A Formal Treatment of the Barendregt Variable Convention in Rule Inductions

Extended Abstract

Christian Urban
Ludwig-Maximilians-University Munich
urban@mathematik.uni-muenchen.de

Michael Norrish
Canberra Research Lab.,
National ICT Australia (NICTA)
Michael.Norrish@nicta.com.au

Abstract

Barendregt's variable convention simplifies many informal proofs in the λ-calculus by allowing the consideration of only those bound variables that have been suitably chosen. Barendregt does not give a formal justification for the variable convention, which makes it hard to formalise such informal proofs. In this paper we show how a form of the variable convention can be built into the reasoning principles for rule inductions. We give two examples explaining our technique.

Categories and Subject Descriptors F.4.1 [*Mathematical Logic*]: Lambda-calculus and related systems; I.2.3 [*Deduction and Theorem Proving*]: Deduction

General Terms Theory, Verification

Keywords Lambda-calculus, nominal logic, POPLmark challenge

1. Introduction

In informal proofs about languages that feature bound variables, one often assumes (explicitly or implicitly) rather convenient conventions about those bound variables. For example, in Barendregt's seminal book [2] about the λ-calculus:

> 2.1.12. CONVENTION. Terms that are α-congruent are identified. So now we write $\lambda x.x \equiv \lambda y.y$, etcetera.

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

Both conventions give rise to very slick informal proofs: the first convention assumes that the "data structure" over which the proofs are done is not that of syntax-trees, but of α-equivalent lambda-terms (or α-equivalence classes). However, the claim to be using α-equivalence classes, rather than syntax-trees, is often blurred by statements like [2]:

> 2.1.14. MORAL. Using conventions 2.1.12 and 2.1.13 one can work with λ-terms in the naive way.

One advantage of using α-equivalence classes is that capture avoiding substitution can be defined as a total function satisfying the following four properties [6, 13]:

- $var(a)[a := N] = N$
- $var(b)[a := N] = var(b)$ provided $b \neq a$
- $app(M_1, M_2)[a := N] = app(M_1[a := N], M_2[a := N])$
- $lam(b, M)[a := N] = lam(b, M[a := N])$
  provided $b \neq a$ and $b \notin FV(N)$

The second convention assumes that binders have always been so chosen that they do not clash with free variables. This avoids having to rename bound variables. When performing a structural induction, renaming bound variables can be handled by switching to inductions on term size, as done by Homeier [7]. When performing rule inductions, proofs typically need to be entirely recast, perhaps by proving properties involving iterated substitutions. In either case, and particularly the latter, the mechanisation can hardly be seen as faithful to the original presentation.

A typical informal proof making use of both conventions is presented in Figure 1.

In this proof, the equational reasoning in the variable-case (1.1.–1.3.) relies on the fact that substitution is a function. In the lambda-case, the reasoning further relies on the variable convention. This gives the assumption that z satisfies freshness constraints which allow the substitutions to be pushed under the binder. Then one can apply the induction hypothesis, and finally pull the substitutions back out from under the binder. In the absence of the variable convention, z

MERLIN'05 September 30, 2005, Tallinn, Estonia.

2.1.16. SUBSTITUTION LEMMA. If $x \not\equiv y$ and $x \notin FV(L)$, then

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

PROOF. By induction on the structure of M.

Case 1: M is a variable.

Case 1.1. $M \equiv x$. Then both sides equal $N[y := L]$ since $x \not\equiv y$.

Case 1.2. $M \equiv y$. Then both sides equal L, for $x \notin FV(L)$ implies $L[x := \ldots] \equiv L$.

Case 1.3. $M \equiv z \not\equiv x, y$. Then both sides equal z.

Case 2: $M \equiv \lambda z.M_1$. By the variable convention we may assume that $z \not\equiv x, y$ and z is not free in N, L. Then by induction hypothesis

$$\begin{aligned} & (\lambda z.M_1)[x := N][y := L] \\ \equiv\ & \lambda z.(M_1[x := N][y := L]) \\ \equiv\ & \lambda z.(M_1[y := L][x := N[y := L]]) \\ \equiv\ & (\lambda z.M_1)[y := L][x := N[y := L]]. \end{aligned}$$

Case 3: $M \equiv M_1 M_2$ The statement follows again from the induction hypothesis.

Figure 1. Barendregt's proof of the Substitution Lemma

might not be distinct from the free variables in the induction ($\{x, y\} \cup FV(N, L)$), and one has to rename z away from that set in order to move the substitutions around.

The approaches reported in [6, 14, 7, 10] show how to deal with the first convention in a formal setting, and do so without having to resort to a nameless de-Bruijn or HOAS-representation for α-equated λ-terms. They construct data structures which allow one to write $var(a)$, $app(M_1, M_2)$ and $lam(a, M)$ to denote α-equated variables, applications and abstractions. Furthermore, one has the equation

$$lam(a, M) = lam(b, N) \qquad (1)$$

whenever the corresponding syntax-trees "$\lambda a.M$" and "$\lambda b.N$" are α-equivalent.

One problem when working with name-carrying α-equivalence classes is that convenient structural induction principles do not come for "free", but need to be derived. Such convenient structural induction principles are derived in [5, 14, 13]. These induction principles are stated in such a way that they come very close to the convenience of the informal reasoning using Barendregt's variable convention. The structural induction principle of Urban and Tasson, for example, is:[1]

$$\frac{\begin{array}{l} \forall x\, z.\ \ P\ (var(z))\ x \\ \forall x\, M\, N. \\ \quad (\forall y.\, P\ M\ y)\ \wedge\ (\forall y.\, P\ N\ y)\ \Rightarrow P\ (app(M, N))\ x \\ \forall x\, z\, M. \\ \quad z \mathrel{\#} x\ \wedge\ (\forall y.\, P\ M\ y)\ \Rightarrow\ P\ (lam(z, M))\ x \end{array}}{P\ M\ x} \qquad (2)$$

where P stands for the property to be proved; M is the variable over which the induction is done, and the variable x for the *context* of the induction. By "the context of an induction", we mean all free variables of the induction hypothesis, except the variable over which the induction is performed. In case of the substitution lemma, the induction hypothesis P is

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

with M being the variable over which the induction is done. So in this case, the context x would need to be instantiated with the tuple (x, y, N, L). Then, when one comes to prove the lambda-case, one can assume in (2) that the binder z in

$$P\ (lam(z, M))\ (x, y, M, N)$$

is fresh w.r.t. (x, y, N, L)—meaning roughly that z cannot be equal to x and y, and that z cannot be a free variable in N and L. In effect, one can formalise Barendregt's slick informal proof without difficulties.

In this paper we show that similar induction principles can be given for rule inductions, provided a certain property holds for the relations over which the rule inductions are performed. We illustrate our technique with two examples: one is the proof of the substitutivity property of the $\underset{1}{\twoheadrightarrow}$ relation (this is one part of the simple Church-Rosser proof due to Tait and Martin-Löf for β-reduction [2]), and the other is the usual proof of the weakening lemma for simple types.

The proof by Tait and Martin-Löf does not show the Church-Rosser property directly for β-reduction, but for a more general reduction relation defined as:

$$\frac{}{M \underset{1}{\twoheadrightarrow} M} \qquad \frac{M \underset{1}{\twoheadrightarrow} M'}{lam(x, M) \underset{1}{\twoheadrightarrow} lam(x, M')}$$
$$\frac{M \underset{1}{\twoheadrightarrow} M' \quad N \underset{1}{\twoheadrightarrow} N'}{app(M, N) \underset{1}{\twoheadrightarrow} app(M', N')} \qquad (3)$$
$$\frac{M \underset{1}{\twoheadrightarrow} M' \quad N \underset{1}{\twoheadrightarrow} N'}{app(lam(x, M), N) \underset{1}{\twoheadrightarrow} M'[x := N']}$$

A central lemma in this Church-Rosser proof is then:

SUBSTITUTIVITY OF $\underset{1}{\twoheadrightarrow}$. If $M \underset{1}{\twoheadrightarrow} M'$ and $N \underset{1}{\twoheadrightarrow} N'$, then $M[x := N] \underset{1}{\twoheadrightarrow} M'[x := N']$.

[1] This is a slightly strengthened version of the induction principle given in [14], which in the light of this work seems more useful than the original version.

This proof proceeds in [2] by an induction over the definition of $M \underset{1}{\twoheadrightarrow} M'$. Though Barendregt does not acknowledge the fact explicitly (as he did in the substitution lemma), there are two places in this proof where the variable convention is used. In the case of the second rule of $\underset{1}{\twoheadrightarrow}$, for example, Barendregt writes (slightly changed to conform with our syntax):

CASE 2. $M \underset{1}{\twoheadrightarrow} M'$ is $lam(y, P) \underset{1}{\twoheadrightarrow} lam(y, P')$ and is a direct consequence of of $P \underset{1}{\twoheadrightarrow} P'$. By induction hypothesis one has

$$P[x := N] \underset{1}{\twoheadrightarrow} P[x := N'] \; .$$

But then

$$lam(y, P[x := N]) \underset{1}{\twoheadrightarrow} lam(y, P'[x := N']) \; ,$$

i.e. $M[x := N] \underset{1}{\twoheadrightarrow} M'[x := N']$.

The last step in this case only works if one knows that

$$\begin{array}{ll} lam(y, P[x := N]) \;\;\, = lam(y, P)[x := N] & \text{and} \\ lam(y, P'[x := N']) = lam(y, P')[x := N'] & \end{array}$$

which only holds for y being not equal to x and not free in N and N'. If y did not satisfy these constraints, one would have to rename first. The contribution of this paper is the technique allowing the derivation of a rule induction principle for $\underset{1}{\twoheadrightarrow}$ in which the above case can be proved under the assumption that $y \neq x$ and $y \notin FV(N, N')$. This additional assumption allows one to reason just like Barendregt in a rigorous, mechanised setting.

Our technique works for relations that are inductively-defined by a set of rules. Such relations come with a notion of rule induction, which can be used to prove theorems of the form

$$R(M_1, \ldots, M_n) \Rightarrow P(M_1, \ldots, M_n)$$

with R the original inductive relation on n arguments, and P another relation, which is shown to be a superset of R. The property we need for deriving our rule induction principles with a built-in form of the variable convention is that the relation R is *equivariant* [3, 12], that

$$\forall \pi.\; R(M_1, \ldots, M_n) \Rightarrow R(\pi \bullet M_1, \ldots, \pi \bullet M_n)$$

Equivariance therefore is the property that a relation is preserved under any permutation π, where permutations are finite bijective mappings from atoms to atoms. Beta-reduction, typing and $\underset{1}{\twoheadrightarrow}$ are instances of equivariant relations. So too are the reduction and typing relations for systems such as the polymorphic λ-calculus. In fact, equivariance is very common. Its absence would imply that a particular relation didn't behave uniformly over choices of the names (atoms) used as variables.

Our technique applies to all equivariant relations.

This paper is organised as follows: Sec. 2 gives a very brief overview about the nominal logic work. Proofs are omitted, but can be found in [3, 12, 14]. Sec. 3 illustrates our technique by deriving improved principles of rule induction for two standard relations from the literature. The ease-of-use of the new principles is also demonstrated. Sec. 4 mentions some related work and Sec. 5 concludes.

2. Nominal Logic

There are two central notions in nominal logic: permutations and support. As mentioned in the previous section, permutations are finite bijective mappings from atoms to atoms, where atoms are drawn from a countably infinite set denoted by $\mathbb{A}$. We represent permutations as finite lists whose elements are swappings (i.e. pairs of atoms). We write such permutations as $(a_1\, b_1)(a_2\, b_2)\cdots(a_n\, b_n)$; the empty list $[]$ stands for the identity permutation. A permutation π *acting* on an atom a is defined as:

$$\begin{array}{rcl} []\bullet a & \stackrel{\text{def}}{=} & a \\ ((a_1\, a_2) :: \pi)\bullet a & \stackrel{\text{def}}{=} & \begin{cases} a_2 & \text{if } \pi\bullet a = a_1 \\ a_1 & \text{if } \pi\bullet a = a_2 \\ \pi\bullet a & \text{otherwise} \end{cases} \end{array}$$

where $(a\, b) :: \pi$ is the composition of a permutation followed by the swapping $(a\, b)$. The composition of π followed by another permutation π' is given by list-concatenation, written as $\pi' @ \pi$, and the inverse of a permutation is given by list reversal, written as π^{-1}. Our representation of permutations as lists does not give unique representatives: for example, the permutation $(a\, a)$ is "equal" to the identity permutation.

We equate the representations of permutations with a relation $\sim$:

Definition 1 (Permutation Equality)**.** *Two permutations are* equal*, written* $\pi_1 \sim \pi_2$*, provided* $\pi_1 \bullet x = \pi_2 \bullet x$*, for all* $x \in \mathbb{A}$*.*

The action of a permutation can be lifted to other types as long as the action on the new type is *sensible*. By this we mean that it has to satisfy the following three properties:

$$\begin{array}{rl} (i) & []\bullet x = x \\ (ii) & (\pi_1 @ \pi_2)\bullet x = \pi_1 \bullet (\pi_2 \bullet x) \\ (iii) & \text{if } \pi_1 \sim \pi_2 \text{ then } \pi_1 \bullet x = \pi_2 \bullet x \end{array}$$

From this we can define "permutation sets" as those having a sensible permutation action:

Definition 2 (PSets)**.** *A set X equipped with a permutation action $\pi\bullet(-)$ is said to be a pset, if for all $x \in X$, the permutation action satisfies the properties (i)-(iii).*

The informal notation $x \in$ *pset* will be adopted whenever it needs to be indicated that x comes from a *pset*. Typical permutation actions permute all atoms in a given *pset*-element. For example, lists, tuples and sets can be seen as *pset*s if their respective permutation actions are defined point-wise:

$$
\begin{array}{lrcl}
\text{lists:} & \pi\bullet[] & \stackrel{\text{def}}{=} & [] \\
& \pi\bullet(x :: t) & \stackrel{\text{def}}{=} & (\pi\bullet x) :: (\pi\bullet t) \\
\text{tuples:} & \pi\bullet(x_1,\ldots,x_n) & \stackrel{\text{def}}{=} & (\pi\bullet x_1,\ldots,\pi\bullet x_n) \\
\text{sets:} & \pi\bullet X & \stackrel{\text{def}}{=} & \{\pi\bullet x \mid x \in X\}
\end{array}
$$

On α-equated λ-terms the permutation action is defined such that it satisfies:

$$
\begin{aligned}
\pi\bullet var(a) &= var(\pi\bullet a) \\
\pi\bullet app(M_1, M_2) &= app(\pi\bullet M_1, \pi\bullet M_2) \\
\pi\bullet lam(a, M) &= lam(\pi\bullet a, \pi\bullet M)
\end{aligned}
\tag{4}
$$

We note the following:

Lemma 1. The following sets are *pset*s: $\mathbb{A}$, the set of α-equated lambda-terms, and every set of lists (similarly tuples and sets) containing elements from *pset*s.

One interesting consequence of nominal logic [3] is that as soon as one fixes the notion of permutation action for a *pset*, then the notion of support, very roughly speaking its set of free atoms, is fixed as well. The support and the derived notion of freshness is defined as follows:

Definition 3 (Support and Freshness)**.** *Given $x \in$ pset, its* support *is defined as:*

$$\mathtt{supp}(x) \stackrel{\mathit{def}}{=} \{a \mid \mathtt{infinite}\{b \mid (a\,b)\bullet x \neq x\}\}\ .$$

An atom a is said to be fresh *for such an x, written $a \# x$, provided $a \notin \mathtt{supp}(x)$.*

With these notions in place we can make the equation in (1) precise, stating when two α-equated abstractions are equal:

$$
\begin{aligned}
& lam(a, M) = lam(b, N) \Leftrightarrow \\
& \quad (a = b \wedge M = N) \vee \\
& \quad (a \neq b \wedge M = (a\,b)\bullet N \wedge a \# N)
\end{aligned}
\tag{5}
$$

In what follows we will often make use the following properties of *pset*s:

Lemma 2. For all $x \in$ *pset*,
- (i) $a \# x$ implies $\pi\bullet a \# \pi\bullet x$
- (ii) if $a \# x$ and $b \# x$, then $(a\,b)\bullet x = x$
- (iii) $a \# (x, y)$ if and only if $a \# x$ and $a \# y$

A further restriction on *pset*s filters out all *pset*s containing elements with infinite support:

Definition 4 (Finitely Supported PSets)**.** *A pset X is said to be an fs-pset if every element in X has finite support.*

We note the following:

Lemma 3. *The following sets are fs-psets: $\mathbb{A}$, the set of α-equated λ-terms, and every set of lists (similarly tuples and finite sets) containing elements from fs-psets.*

Since the set of atoms $\mathbb{A}$ is infinite, the most important property of *fs-pset*s is that for each element one can choose a fresh atom.

Lemma 4. *For all $x \in$ fs-pset, there exists an atom a such that $a \# x$.*

Unwinding the definitions for permutation actions and support one can often easily calculate the support for *fs-pset*-elements:

$$
\begin{array}{ll}
\text{atoms:} & \mathtt{supp}(a) = \{a\} \\
\text{tuples:} & \mathtt{supp}(x_1,\ldots,x_n) = \mathtt{supp}(x_1) \cup \ldots \cup \mathtt{supp}(x_n) \\
\text{lists:} & \mathtt{supp}([]) = \varnothing \\
& \mathtt{supp}(x :: xs) = \mathtt{supp}(x) \cup \mathtt{supp}(xs) \\
\text{finite sets:} & \mathtt{supp}(X) = \bigcup_{x \in X} \mathtt{supp}(x)
\end{array}
$$

α-equated lambda-terms:

$$
\begin{array}{l}
\mathtt{supp}(var(a)) = \{a\} \\
\mathtt{supp}(app(M, N)) = \mathtt{supp}(M) \cup \mathtt{supp}(N) \\
\mathtt{supp}(lam(a, M)) = \mathtt{supp}(M) - \{a\}
\end{array}
$$

The last three equations show that the support of α-equated lambda-terms coincides with the usual notion of free variables. In turn, $a \# M$ with M being an α-equated lambda-term coincides with a not being free in M. If b is an atom, then $a \# b$ coincides with $a \neq b$.

3. Rule Inductions

Inductions over inductively-defined relations, also called rule inductions, are important reasoning tools in the λ-calculus and programming languages [1]. Here we provide two small, but typical, examples of such rule inductions, and illustrate our technique on both.

3.1 Weakening for Simple Types

Terms of the λ-calculus can be given types with respect to contexts (for example finite sets of name-type pairs). Types are of the form

$$ty: \quad \tau \ ::= \ X \mid \tau \to \tau$$

Contexts are *valid* if no variable occurs twice:

$$\frac{}{valid(\varnothing)} \qquad \frac{a \# \Gamma \quad valid(\Gamma)}{valid(a : \tau, \Gamma)}$$

The relation associating terms and types is straightforward to define:

$$\frac{valid(\Gamma) \quad (a : \tau) \in \Gamma}{\Gamma \vdash var(a) : \tau} \qquad \frac{a \# \Gamma \quad a : \tau, \Gamma \vdash M : \sigma}{\Gamma \vdash lam(a, M) : \tau \to \sigma}$$

$$\frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash app(M, N) : \sigma}$$

Making such a definition also results in the proof of the associated induction principle, with P a three-place, curried predicate:

$$\frac{\begin{array}{l}\forall \Gamma\, a\, \tau.\;\; \mathit{valid}(\Gamma) \wedge (a:\tau) \in \Gamma \Rightarrow P\ \Gamma\ (\mathit{var}(a))\ \tau \\ \forall \Gamma\, M\, N\, \tau\, \sigma.\;\; \Gamma \vdash M : \tau \to \sigma \wedge P\ \Gamma\ M\ (\tau \to \sigma) \wedge \\ \qquad\qquad\quad \Gamma \vdash N : \tau \wedge P\ \Gamma\ N\ \tau \Rightarrow \\ \qquad\qquad\quad P\ \Gamma\ (\mathit{app}(M,N))\ \sigma \\ \forall \Gamma\, a\, M\, \tau\, \sigma.\;\; a \# \Gamma \wedge (a:\tau,\Gamma) \vdash M : \sigma \wedge \\ \qquad\qquad\quad P\ (a:\tau,\Gamma)\ M\ \sigma \Rightarrow \\ \qquad\qquad\quad P\ \Gamma\ (\mathit{lam}(a,M))\ (\tau \to \sigma)\end{array}}{\Gamma \vdash M : \tau \;\Rightarrow\; P\ \Gamma\ M\ \tau} \tag{6}$$

We wish to prove the following property, where a context Γ_2 is weaker than Γ_1 (written $\Gamma_1 \subseteq \Gamma_2$), if every name-type pair in Γ_1 also appears in Γ_2:

Lemma 5 (Weakening Lemma)**.** *If $\Gamma_1 \vdash M : \tau$ is derivable, and $\Gamma_1 \subseteq \Gamma_2$ with Γ_2 being valid, then $\Gamma_2 \vdash M : \tau$ is also derivable.*

Proofs of this lemma are often claimed to be straightforward (e.g. [11]). The informal proof usually goes as follows:

INFORMAL PROOF OF THE WEAKENING LEMMA. By rule induction over $\Gamma_1 \vdash M : \tau$.

CASE 1: $\Gamma_1 \vdash M : \tau$ is $\Gamma_1 \vdash \mathit{var}(a) : \tau$. By assumption we know $\mathit{valid}(\Gamma_2)$, $(a : \tau) \in \Gamma_1$ and $\Gamma_1 \subseteq \Gamma_2$. Therefore we can use the typing rules to derive $\Gamma_2 \vdash \mathit{var}(a) : \tau$.

CASE 2: $\Gamma_1 \vdash M : \tau$ is $\Gamma_1 \vdash \mathit{app}(M_1, M_2) : \tau$. Case follows from the induction hypotheses and the typing rules.

CASE 3: $\Gamma_1 \vdash M : \tau$ is $\Gamma_1 \vdash \mathit{lam}(a, M_1) : \tau \to \sigma$. Although, one has to prove this case for all a, using the variable convention we assume that a does not occur in Γ_2. Then we know by the induction hypothesis that $(a : \tau, \Gamma_2) \vdash M_1 : \sigma$ holds. Hence also $\Gamma_2 \vdash \mathit{lam}(a, M_1) : \tau \to \sigma$ by the typing rules. "$\Box$"

Because of the arguably questionable use of the variable convention in the third case, this informal proof is painful to formalise using the original induction principle (6), see for example [4]. In particular, the abstraction case in a formal proof will typically require the abstraction to have its bound variable renamed to be suitably fresh. The proof of the lemma also then requires an equivariance result

$$\Gamma \vdash M : \tau \;\Rightarrow\; (\pi \bullet \Gamma) \vdash (\pi \bullet M) : \tau \tag{7}$$

to be shown.[2] This lemma is generally useful, and because of the existence of inverses for permutations can be recast as

$$(\pi \bullet \Gamma) \vdash (\pi \bullet M) : \tau \;\Leftrightarrow\; \Gamma \vdash M : \tau$$

or even

$$(\pi \bullet \Gamma) \vdash M : \tau \;\Leftrightarrow\; \Gamma \vdash (\pi^{-1} \bullet M) : \tau$$

which is a useful theorem to rewrite with because it collects all of the permutations in a typing judgement and moves them so that they apply only to the term argument.

[2] The proof is an easy induction using (6).

(Another possibility when formalising results such as these is to show that the relation is preserved under iterated variable-for-variable substitutions. This result applies when the body of an abstraction acquires an extra substitution through renaming. Because substitutions are harder to reason with than permutations, this approach is usually less attractive than performing the body's renaming with a permutation.)

The painful requirement to rename the bound variable in the rule induction tends to happen every time a rule induction with (6) is performed (though not in the proof of (7), pleasantly). We can avoid this by proving a more useful induction hypothesis once and for all:

Theorem 1. *For all typing contexts Γ, all α-equated λ-terms M, all $\tau \in ty$ and all contexts $x \in$ fs-pset, the following implication holds:*

$$\frac{\begin{array}{l}\forall x\, \Gamma\, a\, \tau.\;\; \mathit{valid}(\Gamma) \wedge (a:\tau) \in \Gamma \Rightarrow P\ \Gamma\ (\mathit{var}(a))\ \tau\ x \\ \forall x\, \Gamma\, M\, N\, \tau\, \sigma. \\ \quad \Gamma \vdash M : \tau \to \sigma \wedge (\forall z.\;\; P\ \Gamma\ M\ (\tau \to \sigma)\ z) \wedge \\ \quad \Gamma \vdash N : \tau \wedge (\forall z.\;\; P\ \Gamma\ N\ \tau\ z) \;\Rightarrow \\ \quad P\ \Gamma\ (\mathit{app}(M,N))\ \sigma\ x \\ \forall x\, a\, \Gamma\, M\, \tau\, \sigma. \\ \quad a \# x \;\wedge a \# \Gamma \wedge (a:\tau,\Gamma) \vdash M : \sigma \wedge \\ \quad (\forall z.\, P\ (a:\tau,\Gamma)\ M\ \sigma\ z) \;\Rightarrow \\ \quad P\ \Gamma\ (\mathit{lam}(a,M))\ (\tau \to \sigma)\ x\end{array}}{\Gamma \vdash M : \tau \;\Rightarrow\; P\ \Gamma\ M\ \tau\ x}$$

Proof. The proof uses the original induction principle (6). We strengthen the goal by aiming to prove

$$\forall \pi\ \Gamma\ M\ \tau\ (x \in \mathit{fs\text{-}pset}).\;\; (\ \ldots\ ) \Rightarrow P\ (\pi \bullet \Gamma)\ (\pi \bullet M)\ \tau\ x\ \ .$$

In the variable-case we need to prove

$$P\ (\pi \bullet \Gamma)\ (\mathit{var}(\pi \bullet a))\ \tau\ x$$

while knowing that $\mathit{valid}(\Gamma)$ and $(a : \tau) \in \Gamma$ hold. Validity of contexts is preserved under permutations, so we have $\mathit{valid}(\pi \bullet \Gamma)$. From $(a : \tau) \in \Gamma$ we can infer $(\pi \bullet a : \tau) \in \pi \bullet \Gamma$ and hence we can use the assumed implication

$$\forall x\, \Gamma\, a\, \tau.\;\; \mathit{valid}(\Gamma) \wedge (a:\tau) \in \Gamma \Rightarrow P\ \Gamma\ \mathit{var}(a)\ \tau\ x$$

to obtain $P\ (\pi \bullet \Gamma)\ (\mathit{var}(\pi \bullet a))\ \tau\ x$.

The application-case is routine. The interesting case is the lambda-case. In this case we need to prove

$$P\ (\pi \bullet \Gamma)\ (\mathit{lam}(\pi \bullet a, \pi \bullet M))\ (\tau \to \sigma)\ x$$

under the assumption that

$$\begin{array}{rl} (i) & a \# \Gamma \\ (ii) & a : \tau, \Gamma \vdash M : \sigma \\ (iii) & \forall \pi\, x.\, P\ (\pi \bullet (a:\tau,\Gamma))\ (\pi \bullet M)\ \sigma\ x \end{array} \tag{8}$$

Since atoms, λ-terms and typing contexts are finitely supported and by assumption also x, there exists by Lem. 4 an atom c with $c \# (\pi\bullet a, \pi\bullet M, \pi\bullet\Gamma, x)$. Using (iii) we can infer

$$\forall x.\ P\ ((c\ \pi\bullet a)\bullet\pi\bullet(a:\tau,\Gamma))\ ((c\ \pi\bullet a)\bullet\pi\bullet M)\ \tau\ x$$

which is

$$\forall x.\ P\ (c:\tau,(c\ \pi\bullet a)\bullet\pi\bullet\Gamma)\ ((c\ \pi\bullet a)\bullet\pi\bullet M)\ \tau\ x \qquad (9)$$

From (i) and Lem. 2(i) follows that $\pi\bullet a \# \pi\bullet\Gamma$. With $c \# \pi\bullet\Gamma$ we can infer using Lem. 2(ii) that $(c\ \pi\bullet a)\bullet\pi\bullet\Gamma = \pi\bullet\Gamma$, and hence simplify (9) further to

$$\forall x.\ P\ (c:\tau,\pi\bullet\Gamma)\ ((c\ \pi\bullet a)\bullet\pi\bullet M)\ \tau\ x \qquad (10)$$

By equivariance of the typing relation (7), we can use (ii) and infer

$$(c\ \pi\bullet a)\bullet\pi\bullet(a:\tau,\Gamma) \vdash (c\ \pi\bullet a)\bullet\pi\bullet M : \sigma$$

which is

$$(c:\tau,\pi\bullet\Gamma) \vdash (c\ \pi\bullet a)\bullet\pi\bullet M : \sigma \qquad (11)$$

Now we can use $c \# x$, (10), (11) and infer from the assumed implication in the lambda-case that

$$P\ (\pi\bullet\Gamma)\ (lam(c,(c\ \pi\bullet a)\bullet\pi\bullet M))\ (\tau\to\sigma)\ x$$

Because $c \# \pi\bullet a$ and $c \# \pi\bullet M$, we know by (5), however, that

$$lam(c,(c\ \pi\bullet a)\bullet\pi\bullet M) = lam(\pi\bullet a,\pi\bullet M)$$

and we are done. □

With this induction principle at our disposal, the proof of the weakening lemma is simple.

Proof of the Weakening Lemma. Perform a rule induction over $\Gamma_1 \vdash M:\tau$ using the induction hypothesis

$$\Gamma_1 \subseteq \Gamma_2 \Rightarrow \mathit{valid}(\Gamma_2) \Rightarrow \Gamma_2 \vdash M:\tau$$

That is, we instantiate Thm. 1 with

$$\begin{array}{lcl} P & = & \lambda\Gamma_1\,M\,\tau\,\Gamma_2.\ \Gamma_1 \subseteq \Gamma_2 \Rightarrow \mathit{valid}(\Gamma_2) \Rightarrow \Gamma_2 \vdash M:\tau \\ \Gamma & = & \Gamma_1 \\ M & = & M \\ \tau & = & \tau \\ x & = & \Gamma_2 \end{array}$$

where $\Gamma_2 \in$ *fs-pset* by Lem. 3. This gives the following three sub-goals:

(1) $\mathit{valid}(\Gamma'_1) \wedge (a:\tau)\in\Gamma'_1 \wedge \Gamma'_1\subseteq\Gamma'_2 \wedge \mathit{valid}(\Gamma'_2)$
$\Rightarrow \Gamma'_2 \vdash var(a):\tau$

(2) $\Gamma'_1 \vdash M_1:\tau\to\sigma \wedge \Gamma'_1\vdash M_2:\tau\ \wedge$
$(\forall\Gamma'_3.\ \Gamma'_1\subseteq\Gamma'_3 \Rightarrow \mathit{valid}(\Gamma'_3)\Rightarrow\Gamma'_3\vdash M_1:\tau\to\sigma)\ \wedge$
$(\forall\Gamma'_3.\ \Gamma'_1\subseteq\Gamma'_3 \Rightarrow \mathit{valid}(\Gamma'_3)\Rightarrow\Gamma'_3\vdash M_2:\tau)\ \wedge$
$\Gamma'_1\subseteq\Gamma'_2 \wedge \mathit{valid}(\Gamma'_2)$
$\Rightarrow \Gamma'_2 \vdash app(M_1,M_2):\sigma$

(3) $a \# \Gamma'_2 \wedge a \# \Gamma'_1 \wedge (a:\tau,\Gamma'_1)\vdash M:\sigma\ \wedge$
$(\forall\Gamma'_3.\ (a:\tau,\Gamma'_1)\subseteq\Gamma'_3 \Rightarrow \mathit{valid}(\Gamma'_3) \Rightarrow \Gamma'_3\vdash M:\sigma)\ \wedge$
$\Gamma'_1\subseteq\Gamma'_2 \wedge \mathit{valid}(\Gamma'_2)$
$\Rightarrow \Gamma'_2 \vdash lam(a,M):\tau\to\sigma$

where the first two are trivial. For (3) we instantiate $\forall\Gamma'_3$ with $(a:\tau,\Gamma'_2)$. The fact $(a:\tau,\Gamma'_1)\subseteq(a:\tau,\Gamma'_2)$ follows from $\Gamma'_1\subseteq\Gamma'_2$; $\mathit{valid}(a:\tau,\Gamma'_2)$ follows from $\mathit{valid}(\Gamma'_2)$ and $a \# \Gamma'_2$. This gives us $(a:\tau,\Gamma'_2)\vdash M:\sigma$. Now we immediately obtain $\Gamma'_2\vdash lam(a,M):\tau\to\sigma$ using the definition of the typing relation and the fact that $a \# \Gamma'_2$. □

This example also shows why the new induction principle needs to have universal quantifications over the contexts in the premises. For example, in the lambda-case the assumed implication has the premise $(\forall z.\ P\ (a:\tau,\Gamma)\ M\ \sigma\ z)$:

$$\begin{array}{l} \forall x\,a\,\Gamma\,M\,\tau\,\sigma.\ a \# x\ \wedge a \# \Gamma \wedge (a:\tau,\Gamma)\vdash M:\sigma\ \wedge \\ \qquad (\forall z.\ P\ (a:\tau,\Gamma)\ M\ \sigma\ z) \Rightarrow \\ \qquad P\ \Gamma\ (lam(a,M))\ (\tau\to\sigma)\ x \end{array}$$

If we had stated the induction principle using the following simpler implication[3]

$$\begin{array}{l} \forall x\,a\,\Gamma\,M\,\tau\,\sigma.\ a \# x\ \wedge a \# \Gamma \wedge (a:\tau,\Gamma)\vdash M:\sigma\ \wedge \\ \qquad P\ (a:\tau,\Gamma)\ M\ \sigma\ x \Rightarrow \\ \qquad P\ \Gamma\ (lam(a,M))\ (\tau\to\sigma)\ x \end{array}$$

then the induction hypothesis with which we proved the weakening lemma, namely

$$\Gamma_1 \subseteq \Gamma_2 \Rightarrow \mathit{valid}(\Gamma_2) \Rightarrow \Gamma_2 \vdash M:\tau$$

would not have been strong enough. We would have to make it stronger as follows

$$\forall\Gamma_2.\ \Gamma_1 \subseteq \Gamma_2 \Rightarrow \mathit{valid}(\Gamma_2) \Rightarrow \Gamma_2 \vdash M:\tau$$

but then we would not be able to use the assumption $a \# \Gamma_2$, which was vital in the weakening lemma to get the lambda-case through. In effect, we would have to perform renamings. With the version of the rule induction we have given, this is unnecessary.

3.2 Substitutivity of One-Reduction

The central lemma in proof given in [2] for the Church-Rosser property of beta-reduction is the substitutivity of the $\underset{1}{\longrightarrow\!\!\!\!\!\to}$ -reduction shown in (3). The induction principle that comes with this definition is as follows:

$$\frac{\begin{array}{l} \forall M.\ P\ M\ M \\ \forall a\,M\,M'. \\ \quad M \underset{1}{\twoheadrightarrow} M' \wedge P\ M\ M' \Rightarrow \\ \quad P\ (lam(a,M))\ (lam(a,M')) \\ \forall M\,M'\,N\,N'. \\ \quad M \underset{1}{\twoheadrightarrow} M' \wedge P\ M\ M' \wedge \\ \quad N \underset{1}{\twoheadrightarrow} N' \wedge P\ N\ N' \Rightarrow \\ \quad P\ (app(M,N))\ (app(M',N')) \\ \forall a\,M\,M'\,N\,N'. \\ \quad M \underset{1}{\twoheadrightarrow} M' \wedge P\ M\ M' \wedge \\ \quad N \underset{1}{\twoheadrightarrow} N' \wedge P\ N\ N' \Rightarrow \\ \quad P\ (app(lam(a,M),N))\ (M'[a:=N']) \end{array}}{M \underset{1}{\twoheadrightarrow} N \Rightarrow P\ M\ N} \qquad (12)$$

[3] This version is similar in style to the structural induction principle in [14].

Our technique derives the following new induction principle:

Theorem 2. *For all α-equated λ-terms M and N and all contexts $x \in$ fs-pset, the following implication holds:*

$$
\dfrac{
\begin{array}{l}
\forall x\, M.\ P\ M\ M\ x \\
\forall x\, a\, M\, M'.\ \\
\quad a \# x \wedge M \longrightarrow_1 M' \wedge (\forall z.\ P\ M\ M'\ z) \Rightarrow \\
\quad P\ (lam(a, M))\ (lam(a, M'))\ x \\
\forall x\, M\, M'\, N\, N'.\ \\
\quad M \longrightarrow_1 M' \wedge (\forall z.\ P\ M\ M'\ z) \wedge \\
\quad N \longrightarrow_1 N' \wedge (\forall z.\ P\ N\ N'\ z) \Rightarrow \\
\quad P\ (app(M, N))\ (app(M', N'))\ x \\
\forall x\, a\, M\, M'\, N\, N'.\ \\
\quad a \# (x, N, N') \wedge M \longrightarrow_1 M' \wedge (\forall z.\ P\ M\ M'\ z) \wedge \\
\quad N \longrightarrow_1 N' \wedge (\forall z.\ P\ N\ N'\ z) \Rightarrow \\
\quad P\ (app(lam(a, M), N))\ (M'[a := N'])\ x
\end{array}
}{
M \longrightarrow_1 N \ \Rightarrow\ P\ M\ N\ x
}
$$

Proof (Sketch). The proof is similar to the proof of Thm. 1. We need to strengthen the induction hypothesis to be of the form:

$$\forall \pi\, M\, N\, x \in \textit{fs-pset}.\ (\ldots) \ \Rightarrow\ P\ (\pi \bullet M)\ (\pi \bullet N)\ x\ .$$

In the second and fourth rule we can chose a fresh atom c w.r.t. (x, N, N') since α-equated λ-terms and x are finitely supported. We must prove equivariance for $\longrightarrow_1$, namely

$$\forall \pi.\ M \longrightarrow_1 N \ \Rightarrow\ (\pi \bullet M) \longrightarrow_1 (\pi \bullet N)$$

in order to apply the assumed implications (this can be easily established using the original induction principle shown in (12)). □

Using the new induction principle to prove

$$M \longrightarrow_1 M' \Rightarrow N \longrightarrow_1 N' \Rightarrow M[x := N] \longrightarrow_1 M'[x := N']$$

leads to a very simple substitutivity proof: all cases are quite simple calculations about substitutions. In the second case, we have $M \longrightarrow_1 M'$ being $lam(a, P) \longrightarrow_1 lam(a, P')$ and $a \# (x, N, N')$. The latter assumption allows us to move the substitutions $[x := N]$ and $[x := N']$ freely under the binder and back out.

In the fourth case, we have $M \longrightarrow_1 M'$ being

$$app(lam(a, P), Q) \longrightarrow_1 P[a := Q]\ .$$

Because of the assumption in this case that a is fresh for (x, Q, Q', N, N') we have

$$
\begin{aligned}
&app(lam(a, P), Q)[x := N] = \\
&\qquad app(lam(a, P[x := N]), Q[x := N])
\end{aligned}
\tag{13}
$$

and know by the induction hypotheses

$$
\begin{array}{l}
N \longrightarrow_1 N' \ \Rightarrow\ P[x := N] \longrightarrow_1 P'[x := N'] \\
N \longrightarrow_1 N' \ \Rightarrow\ Q[x := N] \longrightarrow_1 Q'[x := N']
\end{array}
$$

(we also have $N \longrightarrow_1 N'$). The definition of $\longrightarrow_1$ tells us that the right-hand side of (13) reduces to:

$$P'[x := N'][a := Q'[x := N']]$$

Now the freshness constraints $a \# x$ and $a \# N'$ match exactly the pre-conditions for the substitution lemma, which gives us

$$P'[a := Q'][x := N']\ .$$

This completes the proof.

4. Related Work

The prettiest formal proof of the weakening lemma we found in the existing literature is that in [12]. Pitts's proof uses the equivariance property of the typing relation, and includes a renaming step using permutations. Because of the pleasant properties that permutations enjoy (they are bijective renamings, in contrast to substitutions which might identify two names), the renaming can be done with minimal overhead. Our contribution is that we have effectively built this renaming into the induction principles once and for all. Proofs using our principles do not need to perform explicit renaming steps at all. Furthermore, we have created a full reasoning framework in HOL4 and Isabelle/HOL in which results about calculi with binders can be proved at least as conveniently as in other mechanisations.

Somewhat similar to our approach is the work of Pollack and McKinna [9]. Starting from the standard induction principle that is associated with an inductive definition, we derived an induction principle that allows emulation of Barendregt's variable convention. Pollack and McKinna, in contrast, gave a "weak" and "strong" version of the typing relation. These versions differ in the way the rule for abstractions is stated:

$$\dfrac{x \# M \quad (x : \tau, \Gamma) \vdash M[y := x] : \sigma}{\Gamma \vdash lam(y, M) : \tau \to \sigma}\ \text{weak}$$

$$\dfrac{\forall x.\ x \# \Gamma \Rightarrow (x : \tau, \Gamma) \vdash M[y := x] : \sigma}{\Gamma \vdash lam(y, M) : \tau \to \sigma}\ \text{strong}$$

They then showed that both versions derive the same typing judgements. With this they proved the weakening lemma using the "strong" version of the principle, while knowing that the result held for the "weak" relation as well. The main difference between this and our work seems to be of convenience: we can relatively easily derive, in a uniform way, an induction principle for equivaraint relations that allows the variable convention (we have illustrated this point with two examples). Achieving the same uniformity in the style of McKinna and Pollack does not seem as straightforward.

5. Conclusion

In the POPLMARK Challenge [1], the proof of the weakening lemma is described as a "straightforward induction". In fact, mechanising this informal proof is *not* straightforward at all (see for example [9, 4, 12]). We have given a novel rule induction principle for the typing relation that makes proving the weakening lemma mechanically as simple as the performing the informal proof. We have also illustrated our technique with another typical proof taken from the λ-calculus. We see no problems in extending this technique to other calculi with bound names.

One remaining challenge is to provide machine support to derive our new rule induction principles automatically. At the moment, we have proved the principles manually. There is clearly a pattern in the statement of the revised principles, and in their proof:

- bound variables that appear in rules can be assumed to be fresh with respect both to free variables in those rules and with respect to an additional "context" parameter; and
- proofs proceed by showing equivariance for the relation, and then using the original induction principles to show that an induction parameter P holds for all possible permutations of its parameters.

The first element of the pattern is not yet rigorous. For example, in the last rule of the principle in Theorem 2, the bound atom a can be assumed fresh with respect to variables N and N', but not M and M'. As a binds over M it is reasonable that it not be forced to be free there, but it is not syntactically clear why M' is excluded. Semantically, it *is* clear: $M \underset{1}{\longrightarrow} M'$, and so a may also appear in M'. For arbitrary relations, this may not always hold, and we may have to content ourselves with syntactic heuristics.

In any case, requiring one simple, stereotyped induction in order to reproduce the ease-of-use of Barendregt's Variable Convention seems a very small price to pay. We have shown this by implementing our results in HOL4 and Isabelle/HOL.

Acknowledgements We are very grateful to Andrew Pitts, for the many discussions with him on the subject of the paper, and to René Vestergaard, who provided us with excellent working conditions at the recent "Binding Workshop" at JAIST. The first author is supported by a fellowship from the Alexander-von-Humboldt Foundation.

References

[1] B. E. Aydemir, A. Bohannon, M. Fairbairn, J. N. Foster, B. C. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, and S. Zdancewic. Mechanized metatheory for the masses: the POPLmark challenge. In Hurd and Melham [8].

[2] H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1981.

[3] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.

[4] J. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper & Row, 1986.

[5] A. D. Gordon. A mechanisation of name-carrying syntax up to alpha-conversion. In *Proceedings of Higher-order Logic Theorem Proving and its Applications (HUG'93)*, volume 780 of *Lecture Notes in Computer Science*, pages 414–426. Springer, 1994.

[6] A. D. Gordon and T. Melham. Five axioms of alpha conversion. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, volume 1125 of *Lecture Notes in Computer Science*, pages 173–190. Springer-Verlag, 1996.

[7] P. Homeier. A proof of the Church-Rosser theorem for the lambda calculus in higher order logic. In R. J. Boulton and P. B. Jackson, editors, *TPHOLs'01: Supplemental Proceedings*, pages 207–222. Division of Informatics, University of Edinburgh, September 2001. Available as Informatics Research Report EDI-INF-RR-0046.

[8] J. Hurd and T. Melham, editors. *Theorem Proving in Higher Order Logics, 18th International Conference*, volume 3603 of *Lecture Notes in Computer Science*. Springer, 2005.

[9] J. McKinna and R. Pollack. Some type theory and lambda calculus formalised. *Journal of Automated Reasoning*, 23(1-4), 1999.

[10] M. Norrish. Mechanising λ-calculus using a classical first order theory of terms with permutation. *Higher Order and Symbolic Computation*. To appear.

[11] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[12] A. M. Pitts. Nominal logic: A first order theory of names and binding. In *Theoretical Aspects of Computer Software, 4th International Symposium, TACS 2001, Sendai, Japan, October 29-31, 2001, Proceedings*, volume 2215 of *Lecture Notes in Computer Science*, pages 219–242. Springer-Verlag, 2001.

[13] A. M. Pitts. Alpha-structural recursion and induction (extended abstract). In Hurd and Melham [8], pages 17–34.

[14] C. Urban and C. Tasson. Nominal techniques in Isabelle/HOL. In *Proc. of the 20th International Conference on Automated Deduction (CADE)*, volume 3632 of *Lecture Notes in Computer Science*, pages 38–53, 2005.