A Proof of the Church-Rosser Theorem for the Lambda Calculus in Higher Order Logic

Peter V. Homeier

U. S. Department of Defense, homeier@saul.cis.upenn.edu
http://www.cis.upenn.edu/~homeier

Abstract. This paper describes a proof of the Church-Rosser theorem within the Higher Order Logic (HOL) theorem prover. This follows the proof by Tait/Martin-Löf, preserving the elegance of the classic presentation by Barendregt. We model the lambda calculus with a name-carrying syntax, as in practical languages. The proof is simplified by forming a quotient of the name-carrying syntax by the α-equivalence relation, thus separating the concerns of α-equivalence and β-reduction.

1 Introduction

The Church-Rosser theorem states the *confluence* property, that if an expression may be evaluated in two different ways, both will lead to the same result. Since the first attempts to prove this in 1936, many improvements have been found, including the Tait/Martin-Löf simplification and the Takahashi Triangle. A classic presentation may be found in Barendregt [1]. The proofs involve sophisticated inductive arguments, whose patterns have also intrigued researchers in mechanically checked proof. The first mechanical proof was presented by Shankar [9], and has been followed by Huet [5], Nipkow [7], Pfenning [8], Vestergaard/Brotherston [11], and Ford/Mason [2]. Of these, only Nipkow extends the work beyond β-reduction to proofs of confluence for η- and $\beta\eta$-reduction.

One key issue in these proofs is whether the syntax of the lambda calculus is represented using names for variables, or a de Bruijn representation, where numbers are used for names. The de Bruijn syntax is more agreeable for the Church-Rosser proof, as it evades the problem of α-equivalence. However, the name-carrying syntax is more realistic, as this is more representative of programming languages in general use. Because of the greater facility, many of the mechanical Church-Rosser proofs mentioned prove confluence for the de Bruijn syntax [5, 7, 8]. However, as in [2], we wish to address the issues of name-carrying syntax, in order to relate more directly to practical programming languages.

The presence of names raises as key issues the definitions of α-equivalence, substitution, and β-reduction. In two of the above proofs where names were treated [9, 11], confluence was proved for an arbitrary intermixture of α- and β-reduction. This intermixture bred an unfortunate complexity.

The elegant presentation by Barendregt [1] axes this complexity by the Barendregt Variable Convention (BVC). Our aim was to follow Barendregt as closely as possible, including mechanizing the BVC. We divided the consideration of α- and β-reduction into two layers, forming a new model of the syntax as the quotient of the original name-carrying syntax divided by the α-equivalence relation. This quotient layer is exactly isomorphic to the de Bruijn syntax and greatly simplifies the Church-Rosser proof. This is the same approach as Ford/Mason [2]. We found that there was at least as much work involved in forming the quotient of the original language as in all the remaining work of proving confluence.

Our HOL proof of Church-Rosser contains 7 main theories, which make 77 definitions and prove 359 theorems in 6 minutes, 54 seconds on a 300 MHz Pentium II. The proofs here may seem to be reasoned in normal lambda calculus, but are actually interpretations of the HOL tactics into mathematical English.

After proving Church-Rosser for β-reduction, we tested the clarity of the foundation by extending this work in two ways: proving the diamond lemma by the Takahashi triangle, and proving Church-Rosser for η- and $\beta\eta$-reduction [1]. The first took one half day, and the second took four days. Space precludes their presentation here, but the HOL proofs are available [6].

The author wishes to thank Randolph Johnson, Bill Legato, Brad Martin, Sylvan Pinsky, and Frank Taylor for many helpful comments and improvements.

2 The Pre-Lambda Calculus

We define the pre-lambda calculus (Λ_1), beginning with the type of terms, `term1`. The type of variables is `var`. We also use Λ_1 to abbreviate `term1`.

Definition 1. $\Lambda_1 \ ::= \ \texttt{var} \mid \Lambda_1\Lambda_1 \mid \lambda\texttt{var}.\Lambda_1$

This defines terms in the lambda calculus inductively as either being variables, applications of a term representing a function to another term representing an argument, or abstractions of terms by a variable, which represent functions of one argument. These terms may be compared for syntactic equality (=).

We will use t, u, e, M, and N as typical variables of type Λ_1, w, x, y, and z as typical variables of type `var`, r for sets of variables, and s for substitutions.

This definition is created in the HOL logic by the code

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

This creates `term1` as a new concrete recursive type within the HOL logic, and Var_1, App_1, and Lam_1 as constructor functions. When no confusion may result, we will use x for $\mathsf{Var}_1\, x$, $t\, u$ for $\mathsf{App}_1\, t\, u$, and $\lambda x.\, t$ for $\mathsf{Lam}_1\, x\, t$. When `term1` is created, `Hol_datatype` automatically proves several theorems that characterize the behavior of values of this new type regarding structural induction, function existence, cases, and constructor distinctiveness and one-to-one properties.

We use the function existance theorem to define the following functions by primitive recursion, by induction on the structure of terms. We here use **max** as an infix operator that yields the maximum of its arguments.

Definition 2 (Height of a term).

$$\begin{aligned}
\mathsf{HEIGHT}_1(x) &\overset{\text{def}}{=} 0\\
\mathsf{HEIGHT}_1(t\ u) &\overset{\text{def}}{=} (\mathsf{HEIGHT}_1\ t\ \mathbf{max}\ \mathsf{HEIGHT}_1\ u) + 1\\
\mathsf{HEIGHT}_1(\lambda x.\ u) &\overset{\text{def}}{=} \mathsf{HEIGHT}_1\ u + 1
\end{aligned}$$

Definition 3 (Free variables of a term).

$$\begin{aligned}
\mathsf{FV}_1(x) &\overset{\text{def}}{=} \{x\}\\
\mathsf{FV}_1(t\ u) &\overset{\text{def}}{=} \mathsf{FV}_1\ t\ \cup\ \mathsf{FV}_1\ u\\
\mathsf{FV}_1(\lambda x.\ u) &\overset{\text{def}}{=} \mathsf{FV}_1\ u - \{x\}
\end{aligned}$$

We express proper substitution on a term using *explicit simultaneous substitutions*, as a separate data structure. These combine a finite number of individual substitutions of expressions for variables into one substitution, where all are applied simultaneously. The actual application of a substitution to an expression is done by versions of the infix binary operator $\lhd$.

We model a simultaneous substitution as type `(var # term1)list`, a list of pairs (x, e) of a variable x and an expression e to be substituted for x.

$$\Sigma_1 \ ::= \ [\,] \ \mid \ (\mathbf{var} := \Lambda_1) \ :: \ \Sigma_1$$

Notation: We will use := to create a single substitution pair, e.g., $(x := e)$, and :: for infix Cons and [ ] for Nil to create lists of pairs, e.g., $(x := e) :: [\,]$, which is the same as $[x := e]$. Longer lists are expressed with commas as $[x_1 := e_1,\ x_2 := e_2,\ x_3 := e_3]$, or, equivalently, as $[x_1, x_2, x_3 := e_1, e_2, e_3]$. Finally, for a substitution of a list of variables ys for another list xs, we will use $[xs := ys]$.

Definition 4 (Substitution applied to a variable).

$$\begin{aligned}
y \lhd_1^v ((x := e) :: s) &\overset{\text{def}}{=} \mathbf{if}\ y = x\ \mathbf{then}\ e\ \mathbf{else}\ y \lhd_1^v s\\
y \lhd_1^v [\,] &\overset{\text{def}}{=} \mathsf{Var}_1\ y
\end{aligned}$$

Definition 5 (Free variables of a substitution on a set of variables).

$$\mathsf{FVsubst}_1\ s\ r \ \overset{\text{def}}{=}\ \bigcup(\mathbf{image}\ (\mathsf{FV}_1 \circ \mathsf{SUB}_1\ s)\ r)$$

where we define $\mathsf{SUB}_1\ s\ y\ =\ y \lhd_1^v s$, as a curried prefix version of $\lhd_1^v$. For every variable in the set r, its image under substitution by s is computed and the free variables of the image are found. All of these free variable sets are unioned for the result. Note if z is not mentioned in s, then $z \lhd_1^v s = z$.

Simultaneous substitutions allow us to define substitution on terms using primitive recursion. For substitutions on abstractions, we carefully calculate a change of bound variable and combine this with the existing substitution before it is applied to the body of the abstraction.

Definition 6 (Substitution on terms).

$$\begin{array}{lcl} x \lhd_1 s & \overset{\text{def}}{=} & x \lhd_1^v s \\ (t\ u) \lhd_1 s & \overset{\text{def}}{=} & (t \lhd_1 s)\ (u \lhd_1 s) \\ (\lambda x.\ u) \lhd_1 s & \overset{\text{def}}{=} & \textbf{let } x' = \mathsf{variant}\ x\ (\mathsf{FVsubst}_1\ s\ (\mathsf{FV}_1\ u - \{x\}))\ \textbf{in} \\ & & \lambda x'.\ (u \lhd_1 ((x := x') :: s)) \end{array}$$

Some other proposals [10, 11] (but not [2] or [4]) define substitution on cases where capture may occur either incorrectly or not at all. The further development must then ensure that substitution is only applied safely.

By contrast, this definition of substitution is complete, correctly avoiding the possibility of capture of bound variables. It chooses a new bound variable using the variant function. We here define variant x r to be x if $x \notin r$, otherwise to choose some variable not in the set r. Thus in any case, variant $x\ r \notin r$. Similarly, if the substitution s does not invite a capture, so that the bound variable x need not change, definition 6 above ensures that in fact $x' = x$.

Note that if a variable $z \neq x$ is free in the abstraction body u but is not explicitly mentioned by the substitution s, then $\mathsf{FVsubst}_1$ delivers z in its result.

At this point we have built the foundational theory of pre-lambda calculus terms. However, it has one crucial flaw. The one-to-one property of the constructors states that $\lambda x_1.\ t_1 = \lambda x_2.\ t_2$ if and only if $x_1 = x_2$ and $t_1 = t_2$. But in fact we want to consider, for example, $\lambda x_1.\ x_1$ and $\lambda x_2.\ x_2$ to mean the same term. Intuitively, it should not matter what name one uses for a bound variable, as long as one is consistent in how it is used. In fact, the Church-Rosser property is not true for the pre-lambda calculus as presented. To prove this property, we must derive a variant where distinctions such as above are blurred. The exact blurring we wish to achieve is called α-equivalence.

3 α-Equivalence

In the past, α-equivalence has been defined as a relation between terms where some bound variables are replaced in a consistent fashion. In Church and others since, the renaming of bound variables was built into the semantic rules, as a reduction relation. The above theory was extended by the axiom scheme

$$\lambda x.\ t \ = \ \lambda y.\ (t \lhd_1 [x := y]) \tag{1}$$

where y is not free in $\lambda x.\ t$ [3].

However, several authors have taken a different route, including Barendregt [1], who prefer to identify α-equivalent terms at the *syntactic* level. Thus $\lambda x.x = \lambda y.y$, etc. This is commonly assumed to be done by forming equivalence classes of the existing terms, according to the α-equivalence relation, and letting these classes be the new terms. In order to form these classes, the relation itself is often defined similarly to (1).

This definition is not unsound. However, we question it on aesthetic grounds. If we later use α-equivalence to simplify substitution, through the BVC, should we first use substitution in defining α-equivalence? This motivated us to search for a definition of α-equivalence independent of substitution. We found this using an auxiliary notion of *contextual α-equivalence*, which relate two terms in the context of a list of lambda-bindings for each term. Shankar [10] is similar.

Definition 7 (Contextual α-equivalence of variables). *Let xs and ys be two lists of variables, and denote the length of xs as $\|xs\|$. The contextual α-equivalence of two variables w and z in the respective contexts xs and ys ($w \;{}^{xs}_{\mathrm{var}}{\equiv}^{ys}_{\alpha}\; z$) is defined recursively on the list structure of xs and ys as*

$$\begin{array}{rcl} w \;{}^{x::xs}_{\mathrm{var}}{\equiv}^{y::ys}_{\alpha}\; z & \stackrel{\mathrm{def}}{=} & (w = x \;\wedge\; z = y \;\wedge\; \|xs\| = \|ys\|) \vee \\ & & (w \neq x \;\wedge\; z \neq y \;\wedge\; w \;{}^{xs}_{\mathrm{var}}{\equiv}^{ys}_{\alpha}\; z) \\ w \;{}^{[\,]}_{\mathrm{var}}{\equiv}^{[\,]}_{\alpha}\; z & \stackrel{\mathrm{def}}{=} & (w = z) \end{array}$$

This definition searches down the two context lists simultaneously to seek a pair of variables which match w and z respectively. w and z are equivalent if the two lists have the same length and if w and z both appear first in the contexts *at the corresponding location*, or if they do not appear at all but are equal.

Lemma 8. $(x \;{}^{xs}_{\mathrm{var}}{\equiv}^{ys}_{\alpha}\; y) \;\Leftrightarrow\; (\, \|xs\| = \|ys\| \;\wedge$
$$\begin{array}{l} (x \lhd_1^v [xs := ys] \;=\; \mathsf{Var}_1\; y) \;\wedge \\ (y \lhd_1^v [ys := xs] \;=\; \mathsf{Var}_1\; x)) \end{array}$$

Proof: by list induction on xs and ys, and then considering cases for x and y.

Definition 9 (Contextual α-equivalence of terms). *Let xs and ys be two lists of variables. The contextual α-equivalence of two terms t and u in the respective contexts xs and ys ($t \;{}^{xs}{\equiv}^{ys}_{\alpha}\; u$) is defined inductively on the structure of the terms t and u by the rules*

$$\frac{x \;{}^{xs}_{\mathrm{var}}{\equiv}^{ys}_{\alpha}\; y}{x \;{}^{xs}{\equiv}^{ys}_{\alpha}\; y} \qquad \frac{t_1 \;{}^{xs}{\equiv}^{ys}_{\alpha}\; t_2 \;,\; u_1 \;{}^{xs}{\equiv}^{ys}_{\alpha}\; u_2}{t_1\, u_1 \;{}^{xs}{\equiv}^{ys}_{\alpha}\; t_2\, u_2} \qquad \frac{t_1 \;{}^{x::xs}{\equiv}^{y::ys}_{\alpha}\; t_2}{\lambda x.\, t_1 \;{}^{xs}{\equiv}^{ys}_{\alpha}\; \lambda y.\, t_2}$$

This maps the test of equivalence down through the structure of the terms, adding context whenever a lambda abstraction is penetrated, resolving eventually to comparisons of the variables in each term.

We have implemented this definition in HOL using Myra VanInwegen's rule induction package. This package automatically proves theorems for the new relation's rules, the inversion of the rules, and weak and strong induction principles. Notably, this package supports defining mutually recursive relations. [6]

Definition 10 (α-equivalence of terms). *The α-equivalence of two terms t and u ($t \equiv_\alpha u$) is defined as*

$$t \equiv_\alpha u \;\stackrel{\mathrm{def}}{=}\; t \;{}^{[\,]}{\equiv}^{[\,]}_{\alpha}\; u$$

Thus we have defined α-equivalence between terms without appeal to substitution. Despite the brevity of the substitution-based definition (1), we believe that this is actually simpler, without hidden subtleties.

It is not hard to prove in HOL that this relation is reflexive, symmetric, and transitive (theorems `ALPHA_REFL`, `ALPHA_SYM`, `ALPHA_TRANS`). Given this α-equivalence relation, we can now form the pure lambda calculus as a quotient.

4 The Pure Lambda Calculus

We define the new type of terms of the pure lambda calculus as a quotient of the pre-term type by the α-equivalence relation, isomorphic to de Bruijn terms.

Definition 11. $\Lambda \stackrel{\text{def}}{=} \Lambda_1 / \equiv_\alpha$

This is accomplished in HOL by a new package to define quotient types [6].

```
val term_QUOTIENT =
    define_quotient_type "term" "term_ABS" "term_REP"
            ALPHA_REFL ALPHA_SYM ALPHA_TRANS;
```

The function `define_quotient_type` defines the new type `term` based on the reflexive, symmetric, and transitive properties of the equivalence relation. It also defines two new functions, an abstraction function `term_ABS` to map from `term1` to `term`, with notation $\lfloor t \rfloor$, and a representation function `term_REP` to map from a `term` to a (fixed) representative `term1`, with notation $\lceil t \rceil$. `define_quotient_type` returns a theorem `term_QUOTIENT` that completely characterizes these functions:

Theorem 12 (Abstraction/representation mappings for terms).

$$(\forall a.\ \lfloor \lceil a \rceil \rfloor = a) \ \wedge \ (\forall r\ r'.\ r \equiv_\alpha r' \Leftrightarrow (\lfloor r \rfloor = \lfloor r' \rfloor))$$

The package also provides functions to prove various other results directly from this theorem, for example,

$$\vdash \forall r.\ \lceil \lfloor r \rfloor \rceil \equiv_\alpha r \qquad \vdash \forall a_1\ a_2.\ (a_1 = a_2) \Leftrightarrow (\lceil a_1 \rceil \equiv_\alpha \lceil a_2 \rceil)$$

In addition to creating the new type `term` (which we abbreviate Λ), we need to recreate the logical environment, with all defined constants and theorems that existed for Λ_1, except for α-equivalence which is represented in Λ by equality.

First, using these abstraction and representation functions and the original constructor functions, we define the corresponding pure constructor functions.

Definition 13 (Term constructors).

$$\begin{aligned}
\mathsf{Var}\ x &\stackrel{\text{def}}{=} \lfloor \mathsf{Var}_1\ x \rfloor \\
\mathsf{App}\ t\ u &\stackrel{\text{def}}{=} \lfloor \mathsf{App}_1\ \lceil t \rceil\ \lceil u \rceil \rfloor \\
\mathsf{Lam}\ x\ t &\stackrel{\text{def}}{=} \lfloor \mathsf{Lam}_1\ x\ \lceil t \rceil \rfloor
\end{aligned}$$

Now we recreate in Λ functions corresponding to those in Λ_1. However, a technical problem arises; not every function definable in Λ_1 can be realized in Λ. In particular, the function must respect α-equivalence in the following sense: if the function is called twice with arguments which are α-equivalent, then the results should be α-equivalent. Of course, if the result type is not Λ_1, the results should be equal. We call such a function *respectful.*

Recreating a function definition in Λ takes three steps; first, prove that the function respects α-equivalence, then define the new function using the abstraction and representation functions, and finally prove as a theorem in Λ the same form of the definition in Λ_1. This pattern is repeated for every function we wish to recreate in Λ. Proving respectfulness may be arbitrarily difficult.

Lemma 14. $t_1 \ {}^{xs}{\equiv}^{ys}_{\alpha}\ t_2 \ \wedge\ x \in \mathsf{FV}_1\ t_1 \ \Rightarrow\ \exists y.\ y \in \mathsf{FV}_1\ t_2 \ \wedge\ x \ {}^{xs}_{\mathrm{var}}{\equiv}^{ys}_{\alpha}\ y$

Proof: by strong rule induction on $t_1 \ {}^{xs}{\equiv}^{ys}_{\alpha}\ t_2$, and definitions 3 and 7.

Definition 15 (Free variables of a term).

Respectfulness:	$t_1 \equiv_\alpha t_2 \ \Rightarrow\ (\mathsf{FV}_1\ t_1 = \mathsf{FV}_1\ t_2)$		
Definition:	$\mathsf{FV}\ t \ \overset{\text{def}}{=}\ \mathsf{FV}_1\ \lceil t \rceil$		
Recreated	$\mathsf{FV}(x)$	$=$	$\{x\}$
definition:	$\mathsf{FV}(t\ u)$	$=$	$\mathsf{FV}\ t\ \cup\ \mathsf{FV}\ u$
	$\mathsf{FV}(\lambda x.\ u)$	$=$	$\mathsf{FV}\ u - \{x\}$

The respectfulness theorem is proven using definition 10, the symmetry of ${}^{xs}{\equiv}^{ys}_{\alpha}$, lemma 14, and the definition of FV_1. The recreated definition is proven using respectfulness, the definition, and the original definition in Λ_1.

The HEIGHT function is recreated in an exactly analogous way. where respectfulness is proven by an easy rule induction on $t_1 \ {}^{xs}{\equiv}^{ys}_{\alpha}\ t_2$.

With substitution things become more complex. The first task is to model the type of substitutions in Λ. Without going into the details, we extend the α-equivalence relation in the obvious way first to pairs of a variable and a term ($\equiv^{\mathrm{pair}}_{\alpha}$), and then to lists of such pairs ($\equiv^{\mathrm{subst}}_{\alpha}$). The quotient package [6] provides tools to create the appropriate mapping functions between the Λ_1 and Λ substitution types. We will use the same $\lfloor s \rfloor$ and $\lceil s \rceil$ notation for the mappings between the substitution types. These tools maintain the pair and list structure, so substitutions may be considered defined by list recursion, with constructors.

Definition 16 (Substitution constructors in Λ).

$$\begin{array}{lcl} (x := e) :: s & \overset{\text{def}}{=} & \lfloor (x := \lceil e \rceil) :: \lceil s \rceil \rfloor \\ [\,] & \overset{\text{def}}{=} & \lfloor [\,] \rfloor \end{array}$$

From this we can derive the standard characterization as in theorem 12.

Definition 17 (Substitution on a variable).

Respectfulness:	$s_1 \equiv^{\mathrm{subst}}_{\alpha} s_2 \ \Rightarrow\ y \lhd^v_1 s_1 \equiv_\alpha y \lhd^v_1 s_2$		
Definition:	$y \lhd^v s \ \overset{\text{def}}{=}\ \lfloor y \lhd^v_1 \lceil s \rceil \rfloor$		
Recreated	$y \lhd^v ((x := e) :: s)$	$=$	**if** $y = x$ **then** e **else** $y \lhd^v s$
definition:	$y \lhd^v [\,]$	$=$	$\mathsf{Var}\ y$

Respectfulness is proven by list induction on the substitutions, the reflexivity of $\equiv_\alpha$, the definition of $\equiv^{\mathrm{subst}}_{\alpha}$, and definition 4. The recreated definition is proven from the definition above and definitions 4 and 16. Analogous to SUB_1, we define $\mathsf{SUB}\ s\ y\ =\ y \lhd^v s$ as a curried prefix version of $\lhd^v$.

Lemma 18. $(\mathsf{FV} \circ \mathsf{SUB}\ s) = (\mathsf{FV}_1 \circ \mathsf{SUB}_1 \lceil s \rceil)$

Proof: By the definitions of SUB and SUB_1, we need to prove $\mathsf{FV}(x \lhd^v s) = \mathsf{FV}_1(x \lhd_1^v \lceil s \rceil)$. By definitions 15 and 17, this is $\mathsf{FV}_1(\lceil \lfloor x \lhd_1^v \lceil s \rceil \rfloor \rceil) = \mathsf{FV}_1(x \lhd_1^v \lceil s \rceil)$. By the respectfulness of FV_1, this follows from $\lceil \lfloor x \lhd_1^v \lceil s \rceil \rfloor \rceil \ \equiv_\alpha\ x \lhd_1^v \lceil s \rceil$. This is true by theorem 12.

Definition 19 (Free variables of a substitution on a set of variables).

Respectfulness:	$s_1 \equiv_\alpha^{\text{subst}} s_2 \ \Rightarrow\ \mathsf{FVsubst}_1\ s_1\ r = \mathsf{FVsubst}_1\ s_2\ r$
Definition:	$\mathsf{FVsubst}\ s\ r \ \stackrel{\text{def}}{=}\ \mathsf{FVsubst}_1\ \lceil s \rceil\ r$
Recreated definition:	$\mathsf{FVsubst}\ s\ r \ =\ \bigcup(\mathbf{image}\ (\mathsf{FV} \circ \mathsf{SUB}\ s)\ r)$

Respectfulness is proven by definition 5, the respectfulness of $\lhd_1^v$ and FV_1, and from definitions 15 and 17. The recreated definition is proven by the definition above, definition 5, and lemma 18.

Before we can define substitution on terms in Λ, we must first prove that substitution in Λ_1 respects α-equivalence. This has an interesting proof.

Theorem 20.
$$\begin{array}{l}((\|xs\| = \|ys\|) \Leftrightarrow (\|xs'\| = \|ys'\|)) \ \wedge \\ (\forall x.\ x \in \mathsf{FV}_1\ t_1 \ \Rightarrow\ x \lhd_1^v [xs := ys] = x \lhd_1^v [xs' := ys']) \ \wedge \\ (\forall y.\ y \in \mathsf{FV}_1\ t_2 \ \Rightarrow\ y \lhd_1^v [ys := xs] = y \lhd_1^v [ys' := xs']) \\ \Rightarrow\ ((t_1\ {}^{xs}{\equiv}_\alpha^{ys}\ t_2) \Leftrightarrow (t_1\ {}^{xs'}{\equiv}_\alpha^{ys'}\ t_2))\end{array}$$

Proof: by structural induction on t_1. We have three cases:

Case 1. $t_1 = x$. We will prove $(t_1\,{}^{xs}{\equiv}_\alpha^{ys} t_2) \Leftrightarrow (t_1\,{}^{xs'}{\equiv}_\alpha^{ys'} t_2)$ as a biconditional.

Subcase 1.1 ($\Rightarrow$) Assume $t_1\ {}^{xs}{\equiv}_\alpha^{ys}\ t_2$. Then t_2 must be of the form y. From the hypotheses, $x \lhd_1^v [xs := ys] = x \lhd_1^v [xs' := ys']$ and $y \lhd_1^v [ys := xs] = y \lhd_1^v [ys' := xs']$. Then by lemma 8, $(x\ {}^{xs}_{\text{var}}{\equiv}_\alpha^{ys}\ y) \Leftrightarrow (x\ {}^{xs'}_{\text{var}}{\equiv}_\alpha^{ys'}\ y)$, so $(x\ {}^{xs}{\equiv}_\alpha^{ys}\ y) \Leftrightarrow (x\ {}^{xs'}{\equiv}_\alpha^{ys'}\ y)$, and $(t_1\ {}^{xs}{\equiv}_\alpha^{ys}\ t_2) \Leftrightarrow (t_1\ {}^{xs'}{\equiv}_\alpha^{ys'}\ t_2)$. *Subcase 1.2* ($\Leftarrow$) Symmetrical.

Case 2. $t_1 = t\ u$.

Subcase 2.1 ($\Rightarrow$) Assume $t_1\ {}^{xs}{\equiv}_\alpha^{ys}\ t_2$. Then t_2 must be of the form $t'\ u'$. From the inductive hypotheses, $(t\ {}^{xs}{\equiv}_\alpha^{ys}\ t') \Leftrightarrow (t\ {}^{xs'}{\equiv}_\alpha^{ys'}\ t')$ and $(u\ {}^{xs}{\equiv}_\alpha^{ys}\ u') \Leftrightarrow (u\ {}^{xs'}{\equiv}_\alpha^{ys'}\ u')$. Then $(t\ u\ {}^{xs}{\equiv}_\alpha^{ys}\ t'\ u') \Leftrightarrow (t\ u\ {}^{xs'}{\equiv}_\alpha^{ys'}\ t'\ u')$ by definition 9 and so $(t_1\ {}^{xs}{\equiv}_\alpha^{ys}\ t_2) \Leftrightarrow (t_1\ {}^{xs'}{\equiv}_\alpha^{ys'}\ t_2)$. *Subcase 2.2* ($\Leftarrow$) Symmetrical.

Case 3. $t_1 = \lambda x.\ t$.

Subcase 3.1 ($\Rightarrow$) Assume $t_1\ {}^{xs}{\equiv}_\alpha^{ys}\ t_2$. Then t_2 must be of the form $t_2 = \lambda y.\ u$. By definition 9, $t\ {}^{x::xs}{\equiv}_\alpha^{y::ys}\ u$, and we need to show $t\ {}^{x::xs'}{\equiv}_\alpha^{y::ys'}\ u$. From the inductive hypothesis, $(t\ {}^{x::xs}{\equiv}_\alpha^{y::ys}\ u) \Leftrightarrow (t\ {}^{x::xs'}{\equiv}_\alpha^{y::ys'}\ u)$ if

$$\begin{array}{l}((\|x :: xs\| = \|y :: ys\|) \Leftrightarrow (\|x :: xs'\| = \|y :: ys'\|)) \ \wedge \\ (\forall x'.\ x' \in \mathsf{FV}_1\ t \ \Rightarrow\ x' \lhd_1^v [x :: xs := y :: ys] = x' \lhd_1^v [x :: xs' := y :: ys']) \ \wedge \\ (\forall y'.\ y' \in \mathsf{FV}_1\ u \ \Rightarrow\ y' \lhd_1^v [y :: ys := x :: xs] = y' \lhd_1^v [y :: ys' := x :: xs'])\end{array}$$

The first conjunct clearly follows from the hypotheses. For the other conjuncts, if $x' = x$, then both substitutions on x' yield y. Likewise if $y' = y$, then both substitutions on y' yield x. If $x' \neq x$ or $y' \neq y$, then the substitutions simplify to the cases covered by the hypotheses, as then $x' \in \mathsf{FV}_1(\lambda x.\ t)$ or $y' \in \mathsf{FV}_1(\lambda y.\ u)$. *Subcase 3.2* ($\Leftarrow$) Symmetrical.

Corollary 21. $\sim(x \in \mathsf{FV}_1\ t_1) \wedge \sim(y \in \mathsf{FV}_1\ t_2) \Rightarrow (t_1\, {}^{x::xs}\!\equiv_\alpha^{y::ys}\, t_2 \Leftrightarrow t_1\, {}^{xs}\!\equiv_\alpha^{ys}\, t_2)$

Proof: Directly from theorem 20, whose antecedents are proved by definition 4.

Definition 22. *This 8-argument notation contextually relates two substitutions s_1, s_2 on sets of variables r_1, r_2, relative to before/after contexts on both sides.*

$$\begin{array}{rl} s_1 \succ r_1\ {}^{xs'}_{xs}\!\equiv^{ys'}_{ys}\ r_2 \prec s_2 \ \overset{\text{def}}{=} & (\|xs'\| = \|ys'\|) \wedge \\ & (\forall x\ y.\ \ x \in r_1\ \wedge\ y \in r_2\ \wedge\ x\ {}^{xs}_{\text{var}}\!\equiv^{ys}_{\alpha}\ y\ \Rightarrow \\ & \qquad (x \lhd_1^v s_1)\ {}^{xs'}\!\equiv_\alpha^{ys'}\ (y \lhd_1^v s_2)) \end{array}$$

Theorem 23 (Contextual respectfulness of substitution).

$$\begin{array}{l} t_1\ {}^{xs}\!\equiv_\alpha^{ys}\ t_2\ \wedge\ s_1 \succ (\mathsf{FV}_1\ t_1)\ {}^{xs'}_{xs}\!\equiv^{ys'}_{ys}\ (\mathsf{FV}_1\ t_2) \prec s_2\ \Rightarrow \\ (t_1 \lhd_1 s_1)\ {}^{xs'}\!\equiv_\alpha^{ys'}\ (t_2 \lhd_1 s_2) \end{array}$$

Proof: by strong rule induction on $t_1\ {}^{xs}\!\equiv_\alpha^{ys}\ t_2$. There are three cases.

Case 1. We have $x\ {}^{xs}\!\equiv_\alpha^{ys}\ y$ and $s_1 \succ \{x\}\ {}^{xs'}_{xs}\!\equiv^{ys'}_{ys}\ \{y\} \prec s_2$, and so $x\ {}^{xs}_{\text{var}}\!\equiv^{ys}_{\alpha}\ y$ by definition 9 and then $(x \lhd_1^v s_1)\ {}^{xs'}\!\equiv_\alpha^{ys'}\ (y \lhd_1^v s_2)$ by definition 22. The goal is $(x \lhd_1 s_1)\ {}^{xs'}\!\equiv_\alpha^{ys'}\ (y \lhd_1 s_2)$, which follows by definition 6.

Case 2. $u_1\ v_1\ {}^{xs}\!\equiv_\alpha^{ys}\ u_2\ v_2$, so $u_1\ {}^{xs}\!\equiv_\alpha^{ys}\ u_2$, $v_1\ {}^{xs}\!\equiv_\alpha^{ys}\ v_2$ by definition 9, and

$$s_1 \succ (\mathsf{FV}_1\ u_1 \cup \mathsf{FV}_1\ v_1)\ {}^{xs'}_{xs}\!\equiv^{ys'}_{ys}\ (\mathsf{FV}_1\ u_2 \cup \mathsf{FV}_1\ v_2) \prec s_2.$$

By definition 22, this implies $s_1 \succ \mathsf{FV}_1\ u_1\ {}^{xs'}_{xs}\!\equiv^{ys'}_{ys}\ \mathsf{FV}_1\ u_2 \prec s_2$ and $s_1 \succ \mathsf{FV}_1\ v_1\ {}^{xs'}_{xs}\!\equiv^{ys'}_{ys}\ \mathsf{FV}_1\ v_2 \prec s_2$. The goal to be shown is

$$\begin{array}{lr} (u_1\ v_1 \lhd_1 s_1)\ {}^{xs'}\!\equiv_\alpha^{ys'}\ (u_2\ v_2 \lhd_1 s_2) & \text{(goal)} \\ \quad \Leftrightarrow\ (u_1 \lhd_1 s_1)\ (v_1 \lhd_1 s_1)\ {}^{xs'}\!\equiv_\alpha^{ys'}\ (u_2 \lhd_1 s_2)\ (v_2 \lhd_1 s_2) & \text{(defn. 6)} \\ \quad \Leftrightarrow\ (u_1 \lhd_1 s_1)\ {}^{xs'}\!\equiv_\alpha^{ys'}\ (u_2 \lhd_1 s_2)\ \wedge\ (v_1 \lhd_1 s_1)\ {}^{xs'}\!\equiv_\alpha^{ys'}\ (v_1 \lhd_1 s_2) & \text{(defn. 9)} \end{array}$$

These last two conjuncts are implied by the inductive hypotheses.

Case 3. $\lambda x.\ u_1\ {}^{xs}\!\equiv_\alpha^{ys}\ \lambda y.\ u_2$, so $u_1\ {}^{x::xs}\!\equiv_\alpha^{y::ys}\ u_2$, and we also have

$$s_1 \succ (\mathsf{FV}_1\ u_1 - \{x\})\ {}^{xs'}_{xs}\!\equiv^{ys'}_{ys}\ (\mathsf{FV}_1\ u_2 - \{y\}) \prec s_2. \tag{1}$$

The goal is $(\lambda x.\ u_1 \lhd_1 s_1)\ {}^{xs'}\!\equiv_\alpha^{ys'}\ (\lambda y.\ u_2 \lhd_1 s_2)$. According to definition 6, let x' and y' be the new bound variables replacing x and y induced by the substitutions s_1 and s_2. Then by definition 9 the goal becomes

$$(u_1 \lhd_1 ((x := x') :: s_1))\ {}^{x'::xs'}\!\equiv_\alpha^{y'::ys'}\ (u_2 \lhd_1 ((y := y') :: s_2))$$

Using the inductive hypothesis, it suffices to prove

$$((x := x') :: s_1) \succ (\mathsf{FV}_1\ u_1)\ {}^{x'::xs'}_{x::xs}\!\equiv^{y'::ys'}_{y::ys}\ (\mathsf{FV}_1\ u_2) \prec ((y := y') :: s_2)$$

Opening up this goal and (1) above by definition 22, this means that given

$$\begin{array}{l} \forall x''\ y''.\ \ x'' \in \mathsf{FV}_1\ u_1 - \{x\}\ \wedge\ y'' \in \mathsf{FV}_1\ u_2 - \{y\}\ \wedge\ x''\ {}^{xs}_{\text{var}}\!\equiv^{ys}_{\alpha}\ y''\ \Rightarrow \\ \qquad (x'' \lhd_1^v s_1)\ {}^{xs'}\!\equiv_\alpha^{ys'}\ (y'' \lhd_1^v s_2) \end{array} \tag{2}$$

we must prove

$$\begin{array}{l}\forall x''\ y''.\ x'' \in \mathsf{FV}_1\ u_1\ \wedge\ y'' \in \mathsf{FV}_1\ u_2\ \wedge\ x''\ {}^{x::xs}_{\text{var}}{\equiv}^{y::ys}_{\alpha}\ y''\ \Rightarrow \\ \qquad (x'' \lhd_1^v ((x := x') :: s_1))\ {}^{x'::xs'}{\equiv}^{y'::ys'}_{\alpha}\ (y'' \lhd_1^v ((y := y') :: s_2))\end{array}$$

This is proven by taking four cases on whether or not $x'' = x$ or $y'' = y$. If both are equal, the goal's consequent is true by definitions 4, 7, and 9. If one is equal and the other not, then the antecedent is false by definition 7. If both are not equal, then the goal simplifies using those definitions and corollary 21 to

$$\begin{array}{l}x'' \in \mathsf{FV}_1\ u_1\ \wedge\ y'' \in \mathsf{FV}_1\ u_2\ \wedge\ x''\ {}^{xs}_{\text{var}}{\equiv}^{ys}_{\alpha}\ y''\ \Rightarrow \\ (x'' \lhd_1^v s_1)\ {}^{xs'}{\equiv}^{ys'}_{\alpha}\ (y'' \lhd_1^v s_2)\end{array}$$

and since $x'' \neq x$ and $y'' \neq y$, this is proven by (2). Corollary 21 applies because x' and y' were chosen by $\mathsf{variant}$ not in the free variables of $u_1 \lhd_1 s_1$ and $u_2 \lhd_1 s_2$.

Corollary 24 (Respectfulness of substitution).

$$\begin{array}{l}t_1 \equiv_\alpha t_2\ \wedge\ (\forall x.\ x \in \mathsf{FV}_1\ t_1\ \Rightarrow\ (x \lhd_1^v s_1) \equiv_\alpha (x \lhd_1^v s_2))\ \Rightarrow \\ (t_1 \lhd_1 s_1) \equiv_\alpha (t_2 \lhd_1 s_2)\end{array}$$

Proof: directly from theorem 23 and definition 22 with empty variable lists. Because of the respectfulness of FV_1, $\mathsf{FV}_1\ t_1 = \mathsf{FV}_1\ t_2$.

This enables us to recreate the definition of substitution on terms in Λ.

Definition 25 (Substitution on terms).

Respectfulness: $t_1 \equiv_\alpha t_2\ \wedge\ s_1 \equiv_\alpha^{\text{subst}} s_2\ \Rightarrow\ t_1 \lhd_1 s_1 \equiv_\alpha t_2 \lhd_1 s_2$

Definition: $t \lhd s\ \overset{\text{def}}{=}\ \lfloor \lceil t \rceil \lhd_1 \lceil s \rceil \rfloor$

Recreated definition:

$$\begin{array}{lcl}x \lhd s & = & x \lhd^v s \\ (t\ u) \lhd s & = & (t \lhd s)\ (u \lhd s) \\ (\lambda x.\ u) \lhd s & = & \textbf{let}\ x' = \mathsf{variant}\ x\ (\mathsf{FVsubst}\ s\ (\mathsf{FV}\ u - \{x\}))\ \textbf{in} \\ & & \lambda x'.\ (u \lhd ((x := x') :: s))\end{array}$$

Respectfulness is proven from corollary 24. The recreated definition is proven from respectfulness, theorem 12, and the definitions.

In addition to recreating these function definitions, we have also recreated the theorems for induction, cases, and distinctiveness and one-to-one of constructors, but *not* existence. For the most part these are direct analogs of Λ_1, except for:

Theorem 26. $(\lambda x_1.\ t_1 = \lambda x_2.\ t_2)\ \Leftrightarrow$
$(t_1 \lhd [x_1 := x_2] = t_2)\ \wedge\ (t_2 \lhd [x_2 := x_1] = t_1)$.

The proof is extensive and omitted for space.

In addition to the normal induction theorem, we have also proven a theorem for induction on the height of a term. This will be frequently used.

Theorem 27 (Term height induction).

$$\begin{array}{ll}\vdash \forall P. & (\forall x.\ P\ (x))\ \wedge \\ & (\forall t\ u.\ P\ t\ \wedge\ P\ u\ \Rightarrow\ P\ (t\ u))\ \wedge \\ & (\forall t.\ (\forall t'.\ \mathsf{HEIGHT}\ t = \mathsf{HEIGHT}\ t'\ \Rightarrow\ P\ t')\ \Rightarrow\ \forall x.\ P\ (\lambda x.\ t)) \\ & \Rightarrow (\forall t.\ P\ t)\end{array}$$

5 Barendregt Variable Convention

Barendregt [1] is the most encyclopedic compilation of lambda calculus theory, and has been widely studied. He raises the issues of capture of bound variables and the fallacies that may result, but then removes those issues by declaring what has become known as the *Barendregt Variable Convention* (BVC):

"2.1.12. Convention. Terms that are α-congruent are identified. . . . "

"2.1.13. Variable convention. If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables. . . . "

"2.1.14. Moral. Using conventions 2.1.12 and 2.1.13 one can work with λ-terms in the naive way. Naive means that substitutions and other operations on terms can be performed without questioning whether they are allowed."

This convention greatly simplifies the proof of the Church-Rosser theorem. However, at first glance, it appears to simply ignore the issues of capture, and some have claimed this may be unsound. [11] presents a "rational reconstruction of the BVC." In contrast, we have found a way to conduct proofs in the style of the BVC using a special-purpose tactic we have written. This tactic first searches the entire proof state for all free variables, then searches the current goal for occurrences of abstractions ($\lambda x.\ M$), chooses new bound variables for the abstractions not appearing in the free variables, shifts the abstractions to the new variables, and finally moves substitutions inside the abstractions.

Here is an example of its use to prove Barendregt's Substitution Lemma.

Lemma 28 (Substitution lemma). *If $x \neq y$ and $x \notin \mathsf{FV}\ L$, then*

$$M \lhd [x := N] \lhd [y := L] \ = \ M \lhd [y := L] \lhd [x := N \lhd [y := L]]$$

Proof: by height induction on the structure of M. There are three cases.

Case 1. M is a variable z. Take cases on $z = x$, $z = y$, or neither (see [1]).

Case 2. $M = M_1\ M_2$. Then the statement follows from the induction hypotheses and the definition of substitution.

Case 3. $M = \lambda x'.\ M_1$. The resulting goal in HOL is

```
Lam x' M <[ [(x,N)] <[ [(y,L)] =
Lam x' M <[ [(y,L)] <[ [(x,N <[ [(y,L)])]
------------------------------------
  0.  !t'.
        (HEIGHT M = HEIGHT t') ==>
        ~(x = y) /\ ~(x IN FV L) ==>
        (t' <[ [(x,N)] <[ [(y,L)] = t' <[ [(y,L)] <[ [(x,N <[ [(y,L)])])
  1.  ~(x = y)
  2.  ~(x IN FV L)
```

The special tactic `SIMPLE_SUBST_TAC` (no arguments) converts this to the goal

```
Lam z (M' <[ [(x,N)] <[ [(y,L)]) =
Lam z (M' <[ [(y,L)] <[ [(x,N <[ [(y,L)])])
------------------------------------
```

```
0.  !t'.
      (HEIGHT M = HEIGHT t') ==>
      ~(x = y) /\ ~(x IN FV L) ==>
      (t' <[ [(x,N)] <[ [(y,L)] = t' <[ [(y,L)] <[ [(x,N <[ [(y,L)])])
1.  ~(x = y)
2.  ~(x IN FV L)
3.  ~(z = x')
4.  ~(z IN FV N)
5.  ~(z IN FV L)
6.  ~(z = y)
7.  ~(z = x)
8.  ~(z IN FV M)
9.  ~(z IN FV M DIFF {x'})
10.  ~(x = y) /\ ~(x IN FV L) ==>
     (M' <[ [(x,N)] <[ [(y,L)] = M' <[ [(y,L)] <[ [(x,N <[ [(y,L)])])
11.  HEIGHT M = HEIGHT M'
12.  Lam x' M = Lam z M'
```

The abstraction $\lambda x'.\ M$ has been shifted to $\lambda z.\ M'$, where z and M' are new. This is stated directly in (12). z has been chosen so as to avoid all free variables present in the proof, as is seen in (3) through (9). (10) is the specialization of the height inductive hypothesis (0) for M', and simplified by (11). Most importantly, for this choice of z and M', the substitutions may be treated naively, as is accomplished in the goal, where they have penetrated to the bodies of the abstractions with no concerns about capture, "in the naive way."

To finish the proof, resolve (10) with (1) and (2) and rewrite the goal.

This achieves almost the same ease and simplicity of reasoning as the BVC, requiring only that we use height induction and that we shift the abstractions away from possible captures with `SIMPLE_SUBST_TAC`.

6 Reduction

Following Barendregt [1] section 3.1, we consider reduction in a general setting.

Definition 29. *A binary relation $\boldsymbol{R}$ on Λ is compatible (with the operations) if for all M, M', $Z \in \Lambda$, $x \in$ `var`,*

$$\boldsymbol{R}\, M\ M' \ \Rightarrow\ \boldsymbol{R}(Z\ M)(Z\ M') \ \wedge\ \boldsymbol{R}(M\ Z)(M'\ Z) \ \wedge\ \boldsymbol{R}(\lambda x.\ M)(\lambda x.\ M')$$

Definition 30. *$\boldsymbol{\beta}$ is defined by rule induction, by the single rule*

$$\frac{}{\boldsymbol{\beta}\ \ ((\lambda x.\ M)\ N)\ \ (M \lhd [x := N])}$$

Definition 31. *If $\rightarrowtail$ is a binary relation, the reflexive closure of $\rightarrowtail$ (notation: $\rightarrowtail^{=}$) is the least relation extending $\rightarrowtail$ that is reflexive. The transitive closure (notation: $\rightarrowtail^{*}$) is defined similarly. $\rightarrowtail^{=*}$ is the reflexive, transitive closure.*

Each closure has its own inductive principle for proofs.

Definition 32. *Let $\boldsymbol{R}$ be a notion of reduction on Λ, that is, a binary relation on Λ. Then $\boldsymbol{R}$ induces the binary relations*

$\rightarrow_R$	*one step R-reduction,*
$\twoheadrightarrow_R$	*R-reduction and*
$=_R$	*R-equality,*

inductively defined by rules as follows.
$\rightarrow_R$ is the compatible closure of $\boldsymbol{R}$:

$$\frac{\boldsymbol{R}\ M\ N}{M \rightarrow_R N} \qquad \frac{M \rightarrow_R N}{Z\ M \rightarrow_R Z\ N} \qquad \frac{M \rightarrow_R N}{M\ Z \rightarrow_R N\ Z} \qquad \frac{M \rightarrow_R N}{\lambda x.\ M \rightarrow_R \lambda x.\ N}.$$

$\twoheadrightarrow_R$ is the reflexive, transitive closure of $\rightarrow_R$:

$$\frac{M \rightarrow_R N}{M \twoheadrightarrow_R N} \qquad \frac{}{M \twoheadrightarrow_R M} \qquad \frac{M \twoheadrightarrow_R N\ ,\ N \twoheadrightarrow_R L}{M \twoheadrightarrow_R L}.$$

$=_R$ is the equivalence relation generated by $\twoheadrightarrow_R$:

$$\frac{M \twoheadrightarrow_R N}{M =_R N} \qquad \frac{M =_R N}{N =_R M} \qquad \frac{M =_R N\ ,\ N =_R L}{M =_R L}.$$

These relations are defined in HOL with Myra VanInwegen's rule induction package [6]. In addition to the weak and strong rule induction principles provided, we have also proved height-based generalizations of these.

Definition 33 (Diamond property). *Let $\rightarrowtail$ be a binary relation on a set. Then $\rightarrowtail$ satisfies the diamond property (notation $\rightarrowtail \models \Diamond$) if*

$$\forall M\ M_1\ M_2.\ M \rightarrowtail M_1\ \wedge\ M \rightarrowtail M_2\ \Rightarrow\ \exists M_3.\ M_1 \rightarrowtail M_3\ \wedge\ M_2 \rightarrowtail M_3$$

Definition 34 (Church-Rosser). *A notion of reduction $\boldsymbol{R}$ is Church-Rosser (CR) if $\twoheadrightarrow_R$ satisfies the diamond property ($\twoheadrightarrow_R \models \Diamond$).*

7 The Church-Rosser Theorem

We follow Barendregt's presentation [1] of the proof by Tait and Martin-Löf.

Theorem 35. *Let $\rightarrowtail$ be a binary relation on a set. Then $\rightarrowtail \models \Diamond\ \Rightarrow\ \rightarrowtail^* \models \Diamond$.*

Proof: by two nested inductions on the transitive relation.

Definition 36 (Parallel reduction). *Let $\twoheadrightarrow\!\!\!\!\!\!+$ be defined by the rules*

$$(1)\ \frac{}{M \Rrightarrow M} \qquad (3)\ \frac{M \Rrightarrow M'\ ,\ N \Rrightarrow N'}{M\ N \Rrightarrow M'\ N'}$$

$$(2)\ \frac{M \Rrightarrow M'}{\lambda x.\ M \Rrightarrow \lambda x.\ M'} \qquad (4)\ \frac{M \Rrightarrow M'\ ,\ N \Rrightarrow N'}{(\lambda x.\ M)\ N\ \Rrightarrow\ M' \lhd [x := N']}\ .$$

This definition is accompanied by strong and weak rule induction principles. In addition, we prove height-based generalizations of these principles.

Lemma 37. $(\lambda x.\, t_1 = \lambda y.\, t_2) \;\Rightarrow\; (t_1 \lhd [x := u] \;=\; t_2 \lhd [y := u])$

Proof: By theorem 26 eliminate t_1, then use height structural induction on t_2.

Theorem 38. $N \twoheadrightarrow\!\!\!\!\!\!\rightarrow N' \;\Rightarrow\; M \lhd [x := N] \twoheadrightarrow\!\!\!\!\!\!\rightarrow M \lhd [x := N']$.

Proof: by height induction on the structure of M. There are three cases:

Case 1. Show $y \lhd [x := N] \twoheadrightarrow\!\!\!\!\!\!\rightarrow y \lhd [x := N']$. If $y = x$, then this simplifies to $N \twoheadrightarrow\!\!\!\!\!\!\rightarrow N'$, which is given. If $y \neq x$, it becomes $y \twoheadrightarrow\!\!\!\!\!\!\rightarrow y$, true by definition 36(1).

Case 2. Show $t\,u \lhd [x := N] \twoheadrightarrow\!\!\!\!\!\!\rightarrow t\,u \lhd [x := N']$. By the inductive hypotheses, $t \lhd [x := N] \twoheadrightarrow\!\!\!\!\!\!\rightarrow t \lhd [x := N']$ and $u \lhd [x := N] \twoheadrightarrow\!\!\!\!\!\!\rightarrow u \lhd [x := N']$. Then the goal follows by definitions 6 and 36(3).

Case 3. Show $\lambda y.\, t \lhd [x := N] \twoheadrightarrow\!\!\!\!\!\!\rightarrow \lambda y.\, t \lhd [x := N']$. We shift $\lambda y.\, t$ to $\lambda y'.\, t'$ such that capture cannot occur (as done in lemma 28). Then the goal is $\lambda y'.\, (t' \lhd [x := N]) \twoheadrightarrow\!\!\!\!\!\!\rightarrow \lambda y'.\, (t' \lhd [x := N'])$. The inductive hypothesis gives us $t' \lhd [x := N] \twoheadrightarrow\!\!\!\!\!\!\rightarrow t' \lhd [x := N']$. The goal is solved by this and definition 36(2).

Lemma 39. $(\lambda x.\, t_1 = \lambda y.\, t_1') \;\wedge\; t_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2 \;\Rightarrow\; (\lambda x.\, t_2 = \lambda y.\, (t_2 \lhd [x := y]))$

Proof: The conclusion is true if $y \notin \mathsf{FV}(\lambda x.\, t_2)$. From $t_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2$, $\mathsf{FV}\ t_2 \subseteq \mathsf{FV}\ t_1$, so it suffices if $y \notin \mathsf{FV}(\lambda x.\, t_1)$. This follows from $\lambda x.\, t_1 = \lambda y.\, t_1'$.

Lemma 40. $(\lambda x.\, t_1 = \lambda y.\, t_1') \;\wedge\; t_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2 \;\Rightarrow\; t_1' \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2 \lhd [x := y]$.

Proof: by theorem 26, $t_1' = t_1 \lhd [x := y]$; rewrite the above goal as $t_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2 \;\Rightarrow\; t_1 \lhd [x := y] \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2 \lhd [x := y]$ and prove by height strong rule induction on $t_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2$.

Theorem 41. $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow M' \;\wedge\; N \twoheadrightarrow\!\!\!\!\!\!\rightarrow N' \;\Rightarrow\; M \lhd [x := N] \twoheadrightarrow\!\!\!\!\!\!\rightarrow M' \lhd [x := N']$.

Proof: by height strong rule induction on $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow M'$.

Case 1. $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow M'$ is $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow M$. Then the goal follows from theorem 38.

Case 2. $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow M'$ is $t_1\, u_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2\, u_2$, and is a direct consequence of $t_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2$, $u_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow u_2$. By the inductive hypotheses, $t_1 \lhd [x := N] \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2 \lhd [x := N']$ and $u_1 \lhd [x := N] \twoheadrightarrow\!\!\!\!\!\!\rightarrow u_2 \lhd [x := N']$. Then $(t_1 \lhd [x := N])\ (u_1 \lhd [x := N]) \twoheadrightarrow\!\!\!\!\!\!\rightarrow (t_2 \lhd [x := N'])\ (u_2 \lhd [x := N'])$, which is $M \lhd [x := N] \twoheadrightarrow\!\!\!\!\!\!\rightarrow M' \lhd [x := N']$.

Case 3. $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow M'$ is $(\lambda y.\, t_1)\, u_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2 \lhd [y := u_2]$, and is a direct consequence of $t_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2$, $u_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow u_2$. We shift $\lambda y.\, t_1$ to $\lambda z.\, t_1'$ to avoid capture. By $t_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2$ and lemmas 39 and 40, $\lambda y.\, t_2 = \lambda z.\, t_2'$ and $t_1' \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2'$, where $t_2' = t_2 \lhd [y := z]$. Then

$$\begin{array}{rcll}
M \lhd [x := N] & = & (\lambda y.\, t_1)\, u_1 \lhd [x := N] & \\
 & = & (\lambda z.\, (t_1' \lhd [x := N]))\ (u_1 \lhd [x := N]) & \\
 & \twoheadrightarrow\!\!\!\!\!\!\rightarrow & t_2' \lhd [x := N'] \lhd [z := u_2 \lhd [x := N']] & (1)\\
 & = & t_2' \lhd [z := u_2] \lhd [x := N'] & (2)\\
 & = & t_2 \lhd [y := u_2] \lhd [x := N'] & (3)\\
 & = & M' \lhd [x := N']. &
\end{array}$$

Notes: (1) by the induction hypotheses on $t_1' \twoheadrightarrow\!\!\!\!\!\!\rightarrow t_2'$ and $u_1 \twoheadrightarrow\!\!\!\!\!\!\rightarrow u_2$, and defn. 36(4).

(2) by lemma 28, since by choice of z, $z \neq x$ and $z \notin \mathsf{FV}\ N'$.

(3) by lemma 37, $t_2 \lhd [y := u_2] = t_2' \lhd [z := u_2]$, since $\lambda y.\, t_2 = \lambda z.\, t_2'$.

Case 4. $M \twoheadrightarrow M'$ is $\lambda y.\, t_1 \twoheadrightarrow \lambda y.\, t_2$, and is a direct consequence of $t_1 \twoheadrightarrow t_2$. We shift the abstractions to $\lambda z.\, t_1'$ and $\lambda z.\, t_2'$ so no captures can occur. By lemma 40 and $t_1 \twoheadrightarrow t_2$, we have $t_1' \twoheadrightarrow t_2 \lhd [y := z]$. By theorem 26, $t_2 \lhd [y := z] = t_2'$, so $t_1' \twoheadrightarrow t_2'$. The ind. hyp. on $t_1' \twoheadrightarrow t_2'$ gives us $t_1' \lhd [x := N] \twoheadrightarrow t_2' \lhd [x := N']$. Then

$$\begin{array}{rcll} M \lhd [x := N] & = & (\lambda y.\, t_1) \lhd [x := N] & \\ & = & \lambda z.\, (t_1' \lhd [x := N]) & \\ & \twoheadrightarrow & \lambda z.\, (t_2' \lhd [x := N']) & \quad \text{by definition 36(2)} \\ & = & (\lambda y.\, t_2) \lhd [x := N'] & \\ & = & M' \lhd [x := N']. & \end{array}$$

Lemma 42. *(i)* $x \twoheadrightarrow t_2 \;\Rightarrow\; t_2 = x$
(ii) $t\; u \twoheadrightarrow t_2 \;\Rightarrow$
$(\exists t'\; u'.\; t_2 = t'\; u' \;\wedge\; t \twoheadrightarrow t' \;\wedge\; u \twoheadrightarrow u') \;\vee$
$(\exists x\; t_1\; t_1'\; u'.\; t = \lambda x.\, t_1 \;\wedge\; t_2 = t_1' \lhd [x := u'] \;\wedge\; t_1 \twoheadrightarrow t_1' \;\wedge\; u \twoheadrightarrow u')$
(iii) $\lambda x.\, t \twoheadrightarrow t_2 \;\Rightarrow\; (\exists t'.\; t_2 = \lambda x.\, t' \;\wedge\; t \twoheadrightarrow t')$

Proof: by an easy application of the inversion theorems of the definition of $\twoheadrightarrow$. For *(iii)*, we have $\lambda x.\, t = \lambda x'.\, t_1'$, $t_2 = \lambda x'.\, t_2'$, and $t_1' \twoheadrightarrow t_2'$. Then by lemmas 39 and 40 with $t_1' \twoheadrightarrow t_2'$, we can take $t' = t_2' \lhd [x' := x]$.

Theorem 43. *$\twoheadrightarrow$ satisfies the diamond property* ($\twoheadrightarrow\; \models \Diamond$).

Proof: by strong rule induction on $M \twoheadrightarrow M_1$ it will be shown that for all $M \twoheadrightarrow M_2$ there is a M_3 such that $M_1 \twoheadrightarrow M_3$ and $M_2 \twoheadrightarrow M_3$.

Case 1. $M \twoheadrightarrow M_1$ because $M = M_1$. Then we can take $M_3 = M_2$.

Case 2. $M \twoheadrightarrow M_1$ is $\lambda x.\, t \twoheadrightarrow \lambda x.\, t'$ and is a direct consequence of $t \twoheadrightarrow t'$. Then by lemma 42(*iii*), $M_2 = \lambda x.\, t''$. By the induction hypothesis there is a term t''' such that $t' \twoheadrightarrow t'''$ and $t'' \twoheadrightarrow t'''$, and we can take $M_3 = \lambda x.\, t'''$.

Case 3. $M \twoheadrightarrow M_1$ is $t\; u \twoheadrightarrow t'\; u'$ and is a direct consequence of $t \twoheadrightarrow t'$, $u \twoheadrightarrow u'$. By lemma 42(*ii*), there are two subcases.

Subcase 3.1. $M_2 = t''\; u''$ with $t \twoheadrightarrow t''$, $u \twoheadrightarrow u''$. Using the induction hypotheses in the obvious way gives us t''' and u''' with $t' \twoheadrightarrow t'''$, $t'' \twoheadrightarrow t'''$, and similarly for the u's. Then we can take $M_3 = t'''\; u'''$.

Subcase 3.2. $t = \lambda x.\, t_1$, $M_2 = t_1'' \lhd [x := u'']$ and $t_1 \twoheadrightarrow t_1''$, $u \twoheadrightarrow u''$. By lemma 42(*iii*), we have $t' = \lambda x.\, t_1'$ with $t_1 \twoheadrightarrow t_1'$. By the definition of $\twoheadrightarrow$, $\lambda x.\, t_1 \twoheadrightarrow \lambda x.\, t_1''$. which with the induction hypothesis for $t \twoheadrightarrow t'$, gives us t''' with $\lambda x.\, t_1' \twoheadrightarrow t'''$, $\lambda x.\, t_1'' \twoheadrightarrow t'''$. By lemma 42(*iii*), $t''' = \lambda x.\, t_1'''$ with $t_1' \twoheadrightarrow t_1'''$ and $t_1'' \twoheadrightarrow t_1'''$. The induction hypothesis for $u \twoheadrightarrow u'$ gives us u''' with $u' \twoheadrightarrow u'''$, $u'' \twoheadrightarrow u'''$. Then by theorem 41, we can take $M_3 = t_1''' \lhd [x := u''']$.

Case 4. $M \twoheadrightarrow M_1$ is $(\lambda x.\, t)\; u \twoheadrightarrow t' \lhd [x := u']$ and is a direct consequence of $t \twoheadrightarrow t'$, $u \twoheadrightarrow u'$. Again, there are two subcases.

Subcase 4.1. $M_2 = (\lambda x.\, t'')\; u''$ with $t \twoheadrightarrow t''$, $u \twoheadrightarrow u''$. Using the induction hypotheses in the obvious way give us t''', u'''. Then by theorem 41, we can take $M_3 = t''' \lhd [x := u''']$.

Subcase 4.2. $M_2 = t_1'' \lhd [x_1 := u'']$ with $t_1 \twoheadrightarrow t_1''$, $u \twoheadrightarrow u''$, and $\lambda x.\, t = \lambda x_1.\, t_1$. By lemmas 39 and 40, $\lambda x.\, t'' = \lambda x_1.\, t_1''$ and $t \twoheadrightarrow t''$ where $t'' = t_1'' \lhd [x_1 := x]$.

Using the induction hypotheses in the evident way gives us t''' and u''' with $t' \twoheadrightarrow\!\!\!\!\!+\, t'''$, $t'' \twoheadrightarrow\!\!\!\!\!+\, t'''$, $u' \twoheadrightarrow\!\!\!\!\!+\, u'''$, $u'' \twoheadrightarrow\!\!\!\!\!+\, u'''$. Since $\lambda x.\, t'' = \lambda x_1.\, t_1''$, by lemma 37 we have $t_1'' \lhd [x_1 := u''] = t'' \lhd [x := u'']$, and then by theorem 41 we can take $M_3 = t''' \lhd [x := u''']$.

The above fills an omission by Barendregt. Though $M = (\lambda x.\, t)\, u$ be the same in $M \twoheadrightarrow\!\!\!\!\!+\, M_1$ and $M \twoheadrightarrow\!\!\!\!\!+\, M_2$, the x's and t's may be different.

Theorem 44. *$\twoheadrightarrow_\beta$ is the transitive closure of $\twoheadrightarrow\!\!\!\!\!+$ ($\twoheadrightarrow_\beta = \twoheadrightarrow\!\!\!\!\!+^*$).*

Proof: Note that as relations $\to_\beta^= \subseteq \twoheadrightarrow\!\!\!\!\!+ \subseteq \twoheadrightarrow_\beta$. Since $\twoheadrightarrow_\beta$ is the transitive closure of $\to_\beta^=$, so it is of $\twoheadrightarrow\!\!\!\!\!+$. The HOL proof [6] uses 12 lemmas and theorems to support this, but for reasons of space, here we give only Barendregt's justification.

Theorem 45 (The Church-Rosser Theorem). *β is CR.*

Proof: by definition 34 and theorems 35, 43, and 44.

8 Summary and Conclusions

This proof of the Church-Rosser theorem modeled a name-carrying syntax, which is relevant to practical programming languages. We separated two concerns, where α-equivalence and β-reduction were analyzed in two distinct layers. As in [2], this modularized and simplified the proof over some previous efforts.

Although not described here, the development has been extended with ease to η-reduction and $\beta\eta$-reduction. This is an example of the simplicity that comes from a separation of concerns, enabled by the quotient library [6].

Soli Deo Gloria.

References

1. Barendregt, H. P.: *The Lambda Calculus.* North-Holland, 1981.
2. Ford, J., Mason, I. A.: Operational Techniques in PVS – A Preliminary Evaluation, in Proceedings of *Australasian Theory Symposium,* (CATS'01), 2001.
3. Gordon, A. D., Melham, T.: Five Axioms of Alpha-Conversion, in Proceedings of TPHOLs'96, LNCS 1125, 173–190, 1996.
4. Hindley, J. R., Lercher, B., Seldin, J. P.: Introduction to Combinatory Logic. Cambridge University Press, 1972.
5. Huet, G.: Residual Theory in λ-Calculus: A Formal Development. *Journal of Functional Programming,* Vol. 4, No. 3, pp. 371-394, 1994.
6. Homeier, P. V.: `http://www.cis.upenn.edu/~hol/lamcr`.
7. Nipkow, T.: More Church-Rosser Proofs (in Isabelle/HOL). In M. McRobbie and J. K. Slaney (eds.), *Automated Deduction – CADE-13,* LNCS 1104, 733–747, 1996.
8. Pfenning, F.: A Proof of the Church-Rosser Theorem and its Representation in a Logical Framework, Technical Report CMU-CS-92-186, CMU, 1992.
9. Shankar, N.: A Mechanical Proof of the Church-Rosser Theorem, *Journal of the ACM* Vol. 35, No. 3, July 1988, 475–522.
10. Shankar, N.: *Metamathematics, Machines, and Gödel's Proof.* Cambridge, 1994.
11. Vestergaard, R., Brotherston, J.: A Formalized First-Order Confluence Proof for the λ-Calculus using One-Sorted Variable Names. To appear in *12th International Conference on Rewriting Techniques and Applications* (RTA 2001).