Locally Nameless Syntax & Semantics

Andrew Pitts
UNIVERSITY OF CAMBRIDGE

Edinburgh, November 2023
Programming language research

new idea

define

prove

The rise of machine-checked formalization (ITPs)

implement

test

tool

HOL4, Isabelle/HOL, PVS,...

Coq, Agda, Lean,...
Programming language research

Some key mathematical ingredients:

- Inductive definitions of types
- Recursive definitions of functions

- Change of equality (quotient types)

Well-supported by current TTPs

Computational logic for them is currently not well developed, so we avoid if possible
Syntax involving name binding operations

Very commonly occurring, so an inductive representation that identifies $\alpha$-equivalent syntax trees is highly desirable.

E.g.

\[\lambda f \rightarrow \lambda x \rightarrow (\lambda x \rightarrow fx)x\]
\[\lambda f \rightarrow \lambda x \rightarrow (\lambda y \rightarrow fy)x\]
\[\lambda x \rightarrow \lambda f \rightarrow (\lambda y \rightarrow xy)f\]

\{ these Haskell functions only differ up to renaming their bound variables \}
Syntax involving name binding operations

Very commonly occurring, so an inductive representation that identifies $\alpha$-equivalent syntax trees is highly desirable.

- There are many ways of achieving that.
- Not all are equal when it comes to formalizing and proving within existing ITPs.
Plan

- Review of *locally nameless* syntax representation, in contrast to *fully nameless* & naive named representations.
  The role of *cofinite quantification*.

- New mathematical foundation for *locally nameless* & its consequences
Named, nameless & locally nameless
syntax representation

Use syntax of *simply typed* λ-calculus with products
as a running example
Named, nameless & locally nameless

Syntax representation

Use syntax of simply typed λ-calculus with products as a running example

\[
A ::= G | A \rightarrow A | A \times A
\]

\[
E ::= x | \lambda x : A . E | E \cdot E | (E, E) | \text{fst} E | \text{snd} E
\]

(G = ground type, x = variable)

Inductive syntax with named free and bound variables

\[
x : A, y : A \vdash \lambda y : B . (x, y) : B \rightarrow A \times B
\]

↑
↑
↑
↑
↑

names range over \( \mathbb{N} \),
only important property is \( \neq \)
Named, nameless & locally nameless
syntax representation

Use syntax of simply typed λ-calculus with products
as a running example

\[ A ::= G \mid A \to A \mid A \times A \]
\[ t ::= x \mid \lambda x : A. t \mid t \circ t \mid (t, t) \mid \text{fst} t \mid \text{snd} t \]
\[ (G = \text{ground type, } x = \text{variable}) \]

Inductive syntax with named free and bound variables

\[ x : A, y : A \vdash \lambda y : B. (x, y) : B \to A \times B \]
\[ x : A, y : A \vdash \lambda z : B. (x, z) : B \to A \times B \]
Named, nameless & locally nameless

Syntax representation

Use syntax of simply typed \( \lambda \)-calculus with products as a running example

\[
A ::= G[A\rightarrow A] \mid A \times A
\]

\[
\mathsf{t} ::= x \mid \lambda x: A. \mathsf{t} \mid \mathsf{tt} \mid (\mathsf{t}, \mathsf{t}) \mid \text{fst} \mid \text{snd}
\]

(\( G = \) ground type, \( x = \) variable)

Quotient of inductive syntax with named free and bound variables

\[
x: A, y: A \vdash [\lambda z: B. (x, z)]_\alpha : B \rightarrow A \times B
\]

\( \alpha \)-equivalence class
**Named, nameless & locally nameless**

**Syntax representation**

Use syntax of simply typed λ-calculus with products as a running example

\[ A ::= G\{ A\to A\} \mid A \times A \]

\[ t ::= x \mid \lambda x : A . t \mid t \times t \mid (t, t) \mid \text{fst } t \mid \text{snd } t \]

\( G = \text{ground type}, \ x = \text{variable} \)

**Quotient of inductive syntax with named free and bound variables**

\[ x : A , y : A \vdash [\lambda z : B . (x, z)]_\alpha : B \rightarrow A \times B \]

is weakening invariant

\[ x : A , y : A , z : C \vdash [\lambda z : B . (x, z)]_\alpha : B \rightarrow A \times B \]
Named, *nameless* & locally nameless

**Syntax representation**

Use syntax of simply typed λ-calculus with products as a running example

\[
A ::= G \mid A \rightarrow A \mid A \times A
\]

\[
t ::= \bar{1} \mid \lambda A.t \mid tt \mid (t,t) \mid ftt \mid \text{snd}t
\]

\(G = \text{ground type}, \bar{1} = \text{deBruijn index}\)

**DeBruijn indices** replace free and bound variables

\[
\begin{align*}
A, A & \vdash \lambda B. (2,0) : B \rightarrow A \times B \\
\text{dangling index}
\end{align*}
\]

\(\check{\text{purely inductive}}\)

Indexes range over \(\mathbb{N}\), only important property is \(<\), use + & \(\div\) operations
Use syntax of simply typed \(\lambda\)-calculus with products as a running example.

Syntax representation

\[ A ::= G \mid A \to A \mid A \times A \]

\[ t ::= \iota \mid A.t \mid \lambda t.1 \mid t \circ t \mid \text{Bewi}i \theta t \theta t \]

De Bruijn indices replace free and bound variables

purely inductive

invariant under weakening, e.g.

purely inductive

not invariant under weakening, e.g.

De Bruijn indices replace free and bound variables
Named, nameless & locally nameless

Syntax representation

Use syntax of simply typed λ-calculus with products as a running example.

\[
A ::= G | A \rightarrow A | A \times A \\
\tau ::= x | i | \lambda x.A \cdot \tau | \tau \cdot \tau | (\tau, \tau) | \text{fst} \cdot \tau | \text{snd} \cdot \tau
\]

(\text{G = ground type, x = variable, i = index})

Free vars named, bound vars indexed

\[
x : A, y : A \vdash \lambda B. (x, 0) : B \rightarrow A \times B
\]

✓ purely inductive
✓ weakening invariant

\[
x : A, y : A, z : C \vdash \lambda B. (x, 0) : B \rightarrow A \times B
\]
Dangling indexes

- used in nameless syntax to represent free variables

\[ \lambda B. (2, 0) \]
Dangling indexes

- used in nameless syntax to represent free variables

- Should be avoided in locally nameless syntax, since we have named variables

  restrict attention to inductively defined subset of locally closed terms
Locally closed terms

t is \text{locally closed} if \( LC_0(t) \) holds, where

\[
\frac{LC_i(x)}{LC_i(\lambda A \cdot t)} \quad \frac{LC_i(s)}{LC_i(st)} \quad \frac{LC_i(t)}{LC_i(fstt)} \quad \frac{LC_i(t)}{LC_i(sndt)}
\]

\[
\text{LC}_{i+1}(t) \quad \frac{LC_i(s) \quad LC_i(t)}{LC_i((s,t))} \quad \frac{LC_i(s)}{LC_i(t)} \quad \frac{LC_i(t)}{LC_i(sndt)}
\]

\[
\frac{LC_i(j)}{LC_i(i)} \quad \frac{LC_i(s) \quad LC_i(t)}{LC_i((s,t))} \quad \frac{LC_i(s)}{LC_i(t)} \quad \frac{LC_i(t)}{LC_i(sndt)}
\]
Locally closed terms

Restricting to such terms avoids the need for tricky (= error prone) index-shifting calculus.

E.g. capture-avoiding substitution:

\[ t[s/x] = \text{substitute } s \text{ for } x \text{ in } t \]

| i[s/x] = i | x[s/x] = s | y[s/x] = y |
| (\lambda A. t)[s/x] = \lambda A. (t[s/x]) |
| (t_1 t_2)[s/x] = (t_1[s/x])(t_2[s/x]) |

etc

Very simple! But only correct if \( LC_0(s) \)
Locally closed terms

restricting to such terms avoids the need for tricky (= error prone) index-shifting calculus

necessitates opening terms with fresh vars when computing under binding operations

\[ \lambda A. t \]

locally closed

not nec. locally closed

\[ \text{LC}_0 (\lambda A. t) \]

\[ \text{LC}_1 (t) \]
Locally nameless: opening & closing ops

given index $i$ & variable $x$, get operations on terms $t$:

$[i \leadsto x]t \triangleq \text{open } t \text{ by replacing index } i \text{ by variable } x$

$[i \leftarrow x]t \triangleq \text{close } t \text{ by replacing variable } x \text{ by index } i$

defined by induction on structure of $t$

$[i \leadsto x](st) = ([i \leadsto x]s)([i \leadsto x]t)$

$[i \leadsto x](\lambda A.t) = \lambda A.[i+1 \leadsto x]t$

$[i \leftarrow x](st) = ([i \leftarrow x]s)([i \leftarrow x]t)$

$[i \leftarrow x](\lambda A.t) = \lambda A.[i+1 \leftarrow x]t$

etc
Locally nameless: opening & closing ops

given index $i$ & variable $x$, get operations on terms $t$:

$i \leadsto x \ t : \triangleq \text{open } t \ \text{by replacing index } i \ \text{by variable } x$

$i \leadsto x \ t : \triangleq \text{close } t \ \text{by replacing variable } x \ \text{by index } i$

E.g.

$[1 \rightarrow x] (\lambda A. (2,0)) = \lambda A. (x,0)$

$[1 \leftarrow x] (\lambda A. (x,0)) = \lambda A. (2,0)$
Locally closed terms

Restricting to such terms avoids the need for tricky (= error prone) index-shifting calculus

Necessitates opening terms with fresh vars when computing under binding operations

E.g. typing rule for λ-abstractions (finitary version)

\[
\begin{array}{c}
\Gamma, x : A \vdash [0 \rightarrow x] t : B \\
\Gamma \vdash \lambda A.t : A \rightarrow B
\end{array}
\]

\( x \notin \Gamma \)

Non-occurrence relation (inductively defined)
Locally closed terms

Restricting to such terms avoids the need for tricky (= error-prone) index-shifting calculus.

Necessitates opening terms with fresh vars when computing under binding operations.

E.g. typing rule for λ-abstractions (infinitary version)

\[
\Gamma, x : A \vdash [0 \to x]t : B \\
\Gamma \vdash \lambda A. t : A \to B
\]
Cofinite quantification

Given a set $A$ and a property $\varphi(x)$ of elements $x \in A$, write $\bigcup_{x \in A} \varphi(x)$ to mean

"$\varphi(x)$ holds for all but finitely many $x \in A$"

Relevance to PL meta-theory recognized independently by McKinna & Pollack, JAR 23 (1999) 373-409
Grubbay & AMP, LICS 1999
Cofinite quantification

Given a set $A$ and a property $\varphi(x)$ of elements $x \in A$, write $\bigcup x \in A . \varphi(x)$ to mean

"$\varphi(x)$ holds for all but finitely many $x \in A$"

Has nice properties when $A$ is "unfinite"

- decidable equality
- finitely inexhaustible

and $\varphi(x)$ is a finitely supported property (à la nominal sets)
Pause

So far, it’s been a summary of what’s known

Now for something new
Locally Nameless Sets

AMP, POPL 2023

New equational axiomatization of open/close operations

some of the (9) axioms are non-obvious, e.g.:

\[ [j \rightarrow y] [j \leftarrow x] [i \rightarrow x] t = [i \rightarrow y] [i \leftarrow x] [j \rightarrow y] t \]

(note that \((i \rightarrow y) \circ (x \rightarrow j) \circ (i \rightarrow x)\)

and \((i \rightarrow y) \circ (x \rightarrow i) \circ (j \rightarrow y)\)

both equal \{i,j,x \rightarrow y\} )
Locally Nameless Sets

AMP, POPL 2023

New equational axiomatization of open/close operations. Combined with $\forall$-quantifier, gives the rest of the locally nameless infrastructure. E.g.

local closure: $LC_i(t) \iff \forall j \geq i. \forall x. [j \mapsto x]t = t$

non-occurrence: $x \not\in t \iff [0 \leftarrow x]t = t$

finite support: $\forall x. x \not\in t$
Locally Nameless Sets

AMP, POPL 2023

New equational axiomatization of open/close operations.

Category $\mathbf{Lns}$ objects = sets equipped with morphisms = functions preserving \{open/close ops\}

Theorem $\mathbf{Lns}$ is equivalent to

\[(\mathbf{Set}^M)_{fp}\]

category (topos) introduced by Staton & Gabbay - Hofmann to model name-for-name substitution

$M$-sets for monoid $M = \text{all functions } \mathbb{N} \to \mathbb{N}$

Full subcat. of finitely supported objects
Locally Nameless Sets

AMP, POPL 2023

New equational axiomatization of open/close operations.

Category $\mathbf{Lns}$

objects = sets equipped with open/close ops

morphisms = functions preserving open/close ops

Theorem: $\mathbf{Lns}$ is equivalent to $(\text{Set}^M)^{\text{fp}}$ and to Popescu's category of finitely supported rensets.

(IJCAR 2022)
Locally Nameless Sets

AMP, POPL 2023

New equational axiomatization of open/close operations.

- Generic, language-independent characterization of binding via "shift" functor $\uparrow : \text{Lns} \to \text{Lns}$ ...

- Initial algebra semantics of syntax/$\simeq_\alpha$

- Non-syntactic examples of locally nameless sets

- Potential for typeclass-style automation of locally nameless "boiler plate" in ITPs
data $S : \text{Set}$ where

\begin{align*}
\text{var} : & \mathbb{V} \to S \\
\text{idx} : & \mathbb{N} \to S \\
\text{app} : & S \times S \to S \\
\text{lam} : & \uparrow S \to S
\end{align*}

\text{deriving} \quad \text{Locally Nameless}

(\text{cf. Escort \& Cockx, ICFP 2022})
Take home

locally nameless + cofinite quantification

works very well for formalized meta-theory of PLs
- purely inductive, but up-to-a
- avoids weakening- and lifting-hell

See:
- Aydemir et al., POPL 2008
- Charguéraud, JAR (2012) 363-408
- AMP, POPL 2023
End
Opening/Closing Axioms

\[
\begin{align*}
[i \rightarrow a][i \rightarrow b]t &= [i \rightarrow b]t \\
[i \leftarrow a][j \leftarrow a]t &= [j \leftarrow a]t \\
[i \leftarrow a][i \rightarrow a]t &= [i \leftarrow a]t \\
[i \rightarrow a][i \leftarrow a]t &= [i \rightarrow a]t
\end{align*}
\]

\[
i \neq j \Rightarrow [i \rightarrow a][j \rightarrow b]t = [j \rightarrow b][i \rightarrow a]t
\]

\[
a \neq b \Rightarrow [i \leftarrow a][j \leftarrow b]t = [j \leftarrow b][i \leftarrow a]t
\]

\[
i \neq j \& a \neq b \Rightarrow [i \rightarrow a][j \leftarrow b]t = [j \leftarrow b][i \rightarrow a]t
\]

\[
[i \rightarrow b][i \leftarrow a][j \rightarrow b]t = [j \rightarrow b][j \leftarrow a][i \rightarrow a]t
\]

\[
[j \leftarrow a][i \rightarrow a][j \leftarrow b]t = [j \leftarrow b][i \rightarrow b][j \leftarrow a]t
\]