Representing Binders: The nominal approach (in Isabelle/HOL)

Most info taken from C. Urban’s talk.
For more info check: http://isabelle.in.tum.de/nominal/
De Brujin indices

2 major flaws (from POPLMark challenge):

- Statements of theorems require complicated clauses involving “shifted” terms and contexts. These extra clauses make it difficult to see the correspondence between informal and formal versions of the same theorem.

- The notational clutter is manageable for “toy” examples of the size of the simply-typed lambda calculus, but becomes a heavy burden even fairly small languages like $F_{<:}$. 
Substitution Lemma: If \( x \not\equiv y \) and \( x \not\in \text{FV}(L) \), 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: \( M \equiv \lambda x.x \). The both sides are equal \( \lambda x.x \) in \( \lambda \)-calculus.

  2.1.12. **Convention:** Terms that are \( \alpha \)-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.

  2.1.14. **Moral:** Using conventions 2.1.12 and 2.1.13 one can work with \( \lambda \)-terms in the naive way.

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

- **Case 3:** \( M \equiv M_1 M_2 \). The statement follows again from the induction hypothesis.

“\( \square \)”
Nominal approach  

(From Urban’s talk)

Nominal Approach - in some sense no original idea: we will just formalise what happens on paper:

- we will work with alpha-equivalence classes (just in a named fashion),
- we will have something like the variable convention,
- we will choose fresh names,...
Atoms

Atoms: Everything that is **bound, binding** and **bindable** is an atom (independent from the language at hand)

**example integrals**

\[
\int_0^1 x^2 + y \, dx
\]

\(x\) is an atom—bound and binding
Swappings

In general, renaming substitutions do not respect $\alpha$-equivalence, e.g.

\[
\begin{align*}
[b := a] \lambda a. b &= \lambda a. a \\
[b := a] \lambda c. b &= \lambda c. a
\end{align*}
\]

\[(b a) \cdot t \overset{\text{def}}{=} \text{swap all occurrences of } b \text{ and } a \text{ in } t\]

be they bound, binding or bindable

Unlike for \([b := a](\_), \text{ for } (b a) \cdot (\_), \text{ we do have if } t =_\alpha t' \text{ then } (b a) \cdot t =_\alpha (b a) \cdot t'.\]
Permutations

- Extend ‘swappings’ to (finite) lists of swappings

\[(a_1 b_1) \ldots (a_n b_n),\]

so called Permutations.

For example:

\[
\pi = \begin{pmatrix}
  a & b \\
  b & a \\
  c & c
\end{pmatrix}
= (c b)(a b)(a c)
Permutations on Atoms

Permutations are **bijective** mappings from atoms to atoms.

A permutation acts on an atom as follows:

\[
[] \cdot a \overset{\text{def}}{=} a
\]

\[
((a_1 a_2) :: \pi) \cdot a \overset{\text{def}}{=} \begin{cases} 
  a_1 & \text{if } \pi \cdot a = a_2 \\
  a_2 & \text{if } \pi \cdot a = a_1 \\
  \pi \cdot a & \text{otherwise}
\end{cases}
\]

- [] stands for the empty list (the identity permutation), and
- \((a_1 a_2) :: \pi\) stands for the permutation \(\pi\) followed by the swapping \((a_1 a_2)\)
Permutations on Atoms

- The composition of two permutations is given by list-concatenation, written as $\pi' @ \pi$.
- The inverse of a permutation is given by list reversal, written as $\pi^{-1}$, and
- The disagreement set of two permutations $\pi$ and $\pi'$ is the set of atoms

$$ds(\pi, \pi') \overset{\text{def}}{=} \{ a \mid \pi \cdot a \neq \pi' \cdot a \}$$

- $\pi \sim \pi' \overset{\text{def}}{=} ds(\pi, \pi') = \emptyset$. 
Permutations: some properties

Here $a$, $b$ and $c$ are arbitrary atoms:

- $(b\ b) \cdot a = a$, $(b\ c) \cdot a = (c\ b) \cdot a$
- $\pi^{-1} \cdot (\pi \cdot a) = a$
- $\pi \cdot a = b$ if and only if $a = \pi^{-1} \cdot b$
- $\pi_1 @ \pi_2 \cdot a = \pi_1 \cdot (\pi_2 \cdot a)$
- $\pi \cdot ((b\ c) \cdot a) = (\pi \cdot b \ \pi \cdot c) \cdot (\pi \cdot a)$

Permutation on lambda-terms

$\pi \cdot (a) \quad \text{given by the action on atoms}$

$\pi \cdot (t_1 \ t_2) \overset{\text{def}}{=} (\pi \cdot t_1)(\pi \cdot t_2)$

$\pi \cdot (\lambda a.\ t) \overset{\text{def}}{=} \lambda(\pi \cdot a).(\pi \cdot t)$
Alpha-equivalence

Take one step back and consider “raw” lambda-terms. The following four rules define $\alpha$-equivalence:

- **$\approx_{\text{var}}$**
  \[ a \approx a \]

- **$\approx_{\text{app}}$**
  \[ t_1 \approx s_1 \quad t_2 \approx s_2 \quad t_1 \, t_2 \approx s_1 \, s_2 \]

- **$\approx_{\text{lam}_1}$**
  \[ t \approx s \quad \lambda a. \, t \approx \lambda a. \, s \]

- **$\approx_{\text{lam}_2}$**
  \[ t \approx (a \, b) \, s \quad a \neq s \quad \lambda a. \, t \approx \lambda b. \, s \]

Assuming $a \neq b$

\[ \lambda a. \, t \approx \lambda b. \, s \] iff $t$ is $\alpha$-equivalent with $s$ in which all occurrences of $b$ have been renamed to $a$. ...oops permuted to $a$. 
Support and Freshness \cite{Pitts2003}

The support of an object $x : \iota$ is a set of atoms $\alpha$:

$$\text{supp}_\alpha x \overset{\text{def}}{=} \{ a \mid \text{infinite}\{ b \mid (a \ b) \cdot x \neq x \} \}$$

This notion corresponds roughly to the usual notion of the set of free variables.

An atom is \textcolor{red}{fresh} for an $x$, if it is not in the support of $x$:

$$a \not\in x \overset{\text{def}}{=} a \not\in \text{supp}_\alpha(x)$$
Nominal datatype package

C.Urban, S.Berghofer, M.Wenzel

The nominal datatype package generates the \( \alpha \)-equivalence classes as a \textit{type} in Isabelle/HOL.

\begin{verbatim}
atom_decl name
nominal_datatype lam =
    Var "name"
    | App "lam" "lam"
    | Lam "<<name>>lam" ("Lam [_._.]" [100,100] 100)
\end{verbatim}

The type is defined so that we have \textit{equations} such as:

\[ \text{Lam} [a].(\text{Var} a) = \text{Lam} [b].(\text{Var} b) \]
Some successful stories

- “Formalising the pi-calculus using nominal logic”: J. Bengtson and J. Parrow
- “Crary’s Completeness proof for equivalence Checking”: C. Urban and J. Narboux
- “Nominal Formalisations of Typical SOS Proofs”: C. Urban and J. Narboux
- ........
- “Implementing Spi-Calculus using nominal techniques”: T. Kahsai and M. Miculan