The Church-Rosser Theorem in Higher Order Logic

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

9/10/01
### Mechanical Proofs of Church-Rosser Theorem

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>System</th>
<th>Reduction Type</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shankar</td>
<td>1988</td>
<td>Boyer-Moore</td>
<td>name-carrying</td>
<td>(\alpha) and (\beta) mixed</td>
</tr>
<tr>
<td>Pfenning</td>
<td>1992</td>
<td>Elf (LF)</td>
<td>de Bruijn</td>
<td></td>
</tr>
<tr>
<td>Huet</td>
<td>1994</td>
<td>Coq</td>
<td>de Bruijn</td>
<td></td>
</tr>
<tr>
<td>Rasmussen</td>
<td>1995</td>
<td>Isabelle-ZF</td>
<td>de Bruijn</td>
<td></td>
</tr>
<tr>
<td>Nipkow</td>
<td>1996</td>
<td>Isabelle-HOL</td>
<td>de Bruijn</td>
<td>(\eta)- and (\eta\beta)-reduction</td>
</tr>
<tr>
<td>Vestergaard/Brotherston</td>
<td>2001</td>
<td>Isabelle-HOL</td>
<td>name-carrying</td>
<td>(\alpha) and (\beta) mixed</td>
</tr>
<tr>
<td>Ford / Mason</td>
<td>2001</td>
<td>PVS</td>
<td>name-carrying</td>
<td>quotients</td>
</tr>
<tr>
<td>Homeier</td>
<td>2001</td>
<td>HOL</td>
<td>name-carrying</td>
<td>quotients</td>
</tr>
</tbody>
</table>
Pre-Lambda Calculus Syntax

Name-carrying λ-calculus syntax:

variables (var): $x, y, z, ...$

terms (term): $\Lambda_i ::= \text{var} \mid \Lambda_i \Lambda_i \mid \lambda \text{var} \Lambda_i$

(variable, application, abstraction)

substitutions (subst): $\Sigma_i ::= [] \mid (\text{var} := \Lambda_i) :: \Sigma_i$

(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 \( \lambda \)-calculus syntax:

\[
\begin{align*}
\text{HEIGHT}_i &: \Lambda \rightarrow \text{num} \quad \text{Height of term, var is 0, else 1+components} \\
\text{FV}_i &: \Lambda \rightarrow \text{var set} \quad \text{Set of free variables of term} \\
_\cdot \langle \cdot ; \cdot \rangle &: \text{var} \rightarrow \Sigma \rightarrow \Lambda \quad \text{Application of a substitution to a variable} \\
_\cdot \langle \cdot \rangle &: \Lambda \rightarrow \Sigma \rightarrow \Lambda \quad \text{Proper application of a substitution to a term}
\end{align*}
\]

\( \text{HEIGHT}_i \) and \( \text{FV}_i \) are defined by primitive recursion on the structure of terms
\( \langle \cdot ; \cdot \rangle \) is defined by list recursion on the structure of the substitution
\( \langle \cdot \rangle \) is defined by primitive recursion on the structure of terms, making use of the simultaneous substitution to add new bindings to properly avoid capture.
Definition of substitution: (Complete)

\[ x \triangleleft_I s = x \triangleleft_I^v s \]

\[ (t \ u) \triangleleft_I s = (t \triangleleft_I^v s) \ (u \triangleleft_I^v s) \]

\[ (\lambda x. \ t) \triangleleft_I s = \text{let } x' = \text{variant } x \ (FV\text{subst}_I \ s \ (FV_I t - \{x\}) \ ) \text{ in } \\
\lambda x'. \ ((t \triangleleft_I ((x := x') :: s)) \]

where

\[ FV\text{subst}_I s r = \bigcup (\text{image } (FV_I \circ \text{SUB}_I s) r) \]

\[ \text{SUB}_I s x = x \triangleleft_I^v s \]

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

\[ (\lambda x. \ t) \triangleleft_I s = \lambda x. \ (t \triangleleft_I s) \]

9/10/01
Almost right, but constructors one-to-one property says that
\[(\lambda x_1. \ t_1 = \lambda x_2. \ t_2) \iff (x_1 = x_2) \land (t_1 = t_2)\]
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 \textit{alpha-equivalence}. 
Alpha-Equivalence

• Church represented as semantic reduction: \( t \rightarrow_{\alpha} t' \)
• More modern approach (Barendreght, 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 \( \equiv_{\alpha} \)?
  – Others used substitution (\( \triangleleft_{l} \)); is it deceptively simple?
  – We used contextual alpha-equivalence, where the contexts are lists of variables denoting bindings present
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:
  \[ \llbracket _ \rrbracket : \Lambda_1 \rightarrow \Lambda \quad \llbracket _ \rrbracket : \Lambda \rightarrow \Lambda_1 \]
  \[ \forall a. \llbracket a \rrbracket = a \quad \land \quad \forall r r'. r \equiv_\alpha r' \iff (\llbracket r \rrbracket = \llbracket r' \rrbracket) \]
- Term constructor functions redefined in \( \Lambda \) using map fns
  E.g., \( \text{Lam} \ x \ t = \llbracket \text{Lam}_1 \ x \ \llbracket t \rrbracket \), which is \( \lambda x. \ t = \llbracket \lambda x. \ \llbracket t \rrbracket \)
Recreating Function Definitions in the Pure Lambda Calculus

• Functions are defined first in $\Lambda_1$ and then recreated in $\Lambda$.
• BUT, not every function definable in $\Lambda_1$ can be recreated!
• Functions must respect alpha-equivalence, e.g.,

\[
\begin{align*}
t_1 & \equiv_\alpha t_2 \Rightarrow \text{FV}_1 t_1 = \text{FV}_1 t_2 \\
\text{FV}_1 t_i & \equiv_\alpha \text{FV}_1 t_j \\
t_1 & \equiv_\alpha t_2 \land s_1 \equiv_\alpha^\text{subst} s_2 \Rightarrow (t_1 \triangleleft_1 s_1) \equiv_\alpha (t_2 \triangleleft_1 s_2)
\end{align*}
\]

• 1) Prove function respects alpha-equivalence (arb. complex)
• 2) Define new function using $\llbracket \_ \rrbracket$ and $\llbracket \_ \rrbracket$
• 3) Prove as theorem in $\Lambda$ the same form as definition in $\Lambda_1$
Recreated Properties in the Pure Lambda Calculus

• Now we have the one-to-one property
  \((\lambda x_1.t_1 = \lambda x_2.t_2) \iff (t_1 \triangleleft [x_1 := x_2] = t_2) \land (t_2 \triangleleft [x_2 := x_1] = t_1)\)

• All other properties and definitions of \(\Lambda_1\) are recreated in \(\Lambda\), except for function existence

• More general term height induction principle:
  \[\vdash \forall P. (\forall x. P x) \land (\forall t u. P t \land P u \Rightarrow P (t \ u)) \land (\forall t. (\forall t'. \text{HEIGHT } t = \text{HEIGHT } t' \Rightarrow P t') \Rightarrow \forall x. P (\lambda x. t)) \Rightarrow (\forall t. P t)\]
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
Semantics of Reduction in Lambda Calculus

• Define $\beta$ 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 $\Lambda$ is compatible (with the operations) if for all $M, M', Z \in \Lambda, x \in \text{var},$

\[ R M M' \Rightarrow R(Z M)(Z M') \land R(M Z)(M' Z) \land R(\lambda x.M)(\lambda x.M') \]

• Given relation $R$, $R$ induces reduction relations:

• $\rightarrow_R$ one step $R$-reduction compatible closure of $R$

• $\Rightarrow_R$ $R$-reduction reflexive, transitive closure of $\rightarrow_R$

• $=_R$ $R$-equality equivalence relation generated by $\Rightarrow_R$

9/10/01
Diamond Property and Church-Rosser Property

• \( \Rightarrow \) satisfies diamond property (\( \Rightarrow \not\vdash \Diamond \)) if
  \[
  \forall M M_1 M_2. M \Rightarrow M_1 \land M \Rightarrow M_2 \Rightarrow \exists M_3. M_1 \Rightarrow M_3 \land M_2 \Rightarrow M_3
  \]

- \( R \) is Church-Rosser if \( \Rightarrow_R \not\vdash \Diamond \); we want to prove \( \Rightarrow_\beta \not\vdash \Diamond \)

9/10/01
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 (\(\leftrightarrow\)) inductively by the rules

\[
\begin{align*}
M \leftrightarrow M' & \quad, \quad N \leftrightarrow N' \\
M \leftrightarrow M' & \quad, \quad N \leftrightarrow N' \\
\lambda x. M \leftrightarrow \lambda x. M' & \quad, \quad (\lambda x. M) N \leftrightarrow M' \triangleleft [x := N']
\end{align*}
\]
Proof of the Church-Rosser Theorem

• Theorem: For all relations $\rightsquigarrow$, $\rightsquigarrow \vdash \diamondsuit \Rightarrow \rightsquigarrow^* \vdash \diamondsuit$

• Theorem: $\rightarrow$ satisfies the diamond property ($\rightarrow \vdash \diamondsuit$)

• Theorem: $\rightarrow_\beta$ is the transitive closure of $\rightarrow$ ($\rightarrow_\beta = \rightarrow^*$)

• Theorem: $\beta$ 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/
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 $\eta$- 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.*