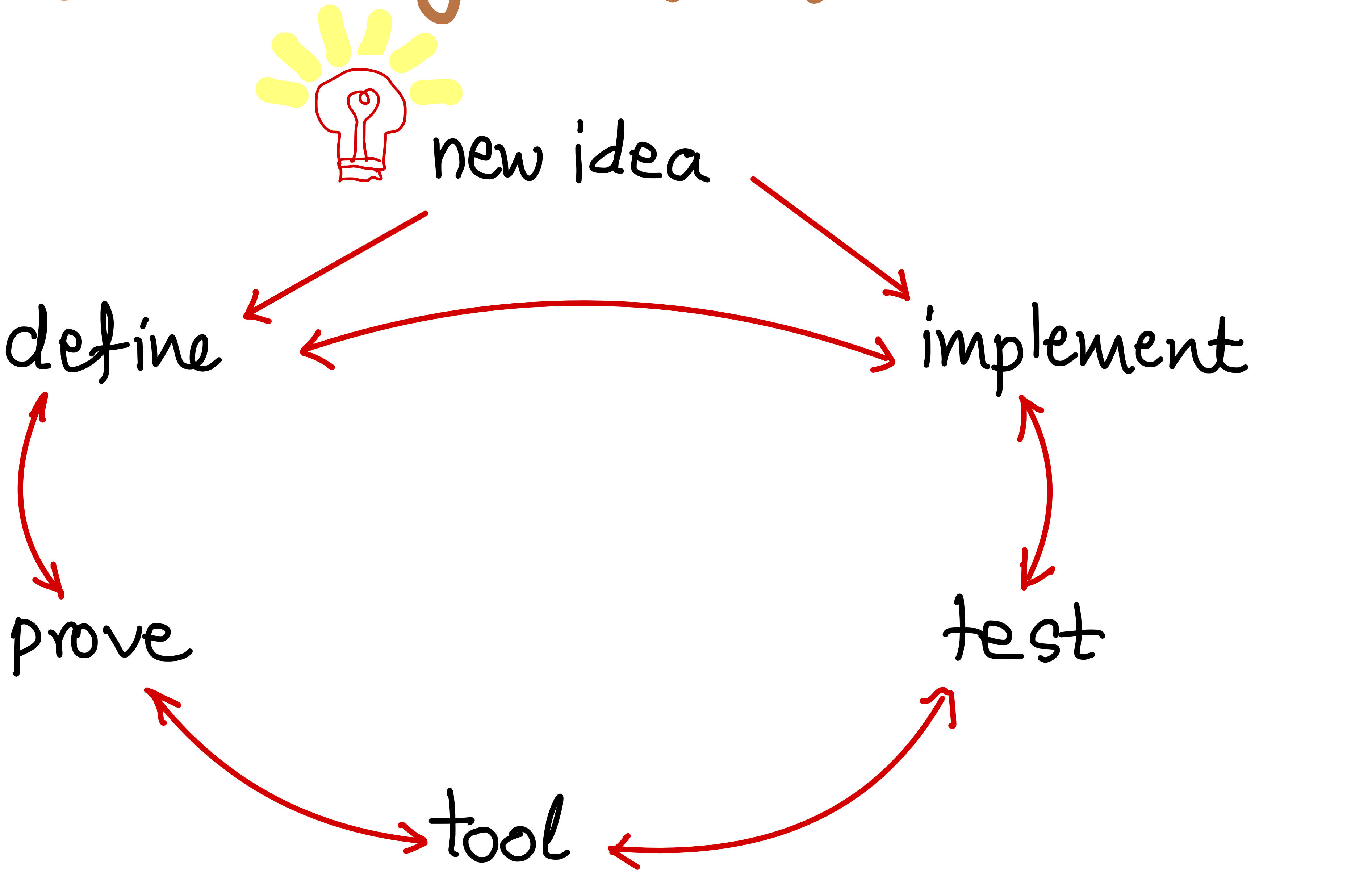
Locally Nameless Syntax & semantics

Andrew Pitts

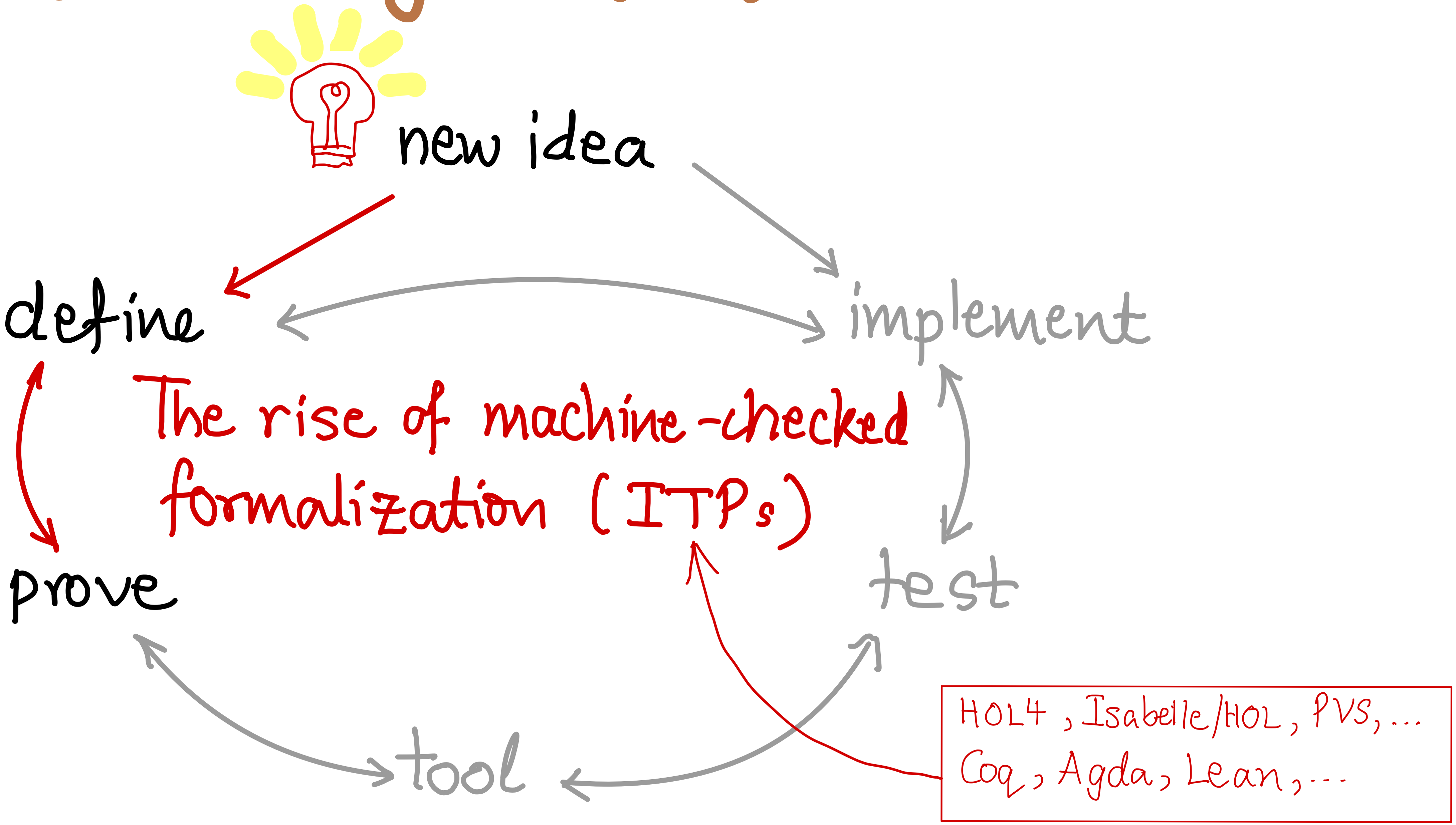
UNIVERSITY OF CAMBRIDGE

Edinburgh, November 2023

Programming language research



Programming language research



Programming language research

Some key mathematical ingredients:

- inductive definitions of types
  recursive definitions of functions

  well-supported by current ITPs

- change of equality (quotient types)

  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 α-equivalent syntax trees is highly desirable.

E.g.

```
\f -> \x -> (\x -> f x) x
\f -> \x -> (\y -> f y) x
\x -> \f -> (\y -> x y) 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 α-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 & naïve 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

$A ::= G \mid A \to A \mid A \times A$

$t ::= x \mid \lambda x : A . t \mid t\, t \mid (t, t) \mid \mathrm{fst}\, t \mid \mathrm{snd}\, t$

(G = ground type, x = variable)

Inductive syntax with named free and bound variables

$$x : A ,\, y : A \vdash \lambda y : B . (x, y) : B \to A \times B$$

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

$A ::= G \mid A \to A \mid A \times A$

$t ::= x \mid \lambda x : A.t \mid t\,t \mid (t, t) \mid \mathrm{fst}\,t \mid \mathrm{snd}\,t$

(G = ground type, x = 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 λ-calculus with products as a running example

$A ::= G \mid A \to A \mid A \times A$

$t ::= x \mid \lambda x : A . t \mid t\, t \mid (t, t) \mid \mathrm{fst}\, t \mid \mathrm{snd}\, t$

(G = ground type, x = variable)

Quotient of inductive syntax with named free and bound variables

$$x : A,\ y : A \vdash \underbrace{[\lambda z : B . (x, z)]_\alpha}_{\alpha\text{-equivalence class}} : B \to A \times B$$

Named, nameless & locally nameless

syntax representation

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

$A ::= G \mid A \to A \mid A \times A$

$t ::= x \mid \lambda x : A.t \mid t\,t \mid (t,t) \mid \mathrm{fst}\,t \mid \mathrm{snd}\,t$

(G = ground type, x = variable)

Quotient of inductive syntax with named free and bound variables

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

is weakening invariant

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

Named, nameless & locally nameless

syntax representation

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

$A ::= G \mid A \to A \mid A \times A$

$t ::= i \mid \lambda A.t \mid t\,t \mid (t,t) \mid \mathrm{fst}\,t \mid \mathrm{snd}\,t$

(G = ground type, i = deBruijn index

deBruijn indices replace free and bound variables

$$\overset{2}{A}, \overset{1}{A} \vdash \lambda \overset{0}{B}.\,(2, 0) : B \to A \times B$$

dangling index

✓ purely inductive

indexes range over $\mathbb{N}$, only important property is $<$, use $+$ & $\dot{-}$ operations

Named, nameless & locally nameless

syntax representation

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

$A ::= G \mid A \to A \mid A \times A$

$t ::= i \mid \lambda A.t \mid t\,t \mid (t,t) \mid \mathrm{fst}\,t \mid \mathrm{snd}\,t$

(G = ground type, i = deBruijn index)

deBruijn indices replace free and bound variables

$$A, A \vdash \lambda B.(2,0) : B \to A \times B$$

✓ purely inductive

× not invariant under weakening, e.g.

$$\overset{3}{A}, \overset{2}{A}, \overset{1}{C} \vdash \lambda \overset{0}{B}.(2+1, 0) : B \to A \times B$$

Named, nameless & locally nameless

syntax representation

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

$A ::= G \mid A \to A \mid A \times A$

$t ::= x \mid i \mid \lambda A.t \mid t\,t \mid (t,t) \mid \mathrm{fst}\,t \mid \mathrm{snd}\,t$

(G = ground type, x = variable, i = index)

free vars named, bound vars indexed

$$x : A,\ y : A \vdash \lambda B.(x, 0) : B \to A \times B$$

✓ purely inductive

✓ weakening invariant

$$x : A,\ y : A,\ z : C \vdash \lambda B.(x, 0) : B \to A \times B$$

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 <u>locally closed</u> if $LC_0(t)$ holds, where

$$\frac{}{LC_i(x)} \qquad \frac{j < i}{LC_i(j)}$$

$$\frac{LC_{i+1}(t)}{LC_i(\lambda A.t)} \qquad \frac{LC_i(s) \quad LC_i(t)}{LC_i(st)}$$

$$\frac{LC_i(s) \quad LC_i(t)}{LC_i((s,t))} \qquad \frac{LC_i(t)}{LC_i(\mathrm{fst}\ t)} \qquad \frac{LC_i(t)}{LC_i(\mathrm{snd}\ t)}$$

Locally closed terms

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

E.g. capture-avioding substitution

$t[s/x]$ = substitute s for x in t

$i[s/x] = i \qquad x[s/x] = s \qquad y[s/x] = y$

$(\lambda A.t)[s/x] = \lambda A.(t[s/x])$

$(t_1 t_2)[s/x] = (t_1[s/x])(t_2[s/x])$

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

t — not nec. locally closed

$$LC_0(\lambda A . t)$$

$$\Uparrow$$

$$LC_1(t)$$

Locally nameless : opening & closing ops

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

$[i \rightsquigarrow x]\, t \triangleq$ open t by replacing index i by variable x

$[i \leftsquigarrow x]\, t \triangleq$ close t by replacing variable x by index i

defined by induction on structure of t

$$[i \rightsquigarrow x](s\,t) = ([i \rightsquigarrow x]s\ ([i \rightsquigarrow x]t)$$

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

$$[i \leftsquigarrow x](s\,t) = ([i \leftsquigarrow x]s)([i \leftsquigarrow x]t)$$

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

etc

Locally nameless : opening & closing ops

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

$i \rightsquigarrow x \quad t \triangleq$ open t by replacing index i by variable x

$i \leftsquigarrow x \quad t \triangleq$ close t by replacing variable x by index i

E.g. $[1 \to x](\lambda A.(2,0)) = \lambda A.(x,0)$

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

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)

$$\boxed{\dfrac{\Gamma, x : A \vdash [0 \to x]t : B \qquad x \# \Gamma}{\Gamma \vdash \lambda A.t : A \to B}}$$

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)

cofinite quantifier

$$\boxed{\dfrac{\forall x \in Var\,(\Gamma, x : A \vdash [0 \to x]t : B)}{\Gamma \vdash \lambda A.t : A \to B}}$$

Cofinite quantification

Given a set A and a property $\varphi(x)$ of elements $x \in A$, write $\boxed{\text{И}\, x \in A \,.\, \varphi(x)}$ 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

Gabbay & AMP, LICS 1999

Cofinite quantification

Given a set A and a property $\varphi(x)$ of elements $x \in A$, write $\boxed{\text{И}\, x \in A \,.\, \varphi(x)}$ 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.:

$$[j \to y][j \leftarrow x][i \to x]\,t = [i \to y][i \leftarrow x][j \to y]\,t$$

(note that $(j \mapsto y) \circ (x \mapsto j) \circ (i \mapsto x)$
and $(i \mapsto y) \circ (x \mapsto i) \circ (j \mapsto y)$
both equal $\{i, j, x \mapsto y\}$)

Locally Nameless Sets

AMP, POPL 2023

New equational axiomatization of open/close operations.

Combined with И-quantifier, gives the rest of the locally nameless infrastructure. E.g.

local closure: $\mathrm{LC}_i(t) \triangleq \forall j \geq i.\ \forall x.\ [j \to x]t = t$

non-occurrence: $x \# t \triangleq [0 \leftarrow x]t = t$

finite support: $\text{И} x.\ x \# t$

Locally Nameless Sets

AMP, POPL 2023

New equational axiomatization of open/close operations.

Category **Lns** objects = sets equipped with } open/close ops
morphisms = functions preserving } open/close ops

$\boxed{\text{Theorem: Lns is equivalent to } (\mathbf{Set}^M)_{fp}}$

Theorem 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 = 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 Lns

objects = sets equipped with
morphisms = functions preserving } open/close ops

Theorem Lns is equivalent to

$(\mathbf{Set}^{\mathcal{M}})_{fp}$ 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 charactization of binding via "shift" functor $\uparrow : Lns \to Lns$ ...
- initial algebra semantics of syntax$/\approx_\alpha$
- non-syntactic examples of locally nameless sets
- potential for typeclass-style automation of locally nameless "boiler plate" in ITPs

```
data S : Set where
  var : 𝕍 → S
  idx : ℕ → S
  app : S × S → S
  lam : ↑S → S
deriving LocallyNameless
```

(cf. Escot & Cockx, ICFP 2022)

Take home

locally nameless + cofinite quantification
works very well for formalized meta-theory of PLs
- purely inductive, but up-to-α
- avoids weakening- and lifting-hell

See: Aydemir et al, POPL 2008
Charguéraud, JAR (2012) 363-408
AMP, POPL 2023

End

Opening/Closing Axioms

$$[i \to a][i \to b]t = [i \to b]t$$
$$[i \leftarrow a][j \leftarrow a]t = [j \leftarrow a]t$$

$$[i \leftarrow a][i \to a]t = [i \leftarrow a]t$$
$$[i \to a][i \leftarrow a]t = [i \to a]t$$

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

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

$$[i \to b][i \leftarrow a][j \to b]t = [j \to b][j \leftarrow a][i \to a]t$$
$$[j \leftarrow a][i \to a][j \leftarrow b]t = [j \leftarrow b][i \to b][i \leftarrow a]t$$