

The Church-Rosser Theorem in Higher Order Logic



Peter Vincent Homeier
U.S. Department of Defense
homeier@saul.cis.upenn.edu

Mechanical Proofs of Church-Rosser Theorem

Shankar	1988	Boyer-Moore	name-carrying	α and β mixed	
Pfenning	1992	Elf (LF)	de Bruijn		
Huet	1994	Coq	de Bruijn		
Rasmussen	1995	Isabelle-ZF	de Bruijn		
Nipkow	1996	Isabelle-HOL	de Bruijn		η - and $\eta\beta$ -reduction
Vestergaard/ Brotherston	2001	Isabelle-HOL	name-carrying	α and β mixed	
Ford / Mason	2001	PVS	name-carrying	quotients	
Homeier	2001	HOL	name-carrying	quotients	η - and $\eta\beta$ -reduction

2

Pre-Lambda Calculus Syntax

Name-carrying λ -calculus syntax:

variables (var): x, y, z, \dots

terms (term_I): $\Lambda_I ::= \text{var} \mid \Lambda_I \Lambda_I \mid \lambda \text{var} . \Lambda_I$
(variable, application, abstraction)

substitutions (subst_I): $\Sigma_I ::= [] \mid (\text{var} := \Lambda_I) :: \Sigma_I$
(nil, cons of (var, term) pair) - a *simultaneous* substitution

Typical meta-variables of types: term: t, u, M, N, L subst: s var set: r

```
val _ = Hol_datatype
  ` term1 = Var1 of var
    | App1 of term1 => term1
    | Lam1 of var => term1 ` ;
```

Hol98 automatically proves term 1) structural induction, 2) function existence, 3) cases, 4) constructors distinctiveness, and 5) constructors one-to-one

Functions on Pre-Lambda Calculus Syntax

Functions on λ -calculus syntax:

$\text{HEIGHT}_I: \Lambda \rightarrow \text{num}$ Height of term, var is 0, else 1+components

$\text{FV}_I: \Lambda \rightarrow \text{var set}$ Set of free variables of term

$_{-}\triangleleft_I^v_{-}: \text{var} \rightarrow \Sigma \rightarrow \Lambda$ Application of a substitution to a variable

$_{-}\triangleleft_I_{-}: \Lambda \rightarrow \Sigma \rightarrow \Lambda$ **Proper** application of a substitution to a term

HEIGHT_I and FV_I are defined by primitive recursion on the structure of terms

$_{-}\triangleleft_I^v_{-}$ is defined by list recursion on the structure of the substitution

$_{-}\triangleleft_I_{-}$ is defined by primitive recursion on the structure of terms, making use of the simultaneous substitution to add new bindings to properly avoid capture.

4

Substitution

Definition of substitution: (Complete)

$$\begin{aligned}x \triangleleft_I s &= x \triangleleft_I^v s \\(t \ u) \triangleleft_I s &= (t \triangleleft_I^v s) (u \triangleleft_I^v s) \\(\lambda x. t) \triangleleft_I s &= \mathbf{let} \ x' = \mathbf{variant} \ x \ (\mathbf{FVsubst}_I \ s \ (\mathbf{FV}_I \ t - \{x\})) \ \mathbf{in} \\&\quad \lambda x'. (t \triangleleft_I ((x := x') :: s))\end{aligned}$$

where

$$\begin{aligned}\mathbf{FVsubst}_I \ s \ r &= \bigcup (\mathbf{image} (\mathbf{FV}_I \circ \mathbf{SUB}_I \ s) \ r) \\ \mathbf{SUB}_I \ s \ x &= x \triangleleft_I^v s\end{aligned}$$

“Naïve” substitution is easy and simple but NOT CORRECT:

$$(\lambda x. t) \triangleleft_I s = \lambda x. (t \triangleleft_I s)$$

Constructors One-to-One Property

Almost right, but constructors one-to-one property says that

$$(\lambda x_1. t_1 = \lambda x_2. t_2) \Leftrightarrow (x_1 = x_2) \wedge (t_1 = t_2)$$

But we want, for example, $\lambda x. x = \lambda y. y$. Just which name is used for the variable should be immaterial, as long as names are changed consistently.

This one-to-one property is too discriminating. We want to create a variant of this calculus to blur such distinctions.

The exact blurring we wish is called *alpha-equivalence*.

6

Alpha-Equivalence

- Church represented as *semantic reduction*: $t \rightarrow_{\alpha} t'$
- More modern approach (Barendregt, Abadi/Cardelli, ...) is to *identify* equivalent terms at *syntactic* level
- Alpha-equivalence: relation on terms; e.g., $\lambda x. x \equiv_{\alpha} \lambda y. y$.
- Design issue: How to define \equiv_{α} ?
 - Others used substitution (\triangleleft_1); is it deceptively simple?
 - We used *contextual alpha-equivalence*, where the contexts are lists of variables denoting bindings present

7

Pure Lambda Calculus

- Pure lambda calculus formed as quotient of pre-lambda calculus by alpha-equivalence:

$$\Lambda = \Lambda_1 / \equiv_\alpha$$

- New type “term” made by new HOL package for quotients
- Produces two mapping functions between term and term1:

$$\begin{aligned} \llbracket _ \rrbracket : \Lambda_1 &\rightarrow \Lambda & \lceil _ \rceil : \Lambda &\rightarrow \Lambda_1 \\ \forall a. \llbracket \lceil a \rceil \rrbracket &= a & \wedge & \forall r r'. r \equiv_\alpha r' \Leftrightarrow (\llbracket r \rrbracket = \llbracket r' \rrbracket) \end{aligned}$$

- Term constructor functions redefined in Λ using map fns
E.g., $\text{Lam } x \ t = \llbracket \text{Lam}_1 \ x \ \lceil t \rceil \rrbracket$, which is $\lambda x. t = \llbracket \lambda x. \lceil t \rceil \rrbracket$

8

Recreating Function Definitions in the Pure Lambda Calculus

- Functions are defined first in Λ_1 and then recreated in Λ
- BUT, not *every* function definable in Λ_1 can be recreated!
- Functions must respect alpha-equivalence, e.g.,

$$t_1 \equiv_{\alpha} t_2 \Rightarrow \text{FV}_1 t_1 = \text{FV}_1 t_2$$

$$t_1 \equiv_{\alpha} t_2 \wedge s_1 \equiv_{\alpha}^{\text{subst}} s_2 \Rightarrow (t_1 \triangleleft_1 s_1) \equiv_{\alpha} (t_2 \triangleleft_1 s_2)$$

- 1) Prove function respects alpha-equivalence (arb. complex)
- 2) Define new function using $\lfloor _ \rfloor$ and $\lceil _ \rceil$
- 3) Prove as theorem in Λ the same form as definition in Λ_1

9

Recreated Properties in the Pure Lambda Calculus

- Now we have the one-to-one property
 $(\lambda x_1.t_1 = \lambda x_2.t_2) \Leftrightarrow (t_1 \triangleleft [x_1 := x_2] = t_2) \wedge (t_2 \triangleleft [x_2 := x_1] = t_1)$
- All other properties and definitions of Λ_1 are recreated in Λ , *except* for function existence
- More general term height induction principle:

$$\begin{aligned} &\vdash \forall P. (\forall x. P x) \wedge \\ &\quad (\forall t u. P t \wedge P u \Rightarrow P (t u)) \wedge \\ &\quad (\forall t. (\forall t'. \text{HEIGHT } t = \text{HEIGHT } t' \Rightarrow P t') \Rightarrow \forall x. P (\lambda x. t)) \\ &\quad \Rightarrow \\ &\quad (\forall t. P t) \end{aligned}$$

Barendregt Variable Convention (BVC)

- Barendregt's *Lambda Calculus: It's Syntax and Semantics*
- The BVC states that in any proof, one can assume that all bound variables are different from all free variables
- Then substitution is simple (naïve), and proofs are elegant
- Controversial; some have suggested the BVC is unsound
- We have found a mechanization within the security of HOL that (partially) justifies the BVC —
A new HOL tactic to shift abstractions away from capture, used along with height-based induction

11

Semantics of Reduction in Lambda Calculus

- Define β as relation on terms such that for all $M, N \in \Lambda$,

$$\beta ((\lambda x. M) N) (M \triangleleft [x := N])$$

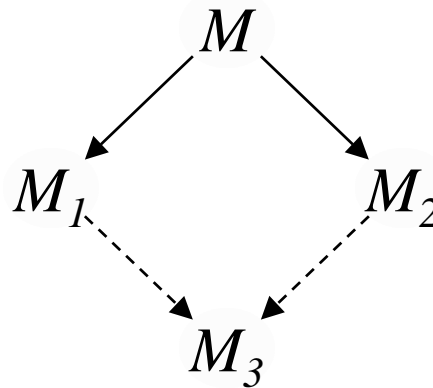
- A relation R on Λ is *compatible* (with the operations) if for all $M, M', Z \in \Lambda, x \in \text{var}$,

$$R M M' \Rightarrow R(Z M)(Z M') \wedge R(M Z)(M' Z) \wedge R(\lambda x.M)(\lambda x.M')$$

- Given relation R , R induces reduction relations:
 - \rightarrow_R one step R -reduction compatible closure of R
 - \twoheadrightarrow_R R -reduction reflexive, transitive closure of \rightarrow_R
 - $=_R$ R -equality equivalence relation generated by \twoheadrightarrow_R

Diamond Property and Church-Rosser Property

- \rightarrow satisfies diamond property ($\rightarrow \models \diamond$) if
$$\forall M M_1 M_2. M \rightarrow M_1 \wedge M \rightarrow M_2 \Rightarrow \exists M_3. M_1 \rightarrow M_3 \wedge M_2 \rightarrow M_3$$



- R is Church-Rosser if $\rightarrow_R \models \diamond$; we want to prove $\rightarrow_\beta \models \diamond$

13

The Church-Rosser Theorem

- Original by Church-Rosser (1936); Schroer (1965) 627 pgs
- Greatly simplified proof found by Martin-Löf (1972), based on ideas of Tait
- Elegant presentation by Barendregt (1981) using the BVC
- Define parallel reduction (\twoheadrightarrow) inductively by the rules

$$\frac{}{M \twoheadrightarrow M}$$

$$\frac{M \twoheadrightarrow M' , N \twoheadrightarrow N'}{M N \twoheadrightarrow M' N'}$$

$$\frac{M \twoheadrightarrow M'}{\lambda x. M \twoheadrightarrow \lambda x. M'}$$

$$\frac{M \twoheadrightarrow M' , N \twoheadrightarrow N'}{(\lambda x. M) N \twoheadrightarrow M' \triangleleft [x := N']}$$

Proof of the Church-Rosser Theorem

- **Theorem:** For all relations \succrightarrow , $\succrightarrow \vDash \diamond \Rightarrow \succrightarrow^* \vDash \diamond$
- **Theorem:** \twoheadrightarrow satisfies the diamond property ($\twoheadrightarrow \vDash \diamond$)
- **Theorem:** \twoheadrightarrow_β is the transitive closure of \twoheadrightarrow ($\twoheadrightarrow_\beta = \twoheadrightarrow^*$)

- **Theorem:** β is Church-Rosser.
Proof: by definition of Church-Rosser and above theorems

HOL Proof of the Church-Rosser Theorem

- 7 main HOL theories (+ 2 auxiliary)
- 3 new types
- 77 new definitions
- 359 theorems proved
- 0 new axioms (secure, conservative extension of HOL)
- 16,735 lines of Standard ML code (including comments)

All theory scripts and associated code, including the new quotient library and mutual recursion tools, are available at

<http://www.cis.upenn.edu/~hol/lamcr/>

16

Conclusions

- Name-carrying syntax is closer to normal programming languages - an issue worth studying.
- Alpha-equivalence and beta-reduction analyzed in two distinct layers.
- Easily extended to η - and $\beta\eta$ -reduction in four days.
- Greater simplicity in proof was enabled by a separation of concerns.
- Natural example of the use of the quotient package.
- *Soli Deo Gloria.*