Proof pearl: abella formalization of lambda-calculus cube property

Beniamino Accattoli

Proof Pearl: Abella Formalization of λ-Calculus Cube Property

Beniamino Accattoli

INRIA and LIX (École Polytechnique) - Palaiseau, France
Carnegie Mellon University - Pittsburgh, PA, USA

Abstract. In 1994 Gerard Huet formalized in Coq the cube property of λ-calculus residuals. His development is based on a clever idea, a beautiful inductive definition of residuals. However, in his formalization there is a lot of noise concerning the representation of terms with binders. We re-interpret his work in Abella, a recent proof assistant based on higher-order abstract syntax and provided with a nominal quantifier. By revisiting Huet's approach and exploiting the features of Abella, we get a strikingly compact and natural development, which makes Huet's idea really shine.

1 Introduction

The confluence or Church-Rosser theorem of λ-calculus has been formalized in several proof assistants, and it is probably the theorem with the highest number of formalized proofs [26,31,21,32,27,30,17,5,16]. In [17] Huet formalizes in Coq also a deeper result, the cube property of λ-calculus residuals (due to Jean-Jacques Lévy [20,4]). This paper presents a new, simple formalization of this result, developed in Abella [9,8], a recent proof-assistant based on higher-order abstract syntax (HOAS) [7,23] and provided with a nominal quantifier [13,12,11,24,10,2].

Residual systems are a standard tool in rewriting [20,18,19,22,15,3,34]. In particular, they are at the basis of the advanced rewriting theory of λ-calculus and orthogonal rewriting systems (standardization, neededness, Lévy's families and optimality, inside-out reductions, see [20,18,19,3,34]). Roughly, one first introduces a mechanism to track redexes along reductions (typically using positions or underlinings), so that it is possible to say which are the residuals of a redex r after another redex. Then, one shows that any given span $u_2 \leftarrow t \rightarrow u_1$ can be closed by simply reducing on both sides the residuals of one redex after the other. The idea is that residuals refine confluence providing a *minimal* closure of confluence diagrams. The abstract theory of residual systems—which is independent from λ-calculus—is based on three axioms, the cube property plus two other minor axioms (see [34], Chapter 8.7). The development in this paper essentially proves that λ-calculus admits a residual system, but we will not enter into the details of the abstract theory.

C. Hawblitzel and D. Miller (Eds.): CPP 2012, LNCS 7679, pp. 173–187, 2012.

A delicate point is how to define residuals for a given calculus. In [17] Huet presents an elegant and compact solution for λ-calculus: he uses a simple ternary relation over terms with underlinings, defined by induction on the structure of the first term. However, Huet represents (marked) λ-terms using de Bruijn indexes, and a relevant part of his development deals with the properties of indexes, substitution and lifting.

Initially, we repeated Huet's development to see how much Abella—being a proof assistant based on HOAS—could help in simplifying Huet's work. All the troubles about indexes, lifting and substitution disappear, this was expected. However, along the way we realized that other simplifications were possible (the first two are independent from Abella):

1. *Marks*: Huet underlines applications, which requires to introduce a notion of well-formed term (*regular terms* in [17]) and to show that various operations preserve well-formedness. According to common practice we rather mark redexes (as in [3], for instance), so that any marked term is well-formed, and some lemmas disappear.
2. *Rewriting*: by analyzing inductions and dependencies between lemmas we simplify the statements and the number of lemmas required to prove the cube property. In particular, Huet recognizes the so-called *prism property* as more fundamental than the cube property, but we show that the direct proof of the cube property is not harder than the proof of the prism property. This also agrees with the clean abstract theory in [34] (which did not exist at the time of [17]), where there are examples of residual systems enjoying the cube property but not the prism property. Actually, we show that for λ-calculus it is possible to prove both properties with the same induction.
3. *Contexts*: in HOAS-based proof assistants α-equivalence and substitution are primitive notions, but induction usually requires to consider predicates inside contexts of local assumptions (called worlds in Twelf [28], schemas in Beluga [29,6]) and prove properties about them. With respect to the untyped λ-calculus these contexts are artifacts, since they do not belong to the informal theory. The nominal quantifier ∇ (nabla) of Abella, not available in other HOAS settings, allows to formalize the untyped λ-calculus circumventing the use of contexts.

The final result is quite striking: we formalize a property subsuming both the cube and prism properties using only two definitions and one auxiliary lemma. Moreover, the development follows exactly the informal, pen-and-paper reasoning: there is no need to care about indexes, α-equivalence, substitution or contexts.

The next section contains an introduction to residuals and the way Huet represents them. In Section 3 we present the formal development, also explaining the representation of λ-terms. Section 4 discusses some variations over our development.

The sources of the development can be found on-line [1].

2 The Diamond and Cube Properties, Informally

The diamond property. A rewriting system $(S, \to)$ is **confluent** when for any $s \in S$:

$$s_2 \,{}^*\!\leftarrow s \to^* s_1 \text{ implies } \exists t \text{ s.t. } s_1 \to^* t \,{}^*\!\leftarrow s_2$$

The corresponding diagram is in Fig. 1.a (solid arrows denote the reductions in the hypothesis, dashed arrows denote the reductions in the conclusion). A stronger notion is the **diamond property** (Fig. 1.b):

$$s_2 \leftarrow s \to s_1 \text{ implies } \exists t \text{ s.t. } s_1 \to t \leftarrow s_2$$

The diamond property implies confluence, but not the converse. Unfortunately, β-reduction does not enjoy the diamond property (because of duplications/erasures). The standard technique (due to Tait and Martin-Löf) for proving confluence of β-reduction is to use a parallel reduction. The idea is to extend β-reduction to a reduction $\Rightarrow$ so that:

1. $\Rightarrow$ enjoys the diamond property, and thus confluence;
2. confluence of $\Rightarrow$ implies confluence of $\to_\beta$.

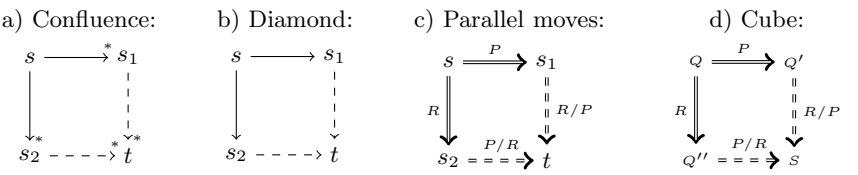

Fig. 1. Diagrams

The parallel reduction $\Rightarrow$ is defined as follows[1]:

$$\frac{}{x \Rightarrow x}\ \Rightarrow\text{-var} \qquad \frac{t \Rightarrow t'}{\lambda x.t \Rightarrow \lambda x.t'}\ \Rightarrow\text{-}\lambda$$

$$\frac{t \Rightarrow t' \qquad u \Rightarrow u'}{t\ u \Rightarrow t'u'}\ \Rightarrow\text{-@} \qquad \frac{t \Rightarrow t' \qquad u \Rightarrow u'}{(\lambda x.t)\ u \Rightarrow t'\{x/u'\}}\ \Rightarrow\text{-}\beta$$

The proof of confluence for $\to_\beta$ is in two parts. The first part is to prove that $\Rightarrow$ has the diamond property. The second part is to deduce confluence of $\to_\beta$

[1] There is a subtlety: sometimes (in [34] for instance) *parallel reduction* denotes the reduction which reduces in parallel disjoint redexes only, while the reduction presented here (which may reduce nested redexes using rule $\Rightarrow$-β, for instance $(\lambda x.x)((\lambda y.y)z) \Rightarrow z$) is called *simultaneous* or *multi-step* reduction. Often, however, the two concepts are not distinguished and the simultaneous reduction is called *parallel* (in [33] for instance). We follow this second tradition.

from the diamond property for $\Rightarrow$. Clearly, one has $\rightarrow_\beta \subseteq \Rightarrow \subseteq \rightarrow_\beta^*$, which implies $\rightarrow_\beta^* \subseteq \Rightarrow^* \subseteq \rightarrow_\beta^*$, *i.e.* $\rightarrow_\beta^* = \Rightarrow^*$. A straightforward induction shows that the diamond property of $\Rightarrow$ implies the diamond property of $\Rightarrow^*$, which is nothing but confluence of $\rightarrow_\beta$.

The first part of the confluence proof uses structural induction over terms with binders, and it is related to the cube property for residuals, so we shall focus on it. The second part is based on the so-called *strip lemma*, it mostly involves first-order reasoning, and its formalization in Abella does not differ much from the other developments of confluence in the literature (for instance [17,27]), and thus it will be omitted[2].

The proof of the diamond property requires a substitution lemma:

Lemma 1 (substitutivity of $\Rightarrow$). *If $t \Rightarrow t'$ and $u \Rightarrow u'$ then $t\{x/u\} \Rightarrow t'\{x/u'\}$.*

Proof. By induction on $t \Rightarrow t'$. Two base cases (rule $\Rightarrow$`-var`): $x \Rightarrow x$, which gives $x\{x/u\} = u \Rightarrow u' = x\{x/u'\}$, and $y \Rightarrow y$ (with $y \neq x$) for which $y\{x/u\} = y \Rightarrow y = y\{x/u'\}$. The cases $\Rightarrow$-λ and $\Rightarrow$-@ follow immediately from the *i.h.*. The case $\Rightarrow$-β: if $t = (\lambda y.s)v$ and $t' = s'\{y/v'\}$ then by *i.h.* $s\{x/u\} \Rightarrow s'\{x/u'\}$ and $v\{x/u\} \Rightarrow v'\{x/u'\}$, then by $\Rightarrow$-β we get $r = (\lambda y.s\{x/u\})v\{x/u\} \Rightarrow s'\{x/u'\}\{y/v'\{x/u'\}\} = r'$. We conclude, since $r = t\{x/u\}$ and $r' = t'\{x/u'\}$

Then (both proofs are formalized in Section 3, Figure 5, page 182):

Theorem 1 (diamond property of $\Rightarrow$). *$s_2 \Leftarrow s \Rightarrow s_1$ implies $\exists t$ s.t. $s_1 \Rightarrow t \Leftarrow s_2$.*

Proof. By induction on $s \Rightarrow s_1$ and case analysis of $s \Rightarrow s_2$, using Lemma 1. If $s = x \Rightarrow x = s_1$ then s_2 can only be x and there is nothing to prove. The $\Rightarrow$-λ case follows by the *i.h.*. Both $\Rightarrow$-@ and $\Rightarrow$-β cases have two subcases, corresponding to $s \Rightarrow s_2$ being $\Rightarrow$-@ or $\Rightarrow$-β. In every subcase one has to use the *i.h.*, and whenever one of the hypothesis is $\Rightarrow$-β it is necessary to apply Lemma 1.

Residuals. The diamond property of $\Rightarrow$ can be strengthened with information about which redexes are reduced. First, one needs to introduce a mechanism for tracing redexes through reductions.

Let us stress that we need to trace *sets* of redexes. Indeed, consider the following reductions, where $I = \lambda z.z$:

$$(\lambda x.xx)\ (II) \rightarrow_\beta (II)(II) \qquad\qquad (\lambda x.y)\ (II) \rightarrow_\beta y$$

The redex II may be duplicated, getting two residuals in the reduct, or erased, having no residual. By the way, this is also the reason why β-reduction does not enjoy the diamond property.

[2] The second part has nonetheless been formalized in Abella, it can be found in [1].

Consider a term s and two sets of redexes R and P in s. If $s \Rightarrow v$ by reducing the redexes in P then there must be a way of describing what is left in v of the redexes in R after the reduction of P, *i.e.* of describing *the set of residuals of* R *after* P, noted R/P.

Now, assume to know how to define and trace residuals, and to have a refined notion of reduction $s \overset{R}{\Rightarrow} v$, which reduces the set R of redexes in s. The diamond property enriched with residuals—usually called the *parallel moves* property—is in Fig. 1.c. The refinement essentially says that residuals allow to close the diagram in a *minimal* way[3].

The point is now how to define residuals and their reduction. Huet uses a brillant method, but unfortunately in [17] his idea does not shine as it could, because of too many technical details. One of the aims of this paper is to bring to the fore the elegance of Huet's approach.

First of all, one needs to represent sets of redexes. This can easily be done introducing a new constructor $\underline{(\lambda x.s)v}$ for marked redexes and say that a set of redexes R in a term t is the term with marks T obtained from t by marking the redexes in R. So we switch to the following grammar of marked terms:

$$R ::= x \mid \lambda x.R \mid RR \mid \underline{(\lambda x.R)R}$$

For instance the four possible sets of redexes of $(\lambda x.(II))I$ are:

$$(\lambda x.(II))I \qquad (\lambda x.(\underline{II}))I \qquad \underline{(\lambda x.(II))I} \qquad \underline{(\lambda x.(\underline{II}))I}$$

In [17] the marks are on applications, which may not be redexes. Marking applications requires a notion of *well-marked term* (*regular terms* in [17]) which comes with various annoying complications. The choice of marking redexes is our first simplification (which is standard, we are not claiming originality).

Huet's contribution is the definition of residuals. For a marked redex R let its **support** be the λ-term obtained by removing all underlinings. Given a term t we want to define R/P, the residuals of the redexes in the set R after the reduction of another set P of redexes of t. Note that both R and P are terms, and that they share the same support t. A first step towards the definition of R/P is to forget t and consider R, which is t plus the information about the redexes we want to track. The idea is to see R/P simply as the target R' of a reduction step $R \overset{P}{\Rightarrow} R'$. Of course, now the point is how to define $R \overset{P}{\Rightarrow} R'$. What is particularly nice is that it can be defined by a simple structural induction over R, exploiting the fact that R and P have the same support:

[3] This minimality can be stated mathematically as the existence of pushouts in a certain category of reductions sequences, but its precise formulation requires to introduce permutation equivalence, which is beyond the scope of this paper.

$$\frac{}{x \overset{x}{\Rightarrow} x} \qquad \frac{R \overset{P}{\Rightarrow} R'}{\lambda x.R \overset{\lambda x.P}{\Rightarrow} \lambda x.R'}$$

$$\frac{R \overset{P}{\Rightarrow} R' \qquad S \overset{Q}{\Rightarrow} S'}{RS \overset{PQ}{\Rightarrow} R'S'} \qquad \frac{R \overset{P}{\Rightarrow} R' \qquad S \overset{Q}{\Rightarrow} S'}{\underline{(\lambda x.R)S} \overset{(\lambda x.P)Q}{\Rightarrow} \underline{(\lambda x.R')S'}}$$

$$\frac{R \overset{P}{\Rightarrow} R' \qquad S \overset{Q}{\Rightarrow} S'}{(\lambda x.R)S \overset{(\lambda x.P)\ Q}{\Rightarrow} R'\{x/S'\}} \qquad \frac{R \overset{P}{\Rightarrow} R' \qquad S \overset{Q}{\Rightarrow} S'}{\underline{(\lambda x.R)S} \overset{(\lambda x.P)Q}{\Rightarrow} R'\{x/S'\}}$$

An example: if $R = (\lambda x.xx)\ (\underline{II})$ and $P = \underline{(\lambda x.xx)\ (II)}$ then R/P is the marked term $R' = (\underline{II})(\underline{II})$, because one easily derives:

$$(\lambda x.xx)\ (\underline{II}) \overset{(\lambda x.xx)\ (II)}{\Rightarrow} (\underline{II})(\underline{II})$$

Now, we have all the ingredients to prove the parallel moves property. However, a stronger property—the *cube* property, due to Jean-Jacques Lévy [20,4]—can now be expressed. Note that in Fig. 1.c the starting term is a λ-term s. Observe that any λ-term s is a marked term: it represents the empty set of redexes of s. By simply replacing s with a generic set of redexes, *i.e.* a marked term Q, we get the cube property (see Fig. 1.d). The cube enriches the parallel moves property with a sort of *contextual coherence*: the two sides of the diagram give the same term *and act in the same way on any other set of redexes in the starting term.*

By repeating Huet's development in Abella and then analyzing the structure of the formal proof we realized that the cube property can be proved exactly as the diamond property of $\Rightarrow$. One needs to first prove the following lemma (called *commutation lemma* in [17]):

Lemma 2 (substitutivity of $\overset{P}{\Rightarrow}$). *If $R \overset{P}{\Rightarrow} R'$ and $S \overset{Q}{\Rightarrow} S'$ then $R\{x/S\} \overset{P\{x/Q\}}{\Rightarrow} R'\{x/S'\}$.*

The proof is a simple induction on $R \overset{P}{\Rightarrow} R'$ (analogously to Lemma 1). Then one gets[4]:

Theorem 2 (cube property of $\overset{P}{\Rightarrow}$). *$Q' \overset{P}{\Leftarrow} Q \overset{R}{\Rightarrow} Q''$ implies $\exists S$ s.t. $Q' \overset{R/P}{\Rightarrow} S \overset{P/R}{\Leftarrow} Q''$.*

The proof is by induction on $Q \overset{P}{\Rightarrow} Q'$ and case analysis of $Q \overset{R}{\Rightarrow} Q''$, using Lemma 2 (their formalization is in the last page, after the bibliography).

The prism property. In [17] Huet argues that the cube property follows from a more primitive property, the prism property. We need a definition: given two

[4] The more accurate but less readable statement is: $Q' \overset{P}{\Leftarrow} Q \overset{R}{\Rightarrow} Q''$ implies $\exists S, R', P'$ s.t. $P \overset{R}{\Rightarrow} P'$, $R \overset{P}{\Rightarrow} R'$, and $Q' \overset{R'}{\Rightarrow} S \overset{P'}{\Leftarrow} Q''$.

$$\frac{}{x \le x} \qquad \frac{R \le R'}{\lambda x.R \le \lambda x.R'} \qquad \frac{R \le R' \qquad S \le S'}{RS \le R'S'}$$

$$\frac{R \le R' \qquad S \le S'}{(\lambda x.R)S \le \underline{(\lambda x.R')S'}} \qquad \frac{R \le R' \qquad S \le S'}{\underline{(\lambda x.R)S} \le \underline{(\lambda x.R')S'}}$$

$$\frac{}{x \cup x = x} \qquad \frac{R \cup P = Q \qquad S \cup T = U}{RS \cup PT = QU} \qquad \frac{R \cup P = Q \qquad S \cup T = U}{(\lambda x.R)S \cup \underline{(\lambda x.P)T} = \underline{(\lambda x.Q)U}}$$

$$\frac{R \cup P = Q}{\lambda x.R \cup \lambda x.P = \lambda x.Q} \qquad \frac{R \cup P = Q \qquad S \cup T = U}{\underline{(\lambda x.R)S} \cup (\lambda x.P)T = \underline{(\lambda x.Q)U}} \qquad \frac{R \cup P = Q \qquad S \cup T = U}{\underline{(\lambda x.R)S} \cup \underline{(\lambda x.P)T} = \underline{(\lambda x.Q)U}}$$

Fig. 2. The order ($\le$) and the union operation ($\cup$) for sets of redexes/marked terms

a) Prism 1: if $P \le R$ b) Prism 2: if $P \le R$ c) Prism 1-2: if $P \le R$ d) Prism-Cube:

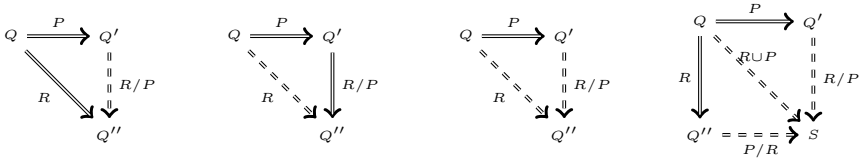

Fig. 3. Prism diagrams

marked terms R and S with the same support, let $R \le S$ hold if S has all the marks in R and possibly more (see Fig. 2 for an inductive definition). Then the **prism property** is given by the two implications in Fig. 3.a-b. The idea is that the prism gives one half of the cube property, which then follows by a symmetry argument (actually one needs only Fig.3.a).

By looking closely at the formalized proofs we realized that both parts of the prism property can be proved by induction on $P \le R$. So that it should rather be stated as in Fig.3.c. The point is that the cube property essentially follows from the same induction.

Let us clarify this point. Given marked redexes R and S sharing the same support, let $R \cup S$ be the marked term with all the marks of R and all the marks of S (see Fig. 2). If $R \le S$ then there is P s.t. $R \cup P = S$. So doing induction on $R \le S$ is essentially equivalent to inducting on $R \cup P$. The cube property in Fig. 1.d can be proved by induction on $R \cup P$. Therefore, the same induction also proves the diagram in Fig. 3.d, which puts together the prism and the cube property. Last, induction on $R \cup P$ can be replaced by induction over $Q \overset{P}{\Rightarrow} Q'$ and case analysis of $Q \overset{R}{\Rightarrow} Q''$. We then get:

Theorem 3 (Prism-cube property of $\overset{P}{\Rightarrow}$). *$Q' \overset{P}{\Leftarrow} Q \overset{R}{\Rightarrow} Q''$ implies $\exists S$ s.t. $Q' \overset{R/P}{\Rightarrow} S \overset{P/R}{\Leftarrow} Q''$ and $Q \overset{R \cup P}{\Rightarrow} S$.*

In the theory of *abstract residuals systems* [34] (Chapter 8.7) the cube property is one of the axioms while the prism property (there called *triangle property*) is not required to hold, and in fact there are examples of residual systems where it fails ([34], 8.7.29, page 440, or the rewriting system obtained orienting the associativity rule, see also [22]).

As for the cube property, it is possible to refine the diamond property into the *prism-diamond property*: if $s_2 \Leftarrow s \Rightarrow s_1$ then $\exists t$ s.t. $s_1 \Rightarrow t \Leftarrow s_2$ *and also* $s \Rightarrow t$. Actually, it is this enriched property—proved exactly as the diamond property—that we have formalized in Abella.

3 The Diamond and Cube Properties, Formally in Abella

Abella. Abella is an interactive theorem prover developed by Andrew Gacek [9,8], and based on the logic G developed by Gacek, Miller, and Nadathur [13,12,11]. Abella uses the higher-order abstract syntax (HOAS) approach to binders [7,23] and it is provided with a nominal quantifier ∇ (nabla) [13,12,11,24,10,2] (which has a subtle proof theory that shall not be treated here). Being based on HOAS, Abella provides a primitive handling of α-equivalence and capture-avoiding substitution. By exploting ∇, Abella is also able to mix inductive and co-inductive reasoning.

Many domains in which Abella has been used involve reasoning about typing judgements. In order to facilitate the treatment of such judgements, a *second logic* is defined within Abella. This second logic, a small intuitionistic *specification logic*, can be used to directly treat a range of typing judgments; this is what is sometimes called the *two-levels approach* [14]. A key property is that the specification logic satisfies cut-elimination. This fact allows to use cut-elimination as a tactic, and derive substitution lemmas *for free*. For instance, subject-reduction proofs can often be written very cleanly.

If one uses Abella to reason on specifications that do not involve typing judgements, then the built-in specification logic is not needed. For example, many properties of the untyped π-calculus have been proved using ∇, induction, and co-induction, without using the two-level logic architecture of Abella [25]. Since our subject here is the untyped λ-calculus, we shall similarly find no need for Abella's second level of logic.

λ-terms. The encoding of λ-terms we use is standard (for HOAS). The encoding of marked terms extends the encoding of λ-terms (whose constructors are now renamed `mabs` and `mapp`) with `mredex R S`, which represents the marked redex $\underline{(\lambda x.R)S}$. The two sets of terms have type `tm` and `mtm` (standing for *term* and *marked term*) and are in Fig. 4.

There is no explicit constructor for variables. This point is a bit delicate. Let us try to explain it. The free variables are handled by the nabla quantifier (see the example in the next paragraph). The bound variables are provided by the HOAS-approach to binders, which codes binders as functions from terms to terms. For instance the term $\lambda x.xx$ is represented as `abs x\ app x x`, *i.e.*

```
kind tm      type.                          kind mtm      type.
type app     tm -> tm -> tm.                type mapp     mtm -> mtm -> mtm.
type abs     (tm -> tm) -> tm.              type mabs     (mtm -> mtm) -> mtm.
                                            type mredex   (mtm -> mtm) -> mtm -> mtm.
```

Fig. 4. Definition of λ-terms and marked λ-terms in Abella

it is obtained by applying the `abs` constructor to the function `x` $\mapsto$ `app x x` (which maps the generic term `x` to the term `app x x`), coded in Abella with `x\ app x x`. In particular, `mabs M` and `mredex M N` are both of type `mtm`, but their subexpression `M` is not a term of type `mtm`, but rather a term of type `mtm->mtm`.

Finally, given a β-redex `app (abs M) N` representing $(\lambda x.M)N$, the reduct $M\{x/N\}$ is denoted by `M N`, *i.e.* the application of the function `M` to `N` (application is given by juxtaposition). The user can forget any trouble with α-equivalence and substitution: Abella takes care of them.

An example. Suppose that we want to write the predicate `tm M` which isolates terms without marked redexes in the larger set of marked terms. In Abella it can be written as follows:

```
Define tm : mtm -> prop by
  nabla x, tm x;
  tm (mabs M) := nabla x, tm (M x);
  tm (mapp M N) := tm M /\ tm N.
```

It corresponds exactly to the common way of defining λ-terms. The first line reads: *if* `x` *is a fresh variable then* `x` *is a term.* The second line: `mabs M` *is a term if given a fresh variable* `x` *the term obtained by applying the function* `M` *to* `x` *is a term.* Informally, one would simply ask that `M` is a term. What we used is nothing but its HOAS formulation, which has to adapt the informal approach because `M` itself is not of type `mtm` (but `mtm->mtm`). The third line: `mapp M N` *is a term if both* `M` *and* `N` *are terms.*

Prism-diamond property. Figure 5 contains the development of the prism-diamond property, where parallel reduction is the predicate `pred`.

Look at the second and the fourth cases of the definition of `pred`. There, `T` and `T'` are functions representing the binders associated to the corresponding abstractions. In the hypothesis of the two rules they are applied to `x`, in order to get a term. Moreover, the fourth case contains `T' U'`, the application of the function `T'` to `U'`: it is where β-reduction and substitution take place.

The substitution lemma follows, proved by simple induction on $T \Rightarrow T'$. Note that `T` is assumed to be a function, since in the conclusion we want to substitute `U` in `T` (and `U'` in `T'`). This is why the first assumption contains `(T x)`, and `x` is bound by ∇ (nabla). The commands `induction on 1`, `intros`, and `case H1` are the Abella code to start an induction on the first hypothesis. Then every line is

```
Define pred : tm -> tm -> prop by
  nabla x, pred x x;
  pred (abs T) (abs T') := nabla x, pred (T x) (T' x);
  pred (app T U) (app T' U') := pred T T' /\ pred U U';
  pred (app (abs T) U) (T' U') := nabla x, pred (T x) (T' x) /\ pred U U'.

Theorem pred_sub : forall T T' U U', nabla x,
  pred (T x) (T' x) -> pred U U' -> pred (T U) (T' U').
    induction on 1. intros. case H1.
      search.
      search.
      apply IH to H3 H2. search.
      apply IH to H3 H2. apply IH to H4 H2. search.
      apply IH to H3 H2. apply IH to H4 H2. search.

Theorem prism-diamond : forall T U1 U2,
  pred T U1 -> pred T U2 -> exists V, pred U1 V /\ pred U2 V /\ pred T V.
    induction on 1. intros. case H1.
      case H2. search.
      case H2. apply IH to H3 H4. search.
      case H2.
        apply IH to H3 H5. apply IH to H4 H6. search.
        case H3. apply IH to H4 H6. apply IH to H7 H5.
                                    apply pred_sub to H12 H9. search.
      case H2.
        case H5. apply IH to H3 H7. apply IH to H4 H6.
                                    apply pred_sub to H8 H11. search.
        apply IH to H3 H5. apply IH to H4 H6. apply pred_sub to H7 H10.
                                          apply pred_sub to H8 H11. search.
```

Fig. 5. Abella development of the prism-diamond property for $t \Rightarrow t'$ (called `pred T T'`)

a case of the induction. The `search` tactic attempts to prove the current goal by an automatic simple search. Clearly, `apply IH` applies the inductive hypothesis.

The diamond property is proved by induction on the first hypothesis and case analysis of the second (note that each subcase starts with `case H2`). The proof uses only the *i.h.* and the substitution lemma, and it is the same proof which appears in [27], which also contains the only complete proof of confluence for λ-calculus based on HOAS of which we are aware.

Prism-cube property. The development for the prism-cube property is in the last page, after the bibliography, where $R \overset{P}{\Rightarrow} R'$ is represented with `res R P R'`. As we explained in Section 2 it follows exactly the same pattern used for the diamond property (plus the additional definition for the union of marked terms). The third component `(R' S')` of the last two cases defining `res` is where β-reduction and substitution are used.

Beyond Huet's original paper [17], the cube property has also been formalized in [31,35]. Our development is the first one using HOAS, and it is sensibly simpler and shorter than the others. Indeed, it fits—in full—into one single page. We believe that this is quite remarkable.

4 Beyond the Pearl

This section contains a few observations and variations over our developments, others can be found in [1].

Confluence by developments. The *(complete) development* t° of a term t is the result of the parallel step which reduces all the redexes in t. The development t° can easily be described by induction on t, as follows:

$$\begin{array}{llll} x^\circ & = x & (tu)^\circ & = t^\circ u^\circ \qquad \text{if } t \neq \lambda x.t' \\ (\lambda x.t)^\circ & = \lambda x.t^\circ & ((\lambda x.t')u)^\circ & = t'^\circ\{x/u^\circ\} \end{array}$$

Developments enjoy the following property, which is a sort of maximal prism property: if $t \Rightarrow u$ then $t \Rightarrow t^\circ$ and $u \Rightarrow t^\circ$. By specializing the prism-diamond property to complete developments, one gets the following *development property*: $s_2 \Leftarrow s \Rightarrow s_1$ implies $s_1 \Rightarrow s^\circ \Leftarrow s_2$ and $s \Rightarrow s^\circ$.

The proof of confluence by developments consists in showing the diamond property by proving the development property. This approach is dual to the use of residuals. Indeed, developments give a sort of *maximum* closure of any span $u_2 \leftarrow t \rightarrow u_1$, while residuals give the *minimum*.

The development property can be proved in Abella essentially as the prism-diamond property (see [1] for details). The formal proof (2 lemmas) is a sensible simplification of the one in Abella by Randy Pollack (18 lemmas), or of the similar one done in Twelf by Dan Licata. They can be found on the websites of the respective proof assistants, and both are based on Takahashi's proof [33] (see also [30]).

Using the specification logic. We now describe the impact of the specification logic provided by Abella on our developments. Let us come back to the toy `tm` predicate of Section 3, which isolates terms among marked terms. At the specification level it takes the following form (the syntax of the specification level is different, an explanation follows):

```
tm (mabs R) :- pi x\ tm x => tm (R x).
tm (mapp M N) :- tm M, tm N.
```

where `pi x\` means $\forall x$, and `tm M, tm N` stays for `tm M` *and* `tm N`. There are two main differences. The first is that there is no case for free variables. This is due to the fact that ∇ is not in the weaker specification logic. It means that we can only represent bound variables, *i.e.* only closed terms. This fact induces the second difference: the first line of the definition says that for proving that `abs R` is a term we need to prove that `R x` is a term *under the assumption that* `x` *is a term*. This new *under the assumption* part has a consequence at the reasoning level: it forces to annotate every use of a predicate involving binders with a *context*, *i.e.* the set of current assumptions under which the predicate holds. Such contexts are what allows to deal with open terms in this weaker setting without nabla. The idea is that to prove `tm (abs R)` we now need to be able to

prove the judgement `tm x |- tm (R x)` (where `|-` is the concrete notation for the turnstile $\vdash$), *i.e.* `tm (R x)` in the context of assumptions `tm x`.

The presence of contexts usually requires to prove some properties about them, to introduce relations between contexts, and to generalize the statements of lemmas and theorems. In [1] we also recast our developments at the specification level. The structure of the proofs is the same, but some complications (*i.e.* additional definitions and lemmas) about contexts arise. Such complications are typically raised by statements having more than one predicate on the same term: the contexts of these predicates have to be related because they refer to the same binding structure. Indeed, the prism-diamond property (which uses only $\Rightarrow$) requires less reasoning on contexts than the development and prism-cube properties (both using two predicates); this is also why in [27] worlds (the analogous of contexts in Twelf) do not require much attention, while they do require it in the proof of the development property by Dan Licata.

One of the aims of this paper is to show that in untyped frameworks the combined use of ∇ and HOAS gets formalizations which are extremely faithful to common pen-and-paper reasoning. Some of the examples dealing with the untyped λ-calculus on Abella website have been *lifted* from the specification to the reasoning logic, getting quite simpler and more readable formalizations, see [1]. It has to be said, however, that there are some untyped specifications that are of an intrinsic *closed nature*. For instance, the examples *equivalence of terms based on paths* and *determinism of translation between HOAS and de Bruijn representations* on the Abella website do not lift in a natural way to the reasoning level.

Substitution lemmas. In our developments we prove explicitly substitution lemmas for $\Rightarrow$ and $\overset{P}{\Rightarrow}$. One of the features of the specification logic is that it allows to get substitution lemmas for free. One would then expect that switching to the specification logic such lemmas disappear, and the formalizations get even shorter. Interestingly, this is not the case. Let us focus on $\Rightarrow$, which is simpler. The clause involving abstractions is:

```
pred (abs U) (abs V) :- pi x\ pred x x => pred (U x) (V x).
```

This implies that the contexts for `pred` contains assumptions of the form `pred x x`. The substitution lemma provided by Abella then says that if $t \Rightarrow t'$ then $t\{x/u\} \Rightarrow t'\{x/u\}$. This is not Lemma 1, because the second term should be $t'\{x/u'\}$ (with $u \Rightarrow u'$). But the assumption `pred x x` forces $u' = u$. Unfortunately, this simpler lemma cannot be used to prove the diamond property. It seems that it is enough to define `pred` as follows:

```
pred (abs U) (abs V) :- pi x\ pi y\ pred x y => pred (U x) (V y).
```

One gets indeed the right substitution lemma. But now it is necessary to prove the reflexivity of `pred` (which before followed implicitly by the definition), which is non-trivial with assumptions of the form `pred x y`.

Summing up, the substitution lemmas for free provided by the specification level do not help in any way in the development under study.

Acknowledgements. To Dale Miller, who supervised me, encouraged me and helped me all along this work. To Kaustuv Chaudhuri and David Baelde for help and discussions about HOAS and Abella. To Fabien Renaud, Stéphane Zimmermann, and the anonymous reviewers for suggesting useful improvements. This work was partially supported by the Qatar National Research Fund under grant NPRP 09-1107-1-168.

References

1. Accattoli, B.: Sources, `https://sites.google.com/site/beniaminoaccattoli/residuals`
2. Baelde, D.: On the expressivity of minimal generic quantification. Electr. Notes Theor. Comput. Sci. 228, 3–19 (2009)
3. Barendregt, H.P.: The Lambda Calculus – Its Syntax and Semantics, vol. 103. North-Holland (1984)
4. Berry, G., Lévy, J.J.: Minimal and optimal computations of recursive programs. In: POPL, pp. 215–226 (1977)
5. Brotherston, J., Vestergaard, R.: A formalised first-order confluence proof for the λ-calculus using one-sorted variable names. Inf. Comput. 183(2), 212–244 (2003)
6. Dunfield, J., Pientka, B.: Beluga: A Framework for Programming and Reasoning with Deductive Systems (System Description). In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS, vol. 6173, pp. 15–21. Springer, Heidelberg (2010)
7. Elliott, C., Pfenning, F.: Higher-order abstract syntax. In: PLDI, pp. 199–208 (1988)
8. Gacek, A.: The Abella Interactive Theorem Prover (System Description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 154–161. Springer, Heidelberg (2008)
9. Gacek, A.: A framework for specifying, prototyping, and reasoning about computational systems. Ph.D. thesis, University of Minnesota (September 2009)
10. Gacek, A.: Relating nominal and higher-order abstract syntax specifications. In: PPDP 2010, pp. 177–186. ACM (July 2010)
11. Gacek, A., Miller, D., Nadathur, G.: Combining generic judgments with recursive definitions. In: LICS, pp. 33–44 (2008)
12. Gacek, A., Miller, D., Nadathur, G.: Reasoning in Abella about structural operational semantics specifications. ENTCS 228, 85–100 (2009)
13. Gacek, A., Miller, D., Nadathur, G.: Nominal abstraction. Inf. Comput. 209(1), 48–73 (2011)
14. Gacek, A., Miller, D., Nadathur, G.: A two-level logic approach to reasoning about computations. J. Autom. Reasoning 49(2), 241–273 (2012)
15. Glauert, J.R.W., Khasidashvili, Z.: Relating conflict-free stable transition and event models via redex families. Theor. Comput. Sci. 286(1), 65–95 (2002)
16. Homeier, P.V.: A proof of the Church-Rosser theorem for the λ-calculus in higher order logic. In: TPHOLs 2001: Supplemental Proceedings, pp. 207–222 (2001)
17. Huet, G.P.: Residual theory in λ-calculus: A formal development. J. Funct. Program. 4(3), 371–394 (1994)

18. Huet, G.P., Lévy, J.J.: Computations in orthogonal rewriting systems, I. In: Computational Logic - Essays in Honor of Alan Robinson, pp. 395–414 (1991)
19. Huet, G.P., Lévy, J.J.: Computations in orthogonal rewriting systems, II. In: Computational Logic - Essays in Honor of Alan Robinson, pp. 415–443 (1991)
20. Lévy, J.J.: Réductions correctes et optimales dans le lambda-calcul. Thése d'Etat, Univ. Paris VII, France (1978)
21. McKinna, J., Pollack, R.: Pure Type Systems Formalized. In: Bezem, M., Groote, J.F. (eds.) TLCA 1993. LNCS, vol. 664, pp. 289–305. Springer, Heidelberg (1993)
22. Melliès, P.-A.: Axiomatic Rewriting Theory VI Residual Theory Revisited. In: Tison, S. (ed.) RTA 2002. LNCS, vol. 2378, pp. 24–50. Springer, Heidelberg (2002)
23. Miller, D., Nadathur, G.: A logic programming approach to manipulating formulas and programs. In: SLP, pp. 379–388 (1987)
24. Miller, D., Tiu, A.: A proof theory for generic judgments. ACM Trans. Comput. Log. 6(4), 749–783 (2005)
25. Miller, D., Tiu, A.: Proof search specifications of bisimulation and modal logics for the π-calculus. ACM Trans. Comput. Log. 11(2) (2010)
26. Nipkow, T.: More Church-Rosser proofs (in Isabelle/HOL). Journal of Automated Reasoning, 733–747 (1996)
27. Pfenning, F.: A proof of the Church-Rosser theorem and its representation in a logical framework. Tech. Rep. CMU-CS-92-186, Carnegie Mellon University (1992)
28. Pfenning, F., Schürmann, C.: System Description: Twelf - A Meta-Logical Framework for Deductive Systems. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 202–206. Springer, Heidelberg (1999)
29. Pientka, B.: Beluga: Programming with Dependent Types, Contextual Data, and Contexts. In: Blume, M., Kobayashi, N., Vidal, G. (eds.) FLOPS 2010. LNCS, vol. 6009, pp. 1–12. Springer, Heidelberg (2010)
30. Pollack, R.: Polishing up the Tait-Martin-Löf proof of the Church-Rosser theorem (1995)
31. Rasmussen, O.: The Church-Rosser theorem in Isabelle: a proof porting experiment. Tech. Rep. 164, University of Cambridge (1995)
32. Shankar, N.: A mechanical proof of the Church-Rosser theorem. J. ACM 35(3), 475–522 (1988)
33. Takahashi, M.: Parallel reductions in λ-calculus. Inf. Comput. 118(1), 120–127 (1995)
34. Terese: Term Rewriting Systems, Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press (2003)
35. Vestergaard, R.: The Primitive Proof Theory of the lambda-Calculus. Ph.D. thesis, Heriot-Watt University, Edinburgh, Scotland (2003)

```
Define res : mtm -> mtm -> mtm -> prop by
  nabla x, res x x x;
  res (mabs R) (mabs P) (mabs R') := nabla x, res (R x) (P x) (R' x);
  res (mapp R S) (mapp P Q) (mapp R' S') := res R P R' /\ res S Q S';
  res (mredex R S) (mapp (mabs P) Q) (mredex R' S') :=
            nabla x, res (R x) (P x) (R' x) /\ res S Q S';
  res (mapp (mabs R) S) (mredex P Q) (R' S') :=
            nabla x, res (R x) (P x) (R' x) /\ res S Q S';
  res (mredex R S) (mredex P Q) (R' S') :=
            nabla x, res (R x) (P x) (R' x) /\ res S Q S'.

Define res_union : mtm -> mtm -> mtm -> prop by
  nabla x, res_union x x x;
  res_union (mabs R) (mabs P) (mabs Q) := nabla x, res_union (R x) (P x) (Q x);
  res_union (mapp R S) (mapp P T) (mapp Q U) :=
        res_union R P Q /\ res_union S T U;
  res_union (mredex R S) (mredex P T) (mredex Q U) :=
        nabla x, res_union (R x) (P x) (Q x) /\ res_union S T U;
  res_union (mapp (mabs R) S) (mredex P T) (mredex Q U) :=
        nabla x, res_union (R x) (P x) (Q x) /\ res_union S T U;
  res_union (mredex R S) (mapp (mabs P) T) (mredex Q U) :=
        nabla x, res_union (R x) (P x) (Q x) /\ res_union S T U.

Theorem res_subst : forall R P R' S Q S', nabla x,
  res (R x) (P x) (R' x) -> res S Q S' -> res (R S) (P Q) (R' S').
    induction on 1. intros. case H1.
      search.
      search.
      apply IH to H3 H2. search.
      apply IH to H3 H2. apply IH to H4 H2. search.
      apply IH to H3 H2. apply IH to H4 H2. search.
      apply IH to H3 H2. apply IH to H4 H2. search.
      apply IH to H3 H2. apply IH to H4 H2. search.

Theorem prism_cube : forall Q P R Q' Q'',
  res Q R Q'' -> res Q P Q' -> exists P' R' RunionP S,
  res P R P'    /\  res R P R'      /\  res Q' R' S     /\
  res Q'' P' S  /\  res Q RunionP S /\  res_union P R RunionP.
    induction on 1. intros. case H1.
      case H2. search.
      case H2. apply IH to H3 H4. search.
      case H2.
        apply IH to H3 H5. apply IH to H4 H6. search.
        case H3. apply IH to H4 H6. apply IH to H7 H5.
                                                    apply res_subst to H16 H10. search.
      case H2.
        apply IH to H3 H5. apply IH to H4 H6. search.
        apply IH to H3 H5. apply IH to H4 H6. apply res_subst to H9 H15. search.
      case H2.
        case H5. apply IH to H3 H7. apply IH to H4 H6.
                                    apply res_subst to H11 H17. search.
        apply IH to H3 H5. apply IH to H4 H6. apply res_subst to H9 H15.
                                              apply res_subst to H10 H16. search.
      case H2.
        apply IH to H3 H5. apply IH to H4 H6. apply res_subst to H10 H16. search.
        apply IH to H3 H5. apply IH to H4 H6. apply res_subst to H9 H15.
                                              apply res_subst to H10 H16. search.
```