

# Mechanising $\lambda$ -calculus using a classical first order theory of terms with permutations

Michael Norrish

© Springer Science + Business Media, LLC 2006

**Abstract** This paper describes the mechanisation in HOL of some basic  $\lambda$ -calculus theory. The proofs are taken from standard sources (books by Hankin and Barendregt), and cover: equational theory, reduction theory, residuals, finiteness of developments, and the standardisation theorem. The issues in mechanising pen-and-paper proofs are discussed; in particular, those difficulties arising from the sources' use of the Barendregt Variable Convention.

## 1. Introduction

If mechanical verification tools are ever to live up to their promise as “proof assistants”, they must fit into the working practices of, and *assist*, the mathematicians who write the proofs. Tools must attempt to adapt to the human's habits as much as possible, not the other way around. Tools must be able to assist with the human's choice of proof strategy (as long as it is sound!). A tool can not tell the human that they must recast their approach in some alien clothes just because that is how the tool prefers to work.

In the absence of live mathematicians willing to undergo the pain of being experimented on by tool-providers, proponents of mechanical techniques can turn to proofs from standard texts. Inasmuch as these reflect standard, human, practice, they also represent a valuable challenge for the mechanical tools. The challenge is

*If my tool was sitting at the mathematician's side while they wrote this text, would they have found the tool even the least bit useful? Would they have been able to use it to check the many inevitable details in the proofs they wrote?*

With the goal of assessing the practicability of this challenge, this paper presents a mechanisation of a substantial, if standard, part of  $\lambda$ -calculus theory. This theory has often been difficult to treat mechanically, mainly due to problems thrown up by the use of binders. In line with the original motivation, the aim was not to have to display any originality or deep

---

M. Norrish (✉)

Canberra Research Laboratory, National ICT Australia, Research School of Information Science and Engineering, Australian National University, Acton 0200, Australia  
e-mail: michael.norrish@nicta.com.au

thought in the transfer of proofs to machine. Having to do so would be a clear indication that the methodology was proving a hindrance; that the tool was not providing much in the way of assistance.

Sadly, but inevitably, a number of proofs did become a great deal larger in their mechanised form. The paper indicates where these increases were caused by difficulties with the method. Elsewhere, such increases were caused by the propensity of authors to brush over tediousness with adjectives such as “obvious”. Section 5.2 discusses the one place where pen-and-paper and machine seemed to diverge significantly.

One area where researchers in this area using pen-and-paper seem to have a significant advantage over mechanisers is in their use of the Barendregt Variable Convention (BVC). Barendregt [2] states the convention thus:

2.1.13. VARIABLE CONVENTION. If  $M_1, \dots, M_n$  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.

After discussing the fundamental choices inherent in representing the types of interest in Section 2, the rest of the paper describes the mechanisation of the source material. The variable convention is used throughout both sources, so discussion of the mechanisation begins with Section 3, which discusses how the variable convention’s “ease-of-proof” can be emulated in a mechanised setting. Section 4 presents the mechanisation of Hankin’s chapters, and Section 5 describes the mechanisation of Barendregt’s proof of the standardisation theorem.

## 2. Foundations

As is often the case, the mechanisation task can be seen as three smaller tasks: definition of new types, definition of functions over those types, and proof. After this section, the rest of the paper concentrates on the last problem. The problem of defining “recursive” functions over  $\lambda$ -calculus terms was addressed in [13], and is briefly discussed below. Definition of relations, such as the various notions of reduction, over terms is straightforward. This section concentrates on the remaining task: the choice and definition of the types needed for  $\lambda$ -calculus theory.

The mechanisation described here works entirely at the level of terms quotiented by  $\alpha$ -equivalence. Un-mechanised presentations tend to argue that this is the level at which they are working (“2.1.12. CONVENTION. Terms that are  $\alpha$ -congruent are identified.”—Barendregt; “From now on, we will assume the variable convention unless otherwise stated.”—Hankin [6, Section 2.3.2]), but mechanisations have often chosen to work with other types as their basis for development. For example, Huet [9] mechanised residual theory using de Bruijn terms, and McKinna and Pollack [11] mechanised standardisation using two-sorted variables, allowing bound and free variables to be syntactically distinguished.

Vestergaard and Brotherston’s work [20] presents an alternative approach to mechanisation. They make a strong case that the (unquotiented) first order abstract syntax should be regarded as primitive. They then carefully present the conditions under which results about confluence will correspond in two related types, show that the unquotiented type and the quotiented type are so related, and then prove confluence at the raw level. Because of the earlier care taken, it follows that this result also holds at the quotiented level. Section 2.1.1 below discusses how the mechanisation described here was similarly shown to be in the correct relation to the unquotiented syntax.

A significant contribution of this paper is to show that it *is* possible, indeed that it is *convenient*, to work at the level of quotiented types. These types remain first-order, include names at the object level, have recursion principles for the definition of functions over the quotiented types, and support inductive reasoning over term-structure and over reduction relations. Further, this ease of use is available within the classical higher-order logic of the HOL system, which includes the Axiom of Choice.

In the source material, there are three different types as objects of study:  $\Lambda$ ,  $\Lambda'$  and  $\Lambda'^*$ . The first is the type of normal  $\lambda$ -calculus terms, the second augments the first with labelled redexes, and the last further adds weights to the terms' variables.

Though the un-mechanised sources implicitly claim that all three of these types are quotients, it is only relatively recently that mechanisations have taken this approach. Ford and Mason [3], and Homeier [7] are examples of two early pioneers (Homeier proves Church-Rosser for  $\beta$ -reduction in HOL; Ford and Mason prove CR for call-by-value  $\beta$ -reduction in PVS). In the absence of support for convenient quotienting in the interactive systems, others have chosen to construct types that are isomorphic to the quotient. For example, Gordon and Melham [4, 5] construct such a type by starting with de Bruijn terms. In more recent work, Urban and Tasson [17] define a type of  $\lambda$ -calculus terms by taking a subset of a model that uses a function space to represent abstractions.

These different approaches represent different authors working with different systems, all aiming at the same destination. Ultimately, their choices of underlying model are not important: having constructed the new type, all of the approaches work exclusively with that new type and show that it has the desired properties. If concerned with adequacy, as discussed by Vestergaard, it is not too difficult to show the necessary properties (see Section 2.1.1).

The rest of this section specifies the required properties of a  $\lambda$ -calculus type, and then describes how the mechanisation achieved this specification. In fact, two separate approaches are described: by quotienting, and by using the Gordon-Melham type. Both approaches were mechanised, and both implemented the specification below, as well as the subsequent proofs.

## 2.1. Specification of $\Lambda$

The specification first requires a type of  $\lambda$ -terms that can be seen as a quotient of the free algebra given by

$$T \simeq S + (T \times T) + (S \times T)$$

where  $S$  is the type of strings.

The type  $\Lambda$  is to have three constructors:

$$\text{VAR} : \text{string} \rightarrow \Lambda$$

$$\text{APP} : \Lambda \rightarrow (\Lambda \rightarrow \Lambda)$$

$$\text{LAM} : \text{string} \rightarrow (\Lambda \rightarrow \Lambda)$$

Each constructor is required to map into distinct subsets of  $\Lambda$ , and all except LAM are required to be injective. For example,  $\forall s \ v \ M. \ \text{VAR } s \neq \text{LAM } v \ M$ , and

$$\forall M_1 \ M_2 \ N_1 \ N_2. (\text{APP } M_1 \ N_1 = \text{APP } M_2 \ N_2) \Leftrightarrow (M_1 = M_2 \wedge N_1 = N_2)$$

This is a first-order picture of  $\lambda$ -calculus terms. The LAM constructor takes as parameters a string and another term, not a meta-level function space, as in higher-order abstract syntax. There will be no problems with exotic terms.

In order to capture the  $\alpha$ -convertibility relationship between terms constructed with LAM, the notions of a term's *free variables* and that of the action of a permutation on a term are required. The free variables function on terms has defining equations:

$$\begin{aligned} \text{FV}(\text{VAR } s) &= \{s\} \\ \text{FV}(\text{APP } M \ N) &= \text{FV}(M) \cup \text{FV}(N) \\ \text{FV}(\text{LAM } v \ M) &= \text{FV}(M) - \{v\} \end{aligned} \tag{1}$$

When  $\pi$  is a finite bijection on strings (alternatively, a bijection that maps all but finitely many strings to themselves), the action of such a permutation on a term  $M$  (written  $\pi \cdot M$ ) is required to satisfy:

$$\begin{aligned} \pi \cdot (\text{VAR } s) &= \text{VAR } (\pi(s)) \\ \pi \cdot (\text{APP } M \ N) &= \text{APP } (\pi \cdot M) \ (\pi \cdot N) \\ \pi \cdot (\text{LAM } v \ M) &= \text{LAM } (\pi(v)) \ (\pi \cdot M) \end{aligned} \tag{2}$$

The  $\alpha$ -equivalence requirement is captured with the following logical equivalence

$$\begin{aligned} (\text{LAM } u \ M = \text{LAM } v \ N) &\Leftrightarrow \\ (u = v \wedge M = N) \vee (u \neq v \wedge u \notin \text{FV}(N) \wedge M = (uv) \cdot N) \end{aligned} \tag{3}$$

where  $(uv) \cdot N$  is the permutation that swaps just  $u$  and  $v$  in term  $N$ .

A structural induction principle (hereafter called the *simple induction principle*) is required:

$$\frac{\begin{aligned} \forall s. P(\text{VAR}(s)) \\ \forall M \ N. P(M) \wedge P(N) \Rightarrow P(\text{APP } M \ N) \\ \forall v \ M. P(M) \Rightarrow P(\text{LAM } v \ M) \end{aligned}}{\forall M. P(M)} \tag{4}$$

A recursion principle is required for the definition of functions with  $\Lambda$  as their domain. Its conclusion is required to be of the form:

$$\begin{aligned} \forall vr \ ap \ lm. \exists f. \\ (\forall s. f(\text{VAR } s) = vr \ s) \wedge \\ (\forall M \ N. f(\text{APP } M \ N) = ap \ (f \ M) \ (f \ N)) \wedge \\ (\forall v \ M. v \notin A \Rightarrow f(\text{LAM } v \ M) = lm \ v \ (f \ M)) \end{aligned} \tag{5}$$

Hypotheses to this theorem may constrain the set of names  $A$ , and the parameter functions ( $vr$ ,  $ap$  and  $lm$ ) in various ways. Nonetheless, the principle must support the definition of substitution,  $\beta$ -normal form,  $\eta$ -normal form and the size of terms. For example, substitution

of  $N$  for variable  $v$  in term  $M$  would be defined by using the following instantiation of (5):

$$\begin{aligned} vr &\mapsto \lambda s. \text{ if } s = v \text{ then } N \text{ else } \text{VAR } s \\ ap &\mapsto \lambda M N. \text{ APP } M N \\ lm &\mapsto \lambda u M. \text{ LAM } u M \\ A &\mapsto \{v\} \cup \text{FV}(N) \end{aligned}$$

This instantiation gives rise to the following equations characterising substitution (with the concrete syntax  $M[v := N]$  representing the substitution of term  $N$  for name  $v$  in term  $M$ ):

$$\begin{aligned} (\text{VAR } v)[v := N] &= N \\ (\text{VAR } u)[v := N] &= \text{VAR } u \quad \text{where } u \neq v \\ (\text{APP } M M')[v := N] &= \text{APP } (M[v := N]) (M'[v := N]) \\ (\text{LAM } u M)[v := N] &= \text{LAM } u (M[v := N]) \quad \text{where } u \neq v, u \notin \text{FV}(N) \end{aligned} \tag{6}$$

### 2.1.1. Adequacy

Vestergaard and Brotherston [20] describe how to define substitution,  $\alpha$ -equivalence ( $\equiv_\alpha$ ) and  $\beta$ -reduction ( $\dashrightarrow_\beta$ ) for an unquotiented type. It is then shown that the relation  $\equiv_\alpha; \dashrightarrow_\beta; \equiv_\alpha$  at the unquotiented level has a corresponding reduction relation  $\rightarrow_\beta$  at the quotiented level. The correspondence is carefully constrained.

*Definition 1.* A map  $\lfloor \_ \rfloor$  from a type  $\tau$  to the raw, unquotiented syntax of  $\lambda$ -terms, and a relation  $\rightarrow_\beta$  on  $\tau$  are *adequate* if

1. The map  $\lfloor \_ \rfloor$  is total and onto
2. The relation  $\rightarrow_\beta$  on  $\tau$  must satisfy

$$\forall M_0 N_0. M_0(\equiv_\alpha; \dashrightarrow_\beta; \equiv_\alpha)N_0 \Leftrightarrow \lfloor M_0 \rfloor \rightarrow_\beta \lfloor N_0 \rfloor$$

where  $M_0$  and  $N_0$  range over terms in the raw syntax, not the type  $\tau$ .

Vestergaard and Brotherston show that if a candidate high-level type  $\tau$ , with operations  $\lfloor \_ \rfloor$  and  $\rightarrow_\beta$  is *adequate*, then confluence results in  $\tau$  exist if and only if they exist in the unquotiented type. This is a powerful sanity check for any mechanisation purporting to be a presentation of the theory of quotiented  $\lambda$ -calculus terms.

**Theorem 1.** *Once  $\rightarrow_\beta$  is defined for  $\Lambda$  (see Section 4.2), the specification of  $\Lambda$  in Section 2.1 is adequate in this sense.*

The proof of this relies heavily on properties of Vestergaard's and Brotherston's unquotiented type, and its notion of substitution and  $\dashrightarrow_\beta$ . The proof's description is omitted, though it forms part of the mechanisation.

## 2.2. Performing the quotient

To define the desired type  $\Lambda$  by quotienting, first establish the existence of the unquotiented, initial type  $\Lambda_0$ , with constructors

$$\begin{aligned} \text{var} &: \text{string} \rightarrow \Lambda_0 \\ \text{app} &: \Lambda_0 \rightarrow (\Lambda_0 \rightarrow \Lambda_0) \\ \text{lam} &: \text{string} \rightarrow (\Lambda_0 \rightarrow \Lambda_0) \end{aligned}$$

Standard theorem-proving tools for defining algebraic types will equip this new type with injectivity and disjointness results, as well as recursion and induction theorems. Using these, it is trivial to define the notion of free variable:

$$\begin{aligned} \text{fv}(\text{var } s) &= \{s\} \\ \text{fv}(\text{app } t \ u) &= \text{fv}(t) \cup \text{fv}(u) \\ \text{fv}(\text{lam } v \ t) &= \text{fv}(t) - \{v\} \end{aligned}$$

Similarly, the definition of `allnames` is straightforward. This function returns the set of all of the strings that appear in a term, whether “bound” by a `lam`-constructor or free. The action of a name-permutation on a term of the  $\Lambda_0$  type is also a standard application of the recursion principle:

$$\begin{aligned} \pi \cdot (\text{var } s) &= \text{var } (\pi(s)) \\ \pi \cdot (\text{app } t \ u) &= \text{app } (\pi \cdot t) \ (\pi \cdot u) \\ \pi \cdot (\text{lam } v \ t) &= \text{lam } (\pi(v)) \ (\pi \cdot t) \end{aligned}$$

Now the  $\equiv_\alpha$  relation can be defined as an inductive relation:

$$\begin{array}{c} \frac{}{\text{var } s \equiv_\alpha \text{var } s} \\ \frac{t \equiv_\alpha t' \quad u \equiv_\alpha u'}{\text{app } t \ u \equiv_\alpha \text{app } t' \ u'} \\ \frac{z \neq v_1, v_2 \quad z \notin \text{allnames}(t_1) \quad z \notin \text{allnames}(t_2) \quad (z v_1) \cdot t_1 \equiv_\alpha (z v_2) \cdot t_2}{\text{lam } v_1 \ t_1 \equiv_\alpha \text{lam } v_2 \ t_2} \end{array}$$

where the expressions  $(z v_i) \cdot t_i$  represent the action of a finite permutation which swaps the strings  $z$  and  $v_i$  in term  $t_i$ . Following Pitts [15], it is not difficult to prove that this relation is *equivariant* (that  $(\pi \cdot t_1 \equiv_\alpha \pi \cdot t_2) \Leftrightarrow (t_1 \equiv_\alpha t_2)$ ), and also an equivalence.

Another paper by Pitts [16] demonstrates how to generate a recursion theorem for  $\Lambda_0$  with a conclusion of the form given in (5), and with the important additional property that the function shown to exist,  $f$ , satisfies the property

$$t_1 \equiv_\alpha t_2 \Rightarrow f(t_1) = f(t_2)$$

Homeier [8] calls this a “respectfulness result”. Each function to be lifted from  $\Lambda_0$  to the quotiented type needs such a result proved of it. Thus, the quotienting process requires

$$\begin{aligned} t_1 \equiv_\alpha t_2 \wedge u_1 \equiv_\alpha u_2 &\Rightarrow (\text{app } t_1 \ u_1 \equiv_\alpha \text{app } t_2 \ u_2) \\ t_1 \equiv_\alpha t_2 &\Rightarrow (\text{lam } v \ t_1 \equiv_\alpha \text{lam } v \ t_2) \end{aligned}$$

to be shown if the constructors `app` and `lam` are to be lifted to become the `APP` and `LAM` of  $\Lambda$ . (The first is immediate given the definition of  $\equiv_\alpha$ ; the second follows given the definition and equivariance of  $\equiv_\alpha$ .) The injectivity and distinctness results for the constructors with respect to  $\equiv_\alpha$  are easy results of the relation's definition.

The respectfulness result for the action of a permutation action is an immediate consequence of the equivariance of  $\equiv_\alpha$ ; the result for the free variable function is

$$t_1 \equiv_\alpha t_2 \Rightarrow \text{fv}(t_1) = \text{fv}(t_2)$$

It is proved by rule induction on the definition of  $\equiv_\alpha$ , and relies on the fact that  $\text{fv}(t) \subseteq \text{allnames}(t)$ .

With the respectfulness theorems proved, the injectivity and disjointness theorems, the recursion theorem, the definitions of  $\pi \cdot \_$ , the free variables function and the induction principle for  $\Lambda_0$ , can all be lifted via the quotienting process (see [8] for details) to give the desired new type  $\Lambda$ , satisfying the specification in Section 2.1.

### 2.3. Using the Gordon-Melham type

Gordon and Melham's five "axioms of  $\alpha$ -conversion" [5] characterise a type of  $\lambda$ -terms which provides a straightforward route to the construction of a type to satisfy Section 2.1's specification for  $\Lambda$ . Gordon and Melham's axioms are theorems about a type (*GM* henceforth) that is isomorphic to an underlying model based on well-formed de Bruijn terms (from Gordon's [4]). There are two important things to note about this construction:

- It does not require the unsupported statement of an axiom. The five axioms are theorems of higher-order logic. Gordon and Melham proved the axioms in HOL, and so they are guaranteed to be sound by virtue of their construction on top of HOL's LCF-style proof kernel. All of the subsequent work was done on top of this purely definitional development, also in HOL.
- The five axioms are a sufficiently complete characterisation of the underlying de Bruijn model that no proof after the establishment of the axioms drops back to the level of the de Bruijn model.

The *GM* type has four constructors: `CON`, `VAR0`, `APP0` and `LAM0`. The extra `CON` constructor is not required in  $\Lambda$ . Fortunately, it is easy to construct a new "CON-free" type from the existing one. First define the predicate `constfree`, such that

$$\begin{aligned} \text{constfree}(\text{VAR}_0 s) &= \top \\ \text{constfree}(\text{CON } k) &= \perp \\ \text{constfree}(\text{APP}_0 M N) &= \text{constfree } M \wedge \text{constfree } N \\ \text{constfree}(\text{LAM}_0 v M) &= \text{constfree } M \end{aligned}$$

Next, use standard type definition technology to create a new type from a non-empty subset of another type. The predicate `constfree` defines such a subset, and the resulting type is in bijection with the subset. In other words, there are two new functions

$$\begin{aligned} \text{to\_term} : GM &\rightarrow \Lambda \\ \text{from\_term} : \Lambda &\rightarrow GM \end{aligned}$$

satisfying the following properties:

$$\begin{aligned} \forall (M : \Lambda). \text{to\_term}(\text{from\_term}(M)) &= M \\ \forall (g : GM). \text{constfree}(g) &\Leftrightarrow (\text{from\_term}(\text{to\_term}(g)) = g) \end{aligned} \tag{7}$$

It is then straightforward to make definitions in the new type  $\Lambda$  to correspond to existing constants in  $GM$ . For example

$$\begin{aligned} \text{VAR } s &= \text{to\_term}(\text{VAR}_0 s) \\ \text{LAM } v t &= \text{to\_term}(\text{LAM}_0 v (\text{from\_term}(t))) \\ \text{FV}(M) &= \text{FV}_0(\text{from\_term}(M)) \\ M[v := N] &= \text{to\_term}((\text{from\_term}(M))[v := \text{from\_term}(N)]) \end{aligned}$$

where the substitution notation on the right of the last definition refers to the substitution already provided for the  $GM$  type.

With definitions such as the above, `constfree` versions of their properties transfer easily to the new type  $\Lambda$ . For example, to prove that

$$\text{FV}(\text{LAM } v M) = \text{FV}(M) - \{v\}$$

rewrite with the definitions of `FV` and `LAM` to obtain

$$\begin{aligned} \text{FV}_0(\text{from\_term}(\text{to\_term}(\text{LAM}_0 v (\text{from\_term}(M)))) &= \\ \text{FV}_0(\text{from\_term}(M)) - \{v\} \end{aligned}$$

Using (7) it follows that the left-hand side of the rewritten form of the equation above is actually equal to

$$\text{FV}_0(\text{LAM}_0 v (\text{from\_term}(M)))$$

from which the desired result follows. These proofs are much more tedious to write out in prose than to mechanise: the HOL simplifier typically proves these results without requiring the human to do anything more than state the desired properties. Proving the required properties of substitution additionally requires the (again, trivial) proof of the fact that the result of a substitution in  $GM$  is `constfree` if the two terms involved are also `constfree`.

The properties desired of  $\Lambda$  have counterparts in  $GM$ , and the properties of  $GM$  are ultimately derived from the Gordon-Melham axioms. These are as follows

*Axiom 1.* A specification of the behaviour of the `FV` (free variables) function. When the clause for `CON` is dropped, it is just as required by the specification (see (1)).

*Axiom 2.* A specification of the substitution function, again as required when lifted to the `constfree` type (see (7)).

*Axiom 3.* A statement of  $\alpha$ -equivalence. When made `constfree`, it has the form

$$v \notin \text{FV}(\text{LAM } u M) \Rightarrow \text{LAM } u M = \text{LAM } v (M[u := \text{VAR } v])$$

The specification's (3) is derived after permutation over terms is defined (see below).  
*Axiom 4.* A unique iteration axiom. When made `constfree` it states:

$$\begin{aligned} &\forall \text{var } \text{app } \text{lam}. \exists ! \text{hom}. \\ &\quad \forall v. \text{hom}(\text{VAR } v) = \text{var}(v) \wedge \\ &\quad \forall M N. \text{hom}(\text{APP } M N) = \text{app}(\text{hom}(M))(\text{hom}(N)) \wedge \\ &\quad \forall v M. \text{hom}(\text{LAM } v M) = \text{lam}(\lambda y. \text{hom}(M[v := \text{VAR } y])) \end{aligned}$$

Note that the  $\lambda$  on the right hand side of the `LAM`-clause is a meta-level  $\lambda$ , creating a HOL abstraction.

Gordon and Melham show how it is possible to derive both an induction and a recursion principle for  $\lambda$ -calculus terms from this axiom. As shown in [13], it is then possible to derive a recursion principle of the form required by Section 2.1 from this theorem. This proof defines the notion of permutation over terms, and also shows the permutation-based notion of  $\alpha$ -equivalence (3).

The *simple induction principle* (4) required by the specification of  $\Lambda$  can be derived from the induction principle accompanying the definition of the predicate `constfree`. Alternatively, it can be derived from the Gordon-Melham induction principle, which states

$$\begin{array}{l} \forall s. P(\text{VAR}_0 s) \\ \forall k. P(\text{CON } k) \\ \forall M N. P(M) \wedge P(N) \Rightarrow P(\text{APP}_0 M N) \\ \forall v M. (\forall y. P(M[v := \text{VAR } y])) \Rightarrow P(\text{LAM}_0 v M) \\ \hline \forall M. P(M) \end{array} \quad (8)$$

When proving that a property holds for all abstractions, this principle allows one to assume that the property holds for all possible renamings of the bound variable in the body of the abstraction. This is stronger than the *simple induction principle*, but is easily reduced to it once the theorem that  $M[v := \text{VAR } v] = M$  is proved. This simple principle, however derived, is easy to lift from *GM* to  $\Lambda$ .

*Axiom 5.* The abstraction axiom (when made `constfree`):

$$\text{LAM } v M = \text{ABS}(\lambda y. M[v := \text{VAR } y])$$

Again, the  $\lambda$  on the right forms a HOL-level abstraction.

This axiom states that all abstraction terms are isomorphic to functions of type *string*  $\rightarrow$  *term* that substitute their arguments (treated as `VARS`) for a given variable in a given term. In this way, one can establish a link between logic-level functions and term-level abstractions. This is reminiscent of higher-order abstract syntax. The `ABS` function is important when defining functions, and is used in the proof (from [13]) that establishes the specification's recursion principle (5).

While the Gordon-Melham type is pragmatically appealing because its construction provides the notion of substitution “for free”, shoe-horning it into the more permutation-oriented specification of Section 2.1 is non-trivial (much of the work is described in [13]). With recent advances in quotienting technology (for example, Homeier [8] and Paulson [14]), and Pitts's contribution [16] to the problem of providing a recursion principles for quotiented types, pragmatic considerations suggest that future work concentrate on quotient constructions.

Another factor in favour of quotienting is that the quotient construction for types other than  $\Lambda$  is equally straightforward. Replaying Gordon’s work [4] with the de Bruijn model for new types is less appealing. It is possible to construct  $\Lambda'$  by using the *GM* type as a foundation, but the predicate that is the analogue of `const.free` is rather more complicated. Deriving a nice recursion theorem for  $\Lambda'$  via this route is also extremely painful. Quotienting provides a much cleaner route to the definition of  $\Lambda'$ , and this route was taken for the version of the mechanisation discussed here.<sup>1</sup>

### 3. Emulating the Barendregt variable convention

There are two standard situations in which the variable convention comes into play during a proof: when performing structural inductions on terms, and when doing rule inductions over inductively defined relations that involve terms. Both sorts of proof occur frequently in the sources. For example, the Substitution Lemma (Lemma 2.11 in Hankin; Lemma 2.1.16 in Barendregt)

$$x \neq y \wedge x \notin \text{FV}(L) \Rightarrow (M[x := N])[y := L] = (M[y := L])[x := N[y := L]]$$

is performed by structural induction.

An example of a rule induction in the sources is the proof that the  $\rightarrow_1$  relation (called “grand reduction” by Hankin, and also known as parallel reduction) satisfies the following property (“substitutivity of  $\rightarrow_1$ ”):

$$M \rightarrow_1 M' \wedge N \rightarrow_1 N' \Rightarrow M[x := N] \rightarrow_1 M'[x := N']$$

This result is Barendregt’s Lemma 3.2.4, and is shown by rule induction on the derivation of  $M \rightarrow_1 M'$ .

#### 3.1. Structural inductions

Urban and Tasson [17] provide a beautiful demonstration that structural inductions can be made to work just as slickly in a mechanised setting as they do in Barendregt’s text. The basis for their proof is a theorem similar to the following result, based on one of Gordon’s results in [4]:

$$\frac{\begin{array}{l} \forall s. P(\text{VAR } s) \\ \forall M N. P(M) \wedge P(N) \Rightarrow P(\text{APP } M N) \\ A \text{ is a finite set of strings} \\ \forall v M. v \notin A \wedge P(M) \Rightarrow P(\text{LAM } v M) \end{array}}{\forall M. P(M)} \tag{9}$$

This induction principle proves the Substitution Lemma by taking  $P$  to be the predicate (over  $M$ ) from the statement of the theorem above, and the set  $A$  to be the set  $\text{FV}(N) \cup \text{FV}(L) \cup \{x, y\}$ . When considering the `LAM` case of the inductive proof, in addition to the inductive hypothesis

$$(M[x := N])[y := L] = (M[y := L])[x := N[y := L]]$$

<sup>1</sup> The *GM* type was used as the basis for the mechanisation of  $\Lambda'$  and standardisation in an earlier version of the mechanisation. This earlier work is described in [12].

and the facts that  $x \neq y$  and  $x \notin \text{FV}(L)$ , one may also assume that  $v \notin \text{FV}(N)$ ,  $\text{FV}(L)$  and that  $v \neq x, y$ . Given all this, it follows that:

$$\begin{aligned} & ((\text{LAM } v \ M)[x := N])[y := L] \\ &= (\text{LAM } v \ (M[x := N]))[y := L] && \text{(def'n of substitution)} \\ &= \text{LAM } v \ ((M[x := N])[y := L]) && \text{(again, def'n of substitution)} \\ &= \text{LAM } v \ ((M[y := L])[x := N[y := L]]) \text{ (IH)} \end{aligned}$$

After showing that

$$v \notin \text{FV}(N) \wedge v \notin \text{FV}(L) \Rightarrow v \notin \text{FV}(N[y := L])$$

(a minor detail that Urban and Tasson point out is absent in Barendregt's proof), one can pull the substitutions back out from underneath the binder, and the proof is complete.

Given the specification of  $\Lambda$  in Section 2.1, what is the proof of the new induction principle (9)?

**Proof:** Assume the hypotheses of (9), and show  $\forall M \ \pi. \ P(\pi \cdot M)$ . Proceed by induction, using the simple induction theorem (4). The  $\text{VAR}$  and  $\text{APP}$  cases follow immediately. In the  $\text{LAM}$  case, the requirement is to show  $P(\pi \cdot (\text{LAM } v \ M))$ . By the definition of permutation (2), this is to show  $P(\text{LAM } (\pi(v)) \ (\pi \cdot M))$ . The relevant assumptions are

$$\begin{aligned} & A \text{ is a finite set of strings} \\ & \forall v \ M. \ v \notin A \wedge P(M) \Rightarrow P(\text{LAM } v \ M) \end{aligned}$$

from (9), with the inductive hypothesis being  $\forall \pi. \ P(\pi \cdot M)$ .

Pick a  $z$  not in  $A \cup \text{FV}(\pi \cdot M)$  and not equal to  $\pi(v)$ . Let  $\pi_0$  be the permutation that swaps  $z$  and  $\pi(v)$ . Then, by the principle of  $\alpha$ -equivalence (3)

$$\text{LAM } (\pi(v)) \ (\pi \cdot M) = \text{LAM } (\pi_0(\pi(v))) \ (\pi_0 \cdot (\pi \cdot M))$$

By the properties of permutations this is equal to

$$\text{LAM } ((\pi_0 \circ \pi)(v)) \ ((\pi_0 \circ \pi) \cdot M)$$

The bound variable  $z$  (now shown to be equal to  $(\pi_0 \circ \pi)(v)$ ) is not in  $A$ , so the implication from (9) can be used. The inductive hypothesis proves  $P((\pi_0 \circ \pi) \cdot M)$ , and the proof is complete.  $\square$

Though the structural induction examples in the mechanisation only require (9), a stronger principle is possible. Rather than fix a particular set  $A$ , the new principle allows the set of names to be avoided to be derived from a function  $f$ , which is applied to an additional parameter of the predicate  $P$ :

$$\begin{aligned} & \forall s \ x. \ P \ (\text{VAR } s) \ x \\ & \forall M \ N \ x. \ (\forall y. \ P \ M \ y) \wedge (\forall z. \ P \ N \ z) \Rightarrow P \ (\text{APP } M \ N) \ x \\ & \forall x. \ \text{finite}(f(x)) \\ & \forall v \ M \ x. \ v \notin f(x) \wedge (\forall y. \ P \ M \ y) \Rightarrow P \ (\text{LAM } v \ M) \ x \\ \hline & \forall M \ x. \ P \ M \ x \end{aligned} \tag{10}$$

The proof of (10) proceeds just does as the proof of (9). The weaker principle can be derived from the stronger by taking  $P$ 's additional parameter to be of the singleton (or “`unit`”) type. Principle (10) is slightly stronger than Lemma 10 in Urban and Tasson [17] because of the nested universal quantifiers in the APP and LAM cases; their principle requires the context to be the same within each case.

Henceforth, those induction principles that allow inconvenient choices of bound variable to be avoided will be referred to as *BVC-compatible*.

(*Aside:* Though these proofs proceed by using permutations, they do also go through if iterated *substitutions* of variables for variables are used instead. The choice of permutations above is simply a result of their being generally easier to reason with.)

### 3.2. Rule inductions

The same “trick” that provides structural principles with more control over the choice of bound variables can be applied to rule induction principles. This construction requires the inductively-defined relation  $R$  to be equivariant (see [15]), that is to satisfy the following

$$R (\pi \cdot x_1) \dots (\pi \cdot x_n) \Leftrightarrow R x_1 \dots x_n$$

where the  $\pi$  may be being applied to parameters other than terms that also support the action of a permutation.

The simplest example in the  $\lambda$ -calculus comes with a definition for  $\beta$ -reduction:

$$\frac{\overline{(\text{APP } (\text{LAM } v M) N) \rightarrow_{\beta} M[v := N]}}{\frac{\frac{M \rightarrow_{\beta} M'}{\text{APP } M N \rightarrow_{\beta} \text{APP } M' N}}{N \rightarrow_{\beta} N'}{\text{APP } M N \rightarrow_{\beta} \text{APP } M N'}}{\frac{M \rightarrow_{\beta} M'}{\text{LAM } v M \rightarrow_{\beta} \text{LAM } v M'}}} \tag{11}$$

The corresponding induction principle states

$$\frac{\forall v M N. P (\text{APP } (\text{LAM } v M) N) (M[v := N]) \quad \forall M M' N. P M M' \Rightarrow P (\text{APP } M N) (\text{APP } M' N) \quad \forall M N N'. P N N' \Rightarrow P (\text{APP } M N) (\text{APP } M N') \quad \forall v M M'. P M M' \Rightarrow P (\text{LAM } v M) (\text{LAM } v M')}{\forall M N. M \rightarrow_{\beta} N \Rightarrow P M N} \tag{12}$$

If one attempts to use (12) to prove that  $\rightarrow_{\beta}$  is *substitutive*, that

$$M \rightarrow_{\beta} M' \Rightarrow M[x := L] \rightarrow_{\beta} M'[x := L]$$

then both the first, “redex”, case and the last, abstraction, case cause difficulties. In the redex case, the goal is to show that

$$(\text{APP } (\text{LAM } v \ M) \ N)[x := L] \rightarrow_{\beta} M[v := N][x := L]$$

One would like to apply the Substitution Lemma, but (12) does not allow one to assume anything about  $v$ ,  $x$  and  $L$ , making it impossible to move the substitution on the left down into the abstraction.

In the abstraction case, the goal is to show that

$$\begin{aligned} (\forall x \ L. \ M[x := L] \rightarrow_{\beta} M'[x := L]) \Rightarrow \\ (\text{LAM } v \ M)[x := L] \rightarrow_{\beta} (\text{LAM } v \ M')[x := L] \end{aligned}$$

and again, lack of knowledge about the relationship between  $v$ ,  $x$  and  $L$  stymies the proof.

The solution to these problems comes by analogy with the solution to structural induction. The analogue of (10) is

$$\begin{array}{l} \forall x. \text{ finite}(f(x)) \\ \forall v \ M \ N \ x. \ v \notin f(x) \cup N \Rightarrow P \ (\text{APP } (\text{LAM } v \ M) \ N) \ (M[v := N]) \ x \\ \forall M \ M' \ N \ x. \ (\forall y. \ P \ M \ M' \ y) \Rightarrow P \ (\text{APP } M \ N) \ (\text{APP } M' \ N) \ x \\ \forall M \ N \ N' \ x. \ (\forall y. \ P \ N \ N' \ y) \Rightarrow P \ (\text{APP } M \ N) \ (\text{APP } M \ N') \ x \\ \forall v \ M \ M' \ x. \ v \notin f(x) \wedge (\forall y. \ P \ M \ M' \ y) \Rightarrow P \ (\text{LAM } v \ M) \ (\text{LAM } v \ M') \ x \\ \hline \forall M \ N \ x. \ M \rightarrow_{\beta} N \Rightarrow P \ M \ N \ x \end{array} \tag{13}$$

To prove the substitutivity result above, it suffices to take  $P$  to be the three-place predicate

$$\lambda M \ N \ (x, L). \ M[x := L] \rightarrow_{\beta} M'[x := L]$$

and the function  $f$  to be  $(\lambda(x, L). \ x \cup \text{FV}(L))$ . With these instantiations, the difficult cases identified above are able to go through as desired because  $v$  is suitably constrained.

The proof of (13) proceeds similarly to the proof of (9), by showing that the hypotheses of (13) imply

$$\forall M \ N \ x \ \pi. \ M \rightarrow_{\beta} N \Rightarrow P \ (\pi \cdot M) \ (\pi \cdot N) \ x$$

and showing this by appeal to the standard induction principle (12). The last, abstraction, case is illustrative:

1. The inductive hypothesis allows the assumption of

$$\forall x \ \pi. \ P \ (\pi \cdot M) \ (\pi \cdot N) \ x$$

2. The new principle allows the assumption of

$$\forall v \ M \ M' \ x. \ v \notin f(x) \wedge (\forall y. \ P \ M \ M' \ y) \Rightarrow P \ (\text{LAM } v \ M) \ (\text{LAM } v \ M') \ x$$

3. The goal to show is

$$P (\text{LAM } (\pi(v)) (\pi \cdot M)) (\text{LAM } (\pi(v)) (\pi \cdot N)) x$$

4. It is possible to choose a  $z$  that satisfies the freshness requirements of the new principle’s hypothesis, and is also free in  $(\pi \cdot M)$  and  $(\pi \cdot N)$ . Let  $\pi_0$  be the permutation that swaps  $z$  and  $\pi(v)$ . Then (by  $\alpha$ -equivalence) the required goal is

$$P (\text{LAM } z ((\pi_0 \circ \pi) \cdot M)) (\text{LAM } z ((\pi_0 \circ \pi) \cdot N)) x$$

5. By appealing to the hypothesis in step 2 above, it suffices to show

$$\forall y. P ((\pi_0 \circ \pi) \cdot M) ((\pi_0 \circ \pi) \cdot N) y$$

This follows immediately from the inductive hypothesis given in step 1.

The proof of (13) implicitly relies on the fact that  $\rightarrow_\beta$  is equivariant. Rather than show  $P M N x$ , the proof shows  $P (\pi \cdot M) (\pi \cdot N) x$ . Because  $\rightarrow_\beta$  is equivariant, then  $P$  will be too. This dependency becomes explicit if one attempts to prove the version of the induction principle that also allows one to assume  $M \rightarrow_\beta N$  alongside  $P M N x$ .

In fact, neither Hankin nor Barendregt show the substitutivity result above with an induction such as the one above. Instead, they define the notion of the *compatible closure* of  $R$ , which repeats the last three rules of the definition of  $\rightarrow_\beta$  (11), and then has

$$\frac{R M N}{M \rightarrow_R N}$$

instead of the  $\beta$ -reduction rule.

Hankin and Barendregt then state the substitutivity result in the form “if  $R$  is substitutive, then so too is  $\rightarrow_R$ .” The  $\beta$  relation, which reduces a redex, is substitutive, so it follows that  $\rightarrow_\beta$  is too.

It should be clear that  $\rightarrow_R$  can only be equivariant if  $R$  is too, and so the stronger induction principle for the compatible closure relation must be

$$\frac{\begin{aligned} &\forall x. \text{finite}(f(x)) \\ &\forall \pi M N. \quad R (\pi \cdot M) (\pi \cdot N) \Leftrightarrow R M N \\ &\forall M N x. \quad R M N \Rightarrow P M N x \\ &\forall M M' N x. (\forall y. P M M' y) \Rightarrow P (\text{APP } M N) (\text{APP } M' N) x \\ &\forall M N N' x. (\forall y. P N N' y) \Rightarrow P (\text{APP } M N) (\text{APP } M N') x \\ &\forall v M M' x. v \notin f(x) \wedge (\forall y. P M M' y) \Rightarrow P (\text{LAM } v M) (\text{LAM } v M') x \end{aligned}}{\forall M N x. M \rightarrow_R N \Rightarrow P M N x} \tag{14}$$

The proof of the general substitutivity result easily follows from the induction result above and the fact that substitutivity implies equivariance. The proof of (14) is very similar to the proof of (13), and is omitted.

The general approach to rule and structural inductions explained here made it possible to mechanise a great many proofs from the pen-and-paper sources very faithfully. The following section describes some other approaches to the same problem.

### 3.3. Other approaches

*Proof using raw syntax.* Vestergaard and Brotherston [19] prove results of both sorts (structural and rule inductions) at the level of raw syntax, where the type of terms is algebraic, and terms are *not* identified up to  $\alpha$ -equivalence. Instead,  $\alpha$ -equivalence is explicitly modelled as a separate equivalence relation, and results are shown for reduction relations that are the composition of some primitive notion ( $\beta$ -reduction where no renamings of bound variables are permitted, for example) and  $\alpha$ -equivalence.

Proofs in this style typically show that the desired property holds of the primitive notion, as long as the starting term is itself in “Barendregt normal form”, where all bound variables have distinct names, and do not overlap with any free variables in the terms under consideration. This normal form is ensured by a sequence of  $\alpha$ -reduction steps. As described above (in Section 2.1.1), this approach is justified by showing (just once) that the operations on the equivalence classes are modelled by the composite relations on the raw syntax. After this, all proofs can be done at the level of the raw syntax.

By working at this level, it is straightforward to use standard theorem-proving technology (induction and recursion principles) to prove the desired results. While the proofs have a rather indirect flavour, and are necessarily set about with explicit considerations of  $\alpha$ -equivalence steps, it is very appealing to be able to assume that a term’s bound variables do indeed conform to the Barendregt Variable Convention.

*Iterated substitutions.* Results such as the substitutivity of  $\rightarrow_\beta$  can be shown by proving the stronger result

$$M \rightarrow_\beta M' \Rightarrow (M \text{ ISUB } R) \rightarrow_\beta (M' \text{ ISUB } R)$$

by rule induction. The ISUB function (written as an infix) is the iterated substitution function: it takes a term and a list of substitutions, and performs each substitution in turn. The proof goes through easily because the inductive hypotheses quantify over  $R$ , meaning that one can assume the property holds for any sequence of substitutions, including those that arise because of forced  $\alpha$ -conversions. There is a corresponding version of the Substitution Lemma that describes the commuting of ISUB terms.

The proof that  $\eta$ -reduction is substitutive can also be done via rule induction with ISUB:

$$M \rightarrow_\eta N \Rightarrow (M \text{ ISUB } R) \rightarrow_\eta (N \text{ ISUB } R)$$

This approach is strong, and fairly direct. It was used in an early version of the mechanisation described in this paper. As already noted, it is also possible to use iterated substitution rather than permutation as the basis for the proofs of the BVC-compatible induction principles above.

*Homeier's BVC-version of size induction.* Homeier [7] approximates the Barendregt Variable Convention in structural inductions by using the following induction principle:

$$\begin{aligned}
 & (\forall s. P(\text{VAR}(s))) \wedge \\
 & (\forall k. P(\text{CON}(k))) \wedge \\
 & (\forall M N. P(M) \wedge P(N) \Rightarrow P(\text{APP } M N)) \wedge \\
 & (\forall v M. (\forall M'. (\text{size}(M') = \text{size}(M)) \Rightarrow P(M')) \Rightarrow P(\text{LAM } v M)) \\
 & \Rightarrow \\
 & \forall t. P(t)
 \end{aligned}$$

This is an easy consequence of the principle of induction on term size but has a pleasant structural appearance, apart from the LAM case. The remaining use of *size* in the LAM case means that awkward  $\alpha$ -conversions can be made to disappear, *as long as one is doing an induction on terms*, rather than, say, a rule induction.

When doing such a proof, one can use a special tactic to bring about the effect of Barendregt Variable Convention. Presented with the need to prove some property  $P$  of  $(\text{LAM } v t)$ , one wants to be able to assume that the bound variable  $v$  is distinct from all others. Homeier's infrastructure first picks such a fresh name,  $z$  say. The original  $(\text{LAM } v t)$  is equal to  $(\text{LAM } z (t[v := \text{VAR}(z)]))$ , and because the body of this term is the same size as  $t$ , the inductive hypothesis is true of it too. Thus, the goal can become  $P(\text{LAM } z t')$ , where  $z$  is known to be suitably fresh, and where the inductive hypothesis holds of  $t'$ . The fact that  $t'$  is itself derived from the original  $t$  by way of a substitution can be forgotten. When applicable, this provides a very appealing approximation to the Barendregt Variable Convention.

Unfortunately, this technique is of no help when the proof is a rule induction. In [7], Homeier has to prove substitutivity of the  $\rightarrow_1$  relation by induction on the term  $M$ , and in this sense his technique does not provide a perfect match for the original source material, where the proof is a rule induction. Nor is this an entirely petty reservation: the elegance of Barendregt's original proof is lost, and one must consider the possible ways in which certain types of term can reduce manually, rather than be led to them directly by the form of the induction principle.

## 4. Mechanising Hankin

Hankin's initial chapters present core results from the theory of the  $\lambda$ -calculus. Initially, the aim of the mechanisation was to work through all of Hankin's book [6], but Hankin's Chapter 3 includes only sketches of proofs for some of the results. For example, Hankin refers the reader to Barendregt for proofs of CR for  $\eta$ - and  $\beta\eta$ -reduction. These proofs are also described in this section. Later, Hankin's reference to Barendregt [2] for the proof of the standardisation theorem was followed, and all of Barendregt's Chapter 11 (except for Section 11.3, which proves the conservation theorem for  $\lambda I$  and is a tangent to the main theorem) was mechanised. This later material is discussed in Section 5.

### 4.1. Chapter 2

Hankin's Chapter 2 introduces the type of  $\lambda$ -terms, discusses substitution, and describes the  $\lambda$ -calculus's basic equational theory, including two treatments of extensionality.

The important result about substitution in this chapter is the Substitution Lemma:

$$x \neq y \wedge x \notin \text{FV}(L) \Rightarrow \\ (M[x := N])[y := L] = (M[y := L])[x := N[y := L]]$$

As discussed above in Section 3, this result is an easy structural induction, once a BVC-compatible principle such as (9) has been proved. This result is proved in 4 lines of proof script.

The equational theory for the  $\lambda$ -calculus that Hankin presents is very easy to mechanise in HOL using an inductive definition. For example, the congruence rule for abstractions

$$\frac{\lambda \vdash M = N}{\lambda \vdash (\lambda x. M) = (\lambda x. N)}$$

was modelled by the following clause of the inductive definition

$$\forall M N x. M \text{ lameq } N \Rightarrow (\text{LAM } x M) \text{ lameq } (\text{LAM } x N)$$

where `lameq` is the HOL constant corresponding to the equational theory's equality. Syntactic equality, usually written  $\equiv$ , is just = in the HOL theory.

When the substitutivity of the `lameq` relation is proved, there is no need for a BVC-compatible form of rule induction. Instead the proof relies on the symmetry and transitivity of `lameq` and is purely equational rather than an induction.

Hankin then introduces two ways of formalising extensionality. One approach adds an explicit extensionality rule:

$$\frac{\lambda + ext \vdash \text{APP } M x = \text{APP } N x \quad x \notin \text{FV}(\text{APP } M N)}{\lambda + ext \vdash M = N}$$

The second approach uses an explicit  $\eta$ -rule:

$$\frac{}{\lambda \eta \vdash \text{LAM } x (\text{APP } M x) = M} \quad x \notin \text{FV}(M)$$

These new systems (the above rules as well as the other rules for the  $\lambda$ -equality) are represented by two new relations in the mechanisation (`lamext` and `lameta`). Two straightforward rule inductions then establish that

$$\text{lamext } M N \Leftrightarrow \text{lameta } M N$$

These proofs do not involve any treatment of substitution, and there is no context that bound variables have to avoid. Because of this, there is no need for BVC-compatible rule induction, and the proofs can use the standard notions of induction that accompany the definitions of `lamext` and `lameta`. The notion of incompatible terms was similarly easy to mechanise.

*List of principal mechanised results.* The substitution lemma; definition of equational theory; equivalence of  $\lambda + ext$  and  $\lambda \eta$ ; notion of incompatibility; incompatibility of  $S$  and  $K$ ; definition of normal forms. (548 lines of proof script.)

### 4.2. Chapter 3

Hankin’s Chapter 3 introduces the basic theory of reduction in the  $\lambda$ -calculus. It proves Church-Rosser for  $\beta$ -reduction (and refers the reader to Barendregt for Church-Rosser of  $\eta$ - and  $\beta\eta$ -reduction). It introduces  $\delta$ -rules, the Hindley-Rosen Lemma, Mitschke’s theorem, residuals, head normal forms, and the standardisation theorem. The mechanisation did not attempt to mechanise Mitschke’s theorem, nor  $\delta$ -rules.

Mechanising the general definitions of reduction, normal form, and compatibility was straightforward. For example, the mechanised version of the compatible closure of a relation  $R$  (written  $\rightarrow_R$  in the sources) is given with the following rules

$$\begin{aligned}
 & (\forall x y. R x y \Rightarrow \text{compat\_closure } R x y) \wedge \\
 & (\forall x y z. \text{compat\_closure } R x y \Rightarrow \\
 & \qquad \qquad \qquad \text{compat\_closure } R (\text{APP } z x) (\text{APP } z y)) \wedge \\
 & (\forall x y z. \text{compat\_closure } R x y \Rightarrow \\
 & \qquad \qquad \qquad \text{compat\_closure } R (\text{APP } x z) (\text{APP } y z)) \wedge \\
 & (\forall x y v. \text{compat\_closure } R x y \Rightarrow \\
 & \qquad \qquad \qquad \text{compat\_closure } R (\text{LAM } v x) (\text{LAM } v y))
 \end{aligned}$$

Section 3 above has already explained how the induction principles associated with reduction relations can be made BVC-compatible. This process applies both to the compatible closure of a relation (as shown in Section 3), and also the  $\rightarrow_1$  relation, which is the other relation defined in this chapter.

All of the inductive proofs from this chapter (and also those such as Church-Rosser for  $\eta$ - and  $\beta\eta$ -reduction which are proved in Barendregt’s Chapter 3) can exploit the various BVC-compatible induction principles, and are faithful transcriptions of the pen-and-paper reasoning. Nonetheless, there are other instances where reasoning with bound variables still presents (relatively minor) obstacles.

#### *Substitutivity of $\beta$*

One such obstacle arises when proving that  $\rightarrow_\beta$  is substitutive. Because of the use of the general notion of compatible closure, which is parameterised by some base reduction relation, it is necessary just to prove

- if  $R$  is substitutive, then so too is  $\rightarrow_R$ ; and
- $\beta$  is substitutive

The first result only needs to be proved once, and is a nice rule induction, using a BVC-compatible principle, as discussed in Section 3.2 above.

The second part of the proof depends on the nature of  $\beta$ . This relation is defined thus

$$\beta M N \Leftrightarrow \exists M_0 P v. \tag{15} \\
 M = (\text{APP } (\text{LAM } v M_0) P) \wedge N = M_0[v := P]$$

Without being able to make typical informal use of Barendregt Variable Convention, this definition is a slightly awkward base on which to prove that  $\beta$  is substitutive. By the definition of substitutivity, this requires a proof of

$$\beta M N \Rightarrow \beta (M[x := L]) (N[x := L])$$

The naïve approach of simply expanding the definition of  $\beta$  above results in a requirement to prove

$$\begin{aligned} \exists M'_0 v'. (\text{LAM } v M_0)[x := L] &= \text{LAM } v' M'_0 \wedge \\ (M_0[v := P])[x := L] &= M'_0[v' := P[x := L]] \end{aligned}$$

Without knowing anything about the variable  $v$ 's relationship to  $x$  and the free variables of  $L$ , the best approach is to choose a suitably fresh  $z$  such that  $(\text{LAM } v M_0)$  is equal to  $(\text{LAM } z (M_0[v := \text{VAR } z]))$  by  $\alpha$ -equivalence. Then take  $M'_0$  to be  $M_0[v := \text{VAR } z][x := L]$ , and  $v'$  to be  $z$ . One must then show

$$\begin{aligned} (\text{LAM } z (M_0[v := \text{VAR } z]))[x := L] &= \text{LAM } z (M_0[v := \text{VAR } z][x := L]) \\ &\wedge \\ M_0[v := P][x := L] &= M_0[v := \text{VAR } z][x := L][z := P[x := L]] \end{aligned}$$

The first equation follows because the substitution on the left-hand side can move underneath the binder ( $z$  is fresh with respect to  $x$  and  $L$  by construction). On the right-hand side of the second equation there is an instance of the Substitution Lemma, which makes the second equation equivalent to

$$M_0[v := P][x := L] = M_0[v := \text{VAR } z][z := P][x := L]$$

The inner pair of substitutions on the right then collapse to give the desired identity because  $z \notin \text{FV}(M_0)$ , again by choice of  $z$ .

This awkwardness can be averted by first proving the following characterisation of  $\beta$ , to be used instead of (15):

$$\begin{aligned} \beta M N \Leftrightarrow \exists M_0 P v. \\ M = (\text{APP } (\text{LAM } v M_0) P) \wedge N = M_0[v := P] \wedge v \notin X \end{aligned} \quad (16)$$

where  $X$  is a finite set of names. Using this result about  $\beta$ , the substitutivity proof doesn't need to involve an  $\alpha$ -renaming. Instead, the new characterisation can be used, with  $X$  taken to be the set  $\{x\} \cup \text{FV}(L)$ . Naturally, the proof of (16) involves an explicit renaming step.

This is thus a demonstration that a little thought can bring about BVC-compatible characterisations of definitions that are not inductive in flavour. Just as the definition of  $\beta$  can be modified to provide a characterisation where the bound variable avoids a wider set, so too can the definition of  $\eta$ , making its substitutivity result a little easier.

*List of principal mechanised results.* Substitutivity of reduction relations; Church-Rosser for  $\beta$ -,  $\eta$ - and  $\beta\eta$ -reduction; Hindley-Rosen Lemma. (1348 lines of proof script.)

## 5. Barendregt—Chapter 11

Just as Barendregt's Chapter 3 was a necessary resource for proofs about  $\eta$ - and  $\beta\eta$ -reduction, Hankin's reference to Barendregt's Chapter 11 was followed for the proof of the standardisation theorem. Barendregt's proof is via the theory of residuals and a proof of the finiteness of developments.

### 5.1. $\lambda$ -terms with labelled redexes

The first significant task in mechanising this material was the establishment of a new type  $\Lambda'$ , corresponding to  $\lambda$ -terms where some redexes are labelled with numbers. The new type has the three constructors (VAR, APP, and LAM) corresponding to those in  $\Lambda$ , and a new constructor for labelled redexes. In the source material, this constructor is written  $(\lambda_i x.M)N$ , where  $i$  is the redex’s label. In the mechanisation, the new terms are of the general form  $\text{LAM}i\ i\ x\ M\ N$ . The new  $\text{LAM}i$  constructor supports its own notion of  $\alpha$ -convertibility ( $\text{LAM}i\ i\ x\ M\ N \equiv_\alpha \text{LAM}i\ i\ y\ (M[x := y])\ N$ , where  $y \notin \text{FV}(M)$ ), and substitution.

The reduction relation on  $\Lambda'$  corresponding to  $\beta$  is written  $\beta'$ , and can be viewed as the union of  $\beta_0$ , which reduces labelled redexes, and  $\beta_1$ , which reduces unlabelled redexes. Specifically,  $\beta_0\ M\ N$  is true if  $M$  is of the form  $(\text{LAM}i\ i\ v\ M_0\ N_0)$  and  $N$  equal to  $M_0[x := N_0]$ . Also,  $\beta_1\ M\ N$  is true if  $M$  is of the form  $(\text{APP}\ (\text{LAM}\ v\ M_0)\ N_0)$  and  $N$  equal to  $M_0[x := N_0]$ . The compatible closure of reduction relations for  $\Lambda'$  (by analogy with that for  $\Lambda$  in Section 4.2 above) is an easy inductive definition, giving  $\rightarrow_{\beta'}, \rightarrow_{\beta_0}$  etc.

The cleanest way of constructing this type is by quotienting an underlying, “raw” type of first-order syntax with respect to the notion of  $\alpha$ -equivalence. In this way,  $\Lambda'$  can be equipped with a permutation action, as well as a principle justifying the definition of recursive equations.

One such equation over  $\Lambda'$  is the  $\varphi$  function which reduces all of a labelled term’s labelled redexes. Its definition is

$$\begin{aligned} \varphi (\text{VAR } s) &= \text{VAR } s \\ \varphi (\text{APP } M\ N) &= \text{APP } (\varphi\ M)\ (\varphi\ N) \\ \varphi (\text{LAM } v\ M) &= \text{LAM } v\ (\varphi\ M) \\ \varphi (\text{LAM}i\ i\ v\ M\ N) &= (\varphi\ M)[v := \varphi\ N] \end{aligned} \tag{17}$$

where the constructors (VAR, APP etc.) on the left are for the type  $\Lambda'$ , and those on the right are for the type  $\Lambda$ .

The definition of  $\varphi$  proceeded easily from the recursion theorem proved for the  $\Lambda'$  type. It was also necessary to prove that every  $M' \in \Lambda'$  could have its labels removed and thus be mapped into a corresponding term in  $\Lambda$ . For example, the  $\text{LAM}i$  clause for this function is

$$\text{strip\_label } (\text{LAM}i\ i\ v\ M\ N) = \text{APP } (\text{LAM } v\ M)\ N$$

Finally, another function was defined, for the reverse direction, to label a term in  $\Lambda$  at a given set of redex positions, producing a term in  $\Lambda'$ .

Barendregt Section 11.1 finishes with a proof of Church-Rosser for  $\beta$ , using the new machinery of labelled terms, the  $\varphi$  function, and a variety of commuting diagrams. The diagrams and the proofs were easy to mechanise. An example is Lemma 11.1.6(i), which states that

$$|M'| \rightarrow_\beta N \Rightarrow \exists N'. M' \rightarrow_{\beta'} N' \wedge N = |N'|$$

where  $|M'|$  is the human-friendly notation for an application of the `strip_label` function to  $M'$ . The proof proceeds by transitivity, and from a simpler result which has the transition from  $|M'|$  to  $N$  be just one step. The simpler result proceeds by a rule-induction on  $\rightarrow_\beta$ , and an analysis of the possible shapes of  $M'$  given that  $|M'|$  is known to have a particular shape in  $\Lambda$ . Mechanising these results did not require the use of BVC-compatible induction principles, and was straightforward.

On the other hand, Lemma 11.1.7(i) states that

$$\varphi (M[x := N]) = (\varphi M)[x := \varphi N]$$

and is proved by structural induction on  $M$ . This requires the use of the BVC-compatible structural induction principle for  $\Lambda'$ , the substitution lemma, the definition of  $\varphi$  (17), and the lemma stating that  $\text{FV}(\varphi(M)) \subseteq \text{FV}(M)$ . The proof is a call to the induction principle, avoiding the context  $\{x\} \cup \text{FV}(N)$ , followed by simplification, taking just three lines.

## 5.2. Labelling reductions and residuals

Barendregt Section 11.2 introduces the notion of *residuals*. Given a finite reduction path in  $\Lambda$ , one determines the residuals of a set of redex positions with respect to that path by labelling the first term of the path with the set, performing the analogous reduction sequence in  $\Lambda'$ , and returning the set of redex positions that are still labelled in the last term of the  $\Lambda'$ -path.

A reduction path is defined not only by the sequence of terms in the path, but also by the labels on the arrows, which identify which redex was contracted to move from one term to the next. Thus

$$M \xrightarrow{\Delta} N$$

means that  $M$  can reduce at redex  $\Delta$  and become  $N$ . This notation is defined by Barendregt's Definition 3.1.17, where the  $\Delta$  is a sub-term of  $M$ . Ignoring the fact that this notation does not distinguish between the two ways of performing

$$\Omega\Omega \xrightarrow{\Omega} \Omega\Omega$$

it also sits very poorly with terms being identified up to  $\alpha$ -equivalence. Consider the reduction

$$(\lambda x. (\lambda y. yx)z) \rightarrow (\lambda x. zx)$$

It seems fair to claim that the redex is  $(\lambda y. yx)z$ , but in writing this, one has made a choice of variable name for the previously bound  $x$ . Because  $(\lambda x. (\lambda y. yx)z) = (\lambda u. (\lambda y. yu)z)$ , the label for this reduction might just as well be  $(\lambda y. yu)z$ . Thus, one reduction can be labelled by infinitely many different terms, and there is an equivalence relation between all of these labels, allowing the renaming of free variables that were bound in the original term (but not those, such as  $z$ , that were free there).

So, an ugly equivalence relation has emerged, despite a motivation of this work being to avoid these, and to work with true equalities wherever possible. One approach to removing this equivalence would be to model labels as terms wrapped in as many additional abstractions as there are variables that can be renamed in the label.<sup>2</sup> If one insisted that the sequence of binders in the label reflected the sequence of binders in the original, then this would be a unique representation for the label.

<sup>2</sup> One could accompany such terms with a number to indicate how many abstractions to strip to get the reduction label, or simply observe that, for  $\beta$ -reduction anyway, reduction labels are always applications, so that all outermost abstractions should be dropped.

One might also use (one-holed) contexts as labels, and avoid the problem with the ambiguous  $\Omega\Omega$  reduction above. Unfortunately, this still leaves open the problem of identifying all of the contexts that are equivalent up to renaming of the free variables that they capture. Moreover, there is no obvious encoding of contexts in simple terms.

In the event, the course chosen was to represent reduction labels (redex positions) as elements of  $\{\text{In}, \text{Lt}, \text{Rt}\}^*$ . That is, reduction labels are sequences of directions (“in”, “left” and “right”) for traversing the syntax tree of a term. This is a simple representation, works for both  $\Lambda$  and  $\Lambda'$  without alteration, and allows for the simple definition of  $<$  on redex positions. Further, there is an injection from reduction to label, which as the  $\Omega\Omega$  case makes clear, is not the case with redexes as labels.

Though this is superficially a significant difference with Barendregt’s presentation, it didn’t produce significant differences in the proofs. Barendregt only requires that his  $\Delta$ s act as positions, or pointers into the structure of terms, and this is just how the “tree-path” representation behaves. Another advantage of this representation is that a reduction’s location can be determined by examination of just the original term and the position. One need not examine the resulting term as well.

After residuals, Barendregt defines *developments*. These are reduction paths that only reduce a subset of those redexes that were present at the start of a path; equivalently, these are  $\beta_0$ -reduction paths. Most of Chapter 11 is concerned with proving that developments are necessarily finite; equivalently, that  $\beta_0$  is strongly normalising.

### 5.3. Finiteness of developments

In the second half of Section 11.2, Barendregt proves that  $\beta_0$  is strongly-normalising by demonstrating that  $\Lambda'$  terms can be given a weighting (a number in  $\mathbb{N}^+$ ) such that  $\beta_0$ -reduction decreases the weighting. The new type of weighted terms is called  $\Lambda'^*$ . A weighted term can be seen as either a pair of a labelled term and a map that gives that term’s variables (free *and* bound) a weight, or a new quotiented type where the VAR constructor takes an additional numeric argument. Barendregt seems to prefer the former view for most of his proofs, so this approach was also taken in the mechanisation. The weight of a term is the sum of the weights of the term’s variables.

Different instances of the same variable (again, whether free or bound) can be assigned different weights, so it is clear that the weightings (which Barendregt describes as being from variable occurrences to weights) are maps from *positions* to weights. The positions used to identify redexes (described in the previous section) can be used as the domains of these maps.

A variety of extra functions on terms is necessary to support this usage. For example, the `v_posn` function takes a variable name and a term, and returns the set of positions where that variable is free in the term. Its defining equations are

$$\begin{aligned} \text{v\_posn } v \text{ (VAR } s) &= \text{if } s = v \text{ then } \{\} \text{ else } \emptyset \\ \text{v\_posn } v \text{ (APP } M \ N) &= \text{consLt}(\text{v\_posn } v \ M) \cup \text{consRt}(\text{v\_posn } v \ N) \\ \text{v\_posn } v \text{ (LAM } u \ M) &= \text{consIn}(\text{v\_posn } v \ M) \quad [\text{where } u \neq v] \end{aligned}$$

where the various `cons $x$`  functions take images of sets of positions under the operation of prepending the element  $x$  to each path in the input sets. The definition of `v_posn` is an easy application of the recursion principle for  $\Lambda$ .

It should be clear that positions can be used to identify any sort of position within a term, whether they be of redexes or variables. Maps from positions to weights are thus a suitable representation for Barendregt’s map from variable occurrences to weights. The value of the

mapping at non-variable positions is ignored, and the weight of a whole term is the sum of its variables' weights.

The proof proceeds by demonstrating that it is always possible to weight a term in a way described as *decreasing*. The mechanisation's definition of decreasing makes more use of positions:

$$\begin{aligned} \text{decreasing}(M, w) = & \\ & \forall p_1 p_2. \\ & p_1 \in \text{lredex}(M) \wedge p_2 \in \text{bv\_posns\_at}(p_1 \frown \langle \text{Lt} \rangle, |M|) \Rightarrow \\ & \text{weight\_at } (p_1 \frown \langle \text{Rt} \rangle) (M, w) < \text{weight\_at } p_2 (M, w) \end{aligned}$$

where

- $\text{lredex}(M)$  returns the set of positions of labelled redexes in  $M$ ,
- $\text{bv\_posns\_at}(p, M)$  returns the set of positions of bound variables under an abstraction assumed to be at position  $p$  in  $M$ , and
- $\text{weight\_at } p (M, w)$  returns the weight of the sub-term at position  $p$  within weighted term  $(M, w)$ .

This captures Barendregt's

The weighting  $I$  is called *decreasing* if for every  $\beta_0^*$  redex  $(\lambda_i x. P)Q$  in  $M$  one has  $\|x\|' > \|Q\|'$  for all occurrences of  $x$  in  $P$ .

where  $\|M\|'$  calculates the total weight of a term  $M$ .

Finally it is shown that if such a weighted term undergoes a  $\beta_0$ -reduction, the resulting term again has a decreasing weighting, and that its total weight has decreased.

The HOL proofs of this property are the longest of the entire mechanisation, and extremely fiddly. The main result, that reduction does reduce the ordering, took 425 lines. An accompanying result, which says that if there is a variable position with a given weight in  $N$ , and  $M \rightarrow N$ , then there is a variable position in  $M$  with the same weight, took 100 lines to prove. Finally, the result characterising how a weighting map has to change to reflect a substitution took 250 lines to prove. It is no surprise that Barendregt's corresponding (partial) pen-and-paper proof is the least convincing of all of those mechanised, simply because the detail on the page becomes overwhelming and one's intuition evaporates. Nor does Barendregt anywhere attempt a formal description of how variable weightings update after a substitution in  $\Lambda^*$ .

It is worth pointing out, however, that this detail and pain have very little to do with the mechanised approach to  $\alpha$ -equivalent terms. In fact, there was just one use of the theorem specifying  $\alpha$ -conversion in the three proofs mentioned above (that being in the substitution characterisation result). Instead, the main source of complexity was reasoning about relative positions within terms, making use of facts such as "if position  $p_1$  is a proper prefix of  $p_2$ , and both are valid positions inside term  $M$ , then  $p_1$  can not be the position of a variable in  $M$ ".

Once Barendregt establishes finiteness of developments, Church-Rosser for  $\beta_0$  follows from Newman's Lemma, and another proof of Church-Rosser for  $\beta$  follows.

#### 5.4. The standardisation theorem

Barendregt Section 11.4 defines the notions of standard and internal reductions, proves a variety of commuting results for internal and head reductions, culminating in the theorem (the "main lemma", 11.4.6) that any reduction can be factored into a sequence of head

reductions followed by internal reductions. He then proves standardisation. There is also a final corollary proved, which states that a term has a finite head reduction path if and only if it has a head normal form.

The section begins by defining what it is to be a *standard reduction*. The notions of head and internal redex are then needed. The notion of a position being a head-redex for a term is a straightforward inductive definition:

$$\frac{\frac{\frac{\text{() is\_hredex\_of APP (LAM } v M) N}{p \text{ is\_hredex\_of } M}}{\text{In} :: p \text{ is\_hredex\_of LAM } v M}}{p \text{ is\_hredex\_of APP } M N}}{\text{lt} :: p \text{ is\_hredex\_of APP (APP } M N) P}$$

The dual notion of a position being an internal redex can then be defined directly, as can the head and internal reduction relations ( $\rightarrow_h$  and  $\rightarrow_i$  respectively). With these notions in place, Barendregt develops a series of lemmas leading to his final result.

The mechanisation of the bulk of this last section proceeded easily, if verbosely. A typical lemma is Barendregt’s 11.4.3(iii), which states that “if  $\Delta_i$  is an internal redex of  $M$ , then all elements of  $\Delta_i/\sigma$  are internal redexes of  $N$ ”, in a setting where  $\sigma$  is an internal reduction from  $M$  to  $N$ , and where the notation  $\Delta_i/\sigma$  denotes the set of residual positions in  $N$  that are descended from the redex position  $\Delta_i$ . Barendregt’s proof of this lemma is the phrase “Equally simple” referring back to his three line proof of 11.4.3(ii). The proof in the mechanisation is 48 lines long, starting with a case-split on the three possible relations ( $<$ ,  $=$ ,  $>$ ) between  $\Delta$  and  $\Delta_i$ .

By this stage of the mechanisation, the proofs rarely need to concern themselves with bound names. Indeed, the proof of this lemma does not directly examine the structure of terms  $M$  and  $N$  at all. Instead, the reasoning happens entirely by reference to earlier results, and the nature of positions within terms.

The only interest specific to the treatment of  $\alpha$ -equivalent terms in this section arises in the final proof of the standardisation theorem. The “main lemma” states

$$M \twoheadrightarrow N \Rightarrow \exists M'. M \twoheadrightarrow_h M' \wedge M' \twoheadrightarrow_i N$$

To prove standardisation, Barendregt takes an arbitrary reduction  $M \twoheadrightarrow N$ , infers the existence of a  $Z$  such that  $M \twoheadrightarrow_h Z \twoheadrightarrow_i N$ , and then discusses the necessary form of  $Z$ . If  $N$  is a variable, then so too is  $Z$ , as no internal reduction can produce a single variable. Otherwise  $N$  is of the form  $(\lambda x_1 \dots x_n. N_0 N_1 \dots N_m)$  with  $n + m > 0$ . In this case, Barendregt concludes that  $Z$  is of the form

$$(\lambda x_1 \dots x_n. Z_0 Z_1 \dots Z_m)$$

with  $Z_i \twoheadrightarrow N_i$ , for  $0 \leq i \leq m$ .

This step is making use of the Barendregt Variable Convention, assuming that the  $\vec{x}$  from  $N$  are disjoint with the free variables of  $Z$ , which has been called into existence through the application of the “main lemma”. This couldn’t be replayed directly in the mechanisation. Instead, it was necessary to prove a result stating how sequences of binders could be  $\alpha$ -converted all at once. The statement of this result is

$$\begin{aligned} & \forall vs \ vs' \ M. \\ & \quad (\text{LENGTH } vs = \text{LENGTH } vs') \wedge \text{ALL\_DISTINCT } vs' \wedge \\ & \quad \text{DISJOINT } (\text{LIST\_TO\_SET } vs') \\ & \quad \quad (\text{LIST\_TO\_SET } vs \text{ UNION FV } M) \Rightarrow \\ & \quad (\text{LAM1 } vs \ M = \\ & \quad \text{LAM1 } vs' \\ & \quad \quad (M \text{ ISUB REVERSE } (\text{ZIP}(\text{MAP VAR } vs', vs)))) \end{aligned}$$

where  $\text{LAM1 } v \ M$  stands for  $(\lambda \vec{v}. M)$ , and the predicate  $\text{ALLDISTINCT}$  is true of a list if it contains no duplicates. The logical variables  $vs$  and  $vs'$  above are both lists of strings, so that the expression  $\text{ZIP}(\text{MAP VAR } vs', vs)$  constructs a list of substitutions: each pair consists of a term and the name of the variable for which it is to be substituted. The infix  $\text{ISUB}$  function computes an iterated substitution.

Appealing to this principle of “vector  $\alpha$ -equivalence”, Barendregt’s proof can be altered by  $\alpha$ -converting both  $Z$  and  $N$ . It was also necessary to prove that

$$\begin{aligned} & \forall vs \ M \ N. \\ & \quad \text{DISJOINT } (\text{LIST\_TO\_SET } vs) \ (\text{FV } M) \wedge \\ & \quad \text{ALL\_DISTINCT } vs \Rightarrow \\ & \quad (M \rightarrow_i \text{LAM1 } vs \ N \Leftrightarrow \\ & \quad \quad \exists M0. (M = \text{LAM1 } vs \ M0) \wedge M0 \rightarrow_i N) \end{aligned}$$

where the theorem repeats part of the precondition of the earlier  $\alpha$ -equivalence result.

Finally, it became clear that the proof of Barendregt’s final corollary (that a term has a head normal form if and only if its head reduction path is finite) could be altered so that it didn’t depend on the standardisation proof at all. It is a corollary of the “main lemma” instead. The implication from left to right is the interesting case. Say  $M = Z$ , with  $Z$  in head normal form. This equality is that of the equational theory, not syntactic equality. Then  $M$  and  $Z$  can both reduce to the same  $Z'$  (by Church-Rosser). By Barendregt’s lemma 11.4.3(i),  $Z'$  is in head normal form, as it is reachable from  $Z$ . Then the “main lemma” tells us there is a  $Z''$ , which is a mid-point between  $M$  and  $Z'$ , with  $M \rightarrow_h Z''$  and  $Z'' \rightarrow_i Z'$ . As  $Z'$  is in head normal form, so too is  $Z''$ , by lemma 11.4.3(ii). Then the path from  $M$  to  $Z''$  is  $M$ ’s head reduction path, and is finite, as required.

*List of principal mechanised results.* Two more proofs of Church-Rosser for  $\beta$ ; introduction of terms with labelled redexes ( $\Lambda'$ ); introduction of residuals; finiteness of developments (FD); uniqueness of complete developments (FD!); standardisation theorem. (11.1: 608 lines; Section 11.2: 4166 lines; 11.4: 2393 lines of proof script.)

## 6. Related work

There are many approaches to mechanising binders and the problems of  $\alpha$ -equivalent terms in the literature. This section describes those that have attacked similar problems (the theory of the  $\lambda$ -calculus), and another that, while not having mechanised the  $\lambda$ -calculus material, uses an underlying model based on Gordon’s work [4].

Mechanisations of  $\lambda$ -calculus theory using name-carrying syntax include Ford and Mason [3], Homeier [7], McKinna and Pollack [11], Urban and Tasson [17], and Vestergaard and Brotherston [20]. The last is the only previous treatment of residual theory using names.

(Earlier work by Huet [9] used de Bruijn indices.) The work presented here is the first to mechanise the finiteness of developments, in any style.<sup>3</sup>

The work of Ambler et al. [1] is based on Gordon [4], but uses that work to implement a higher-order abstract syntax machinery. They show that their approach is well-suited to mechanising a number of different areas of computer science mathematics, including some properties of a lazy  $\lambda$ -calculus. This work's use of its HOAS machinery gives it a different feel to the mechanisation presented here, but it can be seen as further evidence that the basic Gordon-Melham model (de Bruijn terms) can be used to model more user-friendly types in a classical logic.

On the basis of theorems proved, the mechanisation of  $\lambda$ -calculus theory presented here has achieved as much as previous mechanisations by others, but it has also done so in a manner that tries to be as faithful as possible to the original authors' approaches and intentions.

## 7. Conclusion

A number of  $\lambda$ -calculus proofs from the standard literature have been successfully mechanised in the HOL system. At least one of these theorems, the finiteness of developments, has been mechanised for the first time. The mechanisation describes two different approaches to the construction of the type of  $\lambda$ -calculus terms ( $\Lambda$ ). One approach features the use of the Gordon-Melham axioms for  $\alpha$ -equivalent terms; the other demonstrates that it is possible to perform quotients in just the way claimed by pen-and-paper authors.

The mechanisation also draws on the idea of permutation and nominal logic due to Gabbay and Pitts (see [15], for example), and shows that these ideas can be fruitfully applied in a classical setting where the Axiom of Choice is assumed. Taking the permutation action from Fraenkel-Mostowski set theory (where the Axiom of Choice leads to contradiction) into the logic of the HOL system results in a powerful tool for working with  $\alpha$ -equivalent terms. There is no need for users of systems assuming the Axiom of Choice to abjure permutations, nor their elegant theory. The presentation in this paper has deliberately included the ideas from nominal logic in as light a manner as possible, but they have informed and influenced the development extensively.

Nominal logic's heaviest use comes in the proof of the various BVC-compatible induction theorems, where it is easier to work with permutations than iterated substitutions, and in the proof of recursion theorems for quotiented types (from Pitts [16]). Both of these activities hold out the promise of complete automation, suggesting that future users of mechanised tools in this setting will be able to remain oblivious to the use of nominal logic.

This work provides an extensive test for one approach to the long-standing problem of mechanised reasoning about languages with binders. It demonstrates that it is possible to work with  $\alpha$ -equivalent quotients directly: one can prove results about them, and mimic standard presentations in doing so. On top of this foundation, one can also define new types with their own binders, and reason about these in turn.

**Availability.** All of the work described in this paper is distributed as part of the HOL4 system, and is also available for download from the CVS repository reachable from `hol.sourceforge.net`.

<sup>3</sup> Vestergaard [18] presents pen-and-paper proofs of the standardisation theorem in the same style as the mechanised proofs in [20]. McKinna and Pollack [11] also prove standardisation, without using residuals or finiteness of developments.

**Acknowledgments** I would like to thank Andy Pitts for many useful discussions about recursion, Christian Urban for many useful discussions about BVC-compatible induction principles, and René Vestergaard, both for discussions about standardisation and adequacy, and for the excellent working conditions at his recent “Binding Challenges” workshop. I would also like to thank the anonymous referees for their helpful comments on paper structure and presentation. National ICT Australia is funded by the Australian Government’s *Backing Australia’s Ability* initiative, in part through the Australian Research Council.

## References

1. Ambler, S.J., Crole, R.L., Momigliano, A.: Combining higher order abstract syntax with tactical theorem proving and (co)induction. In: Carreño, V.A., Muñoz, C.A., Tahar, S. (eds.), *Theorem Proving in Higher Order Logics*, 15th International Conference, vol. 2410 of *Lecture Notes in Computer Science*, pp. 13–30 (2002)
2. Barendregt, H.P.: *The lambda calculus: its syntax and semantics*, vol. 103 of *Studies in logic and the foundations of mathematics*. Elsevier, Amsterdam: revised edition (1984)
3. Ford, J.M., Mason, I.A.: Operational techniques in PVS: A preliminary evaluation. In: Fidge, C. (ed.), *Computing: The Australasian Theory Symposium (CATS 2001)*, vol. 42 of *Electronic Notes in Theoretical Computer Science* (2001)
4. Gordon, A.D.: A mechanisation of name-carrying syntax up to alpha-conversion. In: Joyce, J.J., Seger, C.J.H. (eds.), *In: Proceedings of Higher-Order Logic Theorem Proving and its Applications (HUG’93)*, vol. 780 of *Lecture Notes in Computer Science*, pp. 413–425 (1994)
5. Gordon, A.D., Melham, T.: Five axioms of alpha conversion. In: von Wright, J., Grundy, J., Harrison, J. (eds.), *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLS’96*, vol. 1125 of *Lecture Notes in Computer Science*, pp. 173–190 (1996)
6. Hankin, C.: *Lambda Calculi: A Guide for Computer Scientists*, vol. 3 of *Graduate Texts in Computer Science*, Clarendon Press, Oxford (1994)
7. Homeier, P.: A proof of the Church-Rosser theorem for the lambda calculus in higher order logic. In: Boulton, R.J., Jackson, P.B. (eds.), *TPHOLS 2001: Supplemental Proceedings*. Available as *Informatics Research Report EDI-INF-RR-0046* pp. 207–222 (2001)
8. Homeier, P.: A design structure for higher order quotients in [10], pp. 130–146 (2005)
9. Huet, G.P.: Residual theory in lambda-calculus: A Formal Development. *Journal of Functional Programming* **4**(3), 371–394 (1994)
10. Hurd, J., Melham, T. (eds.): *Theorem Proving in Higher Order Logics 18th International Conference*, *Lecture Notes in Computer Science*, Springer, vol. 3603 (2005)
11. McKinna, J., Pollack, R.: Some lambda calculus and type theory formalized. *Journal of Automated Reasoning* **23**(3–4), 373–409 (1999)
12. Norrish, M.: Mechanising Hankin and Barendregt Using the Gordon-Melham Axioms. In: Honsell, F., Miculan, M., Momigliano, A. (eds.), *Merlin 2003, Proceedings of the Second ACM SIGPLAN Workshop on Mechanized Reasoning about Languages with Variable Binding*. Available at <http://doi.acm.org/10.1145/976571.976577> (2003)
13. Norrish, M.: Recursive function definition for types with binders. In: Slind, K., Bunker, A., Gopalakrishnan, G. (eds.) *Theorem Proving in Higher Order Logics*, 17th International Conference, vol. **3223** of *LNCS*. pp. 241–256 (2004)
14. Paulson, L.: Defining functions on equivalence classes. *ACM Transactions on Computational Logic*. To appear.
15. Pitts, A.M.: Nominal Logic, A first order theory of names and binding. *Information and Computation* **186**, 165–193 (2003)
16. Pitts, A.M.: Alpha-Structural Recursion and Induction (Extended Abstract) in [10], pp. 17–34 (2005)
17. Urban, C., Tasson, C.: Nominal techniques in Isabelle/HOL. In: Nieuwenhuis, R. (ed.), *Proceedings of the 20th International Conference on Automated Deduction (CADE)*, vol. **3632** of *Lecture Notes in Computer Science*, pp. 38–53 (2005)
18. Vestergaard, R.: *The primitive proof theory of the  $\lambda$ -calculus*. Ph.D. thesis, School of Mathematical and Computer Sciences, Heriot-Watt University (2003)
19. Vestergaard, R., Brotherston, J.: A formalised first-order confluence proof for the  $\lambda$ -calculus using one-sorted variable names (Barendregt was right after all ... almost). In: Middeldorp, A. (ed.), *Proceedings of RTA-12*, vol. 2051 of *LNCS* (2001a)
20. Vestergaard, R., Brotherston, J.: The mechanisation of barendregt-style equational proofs (The Residual Perspective). *Electronic Notes in Theoretical Computer Science* **58**(1) (2001b)