More Church-Rosser Proofs in BELUGA

Alberto Momigliano[*]
Dipartimento di Informatica,
Università degli Studi di Milano, Italy

Martina Sassella
Dipartimento di Matematica
Università degli Studi di Milano, Italy

We report on yet another formalization of the Church-Rosser property in lambda-calculi, carried out with the proof environment BELUGA. After the well-known proofs of confluence for β-reduction in the untyped settings, with and without Takahashi's complete developments method, we concentrate on η-reduction and obtain the result for $\beta\eta$ modularly. We further extend the analysis to typed-calculi, in particular System F. Finally, we investigate the idea of pursuing the encoding directly in BELUGA's meta-logic, as well as the use of BELUGA's logic programming engine to search for counterexamples.

1 Introduction

Stop me if you heard this before: the Church-Rosser theorem for β-reduction (CR(β)) is a good case study for proof assistants. In fact, Church-Rosser theorems are arguably the most formalized results in mechanized meta-theory of deductive systems.

In the beginning, the thrust was to see whether the theorem could be formalized at all: Shankar's proof in the Boyer-Moore theorem-prover [25] was a break-through and a tour de (brute) force. A few years later, Nipkow [17] made the proof much more abstract and extended it to $\beta\eta$-reduction. The workhorse encoding technique was de Bruijn indices and it was a bullet that one just had to bite.

In the following years, the Church-Rosser theorem became a benchmark to showcase how to mechanize the variable binding problem. Some partial data-points, with the understanding that these are not exhaustive not disjoint: if interested in mirroring the informal practice of working mathematicians, see the paper by Vestergaard and Brotherston [27] and the recent work by Copello et al. [6]. If you want to reason about α-conversion explicitly via quotients, see Ford and Mason's [9]. For the *locally nameless* representation, see McKinna and Pollack [14] and its modern take [5]. Library support is showcased by AUTOSUBST [23]. Nominal techniques are also well-represented, see the recent proof by Nagele and al. [16].

We shall use higher-order abstract syntax (HOAS) following up on the seminal proof of CR(β) by Pfenning in 1992 [18], when Twelf was merely Elf. Once the non-trivial issue of totality-checking was settled, that proof stood as a shining example of the benefits of HOAS, which we shall not repeat here. Confluence results did not attract further attention in this community, until Accattoli, in his proof pearl [2], liberated Huet's Coq encoding of residual theory [10] from its concrete-syntaxed infrastructure.

So why bother with yet another HOAS-based encoding? Our aim was to go beyond β-reduction and replicate Nipkow's results in the HOAS setting. Further, to extend it to typed calculi, with a minimum amount of change. One way to achieve that is via *intrinsically-typed terms* [3] and for the latter, BELUGA [20] is the natural (if not only) modern system that natively supports HOAS together with dependent types. We were also curious to see if we could achieve a significant formalization as BELUGA's novices,

[*]Member of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM)

with just a cursory understanding of its intricate meta-theory.[1] This is part of a more general project of developing a curriculum to teach the classic theory of the lambda-calculus to graduate students.

In passing, we evaluate (in the negative) Accattoli's suggestion [2] that even in two-level systems such as BELUGA, the specification of the semantics of the (untyped) lambda-calculus should be carried out directly in the meta-logic. In the Appendix, we explore the concept of *mechanized meta-theory model-checking* within the Beluga framework. This approach aligns with the increasing tendency to enhance proof assistants with a form of *counterexample* search. QuickChick serves as one example of this trend [12]..

In this short paper, we assume knowledge of confluence in the lambda-calculus, for which we refer to the concise presentation in [24]. We recall some basic notions; given binary relations over a set A, and their Kleene and reflexive closure $(\_)^*$, $(\_)^=$, we define:

$$\begin{tikzcd} {} \arrow[r, "\mathrm{R}"] \arrow[d, "\mathrm{R}"'] & {} \arrow[d, dashed, "\mathrm{R}"] \\ {} \arrow[r, dashed, "\mathrm{R}"'] & {} \end{tikzcd}$$

diamond for R

$$\begin{tikzcd} {} \arrow[r, "S^*"] \arrow[d, "R^*"'] & {} \arrow[d, dashed, "R^*"] \\ {} \arrow[r, dashed, "S^*"'] & {} \end{tikzcd}$$

R and S commute

$$\begin{tikzcd} {} \arrow[r, "\mathrm{S}"] \arrow[d, "\mathrm{R}"'] & {} \arrow[d, dashed, "R^="] \\ {} \arrow[r, dashed, "S^*"'] & {} \end{tikzcd}$$

R and S strongly commute

We say that R is *confluent* if and only it commutes with itself.

We also assume a passing familiarity with BELUGA. We simply recall that the specification of the syntax and judgments of the system under study is done in (contextual) LF, while theorems are realized as total functions in BELUGA's meta-logic. LF contexts are reified into first-class objects that can be abstracted and quantified over, being classified by context *schemas*. First-class substitutions map contexts to each other realizing properties such as weakening, strengthening and subsumption. In this development, we have taken care to state every theorem quantifying over the smallest context schema where it makes sense [8].

2 Sketch of the Formal Development

We will only sketch some of the highlights of the encoding, referring the reader to the repository[2] for most, if not all the details.

2.1 CR(β)

Pientka [19] ported to BELUGA Pfenning's encoding [18] in Twelf of the traditional parallel reduction proof à la Tait/Martin-Löf. We did the same for Licata's proof [13] via Takahashi's *complete developments* [26] This porting was similarly uneventful, save of course for the improvements that the BELUGA brings in w.r.t. Twelf. This applies in particular to the streamlined handling of context reasoning in contrast to Twelf's rather fragile notion of *regular worlds*. A detailed comparison of the relative merits of BELUGA vs. Twelf, among others, can be found in [7].

[1] With the partial exception of Kaiser's dissertation [11], all existing BELUGA's code originates exclusively from Pientka and her students.

[2] `https://github.com/martinasassella/More_CR_Proofs_Beluga`

In both proofs, there are *no* technical lemmas about variables, renamings etc. We do have to prove that parallel reduction is stable under substitution, but it is from first principles, meaning it does nor rely on properties such as weakening and exchange. The direct proof of the diamond property for parallel reduction has a complex case-analysis, just as in the informal case. Takahashi's proof is based on a *relational* rather than functional encoding of complete developments, which is fine as the proof only needs totality, not uniqueness of developments.

For future reference, we list here the HOAS encoding of the syntax of untyped lambda terms and of parallel reduction, featuring a crucial use of hypothetical judgments to internalize the variable cases:

```
LF term : type =
 | lam : (term → term) → term
 | app : term → term → term;

LF pred : term → term → type =
 | beta : ({x:term} pred x x → pred (M1 x) (M1' x)) → pred M2 M2'
                                 → pred (app (lam M1) M2) (M1' M2')
 | lm : ({x:term} pred x x → pred (M x) (M' x)) → pred (lam M) (lam M')
 | ap : pred M1 M1' → pred M2 M2' → pred (app M1 M2) (app M1' M2');
```

2.2 CR(η)

While CR(η), as well as CR($\beta\eta$) can be proved via complete developments, here we follow Nipkow's more modular *commutation* approach. We separately consider η-reduction as the congruence over the η rule $\Gamma \vdash \lambda x.M\ x \longrightarrow_\eta M$, provided $x \notin FV(M)$: η-reduction (or η-expansion, if viewed from the right to the left) is encoded in BELUGA as the type family

```
LF eta_red : term → term → type =
 | eta : eta_red (lam \x.(app M x)) M % … congruence rules as above, omitted
```

Note how the above proviso is realized within HOAS as the meta-variable `M` *not* depending on `x` in the LF function `\x.(app M x)`. There is nothing "tricky" in this encoding, as per Nipkow's discussion (§ 4.3 of op.cit.). Only one technical lemma is required:

Lemma 2.1 *(Strengthening) If $\Gamma, x \vdash M \longrightarrow_\eta N$, and M does not depend on x, neither does N and $\Gamma \vdash M \longrightarrow_\eta N$.*

Formalizing this takes a little thought, but BELUGA's primitives make it relatively easy to state and prove.

The main proof strategy relies on a classic result [22] that provides a sufficient condition for *commutation*:

Lemma 2.2 *(Commutation – Hindley-Rosen) Two strongly commuting reductions commute.*

Since both LF and BELUGA are (roughly) first-order type theories, we cannot express operations on abstract relations, in particular closures, nor can we prove the above lemma once and for all for any such relation. Hence, we instantiate R and S to η-reduction:

$$\begin{tikzcd}
\cdot \arrow[r, "\eta"] \arrow[d, "\eta"'] & \cdot \arrow[d, dashed, "\eta^="] & & \cdot \arrow[r, "\eta^*"] \arrow[d, "\eta^*"'] & \cdot \arrow[d, dashed, "\eta^*"] \\
\cdot \arrow[r, dashed, "\eta^*"'] & \cdot & \Longrightarrow & \cdot \arrow[r, dashed, "\eta^*"'] & \cdot
\end{tikzcd}$$

We define the relevant closures at the LF level, with families `eta_red*` and `eta_red=`, and prove that η strongly commutes with itself as the function `square`. Recall that neither LF nor BELUGA have first-class sigma types and therefore existential propositions need to be encoded separately:

```
LF eta*_eta=_joinable : term → term → type =
 | eta*_eta=_result : eta_red* M1 N → eta_red= M2 N → eta*_eta=_joinable M1 M2;

schema ctx = term

rec square : (γ:ctx) {M : [γ ⊢ term]}{M1 : [γ ⊢ term]}{M2 : [γ ⊢ term]}
          [γ ⊢ eta_red M M1] → [γ ⊢ eta_red M M2] → [γ ⊢ eta*_eta=_joinable M1 M2] = …
```

This is proven by induction on `[γ ⊢ eta_red M M1]` and inversion on `[γ ⊢ eta_red M M2]`, with an appeal to strengthening when a critical pair is created by the η and ξ rules. We then state, in the rather long-winded way that BELUGA requires, the above instance of the Commutation lemma:

```
LF confl_prop : term → term → type =
 | confl_result : eta_red* M1 N → eta_red* M2 N → confl_prop M1 M2;

rec commutation_lemma:(γ:ctx)({M: [γ ⊢ term]}{M1: [γ ⊢ term]}{M2: [γ ⊢ term]}
        [γ ⊢ eta_red M M1] → [γ ⊢ eta_red M M2] → [γ ⊢ eta*_eta=_joinable M1 M2])
            → ({M': [γ ⊢ term]}{M1': [γ ⊢ term]}{M2': [γ ⊢ term]}
            [γ ⊢ eta_red* M' M1'] → [γ ⊢ eta_red* M' M2'] → [γ ⊢ confl_prop M1' M2']) = …
```

The proof approximates Nipkow's diagrammatic way via a concrete instance of the *Strip* lemma. In the end, what we decisively gained in elegance and succinctness with HOAS vs de Bruijn, we somewhat lose by the limitations of a first-order framework.

2.3 CR($\beta\eta$)

We take $\beta\eta$-reduction as the union of β and η-reductions. Since we have already shown confluence for each relation separately, it makes sense to exploit this other classic [22] result:

Lemma 2.3 (Commutative Union) *If R and S are confluent and commute, then $R \cup S$ is confluent.*

To establish the commutation assumption in the above lemma, we will again appeal to the Commutation lemma 2.2, hence we start with a version of the `square` function above, with $R := \beta$ and $S := \eta$. The proof requires a strengthening lemma for β-reduction, as well as two beautifully clean substitution lemmas for η and η^*. Again, this beauty is short-lived, as we have to replay the diagrammatic proof of the Commutation lemma with the current instantiation; this is a routine rework of the η case, but tedious.

The final ingredient is the proof of lemma 2.3 for $\beta\eta$, which is entailed by the following steps:

1. $(\beta^* \cup \eta^*)^* = (\beta \cup \eta)^*$;
2. $\beta^* \cup \eta^*$ satisfies the diamond property;
3. a Strip lemma for the above two relations.

2.4 Typed Calculi

We now switch gears and address confluence in *typed* lambda-calculi. While the main definitions still apply, we must be mindful to reduce only well-typed terms [24]. In a dependently typed proof environment such as BELUGA, this can be very elegantly accomplished using *intrinsically-typed terms* [3], that is, by ruling out pre-terms and indexing the judgments under study by well-typed terms only.

To illustrated the idea, we list the specification of well-typed terms in the polymorphic lambda-calculus (System F) and a fragment of parallel reduction (congruence rules for type abstraction and application together with the two beta rules):

```
LF ty : type =                      LF tm : ty → type =
 | arr : ty → ty → ty                 | lam : (tm A → tm B) → tm (arr A B)
 | all : (ty → ty) → ty;              | app : tm (arr A B) → tm A → tm B
                                      | tlam : ({a:ty} tm (A a)) → tm (all A)
                                      | tapp : tm (all A) → {B:ty} tm (A B);

LF pred : tm A → tm A → type =
 | tlm : ({a:ty} pred (M a) (M’ a)) → pred (tlam M) (tlam M’)
 | tap : pred M M’ → pred (tapp M A) (tapp M’ A)
 | beta : ({x:tm A} pred x x → pred (M1 x) (M1’ x)) → pred M2 M2’
                                  → pred (app (lam M1) M2) (M1’ M2’)
 | tbeta : ({a:ty} pred (M1 a) (M1’ a)) → pred (tapp (tlam M1) A) (M1’ A) …
```

Note how the signature of `pred` enforces the invariant that (parallel) reduction preserves typing. Since we now have two kinds of variables, the judgment is hypothetical both on terms and on types. On the reasoning level, this induces a context schema that accounts for this alternation, namely

```
schema pctx = some [A:ty] block(x:tm A, v:pred x x) + ty
```

What is remarkable is that literally the same proof structure of the Church-Rosser theorem carries over from the untyped case to the typed one. To wit, we have formalized Takahashi’s style CR(β) for System F and the proof is *conservative* in a very strong sense: not only do we use the same sequence of lemmata, but, being BELUGA’s scripts explicit proof-terms, the derivations for the untyped calculus directly *embed* into the derivations for System F: the user just needs to refine the statements of the theorems with the indexed judgments, on occasion making some implicit arguments explicit to aid type inference and of course adding cases for the new constructors. The proof of CR(β) for System F consists of around 460 loc, as opposed to 340 for the untyped case.

2.5 CR(β) at the Meta-Level

In the two-level approach adopted by BELUGA, the syntax and the semantics of a formal system are specified at the LF level, whereas reasoning is carried out at the computational level in form of recursive functions. LF features hypothetical judgments and those give for free properties of contexts such as exchange, weakening and substitution; by “for free”, we mean that they need not to be proved on a case by case base. Nonetheless, the communication of context information to the reasoning level requires the definition of possibly complex context schemas and relations among them that may complicate the statements and pollute the proofs. This may be particularly annoying in case studies such as the untyped lambda calculus, where contexts do not seem to play a large part, as observed by Accattoli in his remarkable paper [2] w.r.t. the “cousin” system Abella.

Since [20], BELUGA allows one to define `inductive` and `stratified` relations at the meta-level. Therefore, we can test Accattoli’s proposal w.r.t. the standard proof of CR(β) for the untyped case. This entail keeping at the LF level only the syntax and move all the other judgments at the computation level: to wit, parallel reduction has now this (non-hypothetical) representation:

```
inductive predM : (γ:ctx) [γ ⊢ term] → [γ ⊢ term] → ctype =
 | var : isVar [γ ⊢ M] → predM [γ ⊢ M] [γ ⊢ M]
 | betaM : predM [g, x:term ⊢ M1[…,x]] [g, x:term ⊢ M1’[…,x]] → predM [γ ⊢ M2] [γ ⊢ M2’]
        → predM [γ ⊢ app (lam \x.M1[…,x]) M2] [γ ⊢ M1’[…,M2’]] % other cases omitted

inductive isVar : (γ : ctx) {M: [γ ⊢ term]} ctype =
 | vp : {#q: [γ ⊢ term]} isVar [γ ⊢ #q]
```

```
schema rctx = block(x:term, t:pred x x)

rec rpar: {g:rctx}{M: [γ ⊢ term]}[γ ⊢ pred M M] =
mlam g ⇒ mlam M ⇒ case [γ ⊢ M] of
 | [γ ⊢ #p.1] ⇒ [γ ⊢ #p.2]
 | [γ ⊢ lam \x.M'[…,x]] ⇒
   let [g, b:block(x:term,v:pred x x) ⊢ IH[…,b.1,b.2]] =
     rpar [g, b:block(x:term,v:pred x x)] [g,b ⊢ M'[…,b.1]] in
       [γ ⊢ lm \x.\v.IH[…,x,v]]
 | [γ ⊢ app M1 M2] ⇒
   let [γ ⊢ IH1] = rpar [g] [γ ⊢ M1] in
   let [γ ⊢ IH2] = rpar [g] [γ ⊢ M2] in
       [γ ⊢ ap IH1 IH2];

rec rpar: {γ:ctx}{M : [γ ⊢ term]} predM [γ ⊢ M] [γ ⊢ M] =
mlam g ⇒ mlam M ⇒ case [γ ⊢ M] of
 | [γ ⊢ #p] ⇒ var (vp _)
 | [γ ⊢ lam \x.M'[…,x]] ⇒
    let h = rpar [g,x:term] [g,x:term ⊢ M'] in
    lm h
 | [γ ⊢ app M1 M2] ⇒
    let h1 = rpar [g] [γ ⊢ M1] in
    let h2 = rpar [g] [γ ⊢ M2] in
    ap h1 h2;
```

Figure 1: Reflexivity of `pred` vs `predM`

We have now a separate case for reducing variables, via the `isVar` judgment — recall that `#q` is a *parameter* variable, ranging over elements of the context. This is forced upon us by the usual positivity restriction on inductive types.

The more ominous consequence of this choice is that now establishing substitution properties for a given judgment requires a proof of *weakening*, and in turn the latter calls for a proof of context *exchange*, which is more delicate than expected. First, we must establish a clear and appropriate definition for determining that the ordering of a context is irrelevant : binary variable-swapping will suffice, although we will need to witness the swapping with a first-class substitution. Unfortunately, showing that reduction is stable under swapping, which basically amounts to equivariance, is a hassle: we have to "explain" to BELUGA's meta-logic what it means to be a variable. For the gory details, please see directory `Beta/beta_comp` in the repository. After that, the main proof proceeds smoothly in the usual way.

Is the switch to the meta-logic worth it in BELUGA? Not really. The elegance of Accattoli's approach stems from meta-level contexts in Abella being equivariant, and this preempts any issue with exchange. On the other hand, the idea works only for contexts that track "bare" bound variables, making it suitable just for (certain) untyped calculi. While BELUGA can easily overcome this via intrinsically typed terms, it seems that we have to rebuild a sort of de Bruijn infrastructure specific to each case study; furthermore, we had to struggle to prove that a substitutive judgment promoted to the meta-level is preserved by swapping. This seems too steep of a price for the mostly cosmetic improvements shown in Figure 1, which depicts the proof of reflexivity of parallel reduction in the two flavors.

3 Conclusions

Beluga's support for HOAS, paired with sophisticated context reasoning, makes the development of the traditional proof of CR(β) very elegant and devoid of technical lemmas foreign to the mathematics of the problem. Since specifications can be dependently-typed, the extension of the proof from the untyped to the typed case is conservative. It would be easy to encode other proof techniques such as establishing confluence for the simply typed case using Newman's lemma and existent SN proofs in BELUGA [1] or other classical results such as η-postponement, standardization, and residuals theory as in [2].

On the flip side, BELUGA (and LF) not allowing quantification over relations prevents us from a more succinct development via abstract rewriting system as per Nipkow's account, see the repetitions around the Commutation lemma. Combining natively HOAS and a full Agda-like type theory is under active research [21]. Reader, you may expect more Church-Rosser proofs in the future.

Acknowledgments Thanks to Brigitte Pientka for her help with the development.

References

[1] Andreas Abel, Guillaume Allais, Aliya Hameer, Brigitte Pientka, Alberto Momigliano, Steven Schäfer & Kathrin Stark (2019): *POPLMark reloaded: Mechanizing proofs by logical relations*. *J. Funct. Program.* 29, p. e19, doi:10.1017/S0956796819000170.

[2] Beniamino Accattoli (2012): *Proof Pearl: Abella Formalization of λ-Calculus Cube Property*. In Chris Hawblitzel & Dale Miller, editors: *Certified Programs and Proofs - Second International Conference, CPP 2012, Kyoto, Japan, December 13-15, 2012. Proceedings*, *Lecture Notes in Computer Science* 7679, Springer, pp. 173–187, doi:10.1007/978-3-642-35308-6_15.

[3] Nick Benton, Chung-Kil Hur, Andrew Kennedy & Conor McBride (2012): *Strongly Typed Term Representations in Coq*. *J. Autom. Reason.* 49(2), pp. 141–159, doi:10.1007/s10817-011-9219-0.

[4] Roberto Blanco, Dale Miller & Alberto Momigliano (2019): *Property-Based Testing via Proof Reconstruction*. In Ekaterina Komendantskaya, editor: *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, ACM, pp. 5:1–5:13, doi:10.1145/3354166.3354170.

[5] Arthur Charguéraud: *Locally nameless representation with cofinite quantification*. `https://www.chargueraud.org/softs/ln`. Accessed: November 2023.

[6] Ernesto Copello, Nora Szasz & Álvaro Tasistro (2021): *Formalization of metatheory of the Lambda Calculus in constructive type theory using the Barendregt variable convention*. *Math. Struct. Comput. Sci.* 31(3), pp. 341–360, doi:10.1017/S0960129521000335.

[7] Amy P. Felty, Alberto Momigliano & Brigitte Pientka (2015): *The Next 700 Challenge Problems for Reasoning with Higher-Order Abstract Syntax Representations - Part 2 - A Survey*. *J. Autom. Reason.* 55(4), pp. 307–372, doi:10.1007/s10817-015-9327-3.

[8] Amy P. Felty, Alberto Momigliano & Brigitte Pientka (2018): *Benchmarks for reasoning with syntax trees containing binders and contexts of assumptions*. *Math. Struct. Comput. Sci.* 28(9), pp. 1507–1540, doi:10.1017/S0960129517000093.

[9] Jonathan M. Ford & Ian A. Mason (2001): *Operational Techniques in PVS - A Preliminary Evaluation*. In Colin J. Fidge, editor: *Computing: The Australasian Theory Symposium, CATS 2001, Gold Coast, Australia, January 29-30, 2001*, *Electronic Notes in Theoretical Computer Science* 42, Elsevier, pp. 124–142, doi:10.1016/S1571-0661(04)80882-X.

[10] Gérard P. Huet (1994): *Residual Theory in lambda-Calculus: A Formal Development*. *J. Funct. Program.* 4(3), pp. 371–394, doi:10.1017/S0956796800001106.

[11] Jonas Kaiser, Brigitte Pientka & Gert Smolka (2017): *Relating System F and Lambda2: A Case Study in Coq, Abella and Beluga*. In Dale Miller, editor: *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*, *LIPIcs* 84, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 21:1–21:19, doi:10.4230/LIPIcs.FSCD.2017.21.

[12] Leonidas Lampropoulos & Benjamin C. Pierce (2023): *QuickChick: Property-Based Testing in Coq*. `https://softwarefoundations.cis.upenn.edu/qc-current`. Accessed: November 2023.

[13] Dan Licata (2008): *Church-Rosser theorem for β-reduction via complete development in Twelf*. `http://twelf.org/wiki/Church-Rosser_via_complete_development`. Accessed: November 2023.

[14] James McKinna & Robert Pollack (1993): *Pure Type Systems Formalized*. In Marc Bezem & Jan Friso Groote, editors: *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, *Lecture Notes in Computer Science* 664, Springer, pp. 289–305, doi:10.1007/BFb0037113.

[15] Alberto Momigliano (2000): *Elimination of Negation in a Logical Framework*. In Peter Clote & Helmut Schwichtenberg, editors: *Computer Science Logic, 14th Annual Conference of the EACSL, Fischbachau, Germany, August 21-26, 2000, Proceedings*, *Lecture Notes in Computer Science* 1862, Springer, pp. 411–426, doi:10.1007/3-540-44622-2_28.

[16] Julian Nagele, Vincent van Oostrom & Christian Sternagel (2016): *A Short Mechanized Proof of the Church-Rosser Theorem by the Z-property for the $\lambda\beta$-calculus in Nominal Isabelle*. *CoRR* abs/1609.03139. Available at `http://arxiv.org/abs/1609.03139`.

[17] Tobias Nipkow (2001): *More Church-Rosser Proofs*. *J. Autom. Reason.* 26(1), pp. 51–66, doi:10.1023/A:1006496715975.

[18] Frank Pfenning (1992): *A Proof of the Church-Rosser Theorem and its Representation in a Logical Framework*. Technical Report, Carnegie Mellon University. Tech. Rep. CMU-CS-92-186.

[19] Brigitte Pientka: *Adaptation to Beluga of Pfenning's proof of the Church-Rosser theorem for β-reduction*. `https://github.com/Beluga-lang/Beluga/tree/master/examples/church-rosser`. Accessed: November 2023.

[20] Brigitte Pientka & Andrew Cave (2015): *Inductive Beluga: Programming Proofs*. In Amy P. Felty & Aart Middeldorp, editors: *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, *Lecture Notes in Computer Science* 9195, Springer, pp. 272–281, doi:10.1007/978-3-319-21401-6_18.

[21] Brigitte Pientka, David Thibodeau, Andreas Abel, Francisco Ferreira & Rébecca Zucchini (2019): *A Type Theory for Defining Logics and Proofs*. In: *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, IEEE, pp. 1–13, doi:10.1109/LICS.2019.8785683.

[22] Barry K. Rosen (1973): *Tree-Manipulating Systems and Church-Rosser Theorems*. *J. ACM* 20(1), pp. 160–187, doi:10.1145/321738.321750.

[23] Steven Schäfer, Tobias Tebbi & Gert Smolka (2015): *Autosubst: Reasoning with de Bruijn Terms and Parallel Substitutions*. In Christian Urban & Xingyuan Zhang, editors: *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, *Lecture Notes in Computer Science* 9236, Springer, pp. 359–374, doi:10.1007/978-3-319-22102-1_24.

[24] Peter Selinger (2013): *Lecture notes on the lambda calculus*. arXiv:0804.3434.

[25] Natarajan Shankar (1988): *A mechanical proof of the Church-Rosser theorem*. *J. ACM* 35(3), pp. 475–522, doi:10.1145/44483.44484.

[26] Masako Takahashi (1995): *Parallel Reductions in lambda-Calculus*. *Inf. Comput.* 118(1), pp. 120–127, doi:10.1006/inco.1995.1057.

[27] René Vestergaard & James Brotherston (2001): *A Formalised First-Order Confluence Proof for the lambda-Calculus Using One-Sorted Variable Names*. In Aart Middeldorp, editor: *Rewriting Techniques and Appli-*

cations, 12th International Conference, RTA 2001, Utrecht, The Netherlands, May 22-24, 2001, Proceedings, *Lecture Notes in Computer Science* 2051, Springer, pp. 306–321, doi:10.1007/3-540-45127-7_23.

A Appendix: Counterexamples Search

The theory of confluence is rife with counterexamples. It is a well-known fact that (single step) β-reduction does *not* satisfy the diamond property, since redexes can be discarded or duplicated, and this is why the notion of parallel reduction is so useful. It would be nice, both for research and educational purposes, for a proof environment to assist us in *refuting* unprovable conjectures and witnessing such counter-examples. A lightweight approach to this endeavor is *property-based testing* (PBT), see for example the integration of *QuickChick* within Coq [12]. If we view a property as a logical formula $\forall x : \tau.P(x) \supset Q(x)$, providing a counter-example consists of negating the property, and searching for a proof of $\exists x : \tau.P(x) \wedge \neg Q(x)$. This points to a logic programming solution, where the specification is a fixed set of assumptions and the negated property is the goal. A full proof-theoretic reconstruction of PBT has been presented in [4] and can be adapted to BELUGA, which in the Twelf's tradition has a logic programming engine built-in.

To exemplify, let us search for a counter-example to diamond(β). First, we need to state the conjecture that we want to test, namely the negation of $M_1 \leftarrow M \rightarrow M_2$ entails $\exists N$, $M_1 \rightarrow N \leftarrow M_2$. This needs a little care, since BELUGA does not have negation — the usual solution in logic programming, i.e. *negation-as-failure*, is incompatible with BELUGA's foundations: indeed, what could be the proof term witnessing a proof failure? One solution [15] is to state in the positive when two terms are *non*-joinable: i.e., when they are different and if they reduce in one step, they do not reduce to the same term. This requires a notion of *inequality* for lambda terms (simplified here from the actual code):

```
LF diff : term → term → type =
 | dal : diff (lam _) (app _ _)
 | dla : diff (app _ _) (lam _)
 | da1 : diff E1 F1 → diff (app E1 E2) (app F1 F2)
 | da2 : diff E2 F2 → diff (app E1 E2) (app F1 F2)
 | dll : ({x:term} → diff (M x) (N x)) → diff (lam M) (lam N);

LF not_joinable : term → term → type =
| nj : diff M1 M2 → step M1 P1 → step M2 P2 → diff P1 P2 → not_joinable M1 M2;
```

The second ingredient is a *generator* for terms. For the sake of this paper we do not implement the full architecture of [4], but simply program an exhaustive height-bounded term generator. We also show the test harness predicate, combining generation with the to-be-tested conjecture:

```
LF heigth : nat → term → type =
 | h1 : heigth H M → heigth H N → heigth (s H) (app M N)
 | h2 : ({x:term} ({h:nat} heigth h x) → heigth H (M x)) → heigth (s H) (lam M);

LF gencex : nat → term → term → term → type =
| cx : not_joinable M1 M2 → step M M1 → step M M2 → hei I M → gencex I M M1 M2;
```

A query to BELUGA's logic programming engine with a bound of 3 will generate the following (pretty-printed) counterexample to diamond(β), namely `M` $=(\lambda x.\, x\, x)(I\, I)$, for I the identity combinator. In fact, it steps to $(I\, I)(I\, I)$ and $(\lambda x.\, x\, x)\, I$.

Another application is witnessing the failure of diamond(η) in a typed calculus with unit and surjective pairing. Here we exploit intrinsically-typed terms to encode typed η-reduction and obtain well-typed term generators for free. All the details in the repository, under directory `PBT`.