Proof Pearl: De Bruijn Terms Really Do Work

Michael Norrish[1] and René Vestergaard[2]

[1] Canberra Research Lab, Nicta
Michael.Norrish@nicta.com.au
[2] School of Information Science, JAIST, Ishikawa, Japan
vester@jaist.ac.jp

Abstract. Placing our result in a web of related mechanised results, we give a direct proof that the de Bruijn λ-calculus (à la Huet, Nipkow and Shankar) is isomorphic to an α-quotiented λ-calculus. In order to establish the link, we introduce an "index-carrying" abstraction mechanism over de Bruijn terms, and consider it alongside a simplified substitution mechanism. Relating the new notions to those of the α-quotiented and the proper de Bruijn formalisms draws on techniques from the theory of nominal sets.

1 Introduction

Implementors and theorists alike have long been in de Bruijn's debt [4]. Representing λ-terms without names is a relatively straightforward way of implementing binders internally in, e.g., theorem-proving systems such as Isabelle and HOL4. Complementing this, the same idea has made it possible to formalise a great many results about syntax that features binders.

However, we believe that the state-of-the-art for formalisation has moved on. Implementors may well keep de Bruijn terms in their systems' internals, but it does now seem as if it is possible to mechanise important results about syntax without needing to prove "painful" and "non-obvious" lemmas about indices [11]. The emerging formalisation technologies are appealing for a range of reasons, but here we focus on just two: i) it is now possible to check that multiple formalisations do actually correspond and ii) "non-obvious" lemmas about indices turn out to be related to general nominal properties of de Bruijn terms.

In particular, we can confirm that de Bruijn terms are a model for the λ-calculus, and can do so in an elegant way. By way of contrast, see for example Appendix C of Barendregt [2], where the proofs are given as "implicit in de Bruijn [4]"; or the pioneering work by Shankar [15], where mechanised proofs of complex theorems establish a connection between *unquotiented* λ-terms and de Bruijn terms.

In fact, there are a number of isomorphism results already extant in the literature. In this paper, we summarise these, and provide a new link between de Bruijn terms and what we hope will be an ever-expanding web of results, and body of understanding.

K. Schneider and J. Brandt (Eds.): TPHOLs 2007, LNCS 4732, pp. 207–222, 2007.

2 Formalising Binding

In this section, we position our result in the space of relationships between ten different formalisations of the λ-calculus, following Figure 1. $\boldsymbol{\lambda^C}$ is Curry's unquotiented λ-terms using variable names and fresh-naming substitution [3] (plus explicit α [7]). $\boldsymbol{\lambda^{Hi}_{\alpha}}$, due to Hindley [7], is a quotient construction saying that an equivalence class reduces to another if there are two syntactic terms that witness the reduction, in this case in λ^C. $\boldsymbol{\lambda^V}$ is similar to λ^C but uses renaming-free substitution (and explicit α), as studied in Vestergaard [17]. $\boldsymbol{\lambda^V_{\alpha}}$ is the "Hindley-quotient" of λ^V. $\boldsymbol{\lambda^{Ho}_{\alpha}}$ is a quotient of Curry's set-up but with substitution rather than reduction lifted from the syntactic to the quotient type, as studied in Homeier [8]. $\boldsymbol{dB_2}$ is the set of well-formed, two-sorted de Bruijn terms in Gordon [5]. $\boldsymbol{\lambda^G_{\alpha}}$ is the type derived from this, in Gordon and Melham [6]. $\boldsymbol{\lambda^N_{\alpha}}$ is a quotient of the first order syntax, with substitution defined directly on the quotiented level, as in Norrish [12]. $\boldsymbol{dB}$ is the type of pure de Bruijn terms (with indices representing free and bound variables), as mechanised in Shankar [15], Huet [9] and Nipkow [11]. Finally, $\boldsymbol{dB'}$ is an intermediate formalism that we define here. It uses constructors that are not injective, hence the oval node: conceptually, $\boldsymbol{dB'}$ is syntactic sugar for $\boldsymbol{dB}$ that looks like λ^N_{α}.

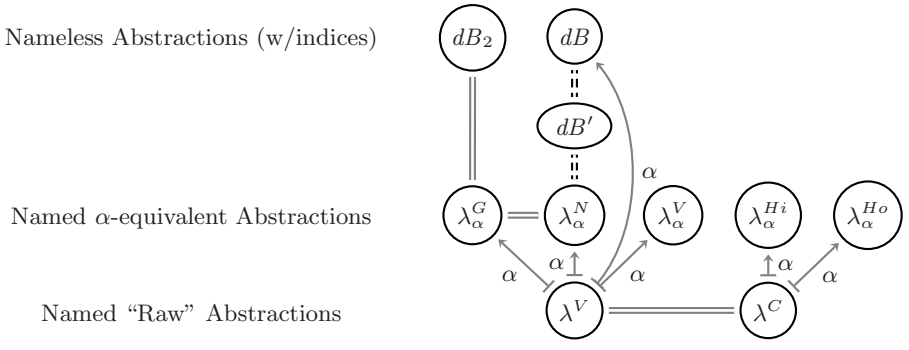

Fig. 1. λ-calculus formalisations and their directly-proved relationships; dashed, dark lines are the results in this paper. Double lines are isomorphisms; arrows α-quotienting.

2.1 Named Abstraction

We use Λ^{var} to refer to the "raw" syntax of the λ-calculus, that is the free algebra generated by the recursion equation

$$\Lambda^{\mathrm{var}} \cong V \;+\; \Lambda^{\mathrm{var}} \times \Lambda^{\mathrm{var}} \;+\; V \times \Lambda^{\mathrm{var}} \tag{1}$$

with V some countably-infinite set of variable names. We use Λ_α to refer to the quotient of Λ^{var} with respect to α-equivalence. The various λ^X_α are reduction systems defined over Λ_α, while λ^V and λ^{C} are defined directly over Λ^{var}.

Table 1. Signatures for formalisms with named abstraction, see Figure 1. The α-collapsed types below a dashed line are bootstrapped from the preceding "raw" type (λ^V/λ^C). For λ^G_α, substitution is bootstrapped from dB_2. See the text for the [proofs: ...]-comments.

λ^V [proofs: $=_{\Lambda^{\mathrm{var}}}$]	substitution: $\Lambda^{\mathrm{var}} \times V \times \Lambda^{\mathrm{var}} \to \Lambda^{\mathrm{var}}$ reduction: $\Lambda^{\mathrm{var}} \times \Lambda^{\mathrm{var}}$
λ^V_α	reduction: $\Lambda_\alpha \times \Lambda_\alpha$
λ^C [proofs: $=_\alpha$]	substitution: $\Lambda^{\mathrm{var}} \times V \times \Lambda^{\mathrm{var}} \to \Lambda^{\mathrm{var}}$ reduction: $\Lambda^{\mathrm{var}} \times \Lambda^{\mathrm{var}}$
λ^{Hi}_α	reduction: $\Lambda_\alpha \times \Lambda_\alpha$
λ^{Ho}_α	substitution: $\Lambda_\alpha \times V \times \Lambda_\alpha \to \Lambda_\alpha$ reduction: $\Lambda_\alpha \times \Lambda_\alpha$
λ^G_α	substitution: $\begin{cases} (dB_2 \times V \times dB_2 \to dB_2) \\ \Lambda_\alpha \times V \times \Lambda_\alpha \to \Lambda_\alpha \end{cases}$ reduction: $\Lambda_\alpha \times \Lambda_\alpha$
λ^N_α	substitution: $\Lambda_\alpha \times V \times \Lambda_\alpha \to \Lambda_\alpha$ reduction: $\Lambda_\alpha \times \Lambda_\alpha$

There are seven different types identified in Figure 1 that use names under abstractions. Though intuition (and, as we shall see, mechanised results) assure us that they are all the "same", their constructions are all different to varying degrees. The differences are most clearly seen in the types' definitions of substitution, see Table 1. At the "raw" level, λ^V defines substitution to be "renaming free", as used to be the tradition in logic books: where a substitution would cause a free variable to become captured, the substitution is instead defined to be the identity function, with the implicit (and satisfiable) requirement that actual uses avoid such situations. This approach finesses many of the problems of λ^C, where an everywhere-correct capture-avoiding substitution mandates fresh-naming (via either simultaneous substitution, as in Homeier [8], or by well-founded recursion on term sizes) and, thus, only allows us to reason "up to α" for most properties [7]. Alternatively, and with substitution defined for Λ^{var}, the type itself can be α-quotiented. In the case of λ^{Ho}_α, Homeier lifts the substitution function and proceeds to define β-reduction at the quotiented type. In the case of λ^{Hi}_α and λ^V_α, Hindley and Vestergaard lift β-reductions from Λ^{var} [7,17].

In λ^G_α, substitution is defined with reference to the substitution function in dB_2. In λ^N_α, substitution is defined directly on Λ_α, using its "native" notion of primitive recursion, as in Norrish [12] and Pitts [14].

2.2 De Bruijn's Anonymous Abstraction, Using Indices

The type of de Bruijn terms is the free algebra generated by the recursion equation

$$\mathtt{dB} \;\cong\; \mathbb{N} + \mathtt{dB} \times \mathtt{dB} + \mathtt{dB} \tag{2}$$

We will use constructors `dV`, `dAPP`, and `dABS` to correspond to the three cases in the equation above, with the idea that, e.g., `dABS (dV 0)` is the *identity* function: "$\lambda x.x$", and `dABS (dABS (dAPP (dV 1) (dV 0)))` is *apply*: "$\lambda f.\lambda x.f\ x$". We will use t, u, z to range over values in `dB`, while M, N will range over values in Λ_α.

Nipkow [11] (following Huet [9]) defines two functions by primitive recursion over this type (see Figure 2). The first is `lift`, which adds one to the value of all free indices that are at least equal to the provided bound. (Shankar [15] calls his version of this function `BUMP`; it is slightly different because he has the first bound index of an abstraction be 1, not 0.)

```
lift (dV i) k       = if i < k then dV i
                      else dV (i + 1)
lift (dAPP t u) k   = dAPP (lift t k) (lift u k)
lift (dABS t) k     = dABS (lift t (k + 1))

nsub z k (dV i)     = if k < i then dV (i - 1)
                      else if i = k then z
                      else dV i
nsub z k (dAPP t u) = dAPP (nsub z k t) (nsub z k u)
nsub z k (dABS t)   = dABS (nsub (lift z 0) (k + 1) t)
```

Fig. 2. Functions from Nipkow's formalisation of de Bruijn terms [11]

Second is substitution, which we call `nsub` here.[1] In order to make their proofs simpler, Nipkow, Huet and Shankar all define a notion of substitution that simultaneously performs a substitution and decrements the indices of the other free variables that appear in the term in which the substitution is occurring. This reflects the fact that the substitution is intended to be used for defining β-contraction, where a `dABS` constructor disappears in the process. As usual, the considered β-reduction relation is given as β-contraction plus congruence rules.

Definition 1 (Nipkow [11]). *$\to_d$ is the least relation satisfying the following rules:*

$$\frac{}{\mathtt{dAPP}\ (\mathtt{dABS}\ t)\ u \;\to_d\; \mathtt{nsub}\ u\ 0\ t}$$

$$\frac{t \;\to_d\; u}{\mathtt{dAPP}\ t\ z \;\to_d\; \mathtt{dAPP}\ u\ z} \qquad \frac{t \;\to_d\; u}{\mathtt{dAPP}\ z\ t \;\to_d\; \mathtt{dAPP}\ z\ u}$$

$$\frac{t \;\to_d\; u}{\mathtt{dABS}\ t \;\to_d\; \mathtt{dABS}\ u}$$

[1] We have altered the order of arguments to be compatible with the other mechanisations: the value being substituted comes first, followed by the index being substituted out, followed by the term in which the substitution is occurring.

2.3 Existing Results

We say that two types of the form we consider, e.g., $\langle X, \rightarrow_x \rangle$, $\langle Y, \rightarrow_y \rangle$, are isomorphic if there exists a bijection, $f : X \longrightarrow Y$, that is homomorphic

$$\forall x_1, x_2 \,.\, x_1 \rightarrow_x x_2 \Leftrightarrow f(x_1) \rightarrow_y f(x_2) \tag{3}$$

In the case of α-quotients, the mappings between the types are surjective and have the same kernels. By the appropriate adaptation of the First Isomorphism Theorem of Algebra,[2] this means that the various range types of each "raw" type are pairwise isomorphic. Once it has been established that the α-equivalences of λ^V and λ^C coincide, this means that the isomorphism of λ^V and λ^C implies that all the range types, i.e., dB and the λ^X_α, are pairwise isomorphic. (In [13], we further show that a broad class of properties is also shared between the elements of the "raw" syntax and the range types.)

As noted, the λ^V-to-λ^V_α and λ^C-to-λ^{Hi}_α α-quotients of the figure are underpinned by an epimorphism (i.e., (3) plus surjectivity) by construction [7,17]. In the remaining "raw-to-quotient" cases, property (3) plus surjectivity have been verified by Norrish [12] (λ^V-to-λ^G_α, λ^V-to-λ^N_α) and Homeier [8] (λ^C-to-λ^{Ho}_α, implicitly).

Vestergaard has mechanised the isomorphism at the base of the figure, between λ^V and λ^C (including their respective α-relations), in Coq. This follows the proof in Vestergaard [17].

The isomorphism at the named α-equivalent abstractions level between λ^G_α and λ^N_α is between the Gordon-Melham terms and the terms formed by quotienting the raw syntax (Λ^{var}). This isomorphism is uninteresting (and implicitly demonstrated in [12]) because the constructors and definitions in each type are immediate images of each other.

The isomorphism between dB_2 and λ^G_α is between the well-formed terms in Gordon's two-sorted de Bruijn type [5], and the terms of Gordon and Melham's subsequent paper [6]. This connection is a bijection by construction: the latter type is established using the HOL principle of type definition, which permits new types to be defined isomorphic to non-empty subsets of old types. Further, the bijection is behaviour-respecting because substitution in the new type is defined by following the bijection back to the old type, performing the substitution there and then returning.

The result most similar to that in this paper is Shankar's link between dB and λ^V [15]. (Strictly speaking Shankar's λ^V is not quite the same as the one used by Vestergaard [17]: where Vestergaard's substitution will go wrong, it returns the term being substituted into unchanged; in the same situation, Shankar returns the result of a recursive descent with free variables incorrectly captured. In neither case does this detail matter: references to both substitution functions are set about with side conditions ensuring they are applied only in situations that make sense.)

Shankar represents the bijection between the two types with a function from λ^V to dB (called `TRANSLATE`) and a function `UNTRANS` for the opposite direction.

[2] The result has been verified in Coq by Vestergaard.

Neither of these can work directly on the values from the types, but need to be accompanied by contextual information. For example, the statement that `TRANSLATE` undoes an `UNTRANS` is given as

```
117. Theorem.  UNTRANS-TRANS (rewrite):
     (IMPLIES (AND (TERMP X) (LEQ (FIND-M X N) M))
              (EQUAL (TRANSLATE (UNTRANS X M N)
                                (BINDINGS M N))
                     X))
```

The other identity is even more complicated, partly because the equality is actually α-equivalence. The subsequent statements that β-reductions in each type match up are similarly complex. (They are used to good effect: the Church-Rosser result for *dB* is transferred back to λ^V.)

As we shall see, our own directly-established connection can be expressed more naturally because we build an isomorphism to λ^N_α instead of λ^V. We also provide an isomorphism result for η-reduction (in Section 4.4).

3 Nominal Reasoning

In this section we provide a brief introduction to the nominal technology that has an important rôle to play in the isomorphism proofs to come. (For more on this, see for example, Pitts [14].)

Permutations. Nominal reasoning proceeds over sets that support a permutation action satisfying certain conditions. We call such sets, *nominal sets*.[3] At the "base level" permutations are defined in terms of their effect on strings.

Definition 2 (Permutations Applied to Strings). *A permutation (written π, π_1 etc) is a bijection on strings that is the identity on all but finitely many strings. We represent a permutation as a list of pairs of strings, and define the action of a permutation π on a string s (written $\pi(s)$) by primitive recursion on the structure of the list:*

$$\begin{aligned} [\,]\,(s) &= s \\ ((x,y) :: \pi)\,(s) &= \mathit{if}\ x = \pi(s)\ \mathit{then}\ y\ \mathit{else\ if}\ y = \pi(s)\ \mathit{then}\ x \\ &\qquad \mathit{else}\ \pi(s) \end{aligned}$$

If permutation π is the singleton list $[(x,y)]$, then π is also written $(x\,y)$.

For a type other than strings to be a nominal set, the type must support a permutation action (written $\pi \cdot x$, for x in the putative nominal set) satisfying the following conditions:

$$\begin{aligned} &[\,] \cdot t = t \\ &\pi_1 \cdot (\pi_2 \cdot t) = (\pi_1 \circ \pi_2) \cdot t \\ &(\forall s.\ \pi_1(s) = \pi_2(s)) \ \Rightarrow\ \pi_1 \cdot t = \pi_2 \cdot t \end{aligned} \tag{4}$$

[3] Pitts only calls sets nominal if each element also has *finite support*, for which see below.

As our permutations are actually lists, permutation composition ($\pi_1 \circ \pi_2$ in the second condition) is actually list concatenation. This is also the reason why the third condition tests for extensional equality (over strings) on the permutations π_1 and π_2.

Examples

- The natural numbers and the booleans are both nominal sets, with a permutation action that is everywhere the identity.
- If A is a nominal set, then so too are lists of values drawn from A. The permutation action over a list value is the permutation of each element.
- If A and B are nominal sets, then so too are functions from A to B. The permutation action is defined to be

$$\pi \cdot (f : A \to B) \;=\; \lambda(x : A).\;\; \pi \cdot (f\ (\pi^{-1} \cdot x))$$

  It follows that $\pi \cdot (f(x)) = (\pi \cdot f)(\pi \cdot x)$

Support. With a permutation action defined, a nominal set immediately gives rise to a notion of "support".

Definition 3. *We say that the set A* supports *a value t if*

$$\forall x\, y \notin A.\ (x\, y) \cdot t = t$$

If there is a finite set that supports a value t, then there is a smallest such supporting set, which we refer to as the *support of t (written* $\mathsf{supp}(t)$*). For types such as λ-calculus terms, support corresponds to the free variables of the term.*

When a value has empty support, it follows that $(x\, y) \cdot t = t$ for all possible x and y. When the value is a function, this gives us a nice result about the distribution of permutations over applications:

$$(x\, y) \cdot (f(v)) \;=\; ((x\, y) \cdot f)((x\, y) \cdot v) \;=\; f((x\, y) \cdot v)$$

This result extends to arbitrary permutations, so that $\pi \cdot f(x) = f(\pi \cdot x)$. Better yet, if the function is actually a relation or predicate (a function whose range is the boolean type), then the outer π disappears and we have that $R\ (\pi \cdot x_1) \ldots (\pi \cdot x_n) \Leftrightarrow R\ x_1 \ldots x_n$. Such a predicate or relation is said to be *equivariant*.

4 Isomorphisms and De Bruijn Terms

Our goal is to show that `dB` is isomorphic to λ_α^N, *i.e.*, a formalism where reductions occur between α-equivalence classes of the raw syntax. The mechanisation was performed by Norrish using the HOL4 proof assistant.

At the base level, a bijection between the terms of the quotiented λ-calculus (Λ_α) and `dB` requires a bijection between strings and natural numbers because

the existing mechanisation of Λ_α uses strings for its variables. Though occasionally awkward, the fact that the two sorts of variables are of different types is a useful sanity check when proofs are being performed. In what follows, s, s', v and v' will range over strings, and i, j and k over natural numbers. The bijections between the types will be written with a hat, so that $\hat{i}$ is the string corresponding to number i, while $\hat{s}$ is the number corresponding to string s.

4.1 An Intermediate Formalism: dB'

We define dB' to match λ_α^N, both in its construction and in its reduction relations. In particular, abstraction in dB' will involve an identifier in the style of "$\lambda 2.2$" (for the *identity* function). This new constructor is necessarily not injective (as, otherwise, we would distinguish, e.g., "$\lambda 2.2$" and "$\lambda 1.1$"). Abstraction in λ_α^N is also not injective but, whereas the λ_α^N-construction is based on a quotient of Λ^{var}, the non-injectivity of the new λ-constructor in dB' follows directly from its definition as syntactic sugar for the existing dABS constructor in dB. Though a separate oval in Figure 1, dB' is based on the same underlying HOL type as dB. The difference arises from its use of different constructors to "cover" the type, and a different reduction relation.

We begin by defining dB-substitution without the index-decrementing used in nsub. We shall see that this function is isomorphic to substitution over Λ_α, while nsub has no natural reading in Λ_α.

Definition 4 (A clean notion of substitution for dB).

$$\begin{aligned}
\mathtt{sub}\ z\ k\ (\mathtt{dV}\ i) &= \mathsf{if}\ i = k\ \mathsf{then}\ z\ \mathsf{else}\ \mathtt{dV}\ i \\
\mathtt{sub}\ z\ k\ (\mathtt{dAPP}\ t\ u) &= \mathtt{dAPP}\ (\mathtt{sub}\ z\ k\ t)\ (\mathtt{sub}\ z\ k\ u) \\
\mathtt{sub}\ z\ k\ (\mathtt{dABS}\ t) &= \mathtt{dABS}\ (\mathtt{sub}\ (\mathtt{lift}\ z\ 0)\ (k+1)\ t)
\end{aligned}$$

The (non-injective) dLAM-function that performs abstraction over a particular index can now be defined as follows.

Definition 5 (Abstraction over an index).

$$\mathtt{dLAM}\ i\ t = \mathtt{dABS}\ (\mathtt{sub}\ (\mathtt{dV}\ 0)\ (i+1)\ (\mathtt{lift}\ t\ 0))$$

All dABS terms can be seen as abstractions over an arbitrary fresh index.

Lemma 6 (dABS terms as dLAM terms). *If i is not among the free indices of* dABS t*, then there exists a t_0 such that* dABS $t =$ dLAM $i\ t_0$*. By the definition of* dLAM *and the injectivity of* dABS *we further know that*

$$t = \mathtt{sub}\ (\mathtt{dV}\ 0)\ (i+1)\ (\mathtt{lift}\ t_0\ 0)$$

Proof. Prove by structural induction on t that there exists a t_0 such that $t =$ sub (dV n) i (lift t_0 n), where i is again suitably fresh. □

Now we define $\rightarrow_{\mathtt{d}'}$, the β-reduction relation for dB'. It mimics the traditional statement in the λ-calculus for both contraction and congruence rules.

Definition 7 (dB' β-reduction).

$$\frac{}{\mathtt{dAPP}\ (\mathtt{dLAM}\ i\ t)\ u\ \rightarrow_{\mathtt{d}'}\ \mathtt{sub}\ u\ i\ t}$$

$$\frac{t\ \rightarrow_{\mathtt{d}'}\ u}{\mathtt{dAPP}\ t\ z\ \rightarrow_{\mathtt{d}'}\ \mathtt{dAPP}\ u\ z} \qquad \frac{t\ \rightarrow_{\mathtt{d}'}\ u}{\mathtt{dAPP}\ z\ t\ \rightarrow_{\mathtt{d}'}\ \mathtt{dAPP}\ z\ u}$$

$$\frac{t\ \rightarrow_{\mathtt{d}'}\ u}{\mathtt{dLAM}\ i\ t\ \rightarrow_{\mathtt{d}'}\ \mathtt{dLAM}\ i\ u}$$

4.2 dB' *vs* $\boldsymbol{\lambda^N_\alpha}$: Nominal Reasoning with Indices

Thus far we have been working entirely with the dB type, where functions are trivial to define by primitive recursion. Now we are faced with a choice: to define a function from dB into Λ_α, or *vice versa*. The former choice is less appealing because a choice of arbitrary fresh name would need to be made in the recursive dABS-clause. One might imagine writing something like

$$f\ (\mathtt{dABS}\ t) = \mathtt{LAM}\ (\mathsf{fresh}(t))\ (f\ (\mathtt{sub}\ (\mathsf{fresh}(t))\ 0\ t))$$

where $\mathsf{fresh}(t)$ generates an index fresh for t. But this clause is not primitive recursive, so one would, e.g., have to write an f that took an additional environment parameter specifying how the bound variables encountered so far should be replaced. Instead, we take as our objective a bijective $f : \Lambda_\alpha \rightarrow \mathtt{dB}$, satisfying the following equations

$$\begin{aligned} f\ (\mathtt{VAR}\ v) &= \mathtt{dV}\ \hat{v} \\ f\ (\mathtt{APP}\ M\ N) &= \mathtt{dAPP}\ (f\ M)\ (f\ N) \\ f\ (\mathtt{LAM}\ v\ M) &= \mathtt{dLAM}\ \hat{v}\ (f\ M) \end{aligned} \tag{5}$$

The existence of such an f is not immediate because Λ_α is not a free algebra (LAM is not injective). However, suitable recursion principles for this type do exist, as shown in Pitts [14].

de Bruijn Terms are a Nominal Set. In order to use recursion principles for Λ_α, it remains to be shown that the range of the desired function is a *nominal set*, as in Section 3. A subtlety derives from the fact that we (necessarily must) send free variables to suitably free indices. Permutation on free variables therefore must be reflected on free indices. Conversely, bound indices are what they are and must be left untouched.

Thus we are required to define a permutation action for de Bruijn terms, and it can not be a trivial one (everywhere the identity, say). Inevitably, the complexity arises in the dABS clause of the definition. When a permutation passes through an abstraction, the free variables it is acting on are now at different index positions, which means that the permutation has to be similarly adjusted.

Definition 8 (Permutation for dB). *Let inc ("increment") over permutations be*

$$\begin{aligned} inc\ [\,] &= [\,] \\ inc\ ((s_1, s_2) :: tl) &= (\widehat{s_1 + 1}, \widehat{s_2 + 1}) :: inc\ tl \end{aligned}$$

The action of a permutation π on a term in dB (written $\pi \cdot t$) is

$$\begin{aligned} \pi \cdot (\mathtt{dV}\ i) &= \mathtt{dV}\ (\widehat{\pi(\hat{i})}) \\ \pi \cdot (\mathtt{dAPP}\ t\ u) &= \mathtt{dAPP}\ (\pi \cdot t)\ (\pi \cdot u) \\ \pi \cdot (\mathtt{dABS}\ t) &= \mathtt{dABS}\ (inc(\pi) \cdot t) \end{aligned}$$

It is straightforward to show that this definition satisfies the requirements to make dB a nominal set. The support for a value in type dB is as one might expect: its set of free indices converted into strings *via* the hat-isomorphism.

Relating dLAM and Permutation. In order to admit (5), we must also characterise the relationship between dLAM (which appears on the RHS of (5)), and dB-permutation. As dLAM is defined in terms of lift and sub, analogous results are required of these constants.

We first characterise the relationship between lift and permutation. This requires the definition of an *inc* that is parameterised by the index below which variables should not be touched, with the already-defined *inc* being inc_0 and with inc_i being the obvious generalisation.

Lemma 9 (Permutations and lift).

$$\mathtt{lift}\ (\pi \cdot t)\ n = (inc_n(\pi)) \cdot (\mathtt{lift}\ t\ n)$$

Proof. By induction on the structure of t. The result first needs

$$inc_0\ (inc_n\ \pi) = inc_{n+1}(inc_0\ \pi)$$

to be shown. This latter is just the sort of tedious result that the POPLmark authors inveigh against so convincingly in [1]. (Note that it is also an equality on the representing lists of pairs, not just extensionally in terms of the permutations' effect on strings.) □

This lemma leads to important results about sub, dLAM and $\rightarrow_{\mathtt{d}'}$:

Theorem 10 (sub and dLAM have empty support; $\rightarrow_{\mathtt{d}'}$ is equivariant).

$$\begin{aligned} \pi \cdot (\mathtt{sub}\ t\ j\ u) &= \mathtt{sub}\ (\pi \cdot t)\ (\widehat{\pi(\hat{j})})\ (\pi \cdot u) \\ \pi \cdot (\mathtt{dLAM}\ i\ t) &= \mathtt{dLAM}\ (\widehat{\pi(\hat{i})})\ (\pi \cdot t) \\ \pi \cdot t\ \rightarrow_{\mathtt{d}'}\ \pi \cdot u &\Leftrightarrow t\ \rightarrow_{\mathtt{d}'}\ u \end{aligned}$$

See Lemma 17 below for the contrast with nsub.

Proof. The substitution result follows from a structural induction on t. The result for dLAM is then immediate. The result for $\rightarrow_{\mathtt{d}'}$ follows from a rule induction showing that $t \rightarrow_{\mathtt{d}'} u \Rightarrow \pi \cdot t \rightarrow_{\mathtt{d}'} \pi \cdot u$. The fact that all permutations have inverses gives the other direction of the if-and-only-if. □

Theorem 11 (dB is in bijection with Λ_α). *There exists a bijective $f : \Lambda_\alpha \rightarrow$ dB with defining equations as in (5).*

Proof. The existence of f now follows once dB is established as a nominal set, and it is shown that

$$\mathtt{supp}(\mathtt{dLAM}\ i\ t) = \mathtt{supp}(t) \backslash \{\hat{\imath}\}$$

This in turn relies on results specifying the free variables (or support) of sub and lift (the latter is from Nipkow).

Injectivity of f follows from a structural induction, and standard properties of substitution. Surjectivity of f follows from a complete induction on the size of the dB term onto which a value in Λ_α maps.[4] □

The Homomorphism between dB' and $\boldsymbol{\lambda^N_\alpha}$. Given its construction, it is not surprising that dB with relation $\rightarrow_{\mathtt{d}'}$ is isomorphic to Λ_α and relation $\rightarrow_\beta$

Theorem 12.

$$M \rightarrow_\beta N \ \Leftrightarrow\ f(M) \rightarrow_{\mathtt{d}'} f(N)$$

Proof. Two straightforward rule inductions, one for each direction. Both results rely on the fact that

$$f\ (M[v := N]) \ =\ \mathtt{sub}\ (f\ N)\ \hat{v}\ (f\ M)$$

which is proved by "BVC-compatible" structural induction (see [12]) on M. □

4.3 The Homomorphism Between dB and dB'

Now we must grasp the nettle, and get to grips with Nipkow's definition of $\rightarrow_{\mathtt{d}}$, and the accompanying nsub function. We will establish that $\rightarrow_{\mathtt{d}}$ and $\rightarrow_{\mathtt{d}'}$ are in fact the same relation, giving us the identity as a homomorphism between dB and dB'.

We begin by proving the following:

Lemma 13.

$$\begin{array}{l} n \leq i \ \Rightarrow \\ \quad \mathtt{sub}\ u\ i\ t = \mathtt{nsub}\ u\ n\ (\mathtt{sub}\ (\mathtt{dV}\ n)\ (i+1)\ (\mathtt{lift}\ t\ n)) \end{array}$$

Proof. By structural induction on t. □

[4] Complete induction allows us to assume the induction hypothesis for all smaller terms, not just those of size one smaller.

Using this, we can show that

$$\texttt{dAPP}\ (\texttt{dLAM}\ i\ t)\ u \quad \rightarrow_{\texttt{d}} \quad \texttt{sub}\ u\ i\ t$$

which is to say that Nipkow's relation $\rightarrow_{\texttt{d}}$ does indeed reduce $\rightarrow_{\texttt{d}'}$-redexes in the correct way. If we expand the definition of `dLAM` on the LHS, we see that the redex is

$$\texttt{dAPP}\ (\texttt{dABS}\ (\texttt{sub}\ (\texttt{dV}\ 0)\ (i+1)\ (\texttt{lift}\ t\ 0))\ u)$$

Using Lemma 13 with $n = 0$ we get the desired identity. The same lemma is used to show the "converse" relation, that

$$\texttt{dAPP}\ (\texttt{dABS}\ t)\ u \quad \rightarrow_{\texttt{d}'} \quad \texttt{nsub}\ u\ 0\ t$$

The congruence rules for `dAPP` are the same in both relations, so our remaining task is to show that the different congruence rules for `dABS` and `dLAM` match up.

Abstraction Congruences Correspond. One of the two correspondences is to show that

$$t \rightarrow_{\texttt{d}'} u \quad \Rightarrow \quad \texttt{dABS}\ t \rightarrow_{\texttt{d}'} \texttt{dABS}\ u$$

In order to apply the abstraction rule for $\rightarrow_{\texttt{d}'}$, we must show that the `dABS` terms correspond to `dLAM` terms. Using Lemma 6, we can pick a suitably fresh i, and show the existence of t_0 and u_0 such that

$$\begin{aligned} \texttt{dABS}\ t &= \texttt{dLAM}\ i\ t_0 \\ \texttt{dABS}\ u &= \texttt{dLAM}\ i\ u_0 \\ t &= \texttt{sub}\ (\texttt{dV}\ 0)\ (i+1)\ (\texttt{lift}\ t_0\ 0) \\ u &= \texttt{sub}\ (\texttt{dV}\ 0)\ (i+1)\ (\texttt{lift}\ u_0\ 0) \end{aligned}$$

We have $t \rightarrow_{\texttt{d}'} u$, and want $t_0 \rightarrow_{\texttt{d}'} u_0$, from which the abstraction rule for $\rightarrow_{\texttt{d}'}$ would give us $\texttt{dABS}\,t \ \rightarrow_{\texttt{d}'}\ \texttt{dABS}\,u$. The problem here is that our assumption is about a large, complicated term, and we want to conclude that the reduction occurs between sub-terms (t_0 and u_0). Standard proofs by rule induction would handle the reverse situation well (results of the form $t \rightarrow u \Rightarrow \texttt{sub}\ w\ i\ t \rightarrow \texttt{sub}\ w\ i\ u$), but our goal looks like the converse.

The Nominal "Aha!". The solution to our problem is to realise that both the substitution and the lifting that occur in terms t and u are no more than variable permutations, and that they are in fact the same permutation applied to t_0 and u_0. Then, we will express our assumption $t \rightarrow_{\texttt{d}'} u$ as $\pi \cdot t_0 \rightarrow_{\texttt{d}'} \pi \cdot u_0$ for some permutation π. Because $\rightarrow_{\texttt{d}'}$ is equivariant (Theorem 10), this gives $t_0 \rightarrow_{\texttt{d}'} u_0$ as desired.

That a substitution of a fresh variable into a term is a permutation is an easy induction. The interest comes in showing that a lifting is also a permutation. First we define a function *lift-pm* that constructs a "lifting permutation". Given a lower bound ℓ and an upper bound m, *lift-pm*(ℓ, m) constructs the permutation that maps $\hat{\imath}$ to $\widehat{i+1}$ if $\ell \leq i \leq m$, maps $\widehat{m+1}$ back to $\hat{\ell}$, and preserves the values of all other arguments.

Lemma 14 (Lifting is a permutation). *If n is at least as large as the largest free index in t, then*

$$\texttt{lift}\ t\ m = (\mathit{lift\text{-}pm}(m, n)) \cdot t$$

Proof. By structural induction on t. The `dABS` case requires another tedious result about index-maneuvering, namely that

$$\mathit{inc}_0(\mathit{lift\text{-}pm}(m, n)) = \mathit{lift\text{-}pm}(m + 1, n + 1)$$

Thankfully, the result does not need to be generalised to cover the behaviour of inc_i. □

We can now prove that abstraction congruences for any equivariant reduction relation must correspond:

Theorem 15 (Abstraction Congruences Correspond). *If R is equivariant, that is $R\ (\pi \cdot t)\ (\pi \cdot u) \Leftrightarrow R\ t\ u$, then*

$$\frac{R\ t\ u}{R\ (\texttt{dABS}\ t)\ (\texttt{dABS}\ u)} \quad \Leftrightarrow \quad \frac{R\ t\ u}{R\ (\texttt{dLAM}\ i\ t)\ (\texttt{dLAM}\ i\ u)}$$

where one can read each inference rule as an implication universally quantified (over t, u and i) at the top level. The generality of the result is useful when we consider η-reduction.

Proof. Immediate, given Lemma 14, and Lemma 6. □

Lemma 16 ($\rightarrow_{\texttt{d}} \subseteq \rightarrow_{\texttt{d'}}$).

$$t \rightarrow_{\texttt{d}} u \ \Rightarrow t \rightarrow_{\texttt{d'}} u$$

Proof. A rule induction over the definition of $\rightarrow_{\texttt{d}}$. The redex case is handled by Lemma 13 (as already discussed). The `dAPP` congruence cases are immediate; the `dABS` congruence case is as just discussed (by Lemma 14, and Theorem 10). □

The Other Inclusion. Showing $\rightarrow_{\texttt{d'}} \subseteq \rightarrow_{\texttt{d}}$, we know that $\rightarrow_{\texttt{d}}$ can perform both $\rightarrow_{\texttt{d'}}$'s redex-reductions (from Lemma 13), and the application congruences. To show the abstraction congruence case, we need to show that $\rightarrow_{\texttt{d}}$ is equivariant in order to apply Theorem 15.

But the definition of $\rightarrow_{\texttt{d}}$ involves `nsub`, so we have to characterise how `nsub` behaves when it interacts with a permutation. This prompts a rather incomprehensible lemma (note the odd way in which the permutation π does not affect the index i):

Lemma 17.

$$\pi \cdot (\texttt{nsub}\ t\ i\ u) = \texttt{nsub}\ (\pi \cdot t)\ i\ (\mathit{inc}_i(\pi) \cdot u)$$

Proof. By structural induction on u. The $\mathtt{dV}$ case requires a number of splits on the ordering possibilities ($<$, $=$, $>$) between the index of the variable and i. The $\mathtt{dABS}$ case uses Lemma 9. □

Now we can conclude that $\rightarrow_{\mathtt{d}}$ is equivariant (by rule induction), and follow immediately with

Lemma 18 ($\rightarrow_{\mathtt{d}'} \subseteq \rightarrow_{\mathtt{d}}$).

$$t \rightarrow_{\mathtt{d}'} u \;\Rightarrow\; t \rightarrow_{\mathtt{d}} u$$

Theorem 19 (Relations $\rightarrow_{\mathtt{d}}$ and $\rightarrow_{\mathtt{d}'}$ Coincide).

$$t \rightarrow_{\mathtt{d}'} u \;\Leftrightarrow\; t \rightarrow_{\mathtt{d}} u$$

Proof. Combining Lemmas 16 and 18. □

Theorem 20 (Isomorphism). *The types dB and λ_α^N, with their respective reduction relations, $\rightarrow_{\mathtt{d}}$ and $\rightarrow_\beta$, are isomorphic, as witnessed by the bijective f:*

$$M \rightarrow_\beta N \;\Leftrightarrow\; f(M) \rightarrow_{\mathtt{d}} f(N)$$

Proof. From Theorems 12 and 19. □

4.4 Bonus: η-Reduction

Nipkow [11] also defines a notion of η-reduction for his terms. The congruence rules are standard. Writing $\rightarrow_{\mathtt{ne}}$ for "Nipkow's η", the redex rule is

$$\frac{0 \notin \mathtt{dFV}(t)}{\mathtt{dABS}\ (\mathtt{dAPP}\ t\ (\mathtt{dV}\ 0)) \;\;\rightarrow_{\mathtt{ne}}\;\; \mathtt{nsub}\ u\ 0\ t}$$

where the choice of u is immaterial. (Because 0 is not a free index of t, $\mathtt{nsub}$ is used only to decrement the free indices of t, which is required as t moves out from underneath an abstraction.)

Using exactly the same techniques as before, it is easy to define a notion of reduction that better emulates the definition in the λ-calculus. This relation (written $\rightarrow_{\mathtt{e}}$), has contraction rule

$$\frac{i \notin \mathtt{dFV}(t)}{\mathtt{dLAM}\ i\ (\mathtt{dAPP}\ t\ (\mathtt{dV}\ i)) \;\;\rightarrow_{\mathtt{e}}\;\; t}$$

It is straightforward to show that $\rightarrow_{\mathtt{e}}$ corresponds to the η-reduction rule in Λ_α. Thanks to Theorem 15, showing that $\rightarrow_{\mathtt{ne}}$ and $\rightarrow_{\mathtt{e}}$ correspond only requires us to show the equivalence of the redex rules, and that $\rightarrow_{\mathtt{ne}}$ and $\rightarrow_{\mathtt{e}}$ are equivariant. These proofs are mainly straightforward. We use some of the lemmas stated in Nipkow, among them

$$\mathtt{nsub}\ u\ n\ (\mathtt{lift}\ t\ n) = t$$

We also state and prove the similar

$$n \notin \mathtt{dFV}(t) \;\Rightarrow\; \mathtt{lift}\ (\mathtt{nsub}\ u\ n\ t)\ n = t$$

5 Conclusion

We have proved what everyone knew (or hoped) to be true from the outset. However, the proofs we have described are not entirely trivial, suggesting that they were indeed worth doing. In fact, the proofs are only made in the least palatable by using insights from the theory of nominal sets.

This theory helps in two important ways: it makes it possible to define the bijection f from Λ_α to `dB` in a natural style; and it helps us perform many of the resulting proofs. In particular, the proofs are helped greatly by the realisation that `lift` is a permutation. To best exploit this, we needed to prove that our reduction relations $\rightarrow_{\mathtt{d}}$ and $\rightarrow_{\mathtt{d}'}$ were equivariant, which in turn prompted the curious Lemma 17.

We hope that this work encourages others to extend the web of equivalence results in Figure 1, which we believe to be a valuable summary of existing work in its own right. In the near future, we hope to consider, for example, the two-sorted calculus of McKinna and Pollack [10], and the weak HOAS style construction of the λ-calculus in Urban and Tasson [16].

Finally, we hope this work will go some way towards bringing about the day when reasoning about de Bruijn terms is never again necessary.

Acknowledgements. NICTA is funded by the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council. We would like to thank James Cheney for valuable discussions in this area, and comments on this paper in particular.

Availability. The sources mechanising this paper's proofs are distributed as part of the HOL4 system, available from `http://hol.sourceforge.net`. (The proofs for β are part of the latest public release (January 2007); the η proofs are in the publicly accessible CVS repository.)

References

1. Aydemir, B.E., Bohannon, A., Fairbairn, M., Nathan Foster, J., Pierce, B.C., Sewell, P., Vytiniotis, D., Washburn, G., Weirich, S., Zdancewic, S.: Mechanized metatheory for the masses: the POPLMARK. In: Hurd, J., Melham, T. (eds.) TPHOLs 2005. LNCS, vol. 3603, Springer, Heidelberg (2005)
2. Barendregt, H.P.: The Lambda Calculus: its Syntax and Semantics. Studies in Logic and the Foundations of Mathematics, vol. 103. revised edn. Elsevier, Amsterdam (1984)
3. Curry, H.B., Feys, R.: Combinatory Logic. North-Holland, Amsterdam (1958)
4. de Bruijn, N.G.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem. Indag. Math. 34, 381–392 (1972)
5. Gordon, A.D.: A mechanisation of name-carrying syntax up to alpha-conversion. In: Joyce, J.J., Seger, C.-J.H. (eds.) HUG 1993. LNCS, vol. 780, pp. 413–425. Springer, Heidelberg (1994)

6. Gordon, A.D., Melham, T.: Five axioms of alpha conversion. In: von Wright, J., Harrison, J., Grundy, J. (eds.) TPHOLs 1996. LNCS, vol. 1125, pp. 173–190. Springer, Heidelberg (1996)
7. Roger Hindley, J.: The Church-Rosser Property and a Result in Combinatory Logic. PhD thesis, University of Newcastle upon Tyne (1964)
8. 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. LNCS, vol. 2152, pp. 207–222. Springer, Heidelberg (2001)
9. Huet, G.P.: Residual theory in lambda-calculus: a formal development. Journal of Functional Programming 4(3), 371–394 (1994)
10. McKinna, J., Pollack, R.: Some lambda calculus and type theory formalized. Journal of Automated Reasoning 23(3–4), 373–409 (1999)
11. Nipkow, T.: More Church-Rosser proofs (in Isabelle/HOL). Journal of Automated Reasoning 26, 51–66 (2001)
12. Norrish, M.: Mechanising λ-calculus using a classical first order theory of terms with permutations. Higher-Order and Symbolic Computation 19, 169–195 (2006)
13. Norrish, M., Vestergaard, R.: Structural preservation and reflection of diagrams. Technical Report IS-RR-2006-011, JAIST (2006)
14. Pitts, A.M.: Alpha-structural recursion and induction. Journal of the ACM 53(3), 459–506 (2006)
15. Shankar, N.: A mechanical proof of the Church-Rosser theorem. Journal of the ACM 35(3), 475–522 (1988)
16. Urban, C., Tasson, C.: Nominal techniques in Isabelle/HOL. In: Nieuwenhuis, R. (ed.) Automated Deduction – CADE-20. LNCS (LNAI), vol. 3632, pp. 38–53. Springer, Heidelberg (2005)
17. Vestergaard, R.: The Primitive Proof Theory of the λ-Calculus. PhD thesis, School of Mathematical and Computer Sciences, Heriot-Watt University (2003)