More Church-Rosser Proofs

(in Isabelle/HOL)

Tobias Nipkow*
Institut für Informatik, Technische Universität München
(`http://www.in.tum.de/~nipkow/`*)*

Abstract. The proofs of the Church-Rosser theorems for β, η and $\beta \cup \eta$ reduction in untyped λ-calculus are formalized in Isabelle/HOL, an implementation of Higher Order Logic in the generic theorem prover Isabelle. For β-reduction, both the standard proof and Takahashi's are given and compared. All proofs are based on a general theory of commutating relations which supports an almost geometric style of reasoning about confluence diagrams.

1. Introduction

The Church-Rosser theorem for β-reduction is one of the key properties of λ-calculus. The de facto standard proof can be found in Barendregt [1]. It has been machine-checked a number of times already [7, 13, 16, 19]. A recently published alternative proof technique [20] prompted me to apply Isabelle/HOL to this problem. The resulting formalization has the following distinctive features:

η-Reduction Not just β but also, for the first time, η and $\beta \cup \eta$ are treated.

Proof techniques In the case of β, two different proof techniques are compared: the standard one and the one due to Takahashi.

Substitution Two different notions of substitution are formalized and related.

Abstractness Based on a theory of binary relations, including transitive and reflexive closure, an abstract theory of commuting relations is developed which supports an almost geometric style of reasoning.

Automation Most proofs are almost automatic, using a few general purpose tactics.

Although the paper provides a complete formalization of the Church-Rosser proofs, it is not meant as an introduction to the underlying

* Research supported by DFG Schwerpunktprogramm *Deduktion*.

intuitions, which the reader is assumed to be familiar with. The purpose of the paper is to contrast textbook mathematics with its formalization in a theorem prover, and to contrast this latest exercise with other published formalizations. The overall structure of the proofs follows Chapter 3 of Barendregt [1], except that all abstract relational reasoning is concentrated in one section.

The complete formalization (including proofs) is available on the web at `http://isabelle.in.tum.de/library/HOL/Lambda/`.

2. Isabelle/HOL

Isabelle is an interactive theorem prover which can be instantiated with different object-logics. One particularly well-developed instantiation is Isabelle/HOL, which supports Church's formulation of Higher Order Logic and is very close to Gordon's HOL system [4]. In the remainder of the paper HOL is short for Isabelle/HOL.

We present no proofs but merely definitions and theorems. Hence it suffices to introduce HOL's surface syntax. A detailed introduction to Isabelle and HOL can be found elsewhere [11, 10].

Formulae The syntax is standard, except that there are two universal quantifiers ($\forall$ and $\bigwedge$), two implications ($\longrightarrow$ and $\Longrightarrow$) and two equalities ($=$ and $\equiv$) which stem from the object and meta-logic, respectively. The distinction can be ignored while reading this paper. The notation $[\![A_1; \ldots; A_n]\!] \Longrightarrow A$ is short for the nested implication $A_1 \Longrightarrow \ldots \Longrightarrow A_n \Longrightarrow A$.

Types follow the syntax for ML-types, except that the function arrow is $\Rightarrow$ rather than $\rightarrow$.

Theories introduce constants with the keyword **consts**, non-recursive definitions with **defs**, and primitive recursive definitions with **primrec**. Further constructs are explained as we encounter them.

Although none of the proofs are presented, their complexity is usually indicated. If it says that some proof is automatic, it means that it was either solved by rewriting or by the "classical reasoner", `fast_tac` in Isabelle parlance [12]. The latter provides a reasonable degree of automation for predicate calculus proofs. Note, however, that its success depends on the right selection of lemmas supplied as parameters. Arithmetic goals are solved by a special tactic based on Fourier-Motzkin elimination implemented by the author.

3. Abstract Reduction Systems

3.1. Relations

The whole development is based on the HOL theory of binary relations, which are simply sets of pairs, i.e. of type $(\tau \times \tau)$ `set`. In particular there are two polymorphic operations * and $^=$ for the transitive-reflexive and the reflexive closure, respectively. They come with a number of theorems, in particular induction for *

```
⟦ (a,b) ∈ R*;
  P a;
  ⋀y z. ⟦ (a,y) ∈ R*; (y,z) ∈ R;  P y ⟧ ⟹ P z
⟧ ⟹ P b
```

and some equational laws like `(R*)* = R*` and `(R=)* = R*`.

3.2. Squares

The rewriting literature is full of confluence-like notions which are defined formally on the predicate calculus level but are accompanied by diagrams in the form of squares. Proofs are almost exclusively performed at the level of diagrams. We mimic this approach by introducing the notion of a `square`:

```
consts square :: (α × α)set ⇒ (α × α)set ⇒ (α × α)set ⇒
                 (α × α)set ⇒ bool
defs   square R S T U ≡
                 ∀x y. (x,y) ∈ R ⟶ (∀z. (x,z) ∈ S ⟶
                       (∃ u. (y,u) ∈ T ∧ (z,u) ∈ U))
```

where α is a type variable, i.e. `square` is polymorphic. Thus the formula `square R S T U` corresponds directly to the diagram

$$\begin{tikzcd} \cdot \arrow[r, "S"] \arrow[d, "R"'] & \cdot \arrow[d, dashed, "U"] \\ \cdot \arrow[r, dashed, "T"'] & \cdot \end{tikzcd}$$

Although `square` is simply an abbreviation and all reasoning could be conducted directly on the predicate calculus level, there are distinct advantages in conducting arguments at the level of squares: for the human, proofs are much easier to find and explain on this level than on the level of quantifiers. In the context of automated deduction, there is another reason: induction, which many automatic systems cannot handle, can be encapsulated as a lemma about squares, which is then amenable to automation (see the Strip Lemma below).

For the convenience of the reader, occurrences of squares in formulae are often replaced by the corresponding diagram.

First we need a few basic lemmas about `square`:

$$\begin{tikzcd} \cdot \arrow[r,"\mathtt{S}"] \arrow[d,"\mathtt{R}"'] & \cdot \arrow[d,dashed,"\mathtt{U}"] \\ \cdot \arrow[r,dashed,"\mathtt{T}"'] & \cdot \end{tikzcd} \implies \begin{tikzcd} \cdot \arrow[r,"\mathtt{R}"] \arrow[d,"\mathtt{S}"'] & \cdot \arrow[d,dashed,"\mathtt{T}"] \\ \cdot \arrow[r,dashed,"\mathtt{U}"'] & \cdot \end{tikzcd} \tag{1}$$

$$\begin{tikzcd} \cdot \arrow[r,"\mathtt{S}"] \arrow[d,"\mathtt{R}"'] & \cdot \arrow[d,dashed,"\mathtt{U}"] \\ \cdot \arrow[r,dashed,"\mathtt{T}"'] & \cdot \end{tikzcd} \wedge\ \mathtt{T} \subseteq \mathtt{T'} \implies \begin{tikzcd} \cdot \arrow[r,"\mathtt{S}"] \arrow[d,"\mathtt{R}"'] & \cdot \arrow[d,dashed,"\mathtt{U}"] \\ \cdot \arrow[r,dashed,"\mathtt{T'}"'] & \cdot \end{tikzcd} \tag{2}$$

$$\begin{tikzcd} \cdot \arrow[r,"\mathtt{S}"] \arrow[d,"\mathtt{R}"'] & \cdot \arrow[d,dashed,"\mathtt{R}^{=}"] \\ \cdot \arrow[r,dashed,"\mathtt{T}"'] & \cdot \end{tikzcd} \wedge\ \mathtt{S} \subseteq \mathtt{T} \implies \begin{tikzcd} \cdot \arrow[r,"\mathtt{S}"] \arrow[d,"\mathtt{R}^{=}"'] & \cdot \arrow[d,dashed,"\mathtt{R}^{=}"] \\ \cdot \arrow[r,dashed,"\mathtt{T}"'] & \cdot \end{tikzcd} \tag{3}$$

$$\begin{tikzcd} \cdot \arrow[r,"\mathtt{S}"] \arrow[d,"\mathtt{R}"'] & \cdot \arrow[d,dashed,"\mathtt{T}"] \\ \cdot \arrow[r,dashed,"\mathtt{S}"'] & \cdot \end{tikzcd} \implies \begin{tikzcd} \cdot \arrow[r,"\mathtt{S}"] \arrow[d,"\mathtt{R}^*"'] & \cdot \arrow[d,dashed,"\mathtt{T}^*"] \\ \cdot \arrow[r,dashed,"\mathtt{S}"'] & \cdot \end{tikzcd} \tag{4}$$

Lemma (4) is a generalized form of what is usually called the *Strip Lemma*. It is easily proved by induction on $\mathtt{R}^*$ and it also encapsulates induction: all of the subsequent lemmas about `square` which require induction are proved by means of the Strip Lemma instead.

As an example of the diagrammatic proof method afforded by `square` we examine the proof of the *Commutation Lemma* due to Hindley [6] and Rosen [17]

$$\mathtt{square\ R\ S\ (S^*)\ (R^=)} \implies \mathtt{square\ (R^*)\ (S^*)\ (S^*)\ (R^*)} \tag{5}$$

which is the key lemma for proving commutation of β and η later on:

$$\begin{tikzcd} \cdot \arrow[r,"\mathtt{S}"] \arrow[d,"\mathtt{R}"'] & \cdot \arrow[d,dashed,"\mathtt{R}^{=}"] \\ \cdot \arrow[r,dashed,"\mathtt{S}^*"'] & \cdot \end{tikzcd} \overset{(3)}{\Longrightarrow} \begin{tikzcd} \cdot \arrow[r,"\mathtt{S}"] \arrow[d,"\mathtt{R}^{=}"'] & \cdot \arrow[d,dashed,"\mathtt{R}^{=}"] \\ \cdot \arrow[r,dashed,"\mathtt{S}^*"'] & \cdot \end{tikzcd} \overset{(1)}{\Longrightarrow} \begin{tikzcd} \cdot \arrow[r,"\mathtt{R}^{=}"] \arrow[d,"\mathtt{S}"'] & \cdot \arrow[d,dashed,"\mathtt{S}^*"] \\ \cdot \arrow[r,dashed,"\mathtt{R}^{=}"'] & \cdot \end{tikzcd} \overset{(4)}{\Longrightarrow}$$

$$\begin{tikzcd} \cdot \arrow[r,"\mathtt{R}^{=}"] \arrow[d,"\mathtt{S}^*"'] & \cdot \arrow[d,dashed,"\mathtt{S}^*"] \\ \cdot \arrow[r,dashed,"\mathtt{R}^{=}"'] & \cdot \end{tikzcd} \overset{(1)}{\Longrightarrow} \begin{tikzcd} \cdot \arrow[r,"\mathtt{S}^*"] \arrow[d,"\mathtt{R}^{=}"'] & \cdot \arrow[d,dashed,"\mathtt{R}^{=}"] \\ \cdot \arrow[r,dashed,"\mathtt{S}^*"'] & \cdot \end{tikzcd} \overset{(4)}{\Longrightarrow} \begin{tikzcd} \cdot \arrow[r,"\mathtt{S}^*"] \arrow[d,"\mathtt{R}^*"'] & \cdot \arrow[d,dashed,"\mathtt{R}^*"] \\ \cdot \arrow[r,dashed,"\mathtt{S}^*"'] & \cdot \end{tikzcd}$$

Trivial as this proof may look, it should be kept in mind that it contains two hidden inductions. We refrain from showing any further such chains, most of which `fast_tac` finds automatically.

3.3. Confluence

The notion of a `square` is sufficient to define the usual confluence-like concepts:

consts `commute` :: $(\alpha \times \alpha)$`set` $\Rightarrow$ $(\alpha \times \alpha)$`set` $\Rightarrow$ `bool`
`confluent, diamond` :: $(\alpha \times \alpha)$`set` $\Rightarrow$ `bool`

defs `commute R S` $\equiv$ `square R S S R`
`diamond R` $\equiv$ `commute R R`
`confluent R` $\equiv$ `diamond`$(\mathtt{R}^*)$

We simply list the key lemmas, which form the basis for the confluence proofs of β, η and $\beta \cup \eta$, respectively:

$$[\![\mathtt{diamond\ R};\ \mathtt{S} \subseteq \mathtt{R};\ \mathtt{R} \subseteq \mathtt{S}^*]\!] \Longrightarrow \mathtt{confluent\ S} \tag{6}$$

follows essentially from the Strip Lemma and the fact that $\mathtt{R}^* = \mathtt{S}^*$.

$$\mathtt{square\ R\ R\ R}^{=}\ \mathtt{R}^{=} \Longrightarrow \mathtt{confluent\ R} \tag{7}$$

follows directly from the Commutation Lemma (5) and (2). The *Commutative Union Theorem* is due to Hindley [6] and Rosen [17]:

$$[\![\mathtt{confluent\ R};\ \mathtt{confluent\ S};\ \mathtt{commute}(\mathtt{R}^*)(\mathtt{S}^*)]\!]$$
$$\Longrightarrow \mathtt{confluent}(\mathtt{R} \cup \mathtt{S}) \tag{8}$$

Finally, we come to the Church-Rosser property itself:

consts `Church_Rosser` :: $(\alpha \times \alpha)$`set` $\Rightarrow$ `bool`

defs `Church_Rosser R` $\equiv$
$\forall$`x y. (x,y)` $\in$ `(R` $\cup$ `converse(R))`* $\longrightarrow$
$(\exists$`z. (x,z)` $\in$ $\mathtt{R}^*$ $\wedge$ `(y,z)` $\in$ $\mathtt{R}^*)$

where `converse R` $\equiv$ `{(y,x) . (x,y)`$\in$`R}`. The proof of the standard result

`confluent R = Church_Rosser R`

is short but requires a certain amount of guidance.

4. Lambda terms and reductions

Lambda terms are represented in de Bruijn notation [3] which is conveniently expressed as an inductive data type with three constructors for variables, application and abstraction:

```
datatype dB = Var nat | $ dB dB (infixl 200) | Abs dB
```

Note that application is written infix. Thus the term $\lambda x.(x\ x)$ is represented as `Abs(Var 0 $ Var 0)`.

Type `nat` is the type of natural numbers with the constructors `0` and `Suc`. Unless `Suc` is required to for pattern matching reasons (e.g. on the left-hand side of a rewrite rule), we use the more familiar `+1`.

4.1. Lifting and substitution

In order to describe β-reduction for de Bruijn terms we need the notions of "lifting" and substitution. The starting point of our formalization is the report by Rasmussen [16] who ported Huet's theory of residuals [7] to Isabelle/ZF, Isabelle's set theory. Initially we benefit a lot from Rasmussen's work but we soon diverge because (a) we take a direct route to the Church-Rosser theorem instead of going via residuals and (b) we also include η-reduction.

We define two functions `lift` and `subst` which perform substitution.

```
consts
  lift :: dB ⇒ nat ⇒ dB
  subst:: dB ⇒ dB ⇒ nat ⇒ dB  (_[_/_] [300,0,0] 300)
primrec
  lift (Var i) k =  (if i < k then Var i else Var(i+1))
  lift (s $ t) k =  (lift s k) $ (lift t k)
  lift (Abs s) k =  Abs(lift s (k+1))
primrec
  (Var i)[s/k]  =  (if k < i then Var(i-1)
                    else if i = k then s else Var i)
  (t $ u)[s/k]  =  t[s/k] $ u[s/k]
  (Abs t)[s/k]  =  Abs (t[lift s 0 / k+1])
```

`lift t i` increments all free variables in `t` which are greater or equal `i`.

`t[s/i]` is syntactic sugar for `subst t s i` and mimics the traditional $t[s/x]$. Observe, however, that `t[s/i]` does not only perform substitution: it also decrements those free variables in `t` which are greater than `i`. This is necessary for both β and η-reduction.

Explicit substitution calculi grate unless they are oiled by a set of well chosen lemmas. In our case the following lemmas suffice, of which the last one is the crucial one.

```
i<k+1 ⟶ lift (lift t i) (Suc k) = lift (lift t k) i
j<i+1 ⟶ lift (t[s/j]) i = (lift t (i+1))[lift s i/j]
```

```
i<j+1 ⟶ lift (t[s/j]) i = (lift t i)[lift s i/j+1]
(lift t k)[s/k] = t
i<j+1 ⟶ t[lift v i/Suc j][u[v/j]/i] = t[u/i][v/j]
```

All lemmas are proved by structural induction on `t`, where the application and abstraction cases go through by simplification. The variable cases also require a call of the arithmetic tactic.

Initially I tried to find and prove these lemmas from scratch but soon decided to steal from Rasmussen's ZF proofs instead, which has obvious advantages:

- I did not have to find this collection of non-obvious lemmas myself.
- In a first version of these proofs [9], before I had implemented the arithmetic tactic, the painfully detailed arithmetic proofs could be ported more or less directly to HOL.

Without this productivity boost, the whole development would have taken much longer than it did.

4.2. β-Reduction

Once we have substitution, β-reduction is easy:

```
consts  beta :: (dB × dB) set
syntax  →β :: dB ⇒ dB ⇒ bool  (infixl 50)
translations  s →β t  ≡  (s,t) ∈ beta
inductive beta
  (Abs s) $ t →β s[t/0]
  s →β t ⟹ s$u →β t$u
  s →β t ⟹ u$s →β u$t
  s →β t ⟹ Abs s →β Abs t
```

This introduces `beta` as a relation on de Bruijn terms and `s` $\to_\beta$ `t` as syntactic sugar for `(s,t)` $\in$ `beta`. Beta-reduction is an inductively defined relation. Except for the syntactic sugar and the trick of using a generic *-operator, this is very much standard.

We also introduce infix syntax for `beta`$^=$ and `beta`*:

```
syntax  →β⁼, ↠β :: dB ⇒ dB ⇒ bool  (infixl 50)
translations  s →β⁼ t  ≡  (s,t) ∈ beta⁼
              s ↠β t  ≡  (s,t) ∈ beta*
```

It helps to know that $\twoheadrightarrow_\beta$ is also a congruence,

```
s ↠β s' ⟹ Abs s ↠β Abs s'
s ↠β s' ⟹ s $ t ↠β s' $ t
t ↠β t' ⟹ s $ t ↠β s $ t'
⟦ s ↠β s'; t ↠β t' ⟧ ⟹ s $ t ↠β s' $ t'
```

which follows almost automatically using a few basic lemmas about *, in particular induction.

4.3. η-Reduction

At this point I had to improvise because there was no role model: η-reduction had not been formalized in a theorem prover before. Although there are two different formalizations of η in the context of de Bruijn notation in the literature, both have their problems (see discussion below). Hence I decided to try a new variation firmly based on the usual definition of η:

$$\lambda x.(s\ x) \to_\eta s \qquad \text{if } x \text{ is not free in } s \tag{η}$$

The definition of freeness is easy:

```
consts free :: dB ⇒ nat ⇒ bool
primrec
  free (Var j) i = (j=i)
  free (s $ t) i = (free s i ∨ free t i)
  free (Abs s) i = free s (i+1)
```

The predicate `free t i` holds iff `i` is free in `t`.

The formalization of the reduction step itself is trickier: we cannot simply replace `Abs(s $ Var 0)` by `s` but we have to decrement the variables in `s` as well, because the enclosing `Abs` has disappeared. Fortunately, substitution also decrements: if `0` is not free in `s`, then `s[u/0]` merely decrements all variables; the value of `u` is immaterial. Thus we can formalize $\to_\eta$ as follows:

```
consts eta  :: (dB × dB) set
syntax  →η :: dB ⇒ dB ⇒ bool (infixl 50)
translations   s →η t  ≡  (s,t) ∈ eta
inductive eta
   ¬free s 0 ⟹ Abs(s $ Var 0) →η s[u/0]          (9)
   s →η t ⟹ s$u →η t$u
   s →η t ⟹ u$s →η u$t
   s →η t ⟹ Abs s →η Abs t
```

Clause (9) corresponds to (η). The fact that `u` is only a dummy variable is the gist of the following lemma:

```
¬free s i ⟹ s[t/i] = s[u/i]
```

It is proved automatically by induction on `s`.

The introduction of the new function `free` necessitates two lemmas expressing how `free` interacts with `lift` and `subst`:

```
free (lift s k) i = (i < k ∧ free s i ∨
                     k < i ∧ free s (i-1))
free (s[t/k]) i = (free s k ∧ free t i ∨
                   free s (if i<k then i else i+1))
```

Both are quite tricky to come up with and are proved by induction on `s` and calls to the arithmetic tactic. The following two lemmas are familiar from the usual treatment of η:

$$\texttt{s} \to_\eta \texttt{t} \Longrightarrow \texttt{free t i = free s i}$$
$$\texttt{s} \to_\eta \texttt{t} \Longrightarrow \texttt{s[u/i]} \to_\eta \texttt{t[u/i]}$$

They show that $\to_\eta$ preserves the set of free variables in a term and is closed under substitution. Both are automatic by induction on $\texttt{s}\to_\eta\texttt{t}$. This concludes our basic stock of lemmas about $\to_\eta$.

We also introduce infix syntax $\to_\eta^=$ and $\twoheadrightarrow_\eta$ for $\texttt{eta}^=$ and $\texttt{eta}^*$, and we show that $\twoheadrightarrow_\eta$ is a congruence, exactly as for β-reduction.

Before we continue in the development, let us briefly compare the above definition of $\to_\eta$ with two other formalizations in the de Bruijn literature.

Briaud [2] also uses substitution to decrement variables. In order to avoid a conditional rule he introduces a fictitious term $\perp$ and defines

$$\texttt{Abs(s \$ Var 0)} \to_\eta \texttt{s[}\perp\texttt{/0]}$$

Although he avoids tricky lemmas about the interaction of `free` with `subst` and `lift`, he pays for it in terms of the complications $\perp$ causes. Since I was able to carry out my proofs quite naturally without $\perp$, I saw no reason to pollute my terms with it.

Hardin [5] briefly considers, in our notation, the following definition of $\to_\eta$:

$$\texttt{Abs(lift s 0 \$ Var 0)} \to_\eta \texttt{s} \tag{10}$$

She rejects it because in her context the subterm `lift s 0` gives rise to critical pairs with the rules for `lift`. Instead she settles for a conditional variant of (10) which leads to quite involved confluence proofs. I was able to show that (10) defines the same relation as (9). The key result is the equivalence

$$\texttt{(}\neg\texttt{free s i) = (}\exists\ \texttt{t. s = lift t i)} \tag{11}$$

which is shown by induction on `s` and a number of case distinctions and explicit instantiations of $\exists$ `t`. As a trivial consequence we obtain

```
(∀s u. (¬free s 0) ⟶ R (Abs(s $ Var 0)) (s[u/0]) )  =
(∀s.                  R (Abs(lift s 0 $ Var 0)) (s) )
```

where `R` is an arbitrary relation. It expresses that (9) and (10) define the same relation.

Although (10) looks very attractive because it avoids the additional function `free`, I found that when I replaced (9) by (10), the confluence proofs became quite tricky. Therefore I did not pursue this alternative any further.

5. Confluence of β

The key idea is to use lemma (6) to reduce confluence of $\rightarrow_\beta$ to the diamond property of a new reduction, say $\Rightarrow_\beta$, such that $\Rightarrow_\beta$ lies in between $\rightarrow_\beta$ and $\twoheadrightarrow_\beta$.

5.1. Parallel β-reduction

It is well known that the following form of parallel (and nested) β-reduction has the desired properties:

consts `par_beta :: (dB × dB) set`

syntax `⇒β :: dB ⇒ dB ⇒ bool  (infixl 50)`

translations `s ⇒β t  ≡  (s,t) ∈ par_beta`

inductive `par_beta`

```
Var n ⇒β Var n
s ⇒β t ⟹ Abs s ⇒β Abs t
⟦ s ⇒β s'; t ⇒β t' ⟧ ⟹ s $ t ⇒β s' $ t'
⟦ s ⇒β s'; t ⇒β t' ⟧ ⟹ (Abs s) $ t ⇒β s'[t'/0]
```

It is a simple inductive consequence that $\Rightarrow_\beta$ is reflexive.

The proofs of `beta` $\subseteq$ `par_beta` and `par_beta` $\subseteq$ `beta`* are again, modulo the application of induction, automatic. Using lemma (6), it only remains to prove the diamond property of $\Rightarrow_\beta$.

However, before we get there, we need two lemmas, namely that $\Rightarrow_\beta$ is compatible with lifting and substitution:

```
t ⇒β t' ⟶ lift t k ⇒β lift t' k
s ⇒β s' ⟶ t ⇒β t' ⟶ t[s/k] ⇒β t'[s'/k]
```

Both lemmas are proved by induction on `t`. The second is the key to the diamond property proof, two different versions of which are now presented.

5.2. Takahashi's proof

Takahashi [20] needs a mere page to sketch the complete confluence proof, without omitting any essential details, which is probably the best one can do on paper. This succinctness is achieved by using the notion of *complete development* of a term:

```
consts  cd  :: dB ⇒ dB
recdef cd (measure size)
  cd(Var n) = Var n
  cd(Var n     $ t) = Var n $ cd t
  cd((s1 $ s2) $ t) = cd(s1 $ s2) $ cd t
  cd(Abs u     $ t) = (cd u)[cd t/0]
  cd(Abs s) = Abs(cd s)
```

The keyword **recdef** starts a recursive definition via pattern matching. In our case totality is (automatically) proved via the size measure on terms. Essentially, `cd` descends recursively through the term contracting all the β-redexes it finds on the way.

It is now straightforward to prove the *completion theorem* for `cd`:

$$\texttt{s} \Rightarrow_\beta \texttt{t} \longrightarrow \texttt{t} \Rightarrow_\beta \texttt{cd s}$$

Apart from the induction on `s` and a case distinction, the proof is automatic. This is the result of combined simplification and logical reasoning.

The proof of `diamond par_beta`, i.e. the diamond property of $\Rightarrow_\beta$, is now completely automatic: the existential variable `u` in the definition of `diamond`, i.e. `square`, is instantiated by `cd x` with the help of the completion theorem and unification.

5.3. The standard proof

Traditionally [1], the diamond property is proved directly. This is less slick than Takahashi's proof because it involves a laborious case distinction on the relative positions of the two redexes contracted at the root of the diamond, and each case leads to a different completion of the diamond. This, however, does not bother Isabelle: `diamond par_beta` is proved automatically, modulo the initial application of induction on `par_beta`. This was a little surprising to me, because of the required existential witnesses and the lack of an obvious candidate `cd x`.

This raises the question if anything has been gained by the formalization of Takahashi's proof. The answer seems to be no: Takahashi requires an additional function `cd` and the proof of the completion theorem for `cd` is in fact a little more complicated than the direct proof of the diamond property of $\Rightarrow_\beta$. The ingenuity of her approach is wasted in the presence of mindless search procedures, aka tactics.

6. Confluence of $\beta \cup \eta$

Confluence of η on its own is an anticlimax: no further lemmas are required, `square eta eta (eta`$^=$`) (eta`$^=$`)`

is proved automatically by induction on `eta`, and `confluent eta` follows directly by lemma (7).

The real challenge is confluence of $\beta \cup \eta$. We follow Hindley and Rosen by combining confluence of β and η (which we already have) with the commutation of $\twoheadrightarrow_\beta$ and $\twoheadrightarrow_\eta$ (which remains to be proved). The latter requires some further lemmas

$$\texttt{s} \rightarrow_\beta \texttt{t} \Longrightarrow \texttt{free t i} \longrightarrow \texttt{free s i}$$
$$\texttt{s} \rightarrow_\beta \texttt{t} \Longrightarrow \texttt{s[u/i]} \rightarrow_\beta \texttt{t[u/i]}$$

which are proved automatically by induction on $\texttt{s} \rightarrow_\beta \texttt{t}$. They express the well known facts that $\rightarrow_\beta$ cannot introduce new free variables and is closed under substitution.

We also need the following very de Bruijn specific lemma

```
t[Var i/i] = t[Var i/i+1]
```

which is proved by induction on `t` and some automatic arithmetic. For the perplexed: the lemma is true because on the left-hand side `Var i` is replaced by `Var i` and all variables above `i` are decremented, and on the right-hand side `Var i` is unchanged, `Var(i+1)` is replaced by `Var i`, and all variables above `i+1` are decremented.

Finally there are two more lemmas about the interaction of $\rightarrow_\eta$ with lifting and substitution

$$\texttt{s} \rightarrow_\eta \texttt{t} \Longrightarrow \texttt{lift s i} \rightarrow_\eta \texttt{lift t i}$$
$$\texttt{s} \rightarrow_\eta \texttt{t} \Longrightarrow \texttt{u[s/i]} \twoheadrightarrow_\eta \texttt{u[t/i]}$$

which are proved by induction on $\texttt{s} \rightarrow_\eta \texttt{t}$ and `u` respectively. The proof of the key commutation lemma

$$\texttt{square beta eta (eta}^*\texttt{) (beta}^=\texttt{)} \tag{12}$$

is by induction on the definition of `beta`. Each resulting subgoal is solved automatically. Combining the Commutation Lemma (5) and the Commutative Union Theorem (8) with the confluence of β and η and lemma (12),

$$\texttt{confluent (beta} \cup \texttt{eta)}$$

follows automatically.

7. An optimization

The code for lifting and substitution given above is used in most theoretical investigations of de Bruijn terms because it only requires successor and predecessor on natural numbers. However, most implementations use an optimized form of substitution and lifting, which replaces n 1-step liftings by 1 n-step lifting:

```
consts  substn :: dB ⇒ dB ⇒ nat ⇒ dB
        liftn  :: nat ⇒ dB ⇒ nat ⇒ dB
primrec
  liftn n (Var i) k = (if i<k then Var i else Var(i+n))
  liftn n (s $ t) k = (liftn n s k) $ (liftn n t k)
  liftn n (Abs s) k = Abs(liftn n s (k+1))
primrec
  substn (Var i) s k = (if k < i then Var(i-1)
                        else if i = k then liftn k s 0
                             else Var i)
  substn (t $ u) s k = (substn t s k) $ (substn u s k)
  substn (Abs t) s k = Abs (substn t s (k+1))
```

An exception is Huet, who uses the above definitions in his proofs, which complicates them severely. On top of the lemmas in Section 4.1, he needs a number of further lemmas dealing with addition, among them the following beauty:

```
substn (substn t s p) u (p+n) =
substn (subst t u (Suc(p+n))) (substn s u n) p
```

We show now that you can have your cake and eat it too: do your proofs with the simple form of substitution and lifting, and prove the optimized versions equivalent later on. This turns out to be much simpler than working with the optimized code directly. The following sequence of lemmas does the trick:

```
liftn 0 t k = t
liftn (Suc n) t k = lift (liftn n t k) k
substn t s k = t[liftn k s 0 / k]
```

All three are proved by induction on `t` followed by simplification. As a corollary to the last lemma we obtain `substn t s 0 = t[s/0]`. Thus we can safely replace `(Abs t) $ s` $\rightarrow_\beta$ `t[s/0]` by `(Abs t) $ s` $\rightarrow_\beta$ `substn t s 0` in the definition of $\rightarrow_\beta$.

8. Comparisons

A comparison with the proofs by Shankar [19], Pfenning [13], Huet [7] and Rasmussen [16] is difficult because the frameworks and aims differ significantly: Shankar uses a much weaker logic than the others (no explicit quantifiers), and Pfenning uses higher-order abstract syntax, thus bypassing de Bruijn terms in favour of the built-in λ-abstraction of ELF. Huet and Rasmussen develop a general theory of residuals, which yields the Church-Rosser theorem as a by-product. Hence we only highlight how our approach differs from the others.

There is also the work by McKinna and Pollack [8, 14] who have proved the Church-Rosser theorem in LEGO using Takahashi's approach. What sets their work apart is that they formalize λ-terms with named variables rather than de Bruijn indices. However, they do not provide many details, since for them the Church-Rosser theorem is only a small part of a much larger development.

Finally there is the formalization of a λ-calculus with explicit substitutions in Coq due to Saïbi [18]. His proof is quite indirect because for the most part he is concerned with showing confluence of the substitution calculus, which he obtaines via termination and local confluence. Because Huet's formalization is also done in Coq but is much closer to ours, we do not discuss Saïbi's work further.

8.1. Proof techniques

We formalized both Takahashi's and the standard proof of the diamond property. Although Takahashi's approach is intellectually appealing, the standard direct proof turned out to be simpler and shorter, since its many case distinctions disappear through automation. For systems with a low degree of automation, for example LEGO and Coq, Takahashi's method may be preferable. It may also be preferable for the Boyer-Moore prover because it yields a simple witness function, which their quantifier-free logic requires.

Of course, Takahashi's approach may well turn out to be significantly better than the traditional methods when we try to formalize more advanced theorems like standardization [20].

8.2. Substitution

We have followed Shankar and Rasmussen in using the 1-step lifting function `lift` to simplify the proofs of the basic substitution lemmas. This paid off because these lemmas required about a third of the work (before arithmetic was automated, and including the failed attempts before porting Rasmussen's proofs) and would have been much harder had we used Huet's optimized versions `liftn` and `substn`. Fortunately we could prove both forms of lifting and substitution equivalent quite easily. This is in accordance with the accepted wisdom that correctness proofs should always start from the simplest version of a system and that optimizations should come later.

8.3. Abstractness

All of Section 3 is independent of λ-calculus. It is part of a general theory of relations and can be reused in many other contexts. Although it has probably not shortened our development by much, this abstractness has definitely made the proofs more transparent and is an investment for the future. The other authors conduct all arguments in terms of the specific relations $\rightarrow_\beta$, $\twoheadrightarrow_\beta$ and $\Rightarrow_\beta$, and even define the transitive-reflexive closure of $\rightarrow_\beta$ via an explicit inductive definition instead of using a generic operation like *.

8.4. Automation

Most of the Isabelle proofs are almost automatic, using a few general purpose tactics. There are 60 theorems that require 140 tactic invocations, i.e. roughly 2.3 tactics per theorem. (If we include theorem (11), which is has nothing to do with the Church-Rosser proof, the average number of tactics per theorem increases to 3.) Loading the complete development takes 66 seconds on a Sun ULTRA 10.

In terms of lemmas required, Shankar's proof has a similar granularity as ours, but each individual lemma is proved automatically. Due to the absence of inductively defined relations and existential quantifiers, the formalization required by the Boyer-Moore prover is not exactly natural. Pfenning's proofs are the opposite: they follow the traditional mathematical argument quite closely, but provide no automation at all. With respect to automation, Huet's proofs are closer to Pfenning's, Rasmussen's are closer to ours.

8.5. Resources — not a comparison

In the first round [9], the complete proof effort took 9 days, of which 3 days were taken up by the ground work including confluence of β, and a further 6 days to develop the theory of squares, η and $\beta \cup \eta$. Of those 6 days, almost 2 were spent developing the de Bruijn theory of η on paper, and one was wasted with an abortive attempt to define η using an explicit decrement function rather than `s[u/0]`.

In a second round, I spent 5 days writing a simple arithmetic tactic and another 2 days cleaning everything up, including a simplification of η.

The size (in number of lines) of the final theories, including all comments, definitions and proofs, is shown in the following table:

Relations	130
Terms, Substitution, β	200
Confluence of β	140
η, Confluence of $\beta \cup \eta$	250

Acknowledgements

Thanks are due to James McKinna for detailed discussions, to Randy Pollack for his short cut in the abstract confluence proofs [15], and to Konrad Slind and Larry Paulson for their excellent suggestions for improvements.

References

1. Hendrik Pieter Barendregt. *The Lambda Calculus, its Syntax and Semantics.* North-Holland, 2nd edition, 1984.
2. Daniel Briaud. An explicit Eta rewrite rule. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Typed Lambda Calculi and Applications*, volume 902 of *Lect. Notes in Comp. Sci.*, pages 94–108. Springer-Verlag, 1995.
3. N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.
4. M.J.C. Gordon and T.F. Melham, editors. *Introduction to HOL: a theorem-proving environment for higher order logic.* Cambridge University Press, 1993.
5. Thérèse Hardin. Eta-conversion for the language of explicit substitutions. In H. Kirchner and G. Levi, editors, *Algebraic and Logic Programming*, volume 632 of *Lect. Notes in Comp. Sci.*, pages 306–321. Springer-Verlag, 1992.
6. J. Roger Hindley. *The Church-Rosser Property and a Result in Combinatory Logic.* PhD thesis, University of Newcastle-upon-Tyne, 1964.
7. Gérard Huet. Residual theory in λ-calculus: A formal development. *J. Functional Programming*, 4:371–394, 1994.
8. James McKinna and Robert Pollack. Pure type systems formalized. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lect. Notes in Comp. Sci.*, pages 289–305. Springer-Verlag, 1993.
9. Tobias Nipkow. More Church-Rosser proofs (in Isabelle/HOL). In M. McRobbie and J.K. Slaney, editors, *Automated Deduction — CADE-13*, volume 1104 of *Lect. Notes in Comp. Sci.*, pages 733–747. Springer-Verlag, 1996.
10. Tobias Nipkow. *Isabelle/HOL. The Tutorial*, 1998. Unpublished Manuscript. Available at `www.in.tum.de/~nipkow/pubs/HOL.html`.
11. Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 1994.
12. Lawrence C. Paulson. Generic automatic proof tools. In R. Veroff, editor, *Automated Reasoning and its Applications: Essays in Honor of Larry Wos*, pages 23–47. MIT Press, 1997.
13. Frank Pfenning. A proof of the Church-Rosser theorem and its representation in a logical framework. *J. Automated Reasoning.* To appear.
14. Robert Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions.* PhD thesis, University of Edinburgh, 1994.
15. Robert Pollack. Polishing up the Tait–Martin-Löf proof of the Church–Rosser theorem. Unpublished manuscript, January 1995.
16. Ole Rasmussen. The Church-Rosser theorem in Isabelle: A proof porting experiment. Technical Report 364, University of Cambridge, Computer Laboratory, May 1995.

17. Barry K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *J. ACM*, 20:160–187, 1973.
18. Amokrane Saïbi. Formalization of a λ-calculus with explicit substitutions in Coq. In P. Dybjer, B. Nordström, and J. Smith, editors, *Types for Proofs and Programs*, volume 996 of *Lect. Notes in Comp. Sci.*, pages 183–202. Springer-Verlag, 1995.
19. N. Shankar. *Metamathematics, Machines, and Gödel's Proof.* Cambridge University Press, 1994.
20. Masako Takahashi. Parallel reductions in λ-calculus. *Information and Computation*, 118:120–127, 1995.

Address for Offprints:
Institut für Informatik
Technische Universität München
80290 München
Germany