Building Decision Procedures in the Calculus of Inductive Constructions

Frédéric Blanqui
LORIA, UMR CNRS 7503
INRIA Lorraine, BP 101
54506 Villers-lès-Nancy, FRANCE

J.-P. Jouannaud and P.-Y. Strub*
LIX, UMR CNRS 7161
École Polytechnique
91128 Plaiseau, FRANCE

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 [8] 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 proof-checker, also called the *kernel* of the proof assistant, implements the inference and deduction rules of the logic on top of a term manipulation layer. Trusting the kernel is a vital need since the mathematical correctness of a proof development relies entirely on the kernel.

The (intuitionist) logic on which Coq is based is the Calculus of Constructions (CC) of Coquand and Huet [9], an impredicative type theory incorporating polymorphism, dependent types and type constructors. As other logics, CC enjoys a computation mechanism 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: 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 such a rule, decidability of type-checking becomes a non-trivial property of the calculus.

The traditional view that computations coincide with β-reductions suffers several drawbacks. A methodological one is that the user must encode other forms of computations as deductions, which is usually done by using appropriate, complex tactics. A practical consequence is that proofs become much larger than necessary, up to a point that they cannot be type-checked anymore. These questions become extremely important when carrying out complex developments involving a large amount of computation as the formal proof of the four colour (now proof-checked) theorem completed by Gonthier and Werner using Coq [13]. The lack of computing power lead Gonthier to use sophisticated detours for specifying the enumeration of all 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 [10]. The recent versions of Coq are based on a slight generalization of this calculus [12]. Besides the β-reduction rule, they also include the so-called ι-reductions which are recursors for terms and types. While the kernel of CC is extremely compact and simple enough to make it easily readable -hence trustable-, the kernel of CIC is much larger and quite complex. Trusting it would require a formal proof, which was done once [3]. Updating that proof for each new release of the system is however unrealistic.

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 Inductive Constructions [5]. Although conceptually quite powerful, since CAC captures CIC [6], this paradigm does not yet fulfill all needs, because the set of user-defined rewrite rules must satisfy several strong assumptions. No implementation of CAC has indeed been released because making type-checking efficient would require compiling the user-defined rules, hence resulting in a kernel too large to be trusted anymore.

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 [16], a framework dubbed *little proof engines* by Shankar [15]: the little proof engines are the decision procedures, required to be convex, combined by Shostak's algorithm. A given deci-

*Project LogiCal, Pôle Commun de Recherche en Informatique du Plateau de Saclay, CNRS, École Polytechnique, INRIA, Univ. Paris-Sud.

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

sion procedure encodes a fixed set of axioms P. But 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 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. Reflexion techniques allow to omit checking this proof term by proving the decision procedure itself, but the soundness of the entire mechanism cannot be guaranteed [11].

A further step in the direction of integrating decision procedures into the Calculus of Constructions is Stehr's Open Calculus of Constructions OCC [17]. Implemented in Maude, OCC is too general to make type checking decidable. In a preliminary work, we also designed a new framework, the Calculus of Congruent Constructions (CCC), which incorporates the congruence closure algorithm in CC's conversion [7], while preserving the good properties of the calculus, including the decidability of type checking.

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 Inductive Constructions which uses the relevant information available from the current context of the proof.

Contributions. Our main contribution is the definition and the meta-theoretical investigation of the Calculus of Congruent Inductive Constructions (CCIC), which incorporates arbitrary *first-order theories* for which entailment is decidable into deductions via an abstract conversion rule of the calculus. A major technical innovation of this work lies in the computation mechanism: 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 CIC does not compromise its main properties: confluency, strong normalization, coherence and decidability of proof-checking are all preserved.

As a second key contribution, we show how a goal to be proved in the Calculus of Inductive Constructions can actually be transformed into a goal in a decidable first-order theory. Based on this transformation, we are currently developing a new version of Coq implementing this calculus.

Finally, we explain why the new system is still trustable, by leaving decision procedures *out* of its kernel, assuming that each procedure delivers a checkable *certificate* which becomes part of the proof. Certificate checkers become themselves part of the kernel, but are usually quite small and efficient and can be added one by one, making this approach a good compromise between CAC and CIC.

We assume some familiarity with typed lambda calculi [2] and the Calculus of Inductive Constructions.

2. The calculus

For ease of the presentation, we restrict ourselves to $\mathrm{CC}_{\mathbb{N}}$, a calculus of constructions with a type nat of natural numbers generated by its two constructors 0 and S and equipped with its weak recursor $\mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}$. Adding the strong recursor in order to have the full Calculus of Inductive Constructions is discussed in Section 5. The calculus is also equipped with a polymorphic equality symbol, written $\doteq$.

Let $\mathcal{S} = \{\star, \Box, \triangle\}$ the set of $\mathrm{CC}_{\mathbb{N}}$ *sorts*. For $s \in \{\star, \Box\}$, $\mathcal{X}^s$ denotes a countably infinite set of *s-sorted variables* s.t. $\mathcal{X}^\star \cap \mathcal{X}^\Box = \emptyset$. The union $\mathcal{X}^\star \cup \mathcal{X}^\Box$ will be written $\mathcal{X}$. Let $\mathcal{A} = \{\mathbf{u}, \mathbf{r}\}$ a set of two constants called *annotations*, totally ordered by $\mathbf{u} \prec_{\mathcal{A}} \mathbf{r}$. We use a for an arbitrary annotation. Annotations are necessary for subject reduction.

Definition 2.1 (Pseudo-terms of $\mathrm{CC}_{\mathbb{N}}$). *We define the* pseudo-terms *of* $\mathrm{CC}_{\mathbb{N}}$ *by the grammar rules:*

$$t, T := x \in \mathcal{X} \mid s \in \mathcal{S} \mid \mathrm{nat} \mid \doteq \mid 0 \mid \mathrm{S} \mid + \mid \mathrm{Eq}(t) \mid t\,u \mid \lambda[x :^a T]t \mid \forall(x :^a T).\,t \mid \mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}(t, T)\{t_0, t_S\}$$

We use $\mathrm{FV}(t)$ *for the set of free variables of* t.

Definition 2.2 (Pseudo-contexts of $\mathrm{CC}_{\mathbb{N}}$). *The* typing environments *of* $\mathrm{CC}_{\mathbb{N}}$ *are defined as* $\Gamma, \Delta := [] \mid \Gamma, [x :^a T]$. *We use* $\mathrm{dom}\,\Gamma$ *for the domain of* Γ.

Definition 2.3 (Syntactic classes). *The pairwise disjoint syntactic classes of* $\mathrm{CC}_{\mathbb{N}}$*, called* objects *($\mathcal{O}$),* predicates *or* types *($\mathcal{P}$),* kinds *($\mathcal{K}$),* externs *($\mathcal{E}$) and* $\triangle$ *are defined as:*

$$\begin{aligned}
\mathcal{O} &:= \mathcal{X}^\star \mid 0 \mid \mathrm{S} \mid \mathcal{O}\,\mathcal{O} \mid \mathcal{O}\,\mathcal{P} \mid [\lambda\mathcal{X}^\star :^a \mathcal{P}]\mathcal{O} \mid \\
&:= [\lambda\mathcal{X}^\Box :^a \mathcal{K}]\mathcal{O} \mid \mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}(\mathcal{O}, \cdot)\{\mathcal{O}, \mathcal{O}\} \\
\mathcal{P} &:= \mathcal{X}^\Box \mid \mathrm{nat} \mid \mathcal{P}\,\mathcal{O} \mid \mathcal{P}\,\mathcal{P} \mid [\lambda\mathcal{X}^\star :^a \mathcal{P}]\mathcal{P} \mid \doteq \mid \\
&:= [\lambda\mathcal{X}^\Box :^a \mathcal{K}]\mathcal{P} \mid (\forall\mathcal{X}^\star :^a \mathcal{P})\mathcal{P} \mid (\forall\mathcal{X}^\Box :^a \mathcal{K})\mathcal{P} \\
\mathcal{K} &:= \star \mid \mathcal{K}\,\mathcal{O} \mid \mathcal{K}\,\mathcal{P} \mid [\lambda\mathcal{X}^\star :^a \mathcal{P}]\mathcal{K} \mid \\
&:= [\lambda\mathcal{X}^\Box :^a \mathcal{K}]\mathcal{K} \mid (\forall\mathcal{X}^\star :^a \mathcal{P})\mathcal{K} \mid (\forall\mathcal{X}^\Box :^a \mathcal{K})\mathcal{K} \\
\mathcal{E} &:= \Box \mid (\forall\mathcal{X}^\star :^a \mathcal{P})\mathcal{E} \mid (\forall\mathcal{X}^\Box :^a \mathcal{K})\mathcal{E} \\
\triangle &:= \triangle
\end{aligned}$$

This enumeration defines a postfixed successor function +1 on classes. We also define $\mathrm{Class}(t) = \mathcal{D}$ *if* $t \in \mathcal{D}$ *and* $\mathcal{D} \in \{\mathcal{O}, \mathcal{P}, \mathcal{K}, \mathcal{E}, \triangle\}$. *Otherwise,* $\mathrm{Class}(t) = \perp$.

Our typing judgments are classically written as $\Gamma \vdash t : T$, meaning that the *term* t is a proof of the proposition T under the assumptions in the *well-formed* environment Γ. *Typing rules* are those of CIC restricted to the single inductive type of natural numbers, with one exception, [CONV], based on an equality relation called *conversion* defined in Section 2.1.

Definition 2.4 (Typing judgements). *Typing rules of* $\mathrm{CC}_{\mathbb{N}}$ *are defined in Figure 1.*

2.1. Computation by conversion

Our calculus has a complex notion of computation reflecting its rich structure made of three different ingredients, the typed lambda calculus, the type nat with its weak recursor and the Presburger arithmetic.

Our typed lambda calculus comes along with the beta-rule. The eta-rule raises known technical difficulties, see [18].

The type nat is generated by the two constructors 0 and S whose typing rules are given in Figure 1. We use $\mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}$ for its recursor whose typing rule is given in Figure 1 as well. Following CIC's tradition, we separate its arguments into two groups, using parentheses for the first two, and curly brackets for the two branches. The type of the second argument $Q : \mathrm{nat} \to s \in \star$ indicates that we restrict ourselves here to the *weak* version of the recursor. Including the strong version for which $Q : \mathrm{nat} \to s \in \Box$ is discussed later. The computation rules of nat are given below:

Definition 2.5 (ι-reduction). *The (weak) ι-reduction is defined by the following rewriting system:*

$$\mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}(0, Q)\{t_0, t_S\} \quad \to_\iota t_0$$
$$\mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}(\mathrm{S}\, t, Q)\{t_0, t_S\} \to_\iota t_S\ t\ (\mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}(t, Q)\{t_0, t_S\})$$

where $t_0, t_S \in \mathcal{O}$.

These rules are going to be part of the conversion $\sim_\Gamma$. Of course, we dot not want to type-check terms at each single step of conversion, we want to type-check only the starting two terms forming the equality goal in [Conv]. But intermediate terms could then be non-typable and strong normalization be lost. Checking in the ι-rules that t_0 and t_S are objects is enough for ensuring strong normalization.

The constructors 0 and S, as well as the additional higher-order constant $+$ are *also* used to build up expressions in the algebraic world of Presburger arithmetic, in which function symbols have arities. We therefore have two different possible views of terms of type nat, either as a term of the calculus of inductive constructions, or as an algebraic term of Presburger arithmetic. We now define precisely this algebraic world and explain in detail how to extract algebraic information from arbitrary terms of $\mathrm{CC}_{\mathbb{N}}$.

[AXIOM-1] $\dfrac{}{\Gamma \vdash \star : \Box}$ [AXIOM-2] $\dfrac{}{\Gamma \vdash \Box : \triangle}$

[$\doteq$-INTRO] $\dfrac{}{\vdash \doteq : \forall(T :^u \star).\, T \to T \to \star}$

[PRODUCT] $\dfrac{\Gamma \vdash T : s_T \quad \Gamma, [x :^a T] \vdash U : s_U}{\Gamma \vdash \forall(x :^a T).\, U : s_U}$

[LAMDA] $\dfrac{\Gamma \vdash \forall(x :^a T).\, U : s \quad \Gamma, [x :^a T] \vdash u : U}{\Gamma \vdash \lambda[x :^a T]u : \forall(x :^a T).\, U}$

[WEAK] $\dfrac{\begin{array}{c}\Gamma \vdash V : s \quad \Gamma \vdash t : T \\ s \in \{\star, \Box\} \quad x \in \mathcal{X}^s - \mathrm{dom}(\Gamma)\end{array}}{\Gamma, [x :^a V] \vdash t : T}$

[VAR] $\dfrac{\Gamma \vdash T : s_x \quad x \in \mathrm{dom}(\Gamma)}{\Gamma \vdash x : x\Gamma}$

[APP] $\dfrac{\Gamma \vdash t : \forall(x :^a U).\, V \quad \Gamma \vdash u : U}{\Gamma \vdash t\, u : V\{x \mapsto u\}}$

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

[0-INTRO] $\dfrac{}{\vdash 0 : \mathrm{nat}}$ [S-INTRO] $\dfrac{}{\vdash \mathrm{S} : \mathrm{nat} \to \mathrm{nat}}$

[EQ-INTRO] $\dfrac{\begin{array}{c}\Gamma \vdash t_1 : T \quad \Gamma \vdash t_2 : T \\ \Gamma \vdash p : \forall(P : T \to \star).\, P\, t_1 \to P\, t_2\end{array}}{\Gamma \vdash \mathrm{Eq}(p) : t_1 \doteq_T t_2}$

[ι-ELIM] $\dfrac{\begin{array}{c}\Gamma \vdash t : \mathrm{nat} \quad \Gamma \vdash Q : \mathrm{nat} \to s \in \star \quad \Gamma \vdash f_0 : \mathrm{nat} \\ \Gamma \vdash f_S : \forall(n :^u \mathrm{nat}).\, Q\, n \to Q\, (\mathrm{S}\, n)\end{array}}{\Gamma \vdash \mathrm{Rec}_{\mathbb{N}}^{\mathcal{W}}(t, Q)\{f_0, f_S\} : Q\, t}$

[CONV] $\dfrac{\Gamma \vdash t : T \quad \Gamma \vdash T' : s' \quad T \sim_\Gamma T'}{\Gamma \vdash t : T'}$

[NAT] $\dfrac{}{\vdash \mathrm{nat} : \star}$ [+-INTRO] $\dfrac{}{\vdash + : \mathrm{nat} \to \mathrm{nat} \to \mathrm{nat}}$

Figure 1. Typing judgment of $\mathrm{CC}_{\mathbb{N}}$

Let $\mathcal{T}$ be the theory of *Presburger arithmetic* defined on the signature $\Sigma = \{0, \mathrm{S}(\_), \_ + \_\}$ and $\mathcal{Y}$ a set of variables distinct from $\mathcal{X}$. Note that we syntactically distinguish the algebraic symbols from the $\mathrm{CC}_\mathbb{N}$ symbols by using a different font (this shows up for the successor symbol only!).

Definition 2.6 (Algebraic terms). *The set* **Alg** *of* $\mathrm{CC}_\mathbb{N}$ algebraic terms *is the smallest subset of* $\mathcal{O}$ *s.t. i)* $\mathcal{X}^\star \subseteq$ **Alg**, *ii)* $0 \in$ **Alg**, *iii)* $\forall t \in \mathrm{CC}_\mathbb{N}. \mathrm{S}\, t \in$ **Alg**, *iv)* $\forall t, u \in \mathrm{CC}_\mathbb{N}. t + u \in$ **Alg**.

Definition 2.7 (Algebraic cap and aliens). *Given a relation* R *on* $\mathrm{CC}_\mathbb{N}$, *let* $\mathcal{R}$ *be the smallest congruence on* $\mathrm{CC}_\mathbb{N}$ *containing* R, *and* $\pi_\mathcal{R}$ *a function from* $\mathrm{CC}_\mathbb{N}$ *to* $\mathcal{Y} \cup \mathcal{X}^\star$ *such that* $t\ \mathcal{R}\ u \iff \pi_\mathcal{R}(t) = \pi_\mathcal{R}(u)$.

The algebraic cap of t modulo $\mathcal{R}$, $\mathrm{cap}_\mathcal{R}(t)$, *is defined by:*

- $\mathrm{cap}_\mathcal{R}(0) = 0$, $\mathrm{cap}_\mathcal{R}(\mathrm{S}\, u) = S(\mathrm{cap}_\mathcal{R}(u))$, $\mathrm{cap}_\mathcal{R}(u + v) = \mathrm{cap}_\mathcal{R}(u) + \mathrm{cap}_\mathcal{R}(v)$,
- *otherwise,* $\mathrm{cap}_\mathcal{R}(t) = t$ *if* $t \in \mathcal{X}^*$ *and else* $\pi_\mathcal{R}(t)$.

We call aliens *subterms of* t *abstracted by a variable in* $\mathcal{Y}$.

Observe that a term not headed by an algebraic symbol is abstracted by a variable from our new set of variables $\mathcal{Y}$ in such a way that $\mathcal{R}$-equivalent terms are abstracted by the same variable.

We can now glue things together to define *conversion*:

Definition 2.8 (Conversion relation). *The family* $\{\sim_\Gamma\}_\Gamma$ *of* Γ-conversions *is defined by the rules of Figure 2.*

This definition is technically complex.

Being a congruence, $\sim_\Gamma$ includes congruence rules. However, all these rules are not quite congruence rules since crossing a binder increases the current context Γ by the new assumption made inside the scope of the binding construct, resulting in a family of congruences. More questions are raised by the three different kinds of basic conversions.

First, $\sim_\Gamma$ includes the rules $\to_\beta$ and $\to_\iota$ of $\mathrm{CC}_\mathbb{N}$. Unlike the beta rule, $\to_\iota$ interacts with first-order rewriting, and therefore the CONV rule of Figure 2 cannot be expressed by $T \leftrightarrow^*_{\beta\iota} \sim_\Gamma \leftrightarrow^*_{\beta\iota} T'$ as one would expect.

Second, $\sim_\Gamma$ includes the relevant assumptions grabbed from the context, this is rule EQ. These assumptions must be of the form $[x :^\mathbf{r} T]$, with the appropriate annotation r, and T must be an equality assumption or otherwise *reduce* to an equality assumption. Note that we use only $\to_\beta$ here. Using $\sim_\Gamma$ recursively instead is indeed an equivalent formulation under our assumptions. Without annotations, $\mathrm{CC}_\mathbb{N}$ does not enjoy subject reduction. Generating appropriate annotations is discussed in Section 5.

Third, with rule [DED], we can also generate new assumptions by using Presburger arithmetic. This rule here uses the property that two algebraic terms are equivalent in $\sim_\Gamma$ if their caps relative to $\sim_\Gamma$ are equivalent in $\sim_\Gamma$ (the converse being false). This is so because the abstraction function $\pi_{\sim_\Gamma}$ abstracts equivalent aliens by the same variable taken from $\mathcal{Y}$. It is therefore the case that deductions on caps made in Presburger arithmetic can be lifted to deductions on arbitrary terms via the abstraction function. As a consequence, the two definitions of the abstraction function $\pi_{\sim_\Gamma}$ and of the congruence $\sim_\Gamma$ are mutually inductive: our conversion relation is defined as a least fixpoint. One may therefore wonder whether $\sim_\Gamma$ is decidable. The answer is positive as shown in Section 3.

2.2 Two simple examples

More automation - smaller proofs. We start with a simple example illustrating how the equalities extracted from a context Γ can be use to deduce new equalities in $\sim_\Gamma$.

$$[\beta\iota]\ \frac{t \leftrightarrow^*_{\beta\iota} u}{t \sim_\Gamma u}$$

$$[\text{EQ}]\ \frac{[x :^\mathbf{r} T] \in \Gamma \quad T \to^*_\beta t_1 \doteq t_2 \quad t_1, t_2 \in \mathcal{O}}{t_1 \sim_\Gamma t_2}$$

$$[\text{DED}]\ \frac{\begin{array}{c} t_1, t_2 \in \mathcal{O} \\ \{\mathrm{cap}_{\sim_\Gamma}(u_1) = \mathrm{cap}_{\sim_\Gamma}(u_2) \mid u_1 \sim_\Gamma u_2\} \vdash_\mathcal{T} \\ \mathrm{cap}_{\sim_\Gamma}(t_1) = \mathrm{cap}_{\sim_\Gamma}(t_2) \end{array}}{t_1 \sim_\Gamma t_2}$$

$$[\text{SYM}]\ \frac{t \sim_\Gamma u}{u \sim_\Gamma t} \qquad [\text{TRANS}]\ \frac{t \sim_\Gamma u \quad u \sim_\Gamma v}{t \sim_\Gamma v}$$

$$[\text{APP}]\ \frac{t_1 \sim_\Gamma t_2 \quad u_1 \sim_\Gamma u_2}{t_1\, u_1 \sim_\Gamma t_2\, u_2}$$

$$[\text{PROD}]\ \frac{T \sim_\Gamma U \quad t \sim_{\Gamma,[x:^a T]} u}{\forall(x :^b T).\, t \sim_\Gamma \forall(x :^b U).\, u}\ [b \prec a]$$

$$[\text{LAM}]\ \frac{T \sim_\Gamma U \quad t \sim_{\Gamma,[x:^a T]} u}{\lambda[x :^b T]t \sim_\Gamma \lambda[x :^b U]u}\ [b \prec a]$$

$$[\text{ELIM}]\ \frac{u \sim_\Gamma t \quad P \sim_\Gamma Q \quad t_0 \sim_\Gamma u_0 \quad t_S \sim_\Gamma u_S}{\mathrm{Rec}^\mathcal{W}_\mathbb{N}(t, P)\{t_0, t_S\} \sim_\Gamma \mathrm{Rec}^\mathcal{W}_\mathbb{N}(u, Q)\{u_0, u_S\}}$$

$$[\mathrm{CC}_\mathbb{N}\text{-EQ}]\ \frac{t \sim_\Gamma u}{\mathrm{Eq}(t) \sim_\Gamma \mathrm{Eq}(u)}$$

Figure 2. Conversion relation $\sim_\Gamma$

$$\Gamma = [x\, y\, t :^{\mathbf{u}} \text{nat}], [f :^{\mathbf{r}} \text{nat} \to \text{nat}],$$
$$[p_1 :^{\mathbf{r}} t \doteq 2], [p_2 :^{\mathbf{r}} (f\,(x+3)) \doteq x+2],$$
$$[p_3 :^{\mathbf{r}} (f\,(y+t))+2 \doteq y], [p_4 :^{\mathbf{r}} y+1 = x]$$

From p_1 and p_4 (extracted from the context by [EQ]), [DED] will deduce that $y+t \sim_\Gamma x+3$, and by congruence, $f\,(y{+}t) \sim_\Gamma f\,(x{+}3)$. Therefore, $\pi_{\sim_\Gamma}$ will abstract $f(x{+}3)$ and $f(y+t)$ by the same constant c, resulting in two new equations: $c = x+2$ and $c+2 = y$. Now, $c = x+2$, $c{+}2 = y$ and $y{+}1 = x$ form a set of unsatisfiable equations and we deduce $0 \sim_\Gamma 1$ by the DED rule: contradiction has been obtained. This shows that we can easily carry out a proof by contradiction in $\mathcal{T}$.

More typable terms. We continue with a second example showing that the new calculus can type more terms that CIC. For the sake of this example we assume that the calculus is extended by dependent lists on natural numbers. We denote by **list** (of type nat $\to \star$) the type of dependent lists and by nil (of type **list** 0) and cons (of type $\forall(n : \text{nat}).\,\mathbf{list}\,n \to \text{nat} \to \mathbf{list}\,(\mathrm{S}\,n)$) the lists constructors. We also add a weak recursor $\text{Rec}_{\mathbb{L}}^{\mathcal{W}}$ such that, given $Q : \forall(n : nat).\,\mathbf{list}\,n \to \star$, $l_0 : P\,0\,\text{nil}$ and $l_S : \forall(n : nat)(l : \mathbf{list}\,n).\,P\,n\,l \to \forall(x : nat).\,P\,(\mathrm{S}\,n)\,(\text{cons}\,n\,x\,l)$, then $\text{Rec}_{\mathbb{L}}^{\mathcal{W}}(l, Q)\{l_0, l_S\}$ has type $P\,n\,l$ for any list l of type $\mathbf{list}\,n$.

Assume now given a dependent reverse function[2] (of type $\forall(n : \text{nat}).\,\mathbf{list}\,n \to \mathbf{list}\,n$) and the list concatenation function :: (of type $\forall(n\,n' : nat), \mathbf{list}\,n \to \mathbf{list}\,n' \to \mathbf{list}\,(n+n')$). We can simply express that a list l is a palindrome: l is a palindrome if $\text{reverse}\ l \doteq l$.

Suppose now that one wants to prove that palindromes are closed under substitution of letters by palindromes. To make it easier, we will simply consider a particular case: the list $l_1 l_2 l_2 l_1$ is a palindrome if l_1 and l_2 are palindromes. The proof sketch is simple: it suffices to apply as many times as needed the lemma $\text{reverse}(ll') = \text{reverse}(l') :: \text{reverse}(l)$ $(*)$. What can be quite surprising is that Lemma $(*)$ is rejected by Coq. Indeed, if l and l' are of length n and n', it is easy to check that $\text{reverse}(ll')$ is of type $\mathbf{list}\,(n+n')$ and $\text{reverse}(l') :: \text{reverse}(l)$ of type $\mathbf{list}\,(n'{+}n)$ which are clearly not $\beta\iota$-convertible. This is not true in our system: $n+n'$ will of course be convertible to $n'+n$ and lemma $(*)$ is therefore well-formed. Proving the more general property needs of course an additional induction on natural numbers to apply lemma $(*)$ the appropriate number of times, which can of course be carried out in our system.

[2] It is somewhat surprising that this function cannot be written in the current version of Coq! On the other hand, It is easy to write it in our calculus by using the weak recursor on lists.

3. Metatheorical properties

3.1. Basic properties of term classes

Definition 3.1. *We define an order $\preceq$ on typing environments as the smallest order s.t.:*

1. $\Gamma \subseteq \Delta \implies \Gamma \preceq \Delta$,
2. $a \preceq b \implies \Gamma_1, [x :^a T], \Gamma_2 \preceq \Gamma_1, [x :^b T], \Gamma_2$.

Lemma 3.2. *Assume $\Gamma \vdash t : T$. Then,*

1. $\text{FV}(t) \cup \text{FV}(T) \subseteq \text{dom}(\Gamma)$,
2. *All subterm of t are well-formed,*
3. $\Delta \vdash t : T$ *if* $\Gamma \preceq \Delta$, *where Δ is well-formed,*

Lemma 3.3. *Assume that $\Gamma_1, [x :^a T], \Gamma_2$ is a well-formed environment. Then $\Gamma_1 \vdash T : s_x$.*

Lemma 3.4. *Let T a $\text{CC}_\mathbb{N}$ term and θ an arbitrary substitution. Then $\text{Class}(T\theta) = \text{Class}(T)$.*

Lemma 3.5. *Assume that $\Gamma \vdash t : T$. Then $\text{Class}(t) \neq \perp$, $\text{Class}(T) \neq \perp$ and $\text{Class}(T) = \text{Class}(t) + 1$.*

3.2. Properties of algebraic caps

Lemma 3.6 (Cap inversion). *Let $v = \text{cap}_\mathcal{R}(t)$.*
i) If $v \in \mathcal{X}^\star$ then $t = v$; ii) if $v = 0$ then $t = 0$; iii) if $v = S(u)$, then $t = \mathrm{S}\,u'$ with $\text{cap}_\mathcal{R}(u') = u$; iv) if $v = t_1 + t_2$ then $t = u_1 + u_2$ with $\text{cap}_\mathcal{R}(u_i) = t_i$; v) if $v \in \mathcal{Y}$ then t does not have an algebraic cap and $\pi_\mathcal{R}(t) = v$.

Lemma 3.7.
1. *Assume that $\text{cap}_\mathcal{R}(t) = \text{cap}_\mathcal{R}(u)$. Then i) $\text{APos}(t) \cup \text{APos}(u) \subseteq \text{Pos}(t) \cap \text{Pos}(u)$ and ii) $\forall p \in \text{APos}(t) \cup \text{APos}(u)$, p is either an alien position of t (resp. u), or a leaf-position in the algebraic cap of t (resp. u) and $t_{|p}\ \mathcal{R}\ u_{|p}$.*
2. *Assume that $\mathcal{R}$ and $\mathcal{R}'$ are two relations s.t. $\mathcal{R} \subseteq \mathcal{R}'$ and $\text{cap}_\mathcal{R}(t) = \text{cap}_\mathcal{R}(u)$. Then $\text{cap}'_\mathcal{R}(t) = \text{cap}'_\mathcal{R}(u)$ and $t\ \mathcal{R}'\ u$ in case $\mathcal{R}'$ is a congruence.*

We now come to a key technical property of caps. Since caps are syntactic expressions of the first-order theory $\mathcal{T}$, we use here the vocabulary of FOL with its usual meaning.

Lemma 3.8. *Let θ be a well-formed substitution of domain included in $\mathcal{X}$ such that $\mathcal{R}\theta \subseteq \mathcal{R}'$ for some congruences $\mathcal{R}$ and $\mathcal{R}'$ on terms of $\text{CC}_\mathbb{N}$. Then, for any $\mathcal{T}$-model $\mathcal{M}$ and $\mathcal{T}$-interpretation $\mathcal{I}$, there exists a $\mathcal{T}$-interpretation $\mathcal{I}_\theta$ s.t.*

$$\forall t \in \text{CC}_\mathbb{N}.\ [\![\text{cap}_\mathcal{R}(t)]\!]^{\mathcal{I}_\theta}_\mathcal{M} = [\![\text{cap}'_\mathcal{R}(t\theta)]\!]^{\mathcal{I}}_\mathcal{M}.$$

Proof. We define $\mathcal{I}_\theta$ by i) if $x \in \mathcal{Y}$, $\mathcal{I}_\theta(x) = \mathcal{I}(\pi'_{\mathcal{R}}(t\theta))$ where $t \in \pi_{\mathcal{R}}^{-1}(x)$, ii) if $x \in \mathrm{dom}(\theta)$, $\mathcal{I}_\theta(x) = [\![\mathrm{cap}'_{\mathcal{R}}(x\theta)]\!]^{\mathcal{I}}_{\mathcal{M}}$ and iii) $\mathcal{I}_\theta(x) = \mathcal{I}(x)$ otherwise.

We show first that $\mathcal{I}_\theta(x)$ does not depend upon the choice of $t \in \pi_{\mathcal{R}}^{-1}(x)$ in case i). Assuming $t, u \in \pi_{\mathcal{R}}^{-1}(x)$, then $t\ \mathcal{R}\ u$, and by assumption , $t\theta\ \mathcal{R}'\ u\theta$. Thus, $\pi'_{\mathcal{R}}(t\theta) = \pi'_{\mathcal{R}}(u\theta)$.

Let now $t \in \mathrm{CC}_{\mathbb{N}}$. We prove that $[\![\mathrm{cap}_{\mathcal{R}}(t)]\!]^{\mathcal{I}_\theta}_{\mathcal{M}} = [\![\mathrm{cap}'_{\mathcal{R}}(t\theta)]\!]^{\mathcal{I}}_{\mathcal{M}}$ by induction on the definition of $\mathrm{cap}_{\mathcal{R}}(t)$:

- If $t = t_1 + t_2$ (or $t = 0$ or $t = \mathrm{S}\,u$), the result is obtained by applying the induction hypothesis after unfolding the definition of $[\![t]\!]^{\mathcal{I}}_{\mathcal{M}}$.
- If $t = x \in \mathrm{dom}(\theta)$, $[\![\mathrm{cap}_{\mathcal{R}}(x)]\!]^{\mathcal{I}_\theta}_{\mathcal{M}} = \mathcal{I}_\theta(x) = [\![\mathrm{cap}'_{\mathcal{R}}(x\theta)]\!]^{\mathcal{I}}_{\mathcal{M}}$.
- If $t = x \in \mathcal{X} - \mathrm{dom}(\theta)$, $[\![\mathrm{cap}_{\mathcal{R}}(x)]\!]^{\mathcal{I}_\theta}_{\mathcal{M}} = \mathcal{I}_\theta(x) = \mathcal{I}(x) = [\![\mathrm{cap}'_{\mathcal{R}}(x)]\!]^{\mathcal{I}}_{\mathcal{M}} = [\![\mathrm{cap}'_{\mathcal{R}}(x\theta)]\!]^{\mathcal{I}}_{\mathcal{M}}$.
- If t does not have an algebraic cap, then $[\![\mathrm{cap}_{\mathcal{R}}(t)]\!]^{\mathcal{I}_\theta}_{\mathcal{M}} = \mathcal{I}_\theta(\pi_{\mathcal{R}}(t)) = \mathcal{I}(\pi'_{\mathcal{R}}(t\theta))$ (since $t \in \pi_{\mathcal{R}}^{-1}(\pi_{\mathcal{R}}(t))$). Since $t\theta$ does not have an algebraic cap either, $\mathcal{I}(\pi'_{\mathcal{R}}(t\theta)) = [\![\mathrm{cap}'_{\mathcal{R}}(t\theta)]\!]^{\mathcal{I}}_{\mathcal{M}}$. □

3.3 Properties of conversion

Lemma 3.9 (Sort compatibility). *1. Assume that $t \sim_\Gamma u$ with $\mathrm{Class}(t) \neq \bot$. Then $\mathrm{Class}(t) = \mathrm{Class}(u)$.*

2. $\forall s \in \mathcal{S}$, $s \sim_\Gamma t$ implies $t \to^*_\beta s$

3. If $t \sim_\Gamma t_1 \doteq t_2$ with $t_1, t_2 \in \mathcal{O}$, then $t \to^*_\beta u_1 \doteq u_2$ with $t_i \sim_\Gamma u_i$.

4. If $t \sim_\Gamma \forall(x :^a U)V$, then $t \to^*_\beta \forall(x :^a U')V'$ with $U \sim_\Gamma U'$ and $V \sim_{\Gamma,[x:^a U]} V'$.

Note that distinct sorts are not convertible to each other.

Definition 3.10. *We define a conversion relation $\sim$ on typing environments as the smallest equivalence relation s.t.:*

$$T \sim_{\Gamma_1} U \implies \Gamma_1, [x :^a T], \Gamma_2 \sim \Gamma_1, [x :^a U], \Gamma_2.$$

Lemma 3.11 (Monotonicity). *If $\Gamma \sim \Delta$, then $\sim_\Gamma = \sim_\Delta$.*

Proof. We prove $t \sim_\Delta u$ by induction on the definition of $t \sim_\Gamma u$. We consider all rules in turn:

- [EQ]. $\Gamma = \Gamma_\alpha, [x :^{\mathbf{r}} T], \Gamma_\beta$ with $T \to^*_\beta t \doteq u$ and $\mathrm{Class}(t) = \mathrm{Class}(u) = \mathcal{O}$.

  The proof is by a induction on $\Gamma \sim \Delta$. The only interesting case is when $\Gamma = \Gamma_\alpha, [x :^{\mathbf{r}} T], \Gamma_\beta$, $\Delta = \Gamma_\alpha, [x :^{\mathbf{u}} U], \Gamma_\beta$ with $T \sim_{\Gamma_1} U$. By compatibility, $U \to^*_\beta t' \doteq u'$ with $t \sim_{\Gamma_1} t'$ and $u \sim_{\Gamma_1} u'$. Since $\Gamma_1 \preceq \Delta$, $t \sim_\Delta t'$ and $u \sim_\Delta u'$. By application of the EQ-rule, $t' \sim_\Delta u'$, and thus, $t \sim_\Delta u$.

- [DED]. $E_{\sim_\Gamma} \vDash \mathrm{cap}_{\sim_\Gamma}(t) = \mathrm{cap}_{\sim_\Gamma}(u)$ where $E_{\sim_\Gamma} = \{\mathrm{cap}_{\sim_\Gamma}(u_1) = \mathrm{cap}_{\sim_\Gamma}(u_2) \mid u_1 \sim_\Gamma u_2\}$ and $t, u \in \mathcal{O}$.

  Thus, there exists $E_1, \ldots, E_n \in E_{\sim_\Gamma}$ s.t. $\mathcal{T} \vDash \forall\overline{x}.\, E_1 \wedge \cdots \wedge E_n \implies \mathrm{cap}_{\sim_\Gamma}(t) = \mathrm{cap}_{\sim_\Gamma}(u)$ $(*)$. Then, by induction hypothesis and application of Lemma 3.8, we can show that $\mathcal{T} \vDash \forall\overline{y}.\, E'_1 \wedge \cdots E'_n \implies \mathrm{cap}_{\sim_\Delta}(t) = \mathrm{cap}_{\sim_\Delta}(u)$ where $E'_i = (\mathrm{cap}_{\sim_\Delta}(w_{1,i}) = \mathrm{cap}_{\sim_\Delta}(w_{2,i}))$ with $E_i = (\mathrm{cap}_{\sim_\Gamma}(w_{1,i}) = \mathrm{cap}_{\sim_\Gamma}(w_{2,i}))$. (See the DED-case of Lemma 3.12 which is similar).

- Other cases follow from the induction hypothesis. □

Lemma 3.12 (Substitutivity). *Let $T, T', \Gamma = \Gamma_1, [w :^a W], \Gamma_2$ such that $T \sim_\Gamma T'$. Assume further that if $a = \mathbf{r}$ and $W \to^*_\beta u_1 \doteq u_2$, then $u_1 \sim_{\Gamma_1} u_2$.*
Then $T\theta \sim_\Delta T'\theta$ where $\theta = \{w \mapsto W\}$ and $\Delta = \Gamma_1, \Gamma_2\theta$.

Proof. By induction on the definition of $T \sim_\Gamma T'$:

- $[\beta\iota]$. By property of $\leftrightarrow^*_{\beta\iota}$.
- [EQ]. $\Gamma = \Gamma_\alpha, [x :^{\mathbf{r}} U], \Gamma_\beta$, with $U \to^*_\beta T \doteq T'$ and $T, T' \in \mathcal{O}$.

  If $[x :^{\mathbf{r}} U]$ is $[w :^a W]$, then $T \sim_{\Gamma_1} T'$ by assumption. Since w cannot appear free in T and T', $T\theta = T \sim_{\Gamma_1} T' = T'\theta$. By Lemma 3.2-3, $T\theta \sim_\Delta T'\theta$.

  Otherwise $x \neq w$ and $x\Delta \to^*_\beta T\theta \doteq T'\theta$. Either $x \in \mathrm{dom}(\Gamma_1)$ and in this case $w \notin \mathrm{FV}(U)$ and $x\Delta = U \to^*_\beta (T \doteq T') = (T\theta \doteq T'\theta)$; or $x \in \mathrm{dom}(\Gamma_2)$, thus $x\Delta = x\Gamma_2\theta = U\theta \to^*_\beta T\theta \doteq T'\theta$. By Lemma 3.4, $\mathrm{Class}(T\theta) = \mathrm{Class}(T) = \mathrm{Class}(T') = \mathrm{Class}(T'\theta) = \mathcal{O}$, hence $T\theta \sim_\Delta T'\theta$ by [EQ],
- [DED]. $E_{\sim_\Gamma} \vDash \mathrm{cap}_{\sim_\Gamma}(T) = \mathrm{cap}_{\sim_\Gamma}(T')$, $T, T' \in \mathcal{O}$ with $E_{\sim_\Gamma} = \{\mathrm{cap}_{\sim_\Gamma}(u_1) = \mathrm{cap}_{\sim_\Gamma}(u_2) \mid u_1 \sim_\Gamma u_2\}$.

  By definition, there exists $E_i = (\mathrm{cap}_{\sim_\Gamma}(w_{1,i}) = \mathrm{cap}_{\sim_\Gamma}(w_{2,i})) \in E_{\sim_\Gamma}$, for $i \in [1..n]$, such that $\mathcal{T} \vDash \forall\overline{x}.\, E_1 \wedge \cdots \wedge E_n \implies \mathrm{cap}_{\sim_\Gamma}(T) = \mathrm{cap}_{\sim_\Gamma}(T')$ $(*)$.

  We show first that $\mathcal{T} \vDash \forall\overline{y}.F$ $(**)$, where F is the formula $E^\theta_1 \wedge \cdots E^\theta_n \implies \mathrm{cap}_{\sim_\Delta}(T\theta) = \mathrm{cap}_{\sim_\Delta}(T'\theta)$, with $E^\theta_i = (\mathrm{cap}_{\sim_\Delta}(w_{1,i}\theta) = \mathrm{cap}_{\sim_\Delta}(w_{2,i}\theta))$. Let $\mathcal{M}$ be a $\mathcal{T}$-model and $\mathcal{I}$ a $\mathcal{T}$-interpretation. If $[\![E^\theta_1 \wedge \cdots \wedge E^\theta_n]\!]^{\mathcal{I}}_{\mathcal{M}} = \bot$, then $[\![F]\!]^{\mathcal{I}}_{\mathcal{M}} = \top$. Otherwise, assume that $[\![\mathrm{cap}_\Delta(T\theta)]\!]^{\mathcal{I}}_{\mathcal{M}} \neq [\![\mathrm{cap}_\Delta(T'\theta)]\!]^{\mathcal{I}}_{\mathcal{M}}$. By induction hypothesis $\sim_\Gamma \subseteq \sim_\Delta\theta$, thus by Lemma 3.8 there exists $\mathcal{I}_\theta$ s.t. $[\![E_1 \wedge \cdots \wedge E_n]\!]^{\mathcal{I}_\theta}_{\mathcal{M}} = \top$ and $[\![\mathrm{cap}_{\sim_\Gamma}(T)]\!]^{\mathcal{I}_\theta}_{\mathcal{M}} \neq [\![\mathrm{cap}_{\sim_\Gamma}(T')]\!]^{\mathcal{I}_\theta}_{\mathcal{M}}$, contradicting $(*)$. Therefore $[\![\mathrm{cap}_\Delta(T\theta)]\!]^{\mathcal{I}}_{\mathcal{M}} = [\![\mathrm{cap}_\Delta(T'\theta)]\!]^{\mathcal{I}}_{\mathcal{M}}$, and $[\![F]\!]^{\mathcal{I}}_{\mathcal{M}} = \top$, ending the proof of $(**)$.

  By induction hypothesis, $\forall i.\, w_{1,i}\theta \sim_\Delta w_{2,i}\theta$. Hence, $\forall i.\, E^\theta_i \in E_{\sim_\Delta} = \{\mathrm{cap}_{\sim_\Delta}(u_1) = \mathrm{cap}_{\sim_\Delta}(u_2) \mid u_1 \sim_\Delta u_2\}$. Therefore $\mathcal{T}, E_{\sim_\Delta} \vDash \mathrm{cap}_{\sim_\Delta}(T\theta) = \mathrm{cap}_{\sim_\Delta}(T'\theta)$ by $(**)$. Hence $T\theta \sim_\Delta T'\theta$ by DED.

- Other cases follow from the induction hypothesis. □

As usual, the substitutivity lemma is used in the proof of subject reduction (for beta) to come later. Because it requires a specific typing property for the equality assumptions annotated by **r**, we need to ensure this property in the application case of the coming inversion lemma used in combination with substitutivity in the subject reduction proof. This is indeed the origin of the similar condition appearing in the typing rule [APP].

Lemma 3.13 (Inversion). *Assume that* $\Gamma \vdash t : T$.

1. *if* $t \in \mathcal{X}^s$, *then* $\Gamma \vdash T : s$ *and* $x\Gamma \sim_\Gamma T$
2. *if* $t \in \mathcal{S}$, *then* $t = \star$ *and* $T = \Box$ *or* $t = \Box$ *and* $T = \Delta$;
3. *if* $t = 0$ *(resp* $t = \mathrm{S}$, $t = +$*), then* $T \sim_\Gamma$ nat *(resp.* $T \sim_\Gamma$ nat $\to$ nat, $T \sim_\Gamma$ nat $\to$ nat $\to$ nat*)*
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* $a = \mathbf{r}$ *and* $V \leftrightarrow^*_\beta t_1 \doteq t_2$ *with* $t_1, t_2 \in \mathcal{O}$, *then* $t_1 \sim_\Gamma t_2$
5. *if* $t = \forall(x : aU).\, V$, *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* $t = \lambda[x : aU]v$, *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)* $\forall(x : aU).\, V \sim_\Gamma T$ *where* s_V *is the sort of* x.

Lemma 3.14 (Type unicity). *If* $\Gamma \vdash t : T_1$ *and* $\Gamma \vdash t : T_2$, *then* $T_1 \sim_\Gamma T_2$.

Proof. By structural induction on t and Lemma 3.13. □

3.4. Conversion as rewriting

We now turn conversion into a rewriting relation in order to prove that our system is logically coherent by analyzing a proof in normal form of $\forall(x :^{\mathbf{u}} \star).\, x$. The notion of a normal proof is of course more complicated than in CIC, since we must account for the congruence $\sim_\Gamma$ associated to an arbitrary context Γ. The difficulty is that the set of equalities assumed in a given environment Γ together with the axioms of the theory $\mathcal{T}$ may be incoherent, making all first-order terms equal in $\sim_\Gamma$ which could break strong normalization of our rewriting relation. Solving this problem is possible because coherence is decidable.

Definition 3.15 (Coherent environment). *A typing environment* Γ *is* $\mathcal{T}$-coherent *if there exist two terms* $t, u \in \mathcal{O}$ *s.t.* $\neg(t \sim_\Gamma u)$.

Lemma 3.16. *If* Γ *is* $\mathcal{T}$*-cohenrent then* $\neg(0 \sim_\Gamma \mathrm{S}\, t)$ *for any term* t.

Definition 3.17 (Weak conversion). *We inductively define a family of* weak conversion relations $\{\cong_\Gamma\}_\Gamma$ *on* $\mathcal{O}^2$ *defined as:* $t \cong_\Gamma u$ *iff* $\mathrm{Eq}(\Gamma) \vdash \mathrm{cap}_\emptyset(t) = \mathrm{cap}_\emptyset(u)$, *where* $\mathrm{Eq}(\Gamma) = \{\mathrm{cap}_\emptyset(w_1) = \mathrm{cap}_\emptyset(w_2) \mid [x :^{\mathbf{r}} w_1 \doteq w_2] \in \Gamma\}$.

Definition 3.18. *We inductively define a family* $\{\to_\Gamma\}_\Gamma$ *of rewriting relations modulo weak-conversion as the smallest rewriting relations satisfying the rules of Figure 3.*

The first rule shows that rewriting is modulo weak conversion in a coherent environment. The second equates all object terms when the environment is incoherent, replacing them by the new constant $\bullet$. The other are as expected.

$$[\text{RW-MOD}]\ \frac{\Gamma \text{ is } \mathcal{T}\text{-coherent} \quad t \cong_\Gamma t' \to_\Gamma u' \cong_\Gamma u}{t \to_\Gamma u}$$

$$[\text{RW-}\bullet]\ \frac{\Gamma \text{ is } \mathcal{T}\text{-incoherent} \quad t \in \mathcal{O} \quad t \neq \bullet}{t \to_\Gamma \bullet}$$

$$[\text{RW-}\beta]\ \frac{t \to_\beta u}{t \to_\Gamma u} \qquad [\text{RW-}\iota]\ \frac{t \to_\iota u}{t \to_\Gamma u}$$

$$[\text{RW-FORWARD}]\ \frac{t \to_\Delta u \quad \Gamma \to_\beta \Delta}{t \to_\Gamma u}$$

$$[\text{W-}\forall]\ \frac{t \to_{\Gamma,[x:^a T]} u}{\forall(x :^b T).\, t \to_\Gamma \forall(x :^b T).\, u}\ [b \preceq a]$$

$$[\text{W-}\lambda]\ \frac{t \to_{\Gamma,[x:^a T]} u}{\lambda[x :^b T].\, t \to_\Gamma \lambda[x :^b T].\, u}\ [b \preceq a]$$

Figure 3. Conversion as a rewriting system

Lemma 3.19. *The rewriting relation* $\to_\Gamma$ *is confluent.*

Proof. This proof is classically done by showing commutation lemmas. □

Lemma 3.20.
1. *If* $t \sim_\Gamma u$ *then* $t \leftrightarrow^*_\Gamma u$.
2. *If* $t \leftrightarrow^*_\Gamma u$ *with* $\bullet \notin t$ *and* $\bullet \notin u$ *then* $t \sim_\Gamma u$.

Lemma 3.21. *If* $\Gamma \vdash t : T$ *with* Γ $\mathcal{T}$*-coherent and* $t \cong_\Gamma u$, *then* $\Gamma \vdash u : T$.

Lemma 3.22 (Subject reduction). *If* $\Gamma \vdash t : T$ *and* $t \to_\Gamma u$ *with* $\bullet \notin u$, *then* $\Gamma \vdash u : T$.

Proof. The proof is standard, by induction on the type derivation of the left-hand side. The interesting case is that when a beta-reduction applies to the top of a term of the

form $(\lambda[x :^a U]v)\ w$ and the typing rule used is [APP]. The inversion Lemma 3.13 (case 4) then provides us with the property needed by the substitutivity lemma 3.12. □

Lemma 3.23. *The rewriting relation $\to_\Gamma$ is strongly normalizing for well formed terms.*

Proof. The proof is a direct application of proof irrelevance [4], because $\sim_\Gamma$ is a congruence generated by equalities between object terms, apart from beta-reduction. What makes this true is that $\text{Rec}_{\mathbb{N}}^{\mathcal{W}}$ is a weak recursor, working at the object level. Including strong elimination rules invalidates this argument. □

We finally conclude that $\text{CC}_{\mathbb{N}}$ is coherent:

Theorem 3.1. *There is no proof of $\vdash t : \forall(x :^{\mathbf{u}} \star).\, x$.*

Proof. If $\Gamma \vdash t : \forall(x :^{\mathbf{u}} \star).\, x$ where t is $\to_\Gamma$ normal and minimal for the subterm order. By typing constraints, t is either an application $u\,v$ or an abstraction.

If $t = u\,v$, t is necessarily a symbol $c \in \{\text{S}, +\}$, since t has no free variables and is normal. Therefore, the type of t is either nat or nat $\to$ nat, which is not convertible to a product type. So this case is impossible.

Otherwise, $t = \lambda[x :^{\mathbf{u}} \star]w$. Applying t to the proposition $0 \doteq 1$ yields a proof of $0 = 1$ is the empty environment, which is impossible by consistency of $\mathcal{T}$. □

4. Deciding type checking in $\text{CC}_{\mathbb{N}}$

Decidability of type checking needs two ingredients. First-of-all, eliminating [CONV], which is non-structural, by incorporating it to [APP]. This is classical, and it is easy to prove decidability of the transformed set of rules for type-checking, assuming $\sim_\Gamma$ is decidable. We therefore concentrate now on the proof of decidability of conversion.

Unfortunately, we cannot use the rewrite system $\to_\Gamma$ for that purpose since the first two rules use the $\mathcal{T}$-coherence of Γ as a perequisite. Instead we will use a saturation based algorithm. The method ressembles very much the one used for combining first-order decision procedures operating on disjoint alphabets [14, 1]. Basic ingredients are: purification of formulae (here equations) by abstracting aliens by new variables; deriving new equalities among variables by using the appropriate decision procedure for pure formulae; propagating these new equalities to the other formulae. It is easy to see that the method terminates, since the set of variables is entirely determined by the first phase and more and more variables become identified.

In our case, there are two different vocabularies, that of nat and the lambda-calculus one. We will purify terms (for convenience, more than needed) by associating a variable to each subterm. There is a difficulty which complicates

$$\frac{[A, N]}{[A, N\{c_1 \to_C c_2\} \cup \{c_1 \to_C N(c_2)\}]}$$
where $A_{|C} \vdash_{\mathcal{T}} c_1 = c_2 \quad (c_1 \to_C N(c_2)) \notin N$, $c_1, c_2 \in \text{Var}(A) \cup \text{Var}(N) \quad c_1 \succ_{\mathcal{C}} N(c_2)$

$$\frac{[A, N \supseteq \{c_1 \to_C t_1, c_2 \to_C t_2\}]}{[A \cup \{c_1 =_C c_2\}, N]}$$
where $c_1 =_C c_2 \notin A \quad A_{|C}$ is $\mathcal{T}$-satisfiable, $N \vdash \{t_1 \equiv_C t_2\} \Rightarrow^* \top$

$$\frac{[A, N \supseteq \{c_1 \to_C t_1, c_2 \to_C t_2\}]}{[A \cup \{c_1 =_C c_2\}, N]}$$
where $c_1 =_C c_2 \notin A \quad A_{|C}$ is $\mathcal{T}$-unsatisfiable, $||t_1|| \leftrightarrow^*_\beta ||t_2||$

$$\frac{[A, N \uplus \{c \to_C \text{Rec}_{\mathbb{N}}^{\mathcal{W}}(c_\iota, c_Q)\{c_0, c_S\}\}]}{[A', N']}$$
where $A_{|C}$ is satisfiable, $A_{|C} \wedge c_\iota \neq 0$ is unsatisfiable, $(A', N') = \text{purify}(\{c =_C c_0\}, A, N)$

$$\frac{[A, N \uplus \{c \to_C \text{Rec}_{\mathbb{N}}^{\mathcal{W}}(c_\iota, c_Q)\{c_0, c_S\}\}]}{[A', N']}$$
where $A_{|C}$ is satisfiable, $A_{|C} \wedge c_\iota \neq S(v)$ is unsatisfiable with v fresh, $E = \{c_\iota =_C \text{S}\,v, c =_C c_S\,v\,\text{Rec}_{\mathbb{N}}^{\mathcal{W}}(v, c_Q)\{c_0, c_S\}\}$, $(A', N') = \text{purify}(E, A, N)$

$$\frac{[A, N \supseteq \{c \to_C (d_1 \doteq d_2)\}]}{[A \cup \{d_1 =_C d_2\}, N]}$$
where $d_1 =_C d_2 \notin A$

$$\frac{[A, N \uplus \{c \to_C A\,B, A \to_C \lambda[x :^a T]D\}]}{[A\{x \mapsto B\}, N\{x \mapsto B\} \cup \{c \to_C B\}]}$$

$$\frac{[A, N]}{\bot} \quad [A_{|\emptyset} \text{ is } \mathcal{T}\text{-unsatisfiable}]$$

Figure 4. Saturation

the derivation of new equalities: the binders which may increase the current set of available equalities when crossed. Besides, new (possibly heterogeneous) terms may be built by beta reductions, generating new variables. Our termination argument will therefore be less straightforward.

Let $\mathcal{C}$ a new set of variables totally ordered by $\succ_{\mathcal{C}}$. A annotated equation (resp. annotated inequation) will be any triple $t =_C u$ (resp. $t \neq_C u$) with $t, u \in \mathrm{CC}_{\mathbb{N}}(\mathcal{X} \cup \mathcal{C})$ and C a sequence over $\mathcal{C}$. An annotated literal is either an annotated equation or annotated inequation written $t \bowtie_C u$.

If E is a set of annotated literals, we write $E_{|C}$ for the set $\{t \bowtie_{C'} u \in E \mid C' \text{ is a prefix of } C\}$.

A term is pure if it is algebraic (without any alien) or if none of its subterms has an algebraic cap.

We first describe (omitting the straightforward rules) *purification* at Figure 5. Purification aims at describing literals by two sets of pure equations belonging either to A, made of pure algebraic literals, or to N, made of equations written $c \to t$ where $c \in \mathcal{C}$ and t is a pure non-algebraic term of depth one.

Definition 4.1. *We say that $[A, N]$ reduce to $[A', N']$ (written $[A, N] \Rightarrow [A', N']$) if $[A', N']$ can be derived from $[A, N]$ by one of the rules of Figure 4.*

Lemma 4.2. *Let Γ a well formed environment, t and u two well formed terms under Γ. Then $t \sim_\Gamma u$ if and only if*

$$\mathrm{purify}(\mathrm{Eq}(\Gamma) \cup \{t \neq u\}, \emptyset, \emptyset) \Rightarrow^* \bot$$

where $\mathrm{Eq}(\Gamma) = \{[x :^{\mathbf{r}} t \doteq u] \in \Gamma\downarrow_\beta \mid t, u \in \mathcal{O}\}$.

Proof. The proof is in three steps: i) the rules preserve the $\mathcal{T}$-models; ii) the rules are terminating: the problem is interpreted by a pair made of the multiset of terms in N and the number of different classes of variables in A with respect to equality. Pairs are compared in the ordering $(\to_{\Gamma mul}, >_{\mathbb{N}})_{lex}$; iii) if $\neg([A, N] \Rightarrow \bot)$ then there is a $\mathcal{T}$-model showing that the initial goal is satisfiable. □

5 Conclusion and discussion

$\mathrm{CC}_{\mathbb{N}}$ is an extension of CIC (restricted to the weak elimination rules of the inductive type nat) by a fragment of Presburger arithmetic (without the strict order $<$ on nat) in which conversion incorporates Presburger arithmetic, beta-reduction and higher-order primitive recursion into a single mechanism. We now discuss in more details how this can be generalized to full CIC, how this can be used in practice, how useful that is, and whether the obtained kernel is trustable.

$$\frac{[E \uplus \{c_1 \bowtie_C c_2\}, A, N]}{[E, A \cup \{c_1 \bowtie_C c_2\}, N]}$$

$$\frac{[E \uplus \{t \bowtie_C C_{\mathcal{A}}[u_1, \ldots, u_n]\}, A, N] \quad c, c_1, \ldots, c_n \text{ fresh}}{[E \cup \{t \bowtie_C c, u_i =_C c_i\}, A \cup \{c =_C C_{\mathcal{A}}[c_1, \ldots, c_n]\}, N]}$$

where $\mid C_{\mathcal{A}}$ is an algebraic context

$$\frac{[E \uplus \{t \bowtie_C u_1\, u_2\}, A, N] \quad u_1\, u_2 \notin \textbf{Alg} \quad c, c_1, c_2 \text{ fresh}}{[E \cup \{t \bowtie_C c, c_i =_C u_i\}, A, N \cup \{c \to_C c_1\, c_2\}]}$$

$$\frac{[E \uplus \{t \bowtie_C \lambda[x :^a T]t\}, A, N] \quad c, c_T, c_t \text{ fresh}}{[E \cup E', A, N \cup \{c \to_C \lambda[x :^a c_T]c_t\}]}$$

where $\mid C' = C, c_T \quad E' = \{t \bowtie_C c, c_T =_C T, c_t =_{C'} t\}$

Figure 5. Purification

Extension to CIC. Building decision procedures in a type-theoretic framework is not that easy. The main difficulty lies in the adequate definition of the congruence $\sim_\Gamma$. Once the definition is obtained, carrying out the technical development is easy in the case of the pure Calculus of Constructions (the congruence becomes quite simpler in this case), difficult in the present case of $\mathrm{CC}_{\mathbb{N}}$ (because of the presence of the weak recursor for nat), no more difficult when other decidable theories are introduced such as lists with their associated weak recursor, but much harder when including strong elimination rules which interact with the first-order theories. In this case, it is necessary to block the congruence below the strong recursor in order to avoid lifting an incoherence from the object level to the predicate level, which would immediately yield paradoxes.

Arbitrary decision procedures. So far, we have considered only decidable equality theories. But is is well-known that a decidable non-equality theory can always be transformed into a decidable equality theory over the type Bool of truth values equipped with its usual operations. This is so because of the decidability assumption.

Relevance. Our second example shows very clearly the expressivity of our calculus with respect to CIC. However, what is done here by a typing rule could be done alternatively in CIC by a tactic. Besides, if one wants to avoid building a proof term which can be quite large and slow down the type-checker, it is possible to prove the tactic and then use a reflection mechanism in order to avoid type-checking the proof each time the tactic is called. In both cases, however, the user must call the tactic explicitly. In our approach, this is completely transparent, and would re-

$$\frac{N \vdash E \uplus \{d \equiv_C d\}}{N \vdash E} \qquad \frac{N \vdash \emptyset}{\top}$$

$$\frac{N \vdash E \uplus \{d_1 \equiv_C d_2\}}{N \vdash E \cup \{N(d_1) \equiv_C N(d_2)\}}$$
where $\mid N(d_1) \neq d_1$ or $N(d_2) \neq d_2$

$$\frac{N \vdash E \uplus \{A\,B \equiv_C A'\,B'\}}{N \vdash E \cup \{A \equiv_C B, A' \equiv_C B'\}}$$

$$\frac{N \vdash E \uplus \{\lambda[x :^a T_1]D_1 \equiv_C \lambda[x :^a T_2]D_2\}}{N \vdash E\{x \mapsto y\} \cup \{T_1 \equiv_C T_2, D_1 \equiv_{C,T_1} D_2\}}$$

$$\frac{N \vdash E \uplus \{\forall[x :^a T_1]D_1 \equiv_C \forall[x :^a T_2]D_2\}}{N \vdash E\{x \mapsto y\} \cup \{T_1 \equiv_C T_2, D_1 =_{C,T_1} D_2\}}$$

Figure 6. Propagation

main transparent in case of a succession of uses of the decision procedure separated by eliminations, since conversion incorporates both, or in case of different decision procedures called successively.

Trusting the kernel. Decision procedures require complex coding. It took a lot of time to get a correct tactic for Presburger arithmetic in Coq. Including a tactic into the kernel of the system is therefore unrealistic, unless it is itself proved correct with a trustable proof assistant. On the other hand, most decision procedures can provide a *certificate* that is quite compact and can be verified by a *certificate-checker* which is usually small, and easy to write and read, and is therefore a trustable piece of code. The reason is that the procedure *searches* for a proof, while the certificate-checker *verifies* that the certificate is correct. A certificate checker looks indeed like a proof-checker. It is then easy to modify the conversion rule so as to output a certificate each time a decision procedure is used. The kernel of $CC_{\mathbb{N}}$ therefore includes a certificate-checker for Presburger arithmetic. In case of CCIC with several decision procedures, the kernel would include one proof-checker for each decision procedure. Besides, the process is incremental: the procedures and the associated proof-checkers can be included one by one, because decision procedures for different inductive types operate on disjoint vocabularies, hence can be combined [14, 1].

An implementation of CCIC has started and should be available soon as a prototype in a version without certificate generation and checking.

References

[1] F. Baader and K. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. In D. Kapur, editor, *Proc. 11th Int. Conf. on Automated Deduction, Saratoga Springs, NY, LNAI 607*, 1992.

[2] H. Barendregt. *Lambda calculi with types*, volume 2 of *Handbook of logic in computer science*. Oxford University Press, 1992.

[3] B. Barras. *Auto-validation d'un système de preuves avec familles inductives*. PhD thesis, University of Paris VII, 1999.

[4] G. Barthe". The relevance of proof irrelevance. In *Proc. 24th Int. Coll. on Automata, Languages and Programming, LNCS 1443*, LNCS, 1998.

[5] F. Blanqui. Definitions by rewriting in the calculus of constructions. *Mathematical Structures in Computer Science*, 15(1):37–92, 2005. Journal version of LICS'01.

[6] F. Blanqui. Inductive types in the calculus of algebraic constructions. *Fundamenta Informaticae*, 65(1-2):61–86, 2005. Journal version of TLCA'03.

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

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

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

[10] T. Coquand and C. Paulin-Mohring. Inductively defined types. In Martin-Löf and G. Mints, editors, *Colog'-88, International Conference on Computer Logic*, volume 417 of *LNCS*, pages 50–66. Springer-Verlag, 1990.

[11] P. Corbineau. *Démonstration automatique en Théorie des Types*. PhD thesis, University of Paris IX, 2005.

[12] E. Giménez. Structural recursive definitions in type theory. In *Proceedings of ICALP'98*, volume 1443 of *LNCS*, pages 397–408, July 1998.

[13] G. Gonthier. The four color theorem in coq. In *TYPES 2004 International Workshop*, 2004.

[14] M. Schmidt-Schauß. Unification in a combination of arbitrary disjoint equational theories. *J. Symbolic Computation*, 8:51–99, 1989. Special issue on Unification.

[15] N. Shankar. Little engines of proof. In G. Plotkin, editor, *Proceedings of the Seventeenth Annual IEEE Symp. on Logic in Computer Science, LICS 2002*. IEEE Computer Society Press, 2002. Invited Talk.

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

[17] M. Stehr. The Open Calculus of Constructions: An equational type theory with dependent types for programming, specification, and interactive theorem proving (part I and II). *To appear in Fundamenta Informaticae*, 2007.

[18] B. Werner. *Une Théorie des Constructions Inductives*. PhD thesis, University of Paris VII, 1994.