More Church-Rosser Proofs (and Refutations) in Beluga

Alberto Momigliano

joint work with Martina Sassella
LSFA 2023

One of the most formalized proof in PL theory

Author	Red	Sys	Style	Aut
Shankar 88	β	Boyer-Moore	DB	2
Huet 94	residuals	Coq	DB	1
Nipkow 01	$\beta\eta$	Isabelle/HOL	DB	2
McKinna et al 98	β/PTS	Lego	LN	2
Chargeraud 09	β	Coq	LN	1
Smolka et al. 15	β	Coq	AutoSubst	2
Brotherston et al. 03	β	Isabelle/HOL	named	1
Ford et al. 01	β	PVS	α-quotient	1
Urban et al. 05	β	Nominal Isabelle	nominal	1
Nagele et al. 17	β	Nominal Isabelle	nominal	1
Gheri et al. 21	β	Isabelle/HOL	nominal	1
Coppello al. 17	β	Agda	BVC	0
Pfenning 92	β	Elf	HOAS	0
Accattoli 12	residuals	Abella	HOAS	1
...				

Aut(omation): 0 – 2 (2, higher), BVC = Barendregt's convention

Why, oh why another HOAS formalization and why in Beluga?

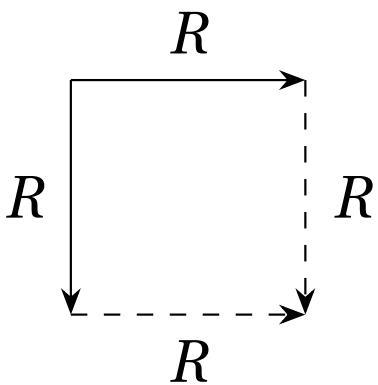
- Part of a more general project of developing a curriculum to teach the classic theory of the lambda-calculus to Master students, who can't do a β-reduction
  - Using a proof assistant in the spirit of UPenn's *Software Foundations*
  - also because (sue me) Barendregt's bible is unreadable
- Go beyond β: consider η and extension to typed calculi
- Complement proofs with *refutations*, i.e., searching for counterexamples, in the spirit of Coq/QuickChick

Why, oh why another HOAS formalization and why in Beluga?

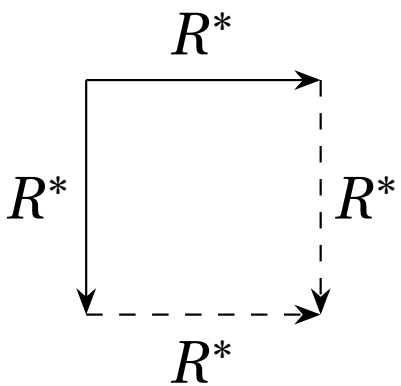
- Part of a more general project of developing a curriculum to teach the classic theory of the lambda-calculus to Master students, who can't do a β-reduction
  - Using a proof assistant in the spirit of UPenn's *Software Foundations*
  - also because (sue me) Barendregt's bible is unreadable
- Go beyond β: consider η and extension to typed calculi
- Complement proofs with *refutations*, i.e., searching for counterexamples, in the spirit of Coq/QuickChick
- With a caveat: we (the authors) have
  - Little understanding of BELUGA's meta-theory
  - Only basic knowledge of that very classic theory of the lambda-calculus that we'd like to teach
  - . . . but we know a thing or two about encoding formal systems with binders.

Abstract reduction systems

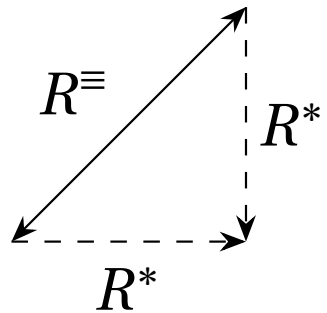
- (A, R): A set, $R \subseteq A \times A$ (think β)
- R^*: reflexive and transitive closure R (think β multi-step)
- $R^{\equiv}$: least equivalence containing R (think β conversion)



diamond



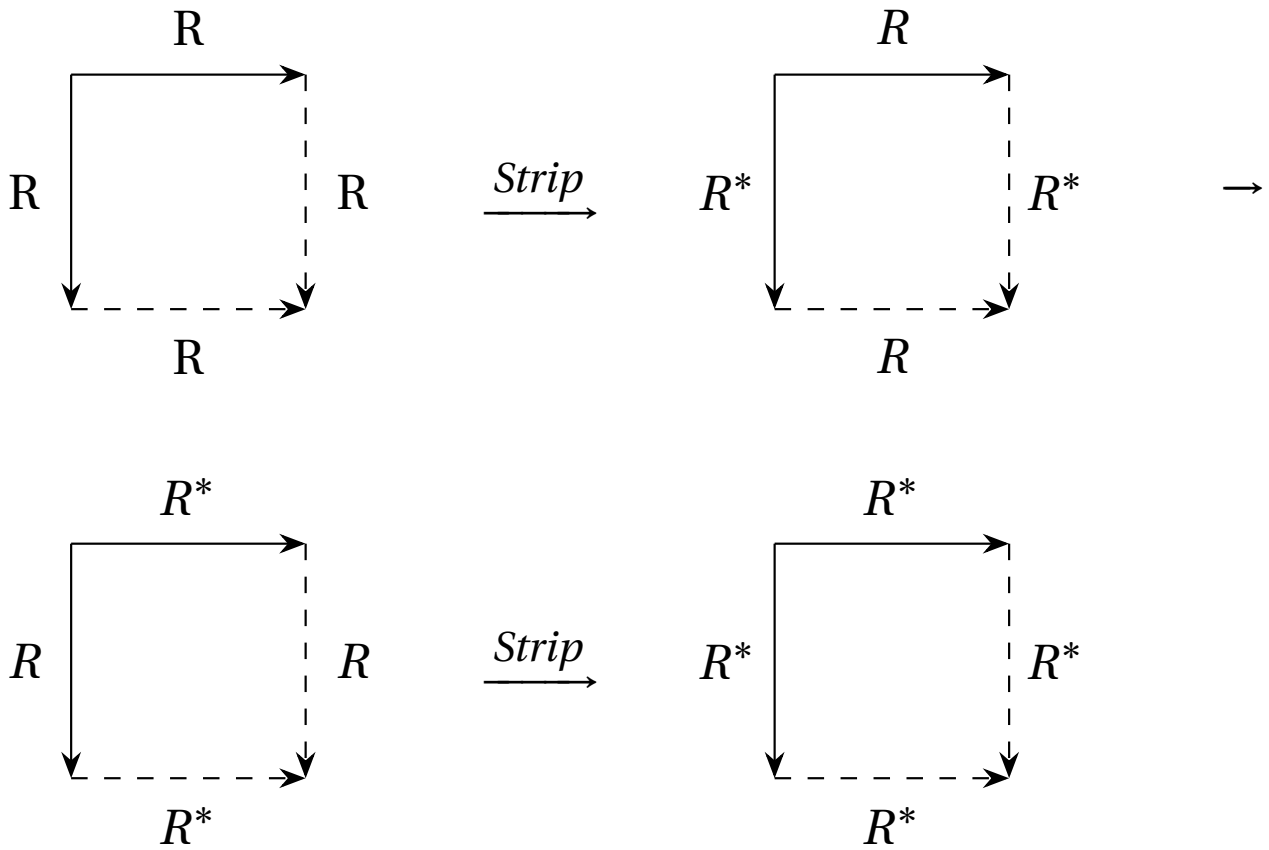
confluence



Church-Rosser

Classic proof strategy for confluence

- diamond $\longrightarrow$ confluence



- confluence $\longleftrightarrow$ CR

Let there be Parallel Reduction

- β does not enjoy diamond
- Idea: find a reduction relation S that does and show that $S^* = \beta^*$

$$\frac{M_1 \Longrightarrow M_1' \qquad M_2 \Longrightarrow M_2'}{(\lambda x.M_1)M_2 \Longrightarrow M_1'[M_2'/x]}\, beta$$

$$\frac{M \Longrightarrow M'}{\lambda x.M \Longrightarrow \lambda x.M'}\, lm$$

$$\frac{M_1 \Longrightarrow M_1' \qquad M_2 \Longrightarrow M_2'}{M_1\, M_2 \Longrightarrow M_1'\, M_2'}\, ap$$

$$\frac{}{x \Longrightarrow x}\, var$$

While contracting a redex, also reduce the subterms, and so for congruence rules

What we have proved

- CR($\boldsymbol{\beta}$) for the untyped lambda calculus
  - direct proof of diamond for parallel reduction following [Pfenning 92]
    1. with relations specified at the LF level
    2. with relations specified in Beluga's meta-level
  - using Takahashi's complete developments
- CR($\boldsymbol{\eta}$) for the untyped lambda calculus via the *Commutation* lemma
- CR($\boldsymbol{\beta\eta}$) for the untyped lambda calculus via the *Commutative Union* lemma.
- CR($\boldsymbol{\beta}$) for typed calculi: simple types and System F, both via complete developments.

Why didn't you . . .

- use a mainstream proof assistant?
  - It's been done to death
  - Binders are still problematic, HOAS only partially supported
  - In curricula such as SF, you end up teaching 80% Coq, 20% PL theory
  - We like proof terms and dependent types w/o the baggage
- just stick to complete developments for $\beta\eta$ as well?
  - There is value in having confluence of $\beta\eta$ emerge from confluence of each relations separately.
  - It also helps in other results such as η-postponement (still to do)

HOAS

- I assume you have some familiarity with the idea of higher-order abstract syntax

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

a term $\lambda x.x$ is represented by a LF term `(lam \x.x)`

HOAS

- I assume you have some familiarity with the idea of higher-order abstract syntax

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

  a term $\lambda x.x$ is represented by a LF term `(lam \x.x)`

- A relation such as parallel reduction is represented with a LF type family with essential use of parametrical-hypothetical judgments:

```
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')
```

  Note again: no case for vars

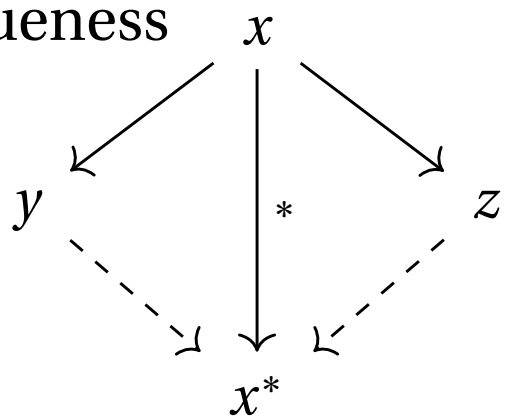
CR(β)

Brief highlights, since we follow existing encodings

- **No** technical lemmas about variables, renamings etc.
- We do have to prove that parallel reduction is stable under substitution, but it's from first principles:

```
rec subst: (g:rctx)[g,x:term,u:pred x x |- pred M[..,x] M'[..,x]]
          -> [g |- pred N N'] -> [g |- pred M[..,N] M'[..,N']] = ..
```

- Direct proof of diamond for `pred` has a complex case-analysis, just like the on paper proof
- Taka's proof goes through a **relational** encoding of complete developments,which is fine as we only need totality, not uniqueness

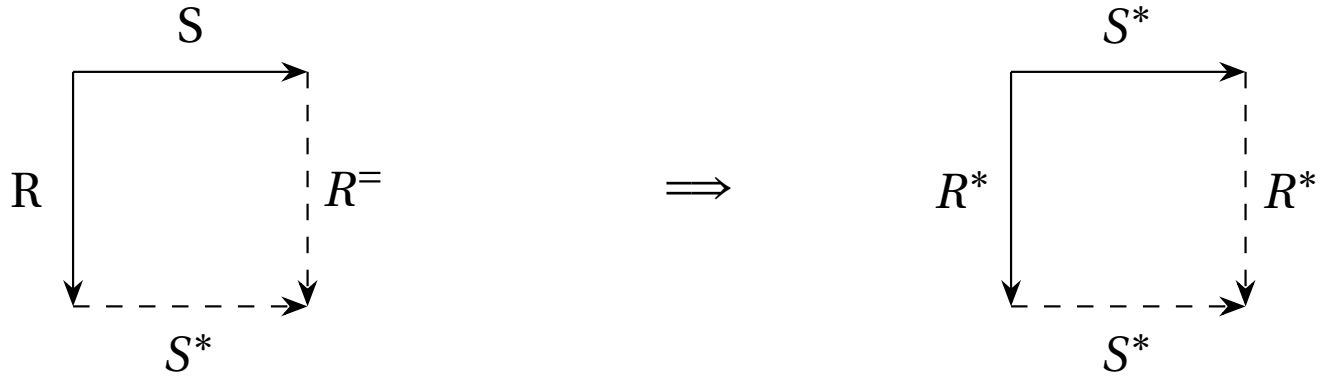


CR(η)

We consider η-reduction in isolation and apply:

Lemma (Commutation – Hidley-Rosen)

Two strongly commuting reductions commute.



Since both LF and BELUGA are (roughly) first order type theories, we can't work on ARS and we instantiate here R and S to $\longrightarrow_\eta$:

- We show that $\longrightarrow_\eta$ strongly commutes with itself
- We (re)prove Commutation for this instance

Encoding of CR(η)

- On the bright size, encoding the proviso in η-reduction is immediate within HOAS as the meta-variable `M` *not* depending on `x` in the LF function `\x.(app M x)`.

```
LF eta_red : term -> term -> type =
| eta : eta_red (lam \x.(app M x)) M
% .. congruence rules omitted
```

- This "non"-occurrence yields **one** technical lemma:

Lemma (Strengthening)

If $\Gamma, x \vdash M \longrightarrow_\eta N$, *and* M *does not depend on* x, *neither does* N *and* $\Gamma \vdash M \longrightarrow_\eta N$.

CR($\beta\eta$)

Since we have confluence for β and η separately, we combine them via the:

Lemma (Commutative Union)

If R and S are confluent and commute, then $R \cup S$ is confluent.

Still some work to do among which:

- strengthening for β
- substitution for η and η^*
- instances of the Commutation and Strip lemmas
- diamond for $\beta^* \cup \eta^*$

System F and its intrinsic encoding

- Similar definitions w.r.t. CR, but reduce only well-typed terms

$$A ::= \alpha \mid A \to B \mid \forall \alpha.A$$
$$M ::= x \mid \alpha \mid M\,N \mid \lambda x^{A}.M \mid M\,A \mid \Lambda \alpha.M$$

- Thanks to dependent types, we can use intrinsically typed terms and index judgments via their object types – this is well-known (see POPLMark reloaded)

```
LF ty : type =
 | arr : ty -> ty -> ty
 | all : (ty -> ty) -> ty
```

```
LF tm : ty -> type =
| abs : (tm A -> tm B) -> tm (arr A B
| app : tm (arr A B) -> tm A -> tm B
| tlam : ({a:ty} tm (A a)) -> tm (all
| tapp : tm (all A) -> {B:ty} tm (A B
```

System F and its intrinsic encoding

- Similar definitions w.r.t. CR, but reduce only well-typed terms

$$A ::= \alpha \mid A \to B \mid \forall \alpha.A$$
$$M ::= x \mid \alpha \mid M\,N \mid \lambda x^{A}.M \mid M\,A \mid \Lambda \alpha.M$$

- Thanks to dependent types, we can use intrinsically typed terms and index judgments via their object types – this is well-known (see POPLMark reloaded)

```
LF ty : type =
 | arr : ty -> ty -> ty
 | all : (ty -> ty) -> ty
```

```
LF tm : ty -> type =
| abs : (tm A -> tm B) -> tm (arr A B
| app : tm (arr A B) -> tm A -> tm B
| tlam : ({a:ty} tm (A a)) -> tm (all
| 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)
 | tbeta : ({a:ty} pred (M1 a) (M1' a))
                -> pred (tapp (tlam M1) A) (M1' A) ..
```

System F: Spot the difference

Reflexivity of parallel reduction in the untyped case

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

rec rpar: {g:rctx}{M: [g |- term]}[g |- pred M M] =
mlam g => mlam M => case [g |- M] of
| [g |- #p.1] => [g |- #p.2]
| [g |- 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
    [g |- lm \x.\v.IH[..,x,v]]
| [g |- app M1 M2] =>
   let [g |- IH1] = rpar [g] [g |- M1] in
   let [g |- IH2] = rpar [g] [g |- M2] in
   [g |- ap IH1 IH2]
```

System F: Spot the difference, ctd.

Reflexivity of parallel reduction in System F

```
schema pctx = some [A:ty] block(x:tm A, v:pred x x) + ty
rec rpar : {g:pctx}{M : [g |- tm A]}[g |- pred M M] =
mlam g => mlam M => case [g |- M] of
| [g |- #p.1] => [g |- #p.2]
| [g |- abs \x.M'[..,x]] =>
    let [g, b:block(x:tm _,v:pred x x) |- IH[..,b.1,b.2]] =
      rpar [g, b:block(x:tm _,v:pred x x)] [g,b |- M'[..,b.1]] in
      [g |- lm \x.\v.IH[..,x,v]]
| [g |- app M1 M2] =>
  let [g |- IH1] = rpar [g] [g |- M1] in
  let [g |- IH2] = rpar [g] [g |- M2] in
  [g |- ap IH1 IH2]
```

System F: Spot the difference, ctd.

Reflexivity of parallel reduction in System F

```
schema pctx = some [A:ty] block(x:tm A, v:pred x x) + ty
rec rpar : {g:pctx}{M : [g |- tm A]}[g |- pred M M] =
mlam g => mlam M => case [g |- M] of
| [g |- #p.1] => [g |- #p.2]
| [g |- abs \x.M'[..,x]] =>
    let [g, b:block(x:tm _,v:pred x x) |- IH[..,b.1,b.2]] =
      rpar [g, b:block(x:tm _,v:pred x x)] [g,b |- M'[..,b.1]] in
      [g |- lm \x.\v.IH[..,x,v]]
| [g |- app M1 M2] =>
  let [g |- IH1] = rpar [g] [g |- M1] in
  let [g |- IH2] = rpar [g] [g |- M2] in
  [g |- ap IH1 IH2]

| [g |- tlam \a.M'] => % new cases
  let [g, a:ty |- IH] =
    rpar _ [g, a:ty |- M'] in  [g |- tlm \a.IH]
| [g |- tapp M' A] =>
  let [g |- IH] =
   rpar _ [g |- M'] in   [g |- tap IH]
```

Conservativity

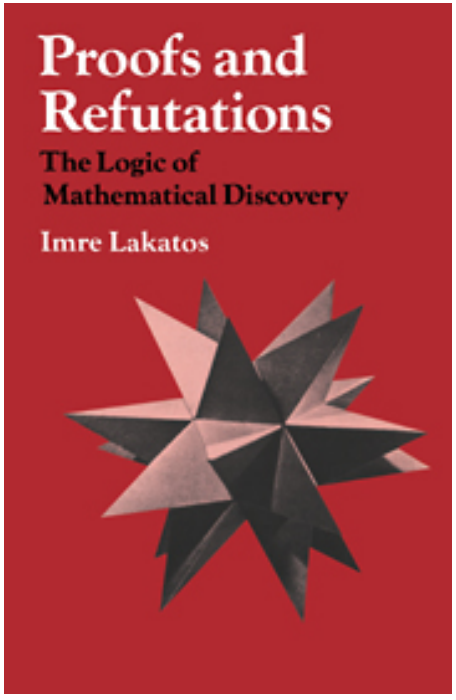
- The proof of CR for the simply typed lambda calculus is literally the same w.r.t. the untyped case, modulo indexing the terms with object types and making explicit some implicit arguments
- The proof for System F simply add cases for the new constructors
- Isn't this remarkable? Is this phenomenon of embedding via indexing common in other instaces?

CR(β) at the meta-level

Forget about it (or ask me after the talk)

Meta-theory model-checking

- People more authoritative than me have argued about the positive interplay of searching for proofs and counterexamples in the same setting
  - Counterexamples are not a final refutations for a theory, but a principled way to refine conjectures and proofs thereof.
- The theory of confluence is rife with counterexamples.
  - β-reductions does not have diamond
  - the same for η in a typed calculus with units and pairs
- Sure, those are well-known, but many others can lurk around



On the proof-theory of property-based testing

- On lightweight idea is PBT and one way to look at it is via proof-theory
- For a property $\forall x : \tau . P(x) \supset Q(x)$, 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 can be achieved via logic programming search, and luckily Beluga has such an engine.
- A general view of the proof-theory of PBT can be carried out in Miller's foundational proof certificates framework [Blanco et al. 19]

Failure of diamond(β)

- State the property:

```
LF not_joinable : term -> term -> type =
| nj : diff M1 M2 -> step M1 P1 -> step M2 P2
          -> diff P1 P2 -> not_joinable M1 M2

LF gencex : nat -> term -> term -> term -> type =
| cx : not_joinable M1 M2 -> step M M1 -> step M M2
   -> height I M -> gencex I M M1 M2
```

- Write a **generator** of lambda terms – here we use an exhaustive enumerator bounded by the `height` of the AST
- Run the query: for β, you get `M` $=(\lambda x.\ x\ x)(I\ I)$, for I the identity combinator, which steps to $(I\ I)(I\ I)$ and $(I\ I)$.

Conclusions

- Church-Rosser, 35 five years in, is still an interesting benchmark for formalization
- Beluga's support for HOAS and context reasoning gives very elegant proofs, with almost no technical lemmas foreign to the mathematics of the problem.
- Intrinsically-typed encodings are good
- Not being able to quantify over relations prevents us from exploiting a general theory of ARS
  $\Rightarrow$ tedious repetition while instantiating general lemmas
- Counter-example search in the same setting is also good.

Future work

- easy: confluence for simply typed using Newman's Lemma (SN & locally confluent entails confluence), using existent BELUGA SN proof
- easy: other results from Taka's & Accattoli's paper: η-postponement, standardization, residuals
- medium: CR(β) for PTS – easy untyped, less so intrinsically-typed
- ? CR for infinitary calculi, Bohm's tree (coinduction)

That's all, folks!