Vérification de propriétés quantitatives et fonctionnelles
Analysis and VERification for the Reliability Of Embedded Systems

Lot 4.3
Technologie de modélisation
Complexité en espace et en temps

Introducing Decision Procedures in the Calculus of Constructions

Description :	We investigate a new version of the calculus of constructions which incorporates arbitrary decision procedures into deductions via the conversion rule of the calculus. Besides the novelty of the problem itself in the context of the calculus of constructions, a major technical innovation of this work lies in the fact that the computation mechanism varies along proof-checking: goals are sent to the decision procedure together with the set of user hypotheses available from the current context. Our main result shows that this extension of the calculus of constructions does not compromise its main properties: confluency, strong normalization and decidability of proof-checking are all preserved.
Auteur(s) :	Frédéric BLANQUI, Jean-Pierre JOUANNAUD Pierre-Yves STRUB
Référence :	AVERROES / Lot 4.3 / Fourniture 3 / V1.0
Date :	25 octobre 2006
Statut :	validé
Version :	1.0

Réseau National des Technologies Logicielles
Projet subventionné par le Ministère de la Recherche et des Nouvelles Technologies

CRIL Technology, France Télécom R&D, INRIA-Futurs, LaBRI (Univ. de Bordeaux – CNRS), LIX (École Polytechnique, CNRS) LORIA, LRI (Univ. de Paris Sud – CNRS), LSV (ENS de Cachan – CNRS)

Historique

juin 2006	V 0.1	version préliminaire
25 octobre 2006	V 1.0	mise au format averroes

Contents

1 Introduction

Background. It is commonly agreed that the success of future proof assistants will rely on their ability to incorporate computations within deductions in order to mimic the mathematician when replacing the proof of a proposition P by the proof of an equivalent proposition P' obtained from P thanks to possibly complex calculations.

Proof assistants based on the Curry-Howard isomorphism such as COQ [7] allow to build the proof of a proposition by applying appropriate proof tactics generating a proof term that can be checked with respect to the rules of logic. The (intuitionnistic) logic on which COQ is based is the Calculus of Constructions (CC) of Coquand and Huet [8], an impredicative type theory incorporating polymorphism, dependent types and type constructors. As other logics, CC enjoys a computation mecanism called cut-elimination, which is nothing but the β-reduction rule of the underlying λ-calculus. But unlike logics without dependent types, CC enjoys also a powerful type-checking rule, called *conversion*, which incorporates computations within deductions as done by the working mathematician: if p is a proof of a proposition P, and P is β-equivalent to P', then p is again a proof of P'. For example, rather than proving that $1 + 3$ is a even number, any math undergrad will prove instead that 4 is an even number. With this rule, the decidability of type-checking becomes a non-trivial property of the calculus.

The traditional view, however, that computations coincide with β-reductions suffers several drawbacks. A methodological one is that the user must encode other forms of computations as deductions. And a practical consequence is that proofs become much larger than necessary, up to a point that they may not be type-checked anymore.

These questions become extremely important when carrying out complex developments involving a large amount of computation. This was recently the case with the first formal proof of the four colour conjecture (now a proof-checked theorem) completed by Gonthier and Werner using COQ [11]. The lack of computing power lead Gonthier to use sophisticated detours in the way he specified the enumeration of the basic maps to be coloured by the system.

The Calculus of Inductive Constructions of Coquand and Paulin was a first attempt to solve this problem by introducing inductive types and the associated elimination rules [6]. The recent versions of COQ are based on a slight generalization of this calculus [10]. Besides the β-reduction rule, they also include the so-called ι-reductions which are recursors for terms and types.

A more general attempt was carried out since the early 90's, by adding user-defined computations as rewrite rules, resulting in the Calculus of Algebraic Constructions [3]. Although conceptually quite powerful, since CAC captures CIC [4], this paradigm does not yet fulfill all needs, because the set of user-defined rewrite rules is fixed and must satisfy several strong assumptions. Besides, the prototype version has not been realised since making type-checking efficient requires novel compilation techniques still under investigation.

The proof assistant PVS uses a potentially stronger paradigm than COQ by combining its deduction mechanism[1] with a notion of computation based on the powerful Shostak's method for combining decision procedures [13], a framework dubbed *little proof engines* by Shankar [14]: the little proof engines are the decision procedures, required to be convex, combined by Shostak's algorithm. While a given decision proof procedures encodes a fixed set of axioms P, an important advantage of the method is that the relevant assumptions A present in the context of the proof are also used by the decision procedure to prove a goal G, and become therefore part of the notion of computation. For example, in the case where the little proof engine is the congruence closure algorithm, the fixed set of axioms P is made of the axioms for equality, A is the set of algebraic ground equalities declared in the context, while the goal G is an equality $s = t$ between two ground expressions. The congruence closure algorithm will then process A and $s = t$ together in order to decide whether or not $s = t$ follows from $P \cup A$. In the Calculus of Constructions, this (possibly long) proof must be constructed by a specific tactic called by the user, which applies the inference rules of CC to the axioms in P and the assumptions in A, and becomes then part of the proof term being built.

[1] PVS logic is not based on Curry-Howard, and proof-checking is not even decidable, which makes both frameworks very different, and difficult to compare.

A step in the direction of integrating decision procedures into the Calculus of Constructions is Stehr's Open Calculus of Constructions OCC [15]. Implemented in Maude, OCC is too general to make type checking decidable. In a preliminary work, we designed a new framework, the Calculus of Congruent Constructions (CCC), which incorporates the congruence closure algorithm in CC's conversion [5].

Problem. The main question investigated in this paper is the incorporation of a general mechanism calling a decision procedure for solving conversion-goals in the Calculus of Algebraic Constructions which uses the relevant information available from the current context of the proof.

Contribution. Our main contribution is the definition and the meta-theoretical investigation of the Calculus of Congruent Constructions (CCC), which incorporates arbitrary decision procedures into deductions via an abstract conversion rule of the calculus. Besides the novelty of the problem itself in the context of the Calculus of Constructions, a major technical innovation of this work lies in the fact that the computation mechanism varies along proof-checking: goals are sent to the decision procedure together with the set of user hypotheses available from the current context. Our main result shows that this extension of the Calculus of Algebraic Constructions does not compromise its main properties: confluency, strong normalization and decidability of proof-checking are all preserved.

2 Calculus of Congruent Constructions

Let Λ be a set of sorts and Σ a Λ-sorted signature. For any $f \in \Sigma$, we write $\mathrm{ar}(f) = (\sigma_1 \times \cdots \times \sigma_n, \sigma)$ the arity of f and define $|\mathrm{ar}(f)|$ by $|\mathrm{ar}(f)| = n$.

Let $\mathcal{S} = \{\star, \Box\}$ - the usual sorts of the calculus of constructions - and for all $s \in \mathcal{S}$, $\mathcal{X}^s$ a countable set of variables (*variables of sort* s) s.t. $\mathcal{X}^\star \cap \mathcal{X}^\Box = \emptyset$. Let $\mathcal{F}^\star = \Sigma$ and $\mathcal{F}^\Box = \{\mathrm{eq}\} \cup \Lambda$. We will write $\mathcal{F}$ (resp. $\mathcal{X}$) for $\mathcal{F}^\star \cup \mathcal{F}^\Box$ (resp. $\mathcal{X}^\star \cup \mathcal{X}^\Box$).

Let $\mathcal{A} = \{\mathrm{r}, \mathrm{u}\}$ a set of *annotations* ordered by $\mathrm{r} \prec_{\mathcal{A}} \mathrm{u}$.

Definition 1 (CCC syntax) *The terms of* CCC *are defined by:*

$$t \in \mathcal{T} ::= s \in \mathcal{S} \mid f \in \mathcal{F} \mid x \in \mathcal{X} \mid tt \mid (\forall x :^a t)t \mid [\lambda x :^a t]t \qquad (a \in \mathcal{A})$$

The usable notions of free variables, substitutions, positions *(...) are defined as usual.*

For each $f \in \mathcal{F}$, we associate a closed type τ_f and sort $s_f \in \mathcal{S}$ as defined bellow:

$[f \in \Sigma]$. *If* $\mathrm{ar}(f) = (\sigma_1 \times \cdots \times \sigma_n, \sigma)$, *then* $\tau_f = \sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to \sigma$ *and* $s_\tau = \star$.
$[\sigma \in \Lambda]$. $\tau_\sigma = \star$ *and* $s_\sigma = \Box$.
$[\mathrm{eq}]$. $\tau_{\mathrm{eq}} = (\forall T :^{\mathrm{u}} \star)T \to T \to \star$ *and* $s_{\mathrm{eq}} = \Box$.

Terms of the form $[\lambda x :^a t]t, (\forall x :^a t)t, tt$ are respectively called abstractions, products, and applications. As usual, we consider terms up to sort-preserving renaming of bound variables, and write $t_1 \to^a t_2$ for $(\forall x :^a t_1)t_2$ whenever $x \notin \mathrm{FV}(t_2)$. A term is *ground* if it has no free variable and *algebraic* when built solely from variables and function symbols in $\mathcal{X}^\star \cup \mathcal{F}^\star$.

Before defining the typing derivation, we define syntactic terms classes:

Definition 2 *We denote by $\mathcal{O}$, $\mathcal{P}$, $\mathcal{K}$, $\mathcal{E}$ the classes of* object level, predicate level, king level *and* extern level *terms, syntactically defined by:*

$$\begin{aligned}
\mathcal{O} &::= \mathcal{X}^\star \mid \mathcal{F}^\star \mid [\lambda x :^a \mathcal{C}]\mathcal{O} \mid [\lambda x :^a \mathcal{K}]\mathcal{O} \mid \mathcal{O}\mathcal{O} \mid \mathcal{O}\mathcal{C} \\
\mathcal{C} &::= \mathcal{X}^\Box \mid \mathcal{F}^\Box \mid (\forall x :^a \mathcal{C})\mathcal{C} \mid (\forall (x :^a \mathcal{K})\mathcal{C} \mid \mathcal{C}\mathcal{O} \mid \mathcal{C}\mathcal{C} \\
\mathcal{K} &::= \star \mid (\forall x :^a \mathcal{C})\mathcal{K} \mid (\forall x :^a \mathcal{K})\mathcal{K} \\
\mathcal{E} &::= \Box
\end{aligned}$$

For any term t, we write i) $\mathcal{CL}(t) = \mathcal{D}$ if $t \in \mathcal{D}$ for $\mathcal{D} \in \{\mathcal{O}, \mathcal{C}, \mathcal{K}, \mathcal{E}\}$; ii) $\mathcal{CL}(t) = \perp$ otherwise. For $\mathcal{D} \in \{\mathcal{O}, \mathcal{C}, \mathcal{K}, \mathcal{E}\}$, we define $\mathcal{D}+1$ by $(\mathcal{O}, \mathcal{C}, \mathcal{K}, \mathcal{E}) + 1 = (\mathcal{C}, \mathcal{K}, \mathcal{E}, \perp)$.

2.1 Typing derivation

Definition 3 *An* environment *is a possibly empty sequence of type declarations of the form* $[x :^a T]$ *($T \in \mathcal{T}$) with no variable declared twice.*

An environment Δ *is an* extension *of an environment* $\Gamma = [x_1 :^a T_1], \dots, [x_n :^a T_n]$*, written* $\Gamma \subseteq \Delta$*, if* Δ *is of the form* $\Delta = \Delta_1, [x_1 :^a T_1], \Delta_2, \dots, \Delta_n, [x_n :^a T_n], \Delta_{n+1}$*.*

In some case, an environment $\Gamma = [x_1 :^a T_1], \dots, [x_n :^a T_n]$ *will be seen as the substitution* $\{x_i \mapsto T_i\}$ *of domain (written* $\mathrm{dom}(\Gamma)$*)* $\{x_1, \dots, x_n\}$*.*

Typing judgements *are written* $\Gamma \vdash t : T$*, where* Γ *is an environment, and* t, T *are terms.*

The typing rules of `CCC` given in Figure 2.1 are very similar to those of CC. Annotations are used to control substitutions as seen from rule APP. CONV uses a yet unspecified family $\{\sim_\Gamma\}_\Gamma$ of congruence relations between terms, refining $=_{\alpha,\beta}$.

$$[\textsc{Symb}]\ \frac{\vdash \tau_f : s_f}{\vdash f : \tau_f} \qquad [(\textsc{Ax})\textsc{iom}]\ \frac{}{\vdash \star : \Box}$$

$$[\textsc{Var}]\ \frac{\Gamma \vdash T : s_x}{\Gamma, [x :^a T] \vdash x : T}\ [x \notin \mathrm{dom}(\Gamma)] \qquad [\textsc{Weak}]\ \frac{\Gamma \vdash t : T \quad \Gamma \vdash U : s_x}{\Gamma, [x :^a U] \vdash t : T}\ [x \notin \mathrm{dom}(\Gamma)]$$

$$[(\textsc{Prod})\textsc{uct}]\ \frac{\Gamma \vdash U : s_U \quad \Gamma, [x :^a U] \vdash V : s_V}{\Gamma \vdash (\forall x :^a U)V : s_V}$$

$$[(\textsc{Abs})\textsc{tract}]\ \frac{\Gamma, [x :^a U] \vdash v : V \quad \Gamma \vdash (\forall x :^a U)V : s}{\Gamma \vdash [\lambda x :^a U]v : (\forall x :^a U)V}$$

$$[(\textsc{App})\textsc{lication}]\ \frac{\Gamma \vdash t : (\forall x :^a U)V \quad \Gamma \vdash u : U}{\Gamma \vdash tu : V\{x \mapsto u\}}$$

$$\text{where} \left|\ \begin{array}{l} \text{if } a = r \text{ and } U \rightarrow^*_\beta \mathrm{eq}\ T\, t_1\, t_2 \text{ with } t_1, t_2 \in \mathcal{O} \\ \text{then } t_1 \sim_\Gamma t_2 \end{array}\right.$$

$$[(\textsc{Conv})\textsc{ersion}]\ \frac{\Gamma \vdash t : T \quad \Gamma \vdash T' : s' \quad T \sim_\Gamma T'}{\Gamma \vdash t : T'}$$

Figure 1: CCC typing rules

2.2 Conversion

Our calculus aims at using an arbitrary decision procedure for some logical theory $\mathcal{T}$ to solve conversion goals. Formulas in this logical theory will be built from the symbols in $\mathcal{F}^\star$. In pratice, the theory will be some decidable fragment of a first-order theory. Another problem is that symbols in `CCC` do not need to be completely applied, while they need when considered as symbols of the theory: we will therefore distinguish the symbols used in $\mathcal{T}$ from those used in `CCC`.

Definition 4 *Assume given a equivalence relation* $\mathcal{R}$ *refining* α*-equivalence and a one-to-one mapping* $\pi_\mathcal{R}$ *from equivalence classes of terms to a set* $\mathcal{Y}$ *of fresh variables. Let* t *be a raw term in* `CCC`*. We define* $\mathrm{cap}_\mathcal{R}(t)$ *by:*

- *Let* $t = x \in \mathcal{X}$*. Then* $\mathrm{cap}_\mathcal{R}(x) = x$ *;*

- *Let* $t = (\ldots(f\ t_1)\ldots)t_n$ *with* $f \in \mathcal{F}^\star$ *and* $|\operatorname{ar}(f)| = n$ *- that is* f fully applied. *Then* $\operatorname{cap}_{\mathcal{R}}(t) = f(\operatorname{cap}_{\mathcal{R}}(t_1), \ldots, \operatorname{cap}_{\mathcal{R}}(t_n))$ *;*
- *Otherwise,* $\operatorname{cap}_{\mathcal{R}}(t) = \pi_{\mathcal{R}}(t)$.

In case of $\mathcal{R}$ *is not an congruence,* $\operatorname{cap}_{\mathcal{R}}(t) = \operatorname{cap}_{\mathcal{S}}(t)$ *and* $\pi_{\mathcal{R}}(t) = \pi_{\mathcal{S}}(t)$ *where* $\mathcal{S}$ *is the smallest congruence relation containing* $\mathcal{R}$*.*

In this transformation, the so-called *alien subterms* of t have been abstracted by a constant, according to their equivalence class modulo $\mathcal{R}$. $\operatorname{cap}_{\mathcal{R}}(t)$ is sometimes called the *cap of t modulo* $\mathcal{R}$.

We can now define the relation $\sim_\Gamma$ for an environment Γ:

Definition 5 *The family of relations* $\{\sim_\Gamma\}$*, indexed by typing environments, is defined by the rules of Figure 2.2.*

$$[\beta]\ \frac{t =_\beta u}{t \sim_\Gamma u} \qquad [\textsc{Eq}]\ \frac{[z :^{\mathrm{r}} W] \in \Gamma \quad W \rightarrow^*_\beta \operatorname{eq} T\, u\, v \quad u, v \in \mathcal{O}}{u \sim_\Gamma v}$$

$$[\textsc{Ded}]\ \frac{\{\operatorname{cap}_{\sim_\Gamma}(u) = \operatorname{cap}_{\sim_\Gamma}(v) \mid u \sim_\Gamma v\} \vdash \operatorname{cap}_{\sim_\Gamma}(s) = \operatorname{cap}_{\sim_\Gamma}(t)}{s \sim_\Gamma t}$$

$$[\textsc{Trans}]\ \frac{t \sim_\Gamma u \quad u \sim_\Gamma v}{t \sim_\Gamma v} \qquad [\textsc{Sym}]\ \frac{t \sim_\Gamma u}{u \sim_\Gamma t} \qquad [\textsc{App}]\ \frac{t_1 \sim_\Gamma u_1 \quad t_2 \sim_\Gamma u_2}{t_1 t_2 \sim_\Gamma u_1 u_2}$$

$$[\textsc{Lam}]\ \frac{t_1 \sim_\Gamma u_1 \quad t_2 \sim_\Gamma u_2}{[\lambda x :^a t_1]t_2 \sim_\Gamma [\lambda x :^a u_1]u_2} \qquad [\textsc{Prod}]\ \frac{t_1 \sim_\Gamma u_1 \quad t_2 \sim_{\Gamma, [x:^a t_1]} u_2}{(\forall x :^b t_1)t_2 \sim_\Gamma (\forall x :^b u_1)u_2}\ [x \notin \operatorname{dom}(\Gamma),\ a \prec b]$$

Figure 2: The Conversion Relation $\sim_\Gamma$

One can wonder wether a simpler definition of conversion would be possible. The answer is no one wants conversion to be non trivial. Defining conversion as a fixpoint actually came from the proof of the substitution lemma to come next. It is also necessary in the proof of confluence in the DED rule.

3 Meta-theorical properties

3.1 Caps and all that...

We will now describe more precisely the notion *cap* and aliens for terms in `CCC`, and enounce some properties about them:

Definition 6 (Cap and aliens positions) *For any term* $t \in \mathcal{T}$*, we define the set of* cap positions - $\operatorname{CPos}(t)$ - *and* alien positions - $\operatorname{APos}(t)$ - *by:*

- *Let* $t = x \in \mathcal{X}$*. Then* $\operatorname{APos}(x) = \emptyset$*;*
- *Let* $t = (\ldots(f\ t_1)\ldots)t_n$ *with* $f \in \mathcal{F}^\star$ *and* f fully applied *- that is* $|\operatorname{ar}(f)| = n$*. Then* $\operatorname{APos}(t) = \bigcup_{i=0}^{n-1} 1^i \cdot 2 \cdot \operatorname{APos}(t_{n-i})$*;*
- *Otherwise,* $\operatorname{APos}(t) = \{\epsilon\}$

and $\operatorname{CPos}(t) = \{p \in \operatorname{Pos}(t) \mid p$ *is a strict prefix of a position in* $\operatorname{APos}(t)\}$.

Lemma 1 *1.* $\forall p, q \in \mathrm{APos}(t), p \neq q \implies$ *p and q are not comparable in the prefix ordering;*

2. $\mathrm{CPos}(t) \cap \mathrm{APos}(t) = \emptyset$ *;*

3. $\forall p \in \mathrm{Pos}(t)$*, if* $p \notin \mathrm{CPos}(t) \cup \mathrm{APos}(t)$ *then* $\exists! q \in \mathrm{APos}(t)$ *s.t. q is a prefix of p ;*

4. $\mathrm{cap}_{\mathcal{R}}(t) = t[\pi_{\mathcal{R}}(t_{|p})]_{p \in \mathrm{APos}(t)}$

Lemma 2 *Let* $t \in \mathcal{T}$ *and* θ *a substitution over* $\mathcal{T}$*. Then:*

1. $\mathrm{APos}(t\theta) = \mathrm{APos}(t) \uplus \{p1 \cdot p2 \mid p1 \in P_{t,\theta}, p_2 \in \mathrm{APos}((t(p_1))\theta)\}$

2. $\mathrm{CPos}(t\theta) = (\mathrm{CPos}(t) - P_{t,\theta}) \uplus \{p1 \cdot p2 \mid p1 \in P_{t,\theta}, p_2 \in \mathrm{CPos}((t(p_1))\theta)\}$

where $P_{t,\theta} = \{p \in \mathrm{CPos}(t) \mid t(p) \in \mathrm{dom}(\theta)\}$.

3.2 General properties

Lemma 3 *Assume* $f : \mathcal{T} \to A$*, where A is an abitrary set, s.t. f is compositional and compatible with algebraic context (i.e.* $f(t) = f(u)$ *implies that* $f(C[t]) = f(C[u])$ *for any terms t, u and algebraic context C). Let* $\mathcal{R}$ *be a binary relation on* $\mathcal{T}$ *compatible with f (i.e.* $u \mathrel{\mathcal{R}} v$ *implies* $f(u) = f(v)$*). If* $\{\mathrm{cap}_{\mathcal{R}}(w_1) = \mathrm{cap}_{\mathcal{R}}(w_2) \mid w_1 \mathrel{\mathcal{R}} w_2\} \vdash \mathrm{cap}_{\mathcal{R}}(u) = \mathrm{cap}_{\mathcal{R}}(v)$ *then* $f(u) = f(v)$.

Proof. By induction on the length n of the equational derivation $\mathrm{cap}_{\mathcal{R}}(u) \leftrightarrow_E^* \mathrm{cap}_{\mathcal{R}}(v)$. □

Lemma 4 (cap θ-monotony) *Given i) a substitution* θ *whose domain is included in* $\mathcal{X}$ *and ii) two binary relations* $\mathcal{R}$ *and* $\mathcal{R}'$ *on* $\mathcal{T}$ *s.t.* $\mathcal{R} \subseteq \mathcal{R}'\theta$ *(where* $\mathcal{S}\theta = \{(t\theta, u\theta) \mid t \mathrel{\mathcal{S}} u\}$*), there exists a substitution* $\xi_{\mathcal{R}}^{\mathcal{R}'}$ *s.t. for any term* $t \in \mathcal{T}$*,* $\mathrm{cap}_{\mathcal{R}'}(t\theta) = \mathrm{cap}_{\mathcal{R}}(t)\xi_{\mathcal{R}}^{\mathcal{R}'}$.

Corollary 1 (Monotony of DED) *Let i)* θ *a substitution whose domain is included in* $\mathcal{X}$ *and ii) two binary relations* $\mathcal{R}$ *and* $\mathcal{R}'$ *on* $\mathcal{T}$ *s.t.* $\mathcal{R} \subseteq \mathcal{R}'\theta$*. Suppose that* $\{\mathrm{cap}_{\mathcal{R}}(s) = \mathrm{cap}_{\mathcal{R}}(t) \mid s \mathrel{\mathcal{R}} t\} \vdash \mathrm{cap}_{\mathcal{R}}(u) = \mathrm{cap}_{\mathcal{R}}(v)$*. Then,* $\{\mathrm{cap}_{\mathcal{R}'}(s) = \mathrm{cap}_{\mathcal{R}'}(t) \mid s \mathrel{\mathcal{R}'} t\} \vdash \mathrm{cap}_{\mathcal{R}'}(u) = \mathrm{cap}_{\mathcal{R}'}(v)$.

3.3 Basic properties

Lemma 5 (Free variables) *Assume that* $\Gamma \vdash t : T$*. Then:*

1. $\mathrm{FV}(t) \cup \mathrm{FV}(T) \subseteq \mathrm{dom}(\Gamma)$.

2. If $\Gamma = \Gamma_1, [x :^a T], \Gamma_2$*, then* $\mathrm{FV}(T) \subseteq \mathrm{dom}(\Gamma_1)$.

Proof. (1) is proved by an immediate induction on $\Gamma \vdash t : T$. (2) is an consequence of (1). □

Lemma 6 *If* $t \sim_\Gamma u$*, then* $\mathcal{CL}(t) = \mathcal{CL}(u)$.

Proof. By induction on $t \sim_\Gamma u$, by case on the last rule used. The DED case is a corollary of Lemma 3. □

Lemma 7 *Assume that* $\Gamma \vdash t : T$*. Then,* $\mathcal{CL}(t) \neq \perp$*,* $\mathcal{CL}(T) \neq \perp$ *and* $\mathcal{CL}(T) = \mathcal{CL}(t) + 1$.

Proof. Direct induction on $\Gamma \vdash t : T$. The CONV case is a corollary of Lemma 6. □

Lemma 8 *Assume that* Γ *and* Δ *are two typing environments s.t. if* $[x :^{\mathrm{r}} T] \in \Gamma$ *with* $T \to_\beta^*$ $\mathrm{eq}\, U\, t_1\, t_2$ *and* $t_1, t_2 \in \mathcal{O}$*, then there exists* $y \in \mathrm{dom}(\Delta)$ *s.t. y is annotated with* r *in* Δ*, and* $y\Delta \to_\beta^* \mathrm{eq}\, V\, t_1\, t_2$*. Then,* $\sim_\Gamma \subseteq \sim_\Delta$.

Proof. By proving that $t \sim_\Gamma u \implies t \sim_\Delta u$, by induction on $t \sim_\Gamma u$. □

Corollary 2 *1. If $\Gamma \subseteq \Delta$, then $\sim_\Gamma \subseteq \sim_\Delta$.*

2. If $\Gamma = [x_1 :^{a_1} T_1] \cdots [x_n :^{a_n} T_n]$ and $\Delta = [x_1 :^{b_1} T_1] \cdots [x_n :^{b_n} T_n]$ with $\forall i.\, b_i \succeq_{\mathcal{A}} a_i$, then $\sim_\Gamma \subseteq \sim_\Delta$.

Lemma 9 (Compatibility) *Assume that $t \sim_\Gamma t'$.*

1. *If $t = (\forall x :^a U)V$, then $t' \rightarrow^*_\beta (\forall x :^a U')V'$ with $U \sim_\Gamma U'$, $V \sim_{\Gamma,[x:^bU]} V'$ and $b \succ_{\mathcal{A}} a$.*
2. *If $t = s \in \mathcal{S}$, then $t' \rightarrow^*_\beta s$.*
3. *If $t = x \in \mathcal{X}^\square$, then $t' \rightarrow^*_\beta x$.*

Proof. 1. By induction on $t \sim_\Gamma t'$, we can prove that if $t \rightarrow^*_\beta (\forall x :^a U)V$ (resp. $t' \rightarrow^*_\beta (\forall x :^a U')V'$), then $t' \rightarrow^*_\beta (\forall x :^a U')V'$ (resp. $t \rightarrow^*_\beta (\forall x :^a U)V$) with $U \sim_\Gamma U'$ and $V \sim_{\Gamma,[x:^bU]} V'$ where $a \succ_{\mathcal{A}} b$. The main arguments are i) since $t \rightarrow^*_\beta (\forall x :^a U)V$, then it is of the form $([\lambda\,(x_1 :^{a_1} T_1) \cdots (x_n :^{a_n} T_n)](\forall x :^a U_\beta)V_\beta)A_1 \cdots A_n$ with $U_\beta \rightarrow^*_\beta U$ and $V_\beta \rightarrow^*_\beta V$ and ii) EQ extract only object level equations. $\square$

Lemma 10 *1. Assume that $\Gamma \vdash t : T$. Then,* $\mathrm{FV}(t) \cup \mathrm{FV}(T) \subseteq \mathrm{dom}(\Gamma)$.

2. Any subterm of a well formed term is well formed.

Lemma 11 (Weakening) *Assume that $\Gamma \vdash t : T$ and $\Delta \supseteq \Gamma$ s.t. there exists at least one term typable under Δ. Then $\Delta \vdash t : T$.*

Lemma 12 *1. If $\Gamma \vdash t : T$ and $\Delta \supseteq \Gamma$ is a well formed environment, then $\Delta \vdash t : T$.*

2. If $\Gamma_1, [x_s :^a T], \Gamma_2$ is a well formed environment, then $\Gamma_1 \vdash T : s$.

Lemma 13 (Substitutivity) *Let $\Gamma = \Gamma_1, [z :^a W], \Gamma_2$, $\theta = \{z \mapsto w\}$, $\Delta = \Gamma_1, \Gamma_2\theta$ and two terms T, T' s.t. i) $T \sim_\Gamma T'$, ii) $\Gamma_1 \vdash w : W$ and iii) if $a = r$ and $W \leftrightarrow^*_\beta \mathrm{eq}\, V\, u\, v$ with $V \leftrightarrow^*_\beta V' \in \Lambda$, then $u \sim_{\Gamma_1} v$. Then, $T\theta \sim_\Delta T'\theta$.*

Proof. For any environment Γ' s.t. $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Gamma') = \emptyset$, we prove the more general statement $T \sim_{\Gamma,\Gamma'} T' \implies T\theta \sim_{\Delta,\Gamma'\theta} T'\theta$, by induction on the proof of $T \sim_{\Gamma,\Gamma'} T'$. By case on the last rule used:

- If $T \sim_{\Gamma,\Gamma'} T'$ is derived from a structural rule (LAM, PROD, APP) or the TRANS rule, we obtain the result by application of the induction hypothesis on the premises.
- If $T \sim_{\Gamma,\Gamma'} T'$ is obtained by the β rule. Then $T =_\beta T'$. By compatibility of the β-reduction with substitutions, we have $T\theta =_\beta T'\theta$ and by application of the β rule, $T\theta \sim_{\Delta,\Gamma'\theta} T'\theta$.
- If $T \sim_{\Gamma,\Gamma'} T'$ is obtained from the EQ rule with the premises $\Gamma, \Gamma' = \Gamma_\alpha, [x :^{\mathrm{r}} V], \Gamma_\beta$ and $V \leftrightarrow^*_\beta \mathrm{eq}\, V'\, u\, v$ with $V' \leftrightarrow^*_\beta V'' \in \Lambda$, we distinguish 2 cases:
  1. $[x \neq z]$. Then $[x :^{\mathrm{r}} V\theta] \in \Delta, \Gamma'\theta$ $(*)$ and $V\theta \leftrightarrow^*_\beta \mathrm{eq}\,(V'\theta)\,(u\theta)\,(v\theta)$ with $V'\theta \leftrightarrow^*_\beta V''\theta \in \Lambda$ $(**)$:
     - $(*)$ If x appears before z in Γ, Γ', then $z \notin \mathrm{FV}(V)$ (Lemma 5) and $\Gamma_\alpha, [x :^{\mathrm{r}} V] \subseteq \Gamma_1$. Thus, $[x :^{\mathrm{r}} V\theta] = [x :^{\mathrm{r}} V] \in \Gamma_1 \subseteq \Delta, \Gamma'\theta$. Otherwise, $[x :^{\mathrm{r}} V], \Gamma_\beta \subseteq \Gamma_2, \Gamma'$, thus $[x :^{\mathrm{r}} V\theta], \Gamma_\beta\theta \subseteq \Gamma_2\theta, \Gamma'\theta \subseteq \Delta, \Gamma'\theta$.
     - $(**)$ We obtain the result by application of the compatibility of the β-reduction with substitutions; and by remarking that since V'' is necesseraly closed ($V'' \in \Lambda$), we have $V''\theta = V'' \in \Lambda$.

  Finally, from $(*)$ and $(**)$, by application of the EQ rule, we have $u\theta \sim_{\Delta,\Gamma'} v\theta$;

2. [$x = z$ and $W \equiv V \leftrightarrow_\beta^* \mathrm{eq}\, V'\, u\, v$ with $V' \leftrightarrow_\beta^* V'' \in \Lambda$]. By confluence of β-reduction, there exists a term T s.t. $V \rightarrow_\beta^* T$ and $\mathrm{eq}\, V'\, u\, v \rightarrow_\beta^* T$. Thus, T is necesseraly of the form $\mathrm{eq}\, T'\, u'\, v'$ with $u \rightarrow_\beta^* u'$, $v \rightarrow_\beta^* v'$ and $T' \leftrightarrow_\beta^* V'' \in \Lambda$. By Eq rule, we deduce that $u' \sim_{\Gamma_1} v'$. From $z \notin \mathrm{FV}(V)$ (Lemma 5) and $V \rightarrow_\beta^* \mathrm{eq}\, T'\, u'\, v'$, we have $z \notin \mathrm{FV}(u') \cup \mathrm{FV}(v')$, implying $u'\theta \sim_{\Gamma_1} v'\theta$. Now, by β-compatibility with substitutions, we have $u\theta \leftrightarrow_\beta^* u'\theta$ (resp. $v\theta \leftrightarrow_\beta^* v'\theta$) and thus (by application of the β rule) $u\theta \sim_{\Gamma_1} u'\theta$ (resp. $v\theta \sim_{\Gamma_1} v'\theta$). Finally, by application of the Trans rule, we obtain $u\theta \sim_{\Gamma_1} v\theta$ and we conclude by Lemma 2.

- If $T \sim_\Gamma T'$ is obtained from the Ded rule, then by application of Lemma 1 to the rule premise, we obtain:

$$\{\mathrm{cap}_{\sim_\Delta}(s) = \mathrm{cap}_{\sim_\Delta}(t) \mid s \sim_\Delta t\} \vdash \mathrm{cap}_{\sim_\Delta}(T) = \mathrm{cap}_{\sim_\Delta}(T') \ (*)$$

  We conclude by applying the Ded rule to $(*)$. □

Corollary 3 *Let $\Gamma = \Gamma_1, [z :^a W], \Gamma_2$, $\theta = \{z \mapsto w\}$, and two terms t, T s.t. i) $\Gamma \vdash t : T$, ii) $\Gamma_1 \vdash w : W$ and iii) if $a = r$ and $W \leftrightarrow_\beta^* \mathrm{eq}\, V\, u\, v$ with $V \leftrightarrow_\beta^* V' \in \Lambda$, then $u \sim_{\Gamma_1} v$. Then, $\Gamma_1, \Gamma_2\theta \vdash t\theta : T\theta$.*

Lemma 14 (Type correctness) *Assume $\Gamma \vdash t : T$. Then, $T = \Box$ or $\Gamma \vdash T : s \in \mathcal{S}$.*

Proof. By induction on $\Gamma \vdash t : T$, by case on the last rule used:

- [App]. i.e. $\Gamma \vdash uv : V\{x \mapsto v\}$ with $\Gamma \vdash t : (\forall x :^a V)W$, $\Gamma \vdash v : V$.

  By induction hypothesis, there exists a sort s s.t. $\Gamma \vdash (\forall x :^a V)W : s$. By inversion, $\exists s_W \in \mathcal{S}$ s.t. $\Gamma, [x :^a V] \vdash W : s_W$ with $s \sim_\Gamma s_W$, that is $s = s_W$ by Lemma 9. By substitutivity lemma, $\Gamma \vdash W\{x \mapsto v\} : s$.

- All other cases are (even more) straightforward. □

Lemma 15 (Inversion) *Assume that $\Gamma \vdash t : T$. Then:*

1. *if $t \in \mathcal{X}^s$, then $\Gamma \vdash T : s$ and $x\Gamma \sim_\Gamma T$*
2. *if $f \in \mathcal{F}$, then $\Gamma \vdash \tau_f : s_f$ and $\tau_f \sim_\Gamma T$*
3. *i) if $t \in \mathcal{S}$, then $t = \star$ and $T = \Box$, ii) it is not possible that $t = \Box$*
4. *if $t = uv$, then i) $\Gamma \vdash u : (\forall x :^a V)W$, ii) $\Gamma \vdash v : V$ and iii) $W\{x \mapsto v\} \sim_\Gamma T$. Moreover, if $a = \mathrm{r}$ and $V \leftrightarrow_\beta^* \mathrm{eq}\, V'\, t_1\, t_2$ with $V' \leftrightarrow_\beta^* V'' \in \Lambda$, then $t_1 \sim_\Gamma t_2$*
5. *if $t = (\forall x :^a U)V$, then i) $\Gamma \vdash U : s_U$, ii) $\Gamma, [x :^a U] \vdash V : s_V$ and iii) $T \sim_\Gamma s_V$*
6. *if $t = [\lambda x :^a U]v$, then i) $\Gamma \vdash U : s_U$, ii) $\Gamma, [x :^a U] \vdash v : V$, iii) $\Gamma, [x :^a U] \vdash V : s_V$, iv) $\Gamma \vdash T : s_V$ and v) $(\forall x :^a U)V \sim_\Gamma T$ where s_V is the sort of x.*

Lemma 16 (Type unicity) *Assume that $\Gamma \vdash t : T$ and $\Gamma \vdash t : T'$. Then $T \sim_\Gamma T'$.*

Proof. Inmmediate by structural induction on t and by inversion lemma. □

Lemma 17 *Assume that $\Gamma \vdash t : T$, $t \rightarrow_\beta t'$ and $T \rightarrow_\beta T'$. Then, $\Gamma \vdash t' : T$ and $\Gamma' \vdash t : T'$.*

Proof. By induction on $\Gamma \vdash t : T$, we prove that i) $t \rightarrow_\beta t' \implies \Gamma \vdash t' : T$, and ii) $\Gamma \rightarrow_\beta \Gamma' \implies \Gamma' \vdash t : T$. By case on the last rule used:

- [Var]. i.e. $t = x \in \mathcal{X}^s$, $\Gamma = \Gamma_1, [x :^a T]$ and $\Gamma_1 \vdash T : s$. i) $x \in \mathrm{SN}(\beta)$. ii) if the reduction is on Γ_1, this is immediate. Else, by induction hypothesis, $\Gamma_1 \vdash T' : s$, and by the Var rule, $\Gamma_1, [x :^a T'] \vdash x : T'$. We conclude by applying the Conv rule.

- [APP]. i.e. $t = uv$, $T = W\{x \mapsto v\}$, $\Gamma \vdash u : (\forall x :^a V)W$, $\Gamma \vdash v : V$. If $u \to_\beta u'$, this is immediate. If $v \to_\beta v'$, then by induction hypothesis, $\Gamma \vdash uv' : W\{x \mapsto v'\}$. By conversion ($W\{x \mapsto v\} \sim_\Gamma W\{x \mapsto v'\}$), we obtain $\Gamma \vdash uv' : W\{x \mapsto v'\}$. Now, if $u = [\lambda x :^a V']w$ and $uv \to_\beta w\{x \mapsto v\}$, by inversion, $\Gamma, [x :^a V'] \vdash w : W'$ with $(\forall x :^a V)W \sim_\Gamma (\forall x :^a V')W'$. Thus, by Lemma 9, $V \sim_\Gamma V'$ and $W \sim_{\Gamma,[x:^b V]} W'$ with $b \succeq_{\mathcal{A}} a$. By conversion, $\Gamma \vdash v : V'$, and by applying $\{x \mapsto v\}$ to $\Gamma, [x :^a V] \vdash w : W'$, we have $\Gamma \vdash w\{x \mapsto v\} : W'\{x \mapsto v\}$. Finally, by conversion, $\Gamma \vdash w\{x \mapsto v\} \vdash W\{\mapsto v\}$.
- all other cases are straightforward. □

3.4 Logical consistency

Lemma 18 (β-normalization) *Any well formed term is strongly normalizing for $\to_\beta$.*

Proof. Following [2], we will prove that $\sim_{[\cdot]}$ are *proof irrelevant* conversions. Let $|\cdot| : \mathcal{T} \to \mathcal{T} \cup \{\circ\}$, defined by:

$$\begin{array}{lll} |t| & = \mathcal{O} & \text{if } t \in \mathcal{O} \\ |v| & = v & \text{if } v \in \mathcal{X}^\Box \cup \mathcal{F}^\Box \cup \{\star, \Box\} \\ |tu| & = |t||u| & \text{if } tu \notin \mathcal{O} \\ |[\lambda x :^a U]t| & = [\lambda x :^a |U|]|t| & \text{if } [\lambda x :^a U]t \notin \mathcal{O} \end{array}$$

Assume that $t \sim_\Gamma u$. By a simple induction on $t \sim_\Gamma u$, we prove that $|t| \leftrightarrow^*_\beta |u|$ (here too, the DED case is a corollary of Lemma 3). Hence, $\sim_\Gamma$ are proof irrevelant relations and, from [2], all well formed terms are strongly normalizing for $\to_\beta$. □

Corollary 4 (Logical consistency) *If the first order theory embedded in the calculus is coherent, there are no proofs of $(\forall P :^{\mathrm{u}} \star)P$ is the empty environment.*

4 Further work

There are a lot of problems to be solved:

Implementation in the COQ System. A basic prototype has been done in the MAUDE system, and we know want to include decision procedures in COQ. This part will required a generalization of the abstract machine of Grégoire [12] in order to obtain a compiled version of COQ.

Inductive Types. Prior to including decision procedures in COQ, this work must be extended to the Calculus of Inductive Constructions - the base logical framework of COQ. This work will be harder due to the presence of strong elimination which prevent us to use the *proof irrelevance* in the proof of logical consistency.

References

[1] H. Barendregt. Lambda calculi with types. In S. Abramski, D. Gabbay, and T. Maibaum, editors, *Handbook of logic in computer science*, volume 2. Oxford University Press, 1992.

[2] G. Barthe. The relevance of proof irrelevance, 1998. In Proc. of the 25th Int. Colloq. on Automata, Languages and Programming, LNCS 1443.

[3] F. Blanqui. Definitions by rewriting in the Calculus of Constructions, 2003. 57 pages. Mathematical Structures in Computer Science.

[4] F. Blanqui. Inductive types in the Calculus of Algebraic Constructions, 2004. 26 pages. Fundamenta Informaticae.

[5] F. Blanqui, J.-P. Jouannaud and P.-Y. Strub. A Calculus of Congruent Constructions, unpublished draft, 2005.

[6] Th. Coquand and C. Paulin-Mohring. Inductively defined types. In P. Martin-Löf and G. Mints, editors, Proceedings of Colog'88, volume 417 of Lecture Notes in Computer Science. Springer-Verlag, 1990.

[7] Coq-Development-Team. *The Coq Proof Assistant Reference Manual - Version 8.0.* INRIA Rocquencourt, France, 2004. at URL http://coq.inria.fr/.

[8] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2-3):95–120, 1988.

[9] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6. North-Holland, 1990.

[10] E. Giménez. Structural Recursive Definitions in Type Theory. Proceedings of ICALP'98, LNCS 1443, July 1998.

[11] G. Gonthier. The four colour theorem in Coq. Presentation at Types 2004.

[12] Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. International Conference on Functional Programming 2002, pages 235–246, ACM Press.

[13] R. E. Shostak. An efficient decision procedure for arithmetic with function symbols. *J. of the Association for Computing Machinery* 26(2):351–360, 1979.

[14] N. Shankar. Little Engines of Proof. Proc. FME 2002.

[15] Mark Oliver Stehr. Explicit substitutions and the Open Calculs of Constructions. Proc. WRLA, 2002.