Principles of Recursion and Induction for Nominal Lambda Calculus.

March 30, 2015

All the code shown is compiled in the last Agda's version 2.4.2.2 and 0.9 standard library, and can be fully accessed at:

https://github.com/ernius/formalmetatheory-nominal

1 Infrastructure

```
data Λ : Set where
  v     : Atom → Λ
  _·_   : Λ → Λ → Λ
  ƛ     : Atom → Λ → Λ
```

Figure 1: Terms

```
data _#_ (a : Atom) : Λ → Set where
  #v    : {b : Atom}          → b ≢ a              → a # v b
  #·    : {M N : Λ }          → a # M → a # N      → a # M · N
  #ƛ≡   : {M : Λ}                                  → a # ƛ a M
  #ƛ    : {b : Atom}{M : Λ}   → a # M              → a # ƛ b M

(_•_)_ : Atom → Atom → Atom → Atom
( a • b )ₐ c  with c ≟ₐ a
... | yes _           = b
... | no  _  with c ≟ₐ b
...          | yes _  = a
...          | no  _  = c

(_•_)_ : Atom → Atom → Λ → Λ
( a • b ) v c      = v (( a • b )ₐ c)
( a • b ) M · N    = (( a • b ) M) ·  (( a • b ) N)
( a • b ) ƛ c M    = ƛ (( a • b )ₐ c) (( a • b ) M)
```

```
_•ₐ_ : Π → Atom → Atom
π •ₐ a = foldr (λ s b → ( proj₁ s • proj₂ s )ₐ b) a π

_•_ : Π → Λ → Λ
π • M = foldr (λ s M → ( proj₁ s • proj₂ s ) M) M π

data _∼α_ : Λ → Λ → Set where
  ∼αv  : {a : Atom} → v a ∼α v a
  ∼α·  : {M M' N N' : Λ} → M ∼α M' → N ∼α N'
         → M · N ∼α M' · N'
  ∼αƛ  : {M N : Λ}{a b : Atom}(xs : List Atom)
         → ((c : Atom) → c ∉' xs → ( a • c ) M ∼α ( b • c ) N)
         → ƛ a M ∼α ƛ b N
```

2 Induction Principles

Primitive induction over terms presented in figure 1:

```
TermPrimInd : (P : Λ → Set)
  → (∀ a → P (v a))
  → (∀ M N → P M → P N → P (M · N))
  → (∀ M b → P M → P (ƛ b M))
  → ∀ M → P M
```

Figure 2: Primitive Induction

Next, we introduce another induction principle over terms with a stronger inductive hypothesis for the abstraction case, which says the property holds for all permutation of names of the body of the abstraction. This principle is proved using previous primitive induction principle.

```
TermIndPerm : (P : Λ → Set)
  → (∀ a → P (v a))
  → (∀ M N → P M → P N →  P (M · N))
  → (∀ M b → (∀ π → P (π • M)) → P (ƛ b M))
  → ∀ M → P M
```

Figure 3: Permutation Induction Principle

α-compatible predicates:

```
αCompatiblePred : (Λ → Set) → Set
αCompatiblePred P = {M N : Λ} → M ∼α N → P M → P N
```

If the predicate is α-compatible then we can prove the following induction principle using previous induction principle. This new principle enables us to

choose the variable of the abstraction case different from a finite list of variables, in this way this principle allow us to emulate Barendregt Variable Convention (BVC).

$$\begin{aligned}
&\mathsf{Term\alpha PrimInd} : (P : \Lambda \to \mathsf{Set}) \to \mathsf{\alpha CompatiblePred}\ P \\
&\quad \to (\forall\ a \to P\ (\mathsf{v}\ a)) \\
&\quad \to (\forall\ M\ N \to P\ M \to P\ N \to P\ (M \cdot N)) \\
&\quad \to \exists\ (\lambda\ vs \to (\forall\ M\ b \to b \notin' vs \to P\ M \to P\ (\mathsf{\lambda}\ b\ M))) \\
&\quad \to \forall\ M \to P\ M
\end{aligned}$$

Again assuming an α-compatible predicate, we can prove the following principle using again the induction principle of figure 3:

$$\begin{aligned}
&\mathsf{Term\alpha IndPerm} : \forall\ P \to \mathsf{\alpha CompatiblePred}\ P \\
&\quad \to (\forall\ a \to P\ (\mathsf{v}\ a)) \\
&\quad \to (\forall\ M\ N \to P\ M \to P\ N \to\ \ P\ (M \cdot N)) \\
&\quad \to \exists\ (\lambda\ as \to (\forall\ M\ b \to b \notin' as \to (\forall\ \pi \to\ \ P\ (\pi \bullet M)) \to P\ (\mathsf{\lambda}\ b\ M))) \\
&\quad \to \forall\ M \to P\ M
\end{aligned}$$

Figure 4: Permutation αInduction

3 Iteration and Recursion Principles

We want to define strong α-compatible functions, that is, functions over the α-equivalence class of terms. This functions can not depend on the abstraction variables of a term. We can resume this concept in the following definition:

$$\begin{aligned}
&\mathsf{strong{\sim}\alpha Compatible} : \{A : \mathsf{Set}\} \to (\Lambda \to A) \to \Lambda \to \mathsf{Set} \\
&\mathsf{strong{\sim}\alpha Compatible}\ f\ M = \forall\ N \to M \sim\alpha\ N \to f\ M \equiv f\ N
\end{aligned}$$

We define an iteration principle over raw terms which always produces α-compatible functions. This is granted because abstraction variables are given by the induction principle, hidding the specific abstraction variables of the given term, in this way the result of a function defined with this iterator has no way to extract any information from abstracted variables. This principle also allow us to give a list of variables from where the abstractions variables will not to be choosen, this will be usefull to define the no capture substitution operation latter. This iteration principle is derived from the last presented induction principle in figure 4.

$$\begin{aligned}
&\Lambda\mathsf{It}\ : (A : \mathsf{Set}) \\
&\to (\mathsf{Atom} \to A) \\
&\to (A \to A \to A) \\
&\to \mathsf{List\ Atom} \times (\mathsf{Atom} \to A \to A) \\
&\to \Lambda \to A
\end{aligned}$$

The following result estabish the strong compatibility of previous iteration principle. This result is proved using the induction principle in figure 4.

```
lemmaItαStrongCompatible : (A : Set)
  → (hv : Atom → A)
  → (h· : A → A → A)
  → (vs : List Atom)
  → (hƛ : Atom → A → A )
  → (M : Λ) → strong∼αCompatible (ΛIt A hv h· (vs , hƛ)) M
```

Figure 5: Strong α Compatibility of the Iteration Principle

From this iteration principle we directly derive the next recursion principle over terms, which also generates strong α-compatible functions.

```
ΛRec : (A : Set)
     → (Atom → A)
     → (A → A → Λ → Λ → A)
     → List Atom × (Atom → A → Λ → A)
     → Λ → A
```

4 Iterator Application

Next we show several iteration principle applications.

4.1 Free Variables

We implement the function that returns the free variables of a term.

```
fv : Λ → List Atom
fv = ΛIt (List Atom) [_] _++_ ([] , λ v r → r - v)
```

As a direct consequence of strong α compatibility of the iteration principle we obtain that α compatible terms have equal free variables.

The relation _*_ holds when a variables ocurrs free in a term.

```
data _*_ : Atom → Λ → Set where
  *v  : {x : Atom}                                → x * v x
  *·l : {x : Atom}{M N : Λ} → x * M               → x * (M · N)
  *·r : {x : Atom}{M N : Λ} → x * N               → x * (M · N)
  *ƛ  : {x y : Atom}{M : Λ} → x * M → y ≢ x       → x * (ƛ y M)
```

We can use the last induction principle (fig. 4) to prove the following proposition:

```
Pfv* : Atom → Λ → Set
Pfv* a M = a ∈' fv M → a * M
```

In the lambda abstraction obligation proof of the induction principle used, we can exclude the variable a from the abstraction variables we need to prove, simplifying in this way the required proof. We have to prove that $\forall b \not\equiv a, a \in \mathsf{fv}\,(\mathsf{ƛ}\,b\,M) \Rightarrow a * \mathsf{ƛ}\,b\,M$, knowing by inductive hypothesis that $\forall \pi, a \in \mathsf{fv}\,(\pi \bullet$

$M) \Rightarrow a * (\pi \bullet M)$. So $a \in \mathsf{fv}\,(\lambda\!\!\!/\, b\, M)$ and $b \not\equiv a$ then we know $a \in \mathsf{fv}\, M$ holds. Now, instantiting the inductive hypothesis with an empty permtutation and the previous result, we have that $a * M$, using again that $b \not\equiv a$, we can then conclude the desired result: $a * \lambda\!\!\!/\, b\, M$.

This flexibility comes with the extra cost that we need to prove that the predicate $\forall a, \mathsf{Pfv}{*}a$ is α-compatible, but this proof is direct because $*$ is an α-compatible relation and the fv function is strong α-compatible.

4.2 Substitution

We implement the no capture substitution operation. We give the substituted variable and free variables of the replaced term as variables to not to choose as abtractions to avoid any variable capture.

```
hvar : Atom → Λ → Atom → Λ
hvar x N y with x ≟ₐ y
... | yes _ = N
... | no  _ = v y
-
_[_:=_] : Λ → Atom → Λ → Λ
M [ a := N ] = ΛIt Λ (hvar a N) _·_ (a :: fv N , λ) M
```

Again because of the strong α-compability of the iteration principle we obtain the following result for free:

```
lemmaSubst1 : {M N : Λ}(P : Λ)(a : Atom)
  → M ∼α N → M [ a := P ] ≡ N [ a := P ]
```

Using the induction principle in figure 3 we prove:

```
lemmaSubst2 : ∀ {N} {P} M x
  → N ∼α P → M [ x := N ] ∼α M [ x := P ]
```

Finally, from the two previous result we directly obtain next substitution lemma.

```
lemmaSubst : {M N P Q : Λ}(a : Atom)
  → M ∼α N → P ∼α Q
  → M [ a := P ] ∼α N [ a := Q ]
lemmaSubst {M} {N} {P} {Q} a M∼N P∼Q
  = begin
      M [ a := P ]
    ≈⟨ lemmaSubst1 P a M∼N ⟩
      N [ a := P ]
    ∼⟨ lemmaSubst2 N a P∼Q ⟩
      N [ a := Q ]
    □
```

Remarkably all this result are directly derived from the first primitive induction principle, and no need of induction on the length of terms or accesible predicates were needed in all of this formalization.