in E; \mathsf{Rew}(\Gamma, R)$ computed using syntactic unification. Critical pairs of $G \vdash l \longrightarrow r$ and $D \vdash g \longrightarrow d$ are tuples $(r\theta, l\theta[d\theta]_p, \Delta)$ for all positions $p$ such that there exists the most general unifier $\theta$ unifying $l|_p$ and $g$ and where $\Delta$ is the subset of $G; D$, such that $dom(\Delta) = FV(l\theta)$*

*A critical pair is* type compatible *if $\Delta$ is type compatible with $l\theta$.*

**Lemma 3 (Unification of left-algebraic rules).** *Let $G \vdash l \longrightarrow r$ and $D \vdash g \longrightarrow d$ be left-algebraic rules, $p$ a position in $l$ and $\theta$ the most general syntactic unifier of $l|_p$ and $g$. Then $l\theta$ is algebraic.*

In the next two sections we prove inductive consequence theorems for two kinds of rewriting systems. Section 6 adresses the case of rewriting on non-functional inductive types and Theorem 1 is an extension of results known for first order rewriting. This covers the majority of practical cases, in particular all examples from this paper apart from the definition by rewriting of `id` on `ord`. Section 7 concerns rewriting on functional inductive types.

## 6 Inductive consequences

For the rest of this section let us assume that $E$ is a closed environment such that $\mathsf{ACC}_E(\Gamma, R)$ and $\mathsf{COMP}_E(\Gamma, R)$ and that $R' \supseteq R$ extends $R$ with some additional rules in such a way that $\mathsf{ACC}_E(\Gamma, R')$ holds.

At the end of this section Theorem 1 states that in the closed environment $E; \mathsf{Rew}(\Gamma, R)$ the rules from $R' - R$ are inductive consequences of the rules from $E; \mathsf{Rew}(\Gamma, R)$ under the assumption that all applications of rules from $R' - R$ in the critical pair diagrams occur on free positions.

**Definition 9 (Free position).** *Let $t$ be a term. A position $q$ is free in $t$ if $t|_q$ is not in the scope of any bound variable from $t$.*

**Definition 10 (Free rewriting).** *A term $s$ free rewrites to $t$ in the rewriting system $R' - R$, denoted by $E; \mathsf{Rew}(\Gamma, R) \vdash s \xrightarrow{\not\chi R'}_{R'-R} t$, if there exists a rule $G \vdash l \longrightarrow r \in R' - R$ a substitution $\gamma : G \to (E; \mathsf{Rew}(\Gamma, R))$ and a free position $q$ in $s$ such that $s|_q = l\gamma$ and $t = s[r\gamma]_q$.*

**Definition 11 ($\not\chi R'$-rewriting).** *A term $s$ $\not\chi R'$-rewrites to $t$, which is denoted by $E; \mathsf{Rew}(\Gamma, R) \vdash s \overset{\not\chi R'}{\rightsquigarrow} t$, if either $E; \mathsf{Rew}(\Gamma, R) \vdash s \longrightarrow t$ or $E; \mathsf{Rew}(\Gamma, R) \vdash s \xrightarrow{\not\chi R'}_{R'-R} t$.*

In other words $\not\chi R'$-rewriting consists in rewriting in the environment $E; \mathsf{Rew}(\Gamma, R)$ with rules from $E; \mathsf{Rew}(\Gamma, R)$ or beta in any context and using rules from $R' - R$ only on free positions.

**Definition 12 ($\not\chi R'$-joinability of critical pairs, critical pairs diagram).** *We say that $(u, v, \Delta)$, a critical pair for $R'$ wrt $R$, is* $\not\chi R'$-joinable *if it is type compatible and there is a term $e$ such that $\Delta \vdash u \overset{\not\chi R'}{\rightsquigarrow} {}^*e$ and $\Delta \vdash v \overset{\not\chi R'}{\rightsquigarrow} {}^*e$ and if for every $R' - R$ rewrite step $\Delta \vdash s[l'\gamma]_p \longrightarrow s[r'\gamma]_p$ in these sequences the substitution $\gamma : G' \to \Delta$ and the term $l'\gamma$ are well-typed, where $G'$ is the local environment of $G' \vdash l' \longrightarrow r' \in R' - R$.*

*The terms $u$, $v$, $e$ (with environment $\Delta$) and the aforementioned reductions between them are called a* critical pair diagram.

Note that we do not assume that all terms in the critical pair diagrams are typable.

**Definition 13 ($\not\chi$-subterm).** *Let $t$ be a term. The term $s = t|_q$ is a* $\not\chi$-subterm *of $t$, denoted by $t \trianglerighteq^{\not\chi} s$, if $q$ is free in $t$.*

It is well-known that the sum of $\triangleright^{\not\chi}$ and any relation that is well-founded and stable by context is also well-founded.

**Theorem 1.** *Suppose that critical pairs for $R'$ wrt $R$ are $\not\chi R'$-joinable. Then for every rule $G \vdash l \longrightarrow r \in R' - R$ and substitution $\sigma : G \to (E; \mathsf{Rew}(\Gamma, R))$, such that $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma : T$ for some $T$, one has $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma \approx r\sigma$.*

*Proof.* By induction on $(\longrightarrow \cup \rhd^{\not\chi})$ where $\longrightarrow$ is the reduction relation corresponding to the environment $E; \mathsf{Rew}(\Gamma, R')$[3]. The relation $\longrightarrow$ is well-founded by assumption $\mathsf{ACC}_E(\Gamma, R')$.

Since $E; \mathsf{Rew}(\Gamma, R)$ is closed and $l\sigma$ is typable, by Corollary 1 the term $l\sigma$ is reducible. There are two possibilities. If the reduction takes place in the substitution then there exists a variable $z$ such that $z : Z \in G$ and $E; \mathsf{Rew}(\Gamma, R) \vdash z\sigma \longrightarrow t'$ for some $t'$. Let us define $\sigma'$ to be $\sigma'(x) = \sigma(x)$ for $x \neq z$ and $\sigma'(z) = t'$. The substitution $\sigma'$ is well-typed since by subject reduction $E; \mathsf{Rew}(\Gamma, R) \vdash t' : Z\sigma'$. The term $l\sigma'$ is also well-typed by subject reduction, since obviously $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma \longrightarrow^+ l\sigma'$. By induction hypothesis applied to $l\sigma'$ we have $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma' \approx r\sigma'$. We have also $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma \approx l\sigma'$ and $E; \mathsf{Rew}(\Gamma, R) \vdash r\sigma \approx r\sigma'$ because they result from rewriting with $R$. By transitivity of $\approx$ we conclude that $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma \approx r\sigma$.

Otherwise, there is a rule $D \vdash g \longrightarrow d$ coming from $E; \mathsf{Rew}(\Gamma, R)$ that has a critical pair with $G \vdash l \longrightarrow r$ at position $p$ in $l$. It means that there exists $\theta$, the most general substitution unifying $l|_p$ and $g$, and $\rho$, such that $\sigma = \theta\rho$, and that the critical pair equals $(l\theta[d\theta]_p, r\theta, \Delta)$ where $dom(\Delta) = FV(l\theta)$. Then $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma \xrightarrow{\not\chi R'}_{R'-R} r\sigma$ and $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma \longrightarrow l\theta[d\theta]_p\rho$. Let us denote $l\theta[d\theta]_p$ by $\hat{l}$. Since critical pairs are joinable there exists a term $e$ such that $E; \mathsf{Rew}(\Gamma, R) \vdash r\sigma \overset{\not\chi R'}{\rightsquigarrow}{}^* e\rho$ and $E; \mathsf{Rew}(\Gamma, R) \vdash \hat{l}\rho \overset{\not\chi R'}{\rightsquigarrow}{}^* e\rho$.

The term $l\theta$ may be not well-typed but it is algebraic by Lemma 3. We know that the term $l\theta\rho$ is well-typed. By Lemma 2, this implies that $\rho$ is a well-typed substitution from $\Delta$ to $E; \mathsf{Rew}(\Gamma, R)$.

Every $R' - R$ step on the path from $r\sigma = r\theta\rho$ or $\hat{l}\rho$ to $e\rho$ is of the form $E; \mathsf{Rew}(\Gamma, R) \vdash s\rho \xrightarrow{\not\chi R'}_{R'-R} t\rho$ where $s = s[l'\gamma]_q$, $t = s[r'\gamma]_q$ for some free position $q$, a rule $G' \vdash l' \longrightarrow r' \in R' - R$, and a substitution $\gamma : G' \to \Delta$ such that $l'\gamma$ is well-typed in $\Delta$. Of course $s\rho = s\rho[l'\gamma\rho]_q$. Since $\rho : \Delta \to E; \mathsf{Rew}(\Gamma, R)$ is well-typed, the substitution $\gamma\rho$ and the term $l'\gamma\rho$ are also well-typed. Hence we may apply the induction hypothesis to $l'\gamma\rho$ (since it is smaller than $l\sigma$ in $(\longrightarrow \cup \rhd^{\not\chi})^+$) and get $E; \mathsf{Rew}(\Gamma, R) \vdash l'\gamma\rho \approx r'\gamma\rho$. Since $\approx$ is stable by context, $E; \mathsf{Rew}(\Gamma, R) \vdash s\rho \approx t\rho$ also holds. Obviously, all rewriting steps corresponding to $E; \mathsf{Rew}(\Gamma, R)$ can be replaced by conversion. Hence $E; \mathsf{Rew}(\Gamma, R) \vdash r\sigma \approx e\rho$ and $E; \mathsf{Rew}(\Gamma, R) \vdash \hat{l}\rho \approx e\rho$. Of course we have also $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma \approx \hat{l}\rho$ because $l\sigma$ rewrites to $\hat{l}\rho$ using a rule from $R$. Consequently by transitivity and symmetry of $\approx$ we get the desired conclusion $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma \approx r\sigma$.

## 7 Functional inductive consequences

Like in the previous section let us assume that $E$ is a closed environment such that $\mathsf{ACC}_E(\Gamma, R)$ and $\mathsf{COMP}_E(\Gamma, R)$ and that $R' \supseteq R$ extends $R$ with some additional rules in such a way that $\mathsf{ACC}_E(\Gamma, R')$ holds.

[3] Since $R$-normal canonical forms are not necessarily $R'$-normal, straightforward $R$-normalization of $l\sigma$ and $r\sigma$ does not always lead to equal terms.

Like before the goal of this section is to check the power of rules from $R' - R$ with respect to those already present in $R$. The main difference is that now we allow for systems whose critical pairs need at least one step of rewriting under a binder. This is often the case when considering rewrite rules involving functional inductive types.

Let us consider the `ord` example from Section 2. The rules for `id` are strongly normalizing and confluent. However, to join the critical pair between the 4th and the 3rd rule of `id` one needs rewriting under an abstraction

```
id (id (lim f))  ⟶  id (lim f)  ⟶  lim (fun n ⇒ id (f n))
id (id (lim f))  ⟶  id (lim (fun n ⇒ id (f n)))
  ⟶+  lim (fun n ⇒ id (id (f n)))  ⟶  lim (fun n ⇒ id (f n))
```

It is easy to check that the first three rules for `id` form a complete subsystem. However, it is not true that for every closed substitution $\sigma$, term $(\texttt{id (id x)})\sigma$ is convertible with $(\texttt{id x})\sigma$ using only the first three rules. Let `n2o` be defined in the following way:

```
Rewriting n2o : nat -> ord
Rules     n2o 0 ⟶ o          n2o (S x) ⟶ S (n2o x)
```

Consider $\sigma = [\,\texttt{lim (fun n} \Rightarrow \texttt{n2o n)} \,/\, \texttt{x}\,]$. Then:

```
lσ = id (id (lim (fun n ⇒ n2o n)))  ⟶+  lim (fun n ⇒ id (id (n2o n)))
rσ = id (lim (fun n ⇒ n2o n))  ⟶+  lim (fun n ⇒ id (n2o n))
```

and these are different normal forms. The reason is that even though the substitution is closed, the corresponding instances of the left- and right-hand side of the 4th rule reduce to terms where the function symbol `id` is applied to open terms. In order to pass from open to closed terms again one can consider a new equivalence, containing the usual conversion and identifying functions that are equal for all closed arguments. Let $\sim_\omega$ be the smallest congruence containing $\approx$ and closed by the following $(\omega)$ rule:

$$\frac{\begin{array}{c} E \text{ closed} \\ E \vdash f : \forall x{:}A.B \qquad E \vdash g : \forall x{:}A.B \\ \forall d \quad (E \vdash d : A \implies E \Vdash f d \sim_\omega g d) \end{array}}{E \Vdash f \sim_\omega g} \quad (\omega)$$

The rule states roughly that functions $f$ and $g$ are equal if all their closed instances are. It is similar in spirit to functional extensionality in a sense that the $\sim_\omega$ equality is roughly the same to inductive consequences as the propositional equality with functional extensionality to the equality without it.

In fact, we do not need the $(\omega)$ rule in its full generality. We will use it only for functions that are arguments of constructors on functional recursive positions, like in `lim (fun n ⇒ n2o n)`. One may also argue that in these places functions are only a means to express infinite branching of a constructor and hence that these functions should be treated extensionally.

Before we can state and prove the theorem corresponding to Theorem 1 from the previous section we need to know more about $\sim_\omega$.

**Lemma 4.** *Let $E$ be a closed environment and let $s[a]_p$, $s[b]_p$ and $T$ be terms such that $E \vdash s[a]_p : T$, $E \vdash s[b]_p : T$. Moreover, suppose that all declarations in $\Gamma(s,p)$ come from abstractions.*

*If $E \Vdash a\delta \sim_\omega b\delta$ holds for all $\delta : \Gamma(s,p) \to E$ then $E \Vdash s[a]_p \sim_\omega s[b]_p$.*

The above lemma states that in order to know $E \Vdash s[a]_p \sim_\omega s[b]_p$, which is in some sense an $\sim_\omega$ equality between open terms $a$ and $b$, it is sufficient to check that $E \Vdash a\delta \sim_\omega b\delta$ holds for all closed substitutions $\delta$. Hence, it shows how to pass from open terms to the closed ones, and in particular from an open instance of the left-hand side of a rule $l'\gamma$ to a closed one $l'\gamma\delta$. Closed instances are necessary since we want to follow the proof of Theorem 1 and use the inductive hypothesis. On the other hand this forces us to use an induction ordering $>$ strong enough to show that $l\sigma > l'\gamma\delta$ for an arbitrary $\delta$, instead of the usual $l\sigma > l'\gamma$.

An example of a well-founded ordering allowing for applications to arbitrary arguments is the constructor subterm ordering on functional types. Taking `ord` for example, the term `lim f` is greater than `f t` for any `t` of type `nat`. Of course this can be done only for recursive arguments of a constructor and because we restrict ourselves to well-typed terms.

Unfortunately constructor subterm is not enough for our needs: we need to use it together with the rewrite relation generated by the environment and with beta reduction. And it is not always the case that the sum of the constructor subterm with a well-founded relation is always well-founded (see an example in the appendix). Fortunately, the sum of the constructor subterm with the rewrite relation generated by rules accepted by HORPO is always well-founded [17]. Our hypothesis is that this can be extended to any well-founded relation containing rules for recursors.

Note that $s|_p\delta$ is smaller than $s$ in the constructor subterm ordering only if on the path from the root to $q$ in $s$ there are only constructors and abstractions and that they appear only on recursive positions. For that reason we restrict critical pair diagrams to be joinable that way.

Let us now introduce formally the notions of a recursive position, constructor rewriting and constructor subterm ordering.

**Definition 14 (Recursive position).** *The $i$-th position of a constructor $c : \forall(\overrightarrow{p : P})(\overrightarrow{z : d}), I(\vec{p})\vec{w}$ of an inductive type $I$ is* recursive *if $d_i$ is of the form $\forall(x_1 : T_1) \dots (x_n : T_n).I(\vec{p})\vec{v}$. It is called a* nonfunctional recursive position *if $n = 0$; otherwise is is called a* functional recursive position*.*

**Definition 15 (Constructor rewriting).** *A term $s$* constructor rewrites *to $t$ in a rewriting system $R' - R$, which is denoted by $E; \mathsf{Rew}(\Gamma, R) \vdash s \xrightarrow{c}_{R'-R} t$, if there exists a rule $G \vdash l \longrightarrow r \in R' - R$, a position $q = q_1 \cdot \ldots \cdot q_m$, and a substitution $\gamma : G \to (E; \mathsf{Rew}(\Gamma, R))$ such that $s = s[l\gamma]_q$, $t = s[r\gamma]_q$, and for every $k = 0..m-1$*

- *either $s|_{q_1 \dots q_k} = c(\vec{a}, \vec{b})$ for some constructor $c$ and $q_{k+1}$ is a recursive position of $c$,*

– *or* $s|_{q_1 \ldots q_k} = \lambda x : T.s|_{q_1 \ldots q_{k+1}}$, $0 < k < m$, $s|_{q_1 \ldots q_{k-1}} = c(\vec{a}, \vec{b})$ *for some constructor* $c$ *and* $q_k$ *is a functional recursive position of* $c$.

**Definition 16 ($cR'$-rewriting).** *A term* $s$ $cR'$*-rewrites to* $t$*, which is denoted by* $E; \mathsf{Rew}(\Gamma, R) \vdash s \overset{cR'}{\rightsquigarrow} t$*, if either* $E; \mathsf{Rew}(\Gamma, R) \vdash s \longrightarrow t$ *or* $E; \mathsf{Rew}(\Gamma, R) \vdash s \xrightarrow[R'-R]{c} t$.

In other words $cR'$-rewriting consists in rewriting in the environment $E; \mathsf{Rew}(\Gamma, R)$ with rules from $E; \mathsf{Rew}(\Gamma, R)$ or beta in any context and using $R' - R$ rules only in contexts built from constructors and abstractions as described in Definition 15.

**Definition 17 ($cR'$-joinability of critical pairs, critical pairs diagram).** *We say that* $(u, v, \Delta)$*, a critical pair for* $R'$ *wrt* $R$*, is* $cR'$-joinable *if it is type compatible and there is a term* $e$ *such that* $\Delta \vdash u \overset{cR'}{\rightsquigarrow} {}^*e$ *and* $\Delta \vdash v \overset{cR'}{\rightsquigarrow} {}^*e$ *and if for every* $R' - R$ *rewrite step in these sequences* $\Delta \vdash s[l'\gamma]_p \longrightarrow s[r'\gamma]_p$

– $\gamma$ *is a well-typed substitution from* $G'$ *to* $\Delta, \Gamma(s, p)$ *and*
– $l'\gamma$ *is a well-typed term in* $\Delta, \Gamma(s, p)$

*where G' is the local environment from* $G' \vdash l' \longrightarrow r' \in R' - R$.

*The terms u, v, e (with environment* $\Delta$*) and the aforementioned reductions between them are called a* critical pair diagram.

Since the constructor subterm ordering is not well-founded on nontypable terms, the definition below depends on environment.

**Definition 18 (Constructor subterm).** *Let* $c : \forall \overrightarrow{(p : P)} \overrightarrow{(z : d)}.I(\vec{p})\vec{w}$ *be a constructor of an inductive type* $I$ *and let* $i$ *be a recursive position of* $c$*. Let* $E$ *be an environment and* $\vec{a}$*,* $\vec{b}$ *be terms such that* $c(\vec{a}, \vec{b})$ *is typable in* $E$.

*Then for every* $\vec{t}$ *such that* $E \vdash b_i\vec{t} : T$ *for some* $T$ *the term* $b_i\vec{t}$ *is a* constructor subterm of $c(\vec{a}, \vec{b})$ in $E$*, denoted by* $E \vdash c(\vec{a}, \vec{b}) \rhd^c b_i\vec{t}$.

**Theorem 2.** *Suppose that all critical pairs for* $R'$ *wrt* $R$ *are* $cR'$*-joinable and that the relation* $(\longrightarrow \cup \rhd^c)$ *is well-founded in* $E; \mathsf{Rew}(\Gamma, R')$*. Then for every rule* $G \vdash l \longrightarrow r \in R' - R$ *and substitution* $\sigma : G \to (E; \mathsf{Rew}(\Gamma, R))$*, such that* $E \vdash l\sigma : T$ *for some* $T$*, one has* $E; \mathsf{Rew}(\Gamma, R) \Vdash l\sigma \sim_\omega r\sigma$.

*Proof (sketch).* By induction on $(\longrightarrow \cup \rhd^c)$ in the environment $E; \mathsf{Rew}(\Gamma, R')$.

The proof follows exactly the schema of the proof of Theorem 1. The difference is that in a critical pair diagram there may be an $R' - R$ rewriting step under a binder, which means that we have an open instance of some left hand-side from $R' - R$ and induction hypothesis cannot be directly applied. We use Lemma 4 to get a closed instance, and then we show that the resulting term is always smaller than $l\sigma$ in the ordering used for induction.

Once we show that induction hypothesis can be applied, the rest of the proof goes as in Theorem 1.

## 8 Conclusions

In this paper we study the calculus of constructions with rewriting and we address the issue of the logical power of such an extension.

We continue the research about rewriting in the calculus of constructions presented in [18] where an algorithm that checks completeness of definitions by rewriting was given. In many cases this algorithm demonstrates that only a basic subset of the rules is sufficient for completeness. In this paper we have shown that the remaining rules are inductive consequences of the basic subset.

The proof is done for two kinds of rewriting systems: when there is no rewriting under a binder in the critical pairs diagrams (Section 6) and for some class of systems where such situation happens (Section 7). In the latter case the conclusion of the inductive consequences lemma must be modified by allowing for a kind of functional extensionality in the corresponding equivalence.

The additional assumptions on rewriting that we impose do not seem restrictive. First of all we require the rewrite rules to be left-algebraic and the critical pairs to be type-compatible, which can be checked easily. Second, there are assumptions on the form of critical pairs diagrams. These are different in Section 6 and Section 7, where the restrictions are mainly due to difficulties in finding a suitable ordering for induction, but they always account for a simple inspection of a diagram.

It is interesting to relate our paper to the PhD work of Oury [13]. He studies CCE, the extensional calculus of constructions, and shows that CCE is conservative with respect to the calculus of inductive constructions extended with three axioms (functional extensionality, Streicher's axiom $K$ and the third technical one). One of the interests of CCE is that it can be seen as a model of the calculus of constructions with rewriting. Assuming that one uses only rewrite rules that are provable as equalities, extending conversion with $l \longrightarrow r$ can be modeled in CCE by adding $l = r$ as an axiom and then using it by extensionality. Consequently, calculus of constructions with rewriting rules which are provable as equalities is conservative with respect to the calculus of inductive constructions extended with the three axioms mentioned above.

Unfortunately we do not prove in this paper that additional rules are provable equalities. We only approach this goal by studying the notion of inductive consequences. In the algebraic setting the three notions, being an additional rule, being an inductive consequence and being an equality proved by induction coincide. In our setting the gap between these notions is not trivial: there are inductive consequences which are not provable as equalities without additional axioms, see `subst` in Section 2 where axiom $K$ is needed, and there are additional rules that are not inductive consequences in the strict sense, see `id` in Section 2 and 7.

It would be interesting to check what happens if we try to prove equalities in a system where axiom $K$ and functional extensionality are present from the start. Especially, since they appear independently in other works on equality in type theory (see e.g. [2]) and that one of them, axiom $K$, can be easily defined by rewriting, see for example [18].

## References


1. A. Abel, T. Coquand, and M. Pagano. A modular type-checking algorithm for type theory with singleton types and proof irrelevance. In P.-L. Curien, editor, *TLCA 2009*, volume 5608 of *LNCS*, pages 5–19. Springer, 2009.
2. T. Altenkirch, C. McBride, and W. Swierstra. Observational equality, now! In *PLPV 2007*, pages 57–68. ACM, 2007.
3. F. Barbanera, M. Fernández, and H. Geuvers. Modularity of strong normalization in the algebraic-$\lambda$-cube. *Journal of Functional Programming*, 7(6):613–660, 1997.
4. F. Blanqui. Definitions by rewriting in the Calculus of Constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005.
5. F. Blanqui, J.-P. Jouannaud, and M. Okada. The Calculus of Algebraic Constructions. In P. Narendran and M. Rusinowitch, editors, *RTA 1999*, volume 1631 of *LNCS*, pages 301–316. Springer, 1999.
6. F. Blanqui, J.-P. Jouannaud, and P.-Y. Strub. Building decision procedures in the calculus of inductive constructions. In J. Duparc and T. A. Henzinger, editors, *CSL 2007*, volume 4646 of *LNCS*, pages 328–342. Springer, 2007.
7. J. Chrząszcz. Modules in Coq are and will be correct. In S. Berardi, M. Coppo, and F. Damiani, editors, *TYPES 2003*, volume 3085 of *LNCS*, pages 130–146. Springer, 2004.
8. J. Chrząszcz. *Modules in Type Theory with Generative Definitions*. PhD thesis, Warsaw Univerity and University of Paris-Sud, 2004.
9. The Coq proof assistant. `http://coq.inria.fr/`.
10. H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1. Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.
11. M. Hofmann and T. Streicher. The groupoid model refutes uniqueness of identity proofs. In *LICS 1994*, pages 208–212. IEEE Computer Society, 1994.
12. P. Letouzey. A new extraction for Coq. In H. Geuvers and F. Wiedijk, editors, *TYPES 2002*, volume 2646 of *LNCS*, pages 200–219. Springer, 2003.
13. N. Oury. Extensionality in the calculus of constructions. In J. Hurd and T. F. Melham, editors, *TPHOLs 2005*, volume 3603 of *LNCS*, pages 278–293. Springer, 2005.
14. C. Paulin-Mohring. Inductive definitions in the system Coq: Rules and properties. In M. Bezem and J. F. Groote, editors, *TLCA 1993*, volume 664 of *LNCS*, pages 328–345. Springer, 1993.
15. Terese, editor. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.
16. D. Walukiewicz-Chrząszcz. Termination of rewriting in the calculus of constructions. *Journal of Functional Programming*, 13(2):339–414, 2003.
17. D. Walukiewicz-Chrząszcz. *Termination of Rewriting in the Calculus of Constructions*. PhD thesis, Warsaw University and University Paris-Sud, 2003.
18. D. Walukiewicz-Chrząszcz and J. Chrząszcz. Consistency and completeness of rewriting in the calculus of constructions. *Logical Methods of Computer Science*, 4(3), 2008.
19. D. Walukiewicz-Chrząszcz and J. Chrząszcz. Inductive consequences in the calculus of constructions. In *ITP 2010*, LNCS. Springer, 2010. To appear.
20. B. Werner. On the strength of proof-irrelevant type theories. *Logical Methods in Computer Science*, 4(3), 2008.

# A More examples

## A.1 Append on dependent vectors

```
Inductive vect : nat → Set  :=
    | vnil : vector 0
    | vcons (a:A) (n:nat), vector n → vector (S n).
Symbol app : forall n:nat, list n → forall m:nat, vect m → vect (n+m)
Rules
  app 0 vnil m v2 ⟶  v2
  app (s n) (vcons a n v1) m v2 ⟶ vcons a (n+m) (app n v1 m v2)
  app _ (app k v1 n v2) m v3 ⟶ app k v1 (n+m) (app n v2 m v3)
  app n v1 0 vnil ⟶ v1
```

The above rewriting system is terminating, confluent, the two first rules completely define `app`. The third and fourth rule are inductive consequences of the first two and, transformed into equations, could be proved by induction from the first two, assuming that $+$, as defined in this paper, is already in the environment. The type correctness of the third rule requires plus to have the associativity rule and the fourth rule require `n+0` to be convertible with `n`.

Note also, that the left hand side of the third rule is not typable (if `_` is treated as a new variable), because `_` should be `n+k`. However the rule is correct because for every typable instance of the left-hand side, the corresponding right hand side has the same type (assuming, again, that associativity of plus is in conversion). This is discussed in Section 2 of [18].

## A.2 Uniqueness of Identity Proofs

```
Symbol UIP : forall (A:Set)(a b:A)(p q: eq A a b), (eq (eq A a b) p q)
Rules
  UIP A a a (refl A a) (refl A a) ⟶ refl (eq A a a) (refl A a).
  UIP A a a p p ⟶ refl (eq A a a) p.
```

The first rule completely defines `UIP` and the second one is its inductive consequence. The type for `UIP` is not inhabited in CCI, because it is equivalent to axiom $K$. Once it is assumed together with the first rule (as equality), the second rule can easily be proved.

## A.3 Polymorphic iterated function

```
Symbol iter : forall A:Set, nat → (A → A) → A → A
Rules
  iter 0 f x ⟶ x
  iter (S n) f x ⟶ f (iter n f x)
  iter n (fun y ⇒ y) x ⟶ x
  iter (add n m) f x ⟶ iter n f (iter m f x)
```

The first two rules completely define `iter`. The third and the fourth one are their inductive consequences, also provable by induction. Note that the fourth rule is confluent with the others only if the definition of addition (`add`) is restricted to the two rules defining it by induction on the first argument.

## B Details from Section 5

Proof of **Lemma 2**.

*Proof.* We have to show that for every $x : T \in G$ one has $G' \vdash x\rho : T\rho$.

By assumption that $G$ is type compatible with $t$, there is a position $p$ such that $t|_p = x$ and $CT(t|_p) = T$.

Since $t\rho$ is typable one has $G' \vdash t\rho|_p : CT(t\rho|_p)$. Note that $t\rho|_p = x\rho$ and $CT(t\rho|_p) = CT(t|_p)\rho$. Hence $G' \vdash x\rho : T\rho$

Proof of **Lemma 3**.

*Proof.* By analysing the unification algorithm one notices that for every $x$ the term $x\theta$ is either algebraic or a variable. Hence, the term $l\theta$ is algebraic.

## C Details from Section 7

### C.1 Discussion on the constructor order

The example below shows that the sum of the constructor subterm with a well-founded relation is not always well-founded.

*Example 1.* Let us consider the following definition $\mathsf{Rew}(f; R)$

```
Symbol f : nat → ord
Rules
  f 0 ⟶ lim (fun x ⇒ f x)
  f (S x) ⟶ lim (fun x ⇒ f x)
```

This rewrite system is obviously terminating, confluent and complete. But if we combine it with constructor subterm we get an infinite reduction sequence starting with : $f\,0 \longrightarrow_R lim(\lambda x\!:\!nat.f\,x) \rhd^c \lambda x\!:\!nat.f\,x@0 \longrightarrow_\beta f\,0$.

On the other hand the rewrite system given above is probably not very interesting if one finds that recursors for inductive types should be "automatically" considered as rewrite rules added to the environment just after the definition of the corresponding inductive type. The reason is that the system containing the rules for $f$ and the rules of recursor for $ord$ is not terminating.

*Example 2.* Let us take $f_3 = \lambda a\!:\!nat \to nat.\lambda b\!:\!nat \to ord.(b@0)$ and arbitrary $P$, $f_2$, $f_1$. Then we get an infinite rewriting sequence which starts as follows: $ord\_rec(P, f_1, f_2, f_3, f\,0) \longrightarrow_R ord\_rec(P, f_1, f_2, f_3, lim(\lambda x : nat.f\,x) \longrightarrow_{ord\_rec} @(f_3, \lambda x : nat.f\,x, \lambda y : nat.ord\_rec(P, f_1, f_2, f_3, \lambda x : nat.f\,x@y) \longrightarrow_\beta @(f_3, \lambda x : nat.f\,x, \lambda y\!:\!nat.ord\_rec(P, f_1, f_2, f_3, f\,y) \longrightarrow_\beta^* ord\_rec(P, f_1, f_2, f_3, f\,0)$

Our hypothesis is that the sum of the constructor subterm with a well-founded relation containing rules for recursors is always well-founded.

In particular, this is true if the well-founded relation in question is the rewrite relation generated by rules accepted by HORPO. In [17] there is a proof that the sum of HORPO, beta reduction and $\rhd^c$ is well-founded on well-typed terms. The same should be true for the General Schema [4]. Moreover it should be true for both styles for defining interpretation of inductive types: introduction and elimination ones.

### C.2 Omitted proofs

Proof of **Lemma 4**.

*Proof.* By induction on the length of $\Gamma(s,p)$. If it is empty then $a$ and $b$ are closed terms and $E \Vdash a \sim_\omega b$ follows from assumption. By stability by context one gets $E \Vdash s[a]_p \sim_\omega s[b]_p$

If $\Gamma(s,p)$ is nonempty let us focus on the innermost $\lambda$ on the path from the root to $p$. Then there exists positions $q'$, $q''$ such that $s[a]_p = s'[\lambda x : A.s''[a]_{q''}]_{q'}$, $s[b]_p = s'[\lambda x : A.s''[b]_{q''}]_{q'}$, $p = q' \cdot 2 \cdot q''$ and $\Gamma(s,p) = \Gamma(s,q'), x : A$.

Let us show that $E \Vdash s'[\lambda x : A.s''[a]_{q''}]_{q'} \sim_\omega s'[\lambda x : A.s''[b]_{q''}]_{q'}$ by proving that for all $\delta' : \Gamma(s,q') \to E$ the judgement $E \Vdash (\lambda x : A.s''[a]_{q''})\delta' \sim_\omega (\lambda x : A.s''[b]_{q''})\delta'$ holds and by applying the induction hypothesis.

We have $E \vdash (\lambda x : A.s''[a]_{q''})\delta' : \forall x : A\delta'.B\delta'$ and $E \vdash (\lambda x : A.s''[a]_{q''})\delta' : \forall x : A\delta'.B\delta'$ for some $B$ since $s[a]_p$, $s[b]_p$ and $\delta'$ are typable in $E$. In order to show $E \Vdash \lambda x : A\delta'.(s''[a]_{q''})\delta' \sim_\omega \lambda x : A\delta'.(s''[b]_{q''})\delta'$ we may use $(\omega)$ rule but first we need to know that for all $d$ such that $E \vdash d : A\delta'$ the judgement $E \Vdash (\lambda x : A\delta'.(s''[a]_{q''})\delta')d \sim_\omega (\lambda x : A\delta'.(s''[b]_{q''})\delta')d$ holds.

Let $\delta_d$ be $\delta'[d/x]$. Then $\delta_d : \Gamma(s,p) \to E$ and by assumption of the lemma $E \Vdash a\delta_d \sim_\omega b\delta_d$. The context $s''\delta_d[\_]_{q''}$ is closed and by stability by context one has also $E \Vdash s''\delta_d[a\delta_d]_{q''} \sim_\omega s''\delta_d[b\delta_d]_{q''}$. But $s''\delta_d[a\delta_d]_{q''} =_\beta (\lambda x : A\delta'.(s''[a]_{q''})\delta')d$ and similarly for $b$. Hence, we get the desired conclusion.

Full proof of **Theorem 2**.

*Proof.* By induction on $(\longrightarrow \cup \rhd^c)$ in the environment $E; \mathsf{Rew}(\Gamma, R')$.

Since $E; \mathsf{Rew}(\Gamma, R)$ is closed and $l\sigma$ is typable, by Corollary 1 the term $l\sigma$ is reducible. There are two possibilities. If the reduction takes place in substitution then there exists a variable $z$ such that $z : Z \in G$ and $E; \mathsf{Rew}(\Gamma, R) \vdash z\sigma \longrightarrow t'$. Let us define $\sigma'$ to be $\sigma'(x) = \sigma(x)$ for $x \neq z$ and $\sigma'(z) = t'$. Of course, $\sigma'$ and $l\sigma'$ are well-typed (this is done in the proof of Theorem 1). By induction hypothesis $E; \mathsf{Rew}(\Gamma, R) \Vdash l\sigma' \sim_\omega r\sigma'$. Of course $E; \mathsf{Rew}(\Gamma, R) \Vdash l\sigma \sim_\omega l\sigma'$ and $E; \mathsf{Rew}(\Gamma, R) \Vdash r\sigma \sim_\omega r\sigma'$ because they result from rewriting with $R$. By transitivity of $\approx_\omega$ we conclude that $E; \mathsf{Rew}(\Gamma, R) \Vdash l\sigma \sim_\omega r\sigma$.

Otherwise, there is a rule $D \vdash g \longrightarrow d$ coming from $E; \mathsf{Rew}(\Gamma, R)$ that has a critical pair with $l \longrightarrow r$ at position $p$ in $l$. It means that there exists the most general substitution $\theta$ unifying $l|_p$ and $g$, and $\rho$ such that $\sigma = \theta\rho$, and that the critical pair equals $(l\theta[d\theta]_p, r\theta, \Delta)$ where $dom(\Delta) = FV(l\theta)$. Then $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma \xrightarrow{c}_{R'-R} r\sigma$ and $E; \mathsf{Rew}(\Gamma, R) \vdash l\sigma \longrightarrow_R l\theta[d\theta]_p\rho$. Let us denote $l\theta[d\theta]_p$ by $\hat{l}$. Since critical pairs are $cR'$-joinable there exists a term $e$ such that $E; \mathsf{Rew}(\Gamma, R) \vdash r\sigma \overset{cR'}{\rightsquigarrow} {}^*e\rho$ and $E; \mathsf{Rew}(\Gamma, R) \vdash \hat{l}\rho \overset{cR'}{\rightsquigarrow} {}^*e\rho$.

The term $l\theta$ may be not well-typed but it is algebraic (by Lemma 3). We know that the term $l\theta\rho$ is well-typed. By Lemma 2, this implies that $\rho$ is a well-typed substitution from $\Delta$ to $E; \mathsf{Rew}(\Gamma, R)$.

Every $R'-R$ step on the path from $r\sigma$ or $\hat{l}\rho$ to $e\rho$ is of the form $E; \mathsf{Rew}(\Gamma, R) \vdash s\rho \xrightarrow{c}_{R'-R} t\rho$ where $s = s[l'\gamma]_q$, $t = s[r'\gamma]_q$ for some rule $G' \vdash l' \longrightarrow r' \in R'-R$, a position $q$ and a substitution $\gamma : G' \to (\Delta; \Gamma(s,q))$. By the definition of

$\xrightarrow{c}_{R'-R}$ on the path from the root to $q$ in $s$ there are only constructors and lambdas and $l'\gamma$ is a well-typed term in $\Delta;\Gamma(s,q)$.

Note that $s\rho = s\rho[l'\gamma\rho]_q$ and $t\rho = s\rho[r'\gamma\rho]_q$. According to Lemma 4 to show that $E;\mathsf{Rew}(\Gamma,R) \Vdash s\rho \sim_\omega t\rho$ it is sufficient to prove that for every substitution $\delta : \Gamma(s,q) \to (E;\mathsf{Rew}(\Gamma,R))$ one has $E;\mathsf{Rew}(\Gamma,R) \Vdash l'\gamma\rho\delta \sim_\omega r'\gamma\rho\delta$.

First we show that $l'\gamma\rho\delta$ is smaller than $l\sigma$ in the ordering used for induction. Note that $E;\mathsf{Rew}(\Gamma,R) \vdash l'\gamma\rho\delta : T'$ for some $T'$ since $l'\gamma$ is well-typed in the environment $\Delta;\Gamma(s,q)$ and $\rho : \Delta \to E;\mathsf{Rew}(\Gamma,R)$ and $\delta : \Gamma(s,q) \to E;\mathsf{Rew}(\Gamma,R)$.

Of course $E;\mathsf{Rew}(\Gamma,R') \vdash l\sigma \longrightarrow^* s\rho$. Let $q = q_1 \cdot \ldots q_m$ and let us denote $s|_{q_1 \ldots q_k}$ by $s^k$ (using this notation $s|_q = s^m$ and $s = s^0$). Let $\delta = [t_1/x_1, \ldots t_n/x_n]$ and let $\delta^k = [t_1/x_1, \ldots t_k/x_k]$. Moreover let $i_1 < \ldots < i_n$ be all indexes such that $s^{i_k} = \lambda x_k{:}T_k.s^{i_k+1}$. By the assumption that critical pairs are $cR'$-joinable the context $s[\ ]_q$ is built only from constructors and lambdas and we have the following:

1. $0 < i_1$ and $i_k + 1 \le i_{k+1} - 1$ for every $k \in [1..n-1]$,
2. $E;\mathsf{Rew}(\Gamma,R) \vdash s\rho \trianglerighteq^c s^{i_1-1}\rho$,
3. $E;\mathsf{Rew}(\Gamma,R) \vdash s^{i_k-1}\rho\delta^{k-1} \rhd^c (s^{i_k}\rho\delta^{k-1})@t_k$ for every $k \in [1..n]$,
4. $E;\mathsf{Rew}(\Gamma,R) \vdash (s^{i_k}\rho\delta^{k-1})@t_k \longrightarrow_\beta s^{i_k+1}\rho\delta^k$ for every $k \in [1..n]$,
5. $E;\mathsf{Rew}(\Gamma,R) \vdash s^{i_k+1}\rho\delta^k \trianglerighteq^c s^{i_{k+1}-1}\rho\delta^k$ for every $k \in [1..n-1]$,
6. $E;\mathsf{Rew}(\Gamma,R) \vdash s^{i_n+1}\rho\delta \trianglerighteq^c s^m\rho\delta$.

The first, the fourth and the last statements are obvious as they correspond to constructor subterm on a nonfunctional recursive position. The second statement corresponds to constructor subterm on functional position and it is correct since the term $t_k$ has the right type ($E;\mathsf{Rew}(\Gamma,R) \vdash t_k : T_k\delta^{k-1}$). The third statement corresponds to the beta reduction step.

Summarizing we get $E;\mathsf{Rew}(\Gamma,R') \vdash l\sigma\,(\longrightarrow \cup \rhd^c)^+ l'\gamma\rho\delta$ and hence we may apply the induction hypothesis and get $E;\mathsf{Rew}(\Gamma,R) \Vdash l'\gamma\rho\delta \sim_\omega r'\gamma\rho\delta$. Since $\delta$ is arbitrary we get $E;\mathsf{Rew}(\Gamma,R) \Vdash s \sim_\omega t$.

Recall that $E;\mathsf{Rew}(\Gamma,R) \vdash s \xrightarrow{c}_{R'-R} t$ is an arbitrary step in the critical pair diagram joining $r\theta\rho$ and $\hat{l}\rho$. Obviously, all rewriting steps corresponding to $E;\mathsf{Rew}(\Gamma,R)$ can be replaced by conversion $\approx$. Hence $E;\mathsf{Rew}(\Gamma,R) \Vdash r\sigma \sim_\omega e\rho$ and $E;\mathsf{Rew}(\Gamma,R) \vdash \hat{l}\rho \sim_\omega e\rho$. Of course we have also $E;\mathsf{Rew}(\Gamma,R) \Vdash l\sigma \sim_\omega \hat{l}\rho$. Consequently by transitivity and symmetry of $\sim_\omega$ we get the desired conclusion $E;\mathsf{Rew}(\Gamma,R) \Vdash r\sigma \sim_\omega l\sigma$.