

de Bruijn Terms Really Do Work

but then, you knew that anyway. . .

Michael Norrish¹ René Vestergaard²

¹NICTA

²JAIST

11 September 2007

Outline

① Introduction

Existing Results

Existing de Bruijn Term Mechanisation

② Giving de Bruijn Terms Named Abstraction

③ Nominal Insights

④ Conclusion

Thesis

To be confident in a rigorous mathematical model, we either

- show it satisfies some desired axiomatic characterisation; or
- show it isomorphic to another, known satisfactory, construction.

Thesis

To be confident in a rigorous mathematical model, we either

- show it satisfies some desired axiomatic characterisation; or
- show it isomorphic to another, known satisfactory, construction.

The former is easier (if a characterisation is to hand!) because only one model is constructed.

(Always the risk of “non-standard” models.)

The latter is painful (for mechanisers) because two models have to be established and compared.

The λ -calculus

A typical λ -calculus mechanisation proceeds:

- 1 Here is my type Λ , and here are the operations defined upon it (substitution, β , η . . .)
- 2 It looks right (or: looks like someone else's mechanisation)
- 3 Here are the expected results (substitution lemma, Church-Rosser . . .)

Step 2 might be an unmechanised isomorphism argument (“adequacy”).

Step 3 appeals to an implicit (and loose) axiomatic characterisation.

We Can Do Better

Though proving “new” results may be more fun,

- Modern technology is capable of mechanised isomorphism arguments
- In fact, mid-80s technology is capable (*cf* Shankar)
- Some modern constructions provide isomorphism results in passing

If a result is trivial, proving it won't hurt.

Contrariwise, if it's not pain-free, it was worth proving.

We Can Do Better

Though proving “new” results may be more fun,

- Modern technology is capable of mechanised isomorphism arguments
- In fact, mid-80s technology is capable (*cf* Shankar)
- Some modern constructions provide isomorphism results in passing

If a result is trivial, proving it won't hurt.

Contrariwise, if it's not pain-free, it was worth proving.

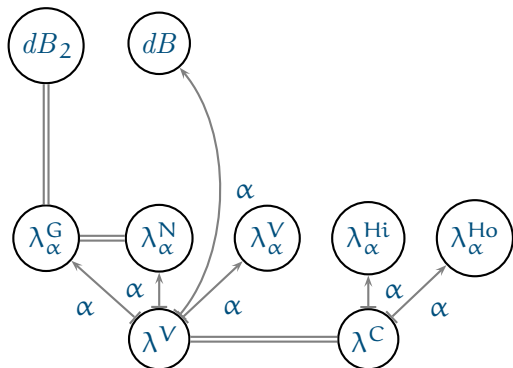
[Or, if it's not pain-free, your tool isn't up to it. . .]

Some Existing Results, Graphically

Nameless Abstractions (w/indices)

Named α -equivalent Abstractions

Named "Raw" Abstractions

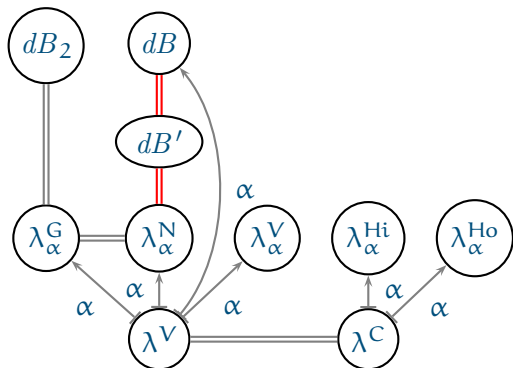


Some Existing Results, Graphically

Nameless Abstractions (w/indices)

Named α -equivalent Abstractions

Named "Raw" Abstractions



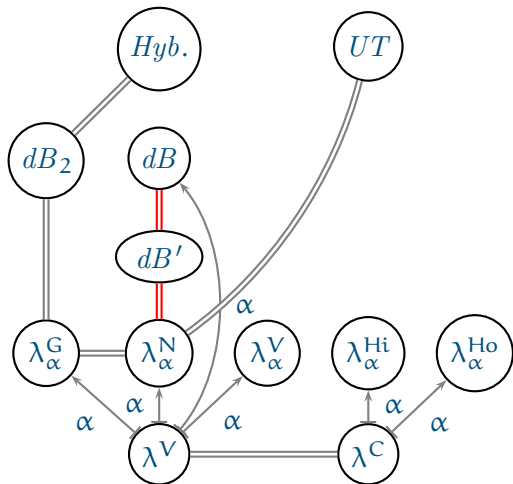
Some Existing Results, Graphically

“HOAS” style

Nameless Abstractions (w/indices)

Named α -equivalent Abstractions

Named “Raw” Abstractions



The Standard de Bruijn Mechanisation

Followed by Shankar (1988), Huet (1994), Nipkow (2001)

$$dB ::= dV \mathbb{N} \mid dAPP \ dB \ dB \mid dABS \ dB$$

```
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)
```

de Bruijn β -Reduction

$$\overline{\text{dAPP } (\text{dABS } t) u \rightarrow_d \text{ nsub } u \ 0 \ t}$$

$$\frac{t \rightarrow_d u}{\text{dAPP } t \ z \rightarrow_d \text{ dAPP } u \ z} \quad \frac{t \rightarrow_d u}{\text{dAPP } z \ t \rightarrow_d \text{ dAPP } z \ u}$$

$$\frac{t \rightarrow_d u}{\text{dABS } t \rightarrow_d \text{ dABS } u}$$

(Nipkow also did η -reduction; see paper for discussion.)

Named Abstractions for de Bruijn Terms

- First stage of an isomorphism is to show a bijection.
- dV and $dAPP$ have obvious counterparts in Λ_α
- We can create a named abstraction in dB to correspond to “normal” abstraction.

Want:

$$dLAM : \mathbb{N} \rightarrow dB \rightarrow dB$$

Note:

- Will also use existing isomorphism result between \mathbb{N} and *string*.
- $dLAM$ won't be injective, as desired

Sensible Substitution for de Bruijn Terms

In order to define `dLAM`, must first define a new version of substitution.

Old substitution:

```
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)
```

Sensible Substitution for de Bruijn Terms

In order to define `dLAM`, must first define a new version of substitution.

New substitution:

$$\text{sub } z \ k \ (\text{dV } i) \quad = \text{ if } i = k \text{ then } z \\ \quad \quad \quad \text{else } \text{dV } i$$
$$\text{sub } z \ k \ (\text{dAPP } t \ u) = \text{dAPP } (\text{sub } z \ k \ t) \ (\text{sub } z \ k \ u) \\ \text{sub } z \ k \ (\text{dABS } t) \quad = \text{dABS } (\text{sub } (\text{lift } z \ 0) \ (k + 1) \ t)$$

This version of substitution has much nicer properties than `nsub`.

Defining dLAM

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

To “abstract out” i from term t :

- lift t , freeing up index 0
- substitute 0 (the bound variable) in for $i + 1$
(which was i before the lift)
- apply dABS

(Also, prove that all dABS terms are dLAM terms.)

A Bijection From Λ_α

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

- \hat{v} is the natural number corresponding to string v
- Establish existence of f by showing that dB is a nominal set (has permutation action and finite support)
- f is injective and onto

An Alternative Definition of β -Reduction

$$\overline{\text{dAPP (dLAM i t) u} \rightarrow_{\text{d}}, \text{sub u i t}}$$

$$\frac{t \rightarrow_{\text{d}}, u}{\text{dAPP t z} \rightarrow_{\text{d}}, \text{dAPP u z}}$$

$$\frac{t \rightarrow_{\text{d}}, u}{\text{dAPP z t} \rightarrow_{\text{d}}, \text{dAPP z u}}$$

$$\frac{t \rightarrow_{\text{d}}, u}{\text{dLAM i t} \rightarrow_{\text{d}}, \text{dLAM i u}}$$

Trivial to show:

$$M \rightarrow_{\beta} N \Leftrightarrow f(M) \rightarrow_{\text{d}}, f(N)$$

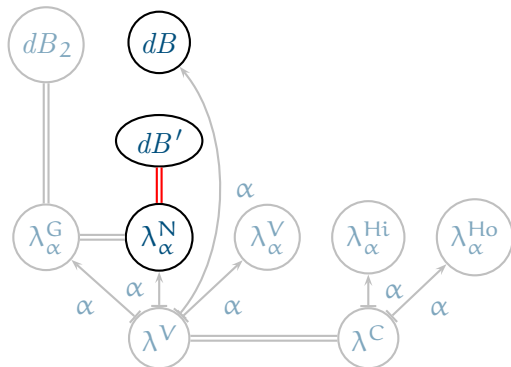
Status

Have that $(\Lambda_\alpha, \rightarrow_\beta)$ is isomorphic to (dB, \rightarrow_d) .

Nameless Abstractions (w/indices)

Named α -equivalent Abstractions

Named "Raw" Abstractions



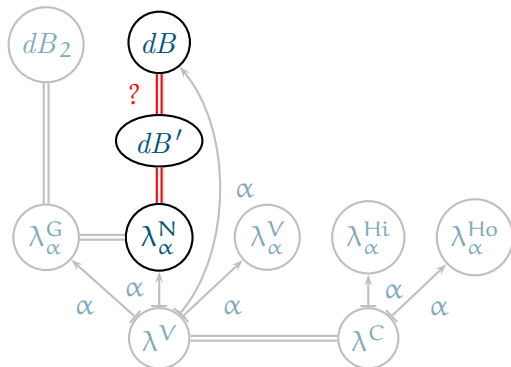
Status

Have that $(\Lambda_\alpha, \rightarrow_\beta)$ is isomorphic to (dB, \rightarrow_d) .

Nameless Abstractions (w/indices)

Named α -equivalent Abstractions

Named "Raw" Abstractions



Still need to show that (dB, \rightarrow_d) is isomorphic to (dB, \rightarrow_d) .

Proof Obligations

Abstraction Congruences Correspond:

$$t \rightarrow_d, u \Rightarrow \text{dABS } t \rightarrow_d, \text{dABS } u$$

$$t \rightarrow_d u \Rightarrow \text{dLAM } i \ t \rightarrow_d \text{dLAM } i \ u$$

Redex Rules Correspond:

$$\text{dAPP } (\text{dLAM } i \ t) \ u \rightarrow_d \text{sub } u \ i \ t$$

$$\text{dAPP } (\text{dABS } t) \ u \rightarrow_d, \text{nsub } u \ 0 \ t$$

Abstraction Congruence 1

Have:

$$\frac{t \rightarrow_d, u}{\text{dLAM } i \ t \rightarrow_d, \text{dLAM } i \ u}$$

Want: $t \rightarrow_d, u$ implies

$$\text{dABS } t \rightarrow_d, \text{dABS } u$$

Luckily, all **dABS** terms are also **dLAM** terms...

Abstraction Congruence 1

Have:

$$\frac{t \rightarrow_d, u}{\text{dLAM } i \ t \rightarrow_d, \text{dLAM } i \ u}$$

Want: $t \rightarrow_d, u$ implies

$$\text{dLAM } i \ t_0 \rightarrow_d, \text{dLAM } i \ u_0$$

where

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

Next, apply congruence rule for \rightarrow_d, \dots

Abstraction Congruence 1

Have:

$$\frac{t \rightarrow_d, u}{\text{dLAM } i \ t \rightarrow_d, \text{dLAM } i \ u}$$

Want: $t \rightarrow_d, u$ implies

$$t_0 \rightarrow_d, u_0$$

where

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

At this point, I felt a sinking feeling in my stomach

The Nominal “Aha!”

Need to show:

$$\begin{aligned} \text{sub } (dV \ 0) \ (i + 1) \ (\text{lift } t_0 \ 0) &\rightarrow_{d'} \text{sub } (dV \ 0) \ (i + 1) \ (\text{lift } u_0 \ 0) \\ &\Rightarrow \\ t_0 &\rightarrow_{d'} u_0 \end{aligned}$$

Realisation:

- Both `sub` and `lift` are permutations
(in case of `sub`, because `dV 0` is fresh for `lift t0 0`)
- The permutations on both sides of the reduction are the same

Permutations Make Life Easier

Goal becomes:

$$\pi \cdot t_0 \rightarrow_d \pi \cdot u_0 \quad \Rightarrow \quad t_0 \rightarrow_d u_0$$

for a permutation π equivalent to the composition of the `lift` and `sub`.

Permutations Make Life Easier

Goal becomes:

$$\pi \cdot t_0 \rightarrow_{d'} \pi \cdot u_0 \quad \Rightarrow \quad t_0 \rightarrow_{d'} u_0$$

for a permutation π equivalent to the composition of the `lift` and `sub`.

Permutations have inverses, so need only prove:

$$\forall t \ u \ \pi. \ t \rightarrow_{d'} \ u \quad \Rightarrow \quad \pi \cdot t \rightarrow_{d'} \ \pi \cdot u$$

(This is a one-liner: “*induct, then simplify*”.)

The Other Abstraction Congruence

Proof is similar, but need to show that \rightarrow_d is equivariant, that

$$t \rightarrow_d u \quad \Rightarrow \quad \pi \cdot t \rightarrow_d \pi \cdot u$$

This is complicated by

$$\frac{}{\text{dAPP } (\text{dABS } t) u \rightarrow_d \text{ nsub } u \ 0 \ t}$$

How do permutations interact with `nsub`?

A Curious Result

For `sub`, have:

$$\pi \cdot (\text{sub } t \ i \ u) = \text{sub } (\pi \cdot t) \ (\pi(i)) \ (\pi \cdot u)$$

(The sort of result one expects.)

A Curious Result

For `sub`, have:

$$\pi \cdot (\text{sub } t \ i \ u) = \text{sub } (\pi \cdot t) \ (\pi(i)) \ (\pi \cdot u)$$

(The sort of result one expects.)

For `nsub`, have:

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

(It surprised me.)

dB As a Waypoint to Λ_α

- One route to Λ_α is by performing a quotient of a raw syntax (easy with HOL4's quotient package)
- Urban's `nominal-isabelle` package goes *via* a “HOAS” type first
- Both constructions involve a redundant type

This work suggests we can start from a de Bruijn type.

- Only need one redundant constructor (`dABS`)
- Construction can allow standard definitions of β -reduction and substitution (no need for `nsub`)

Conclusion

- Today's result is strictly redundant given work by Shankar, and others
- The proof was fun, thanks to insights from the nominal framework

- Yes, de Bruijn terms really do work.

Thank You