Supernominal Datatypes and Codatatypes

Andrei Popescu
University of Sheffield, UK

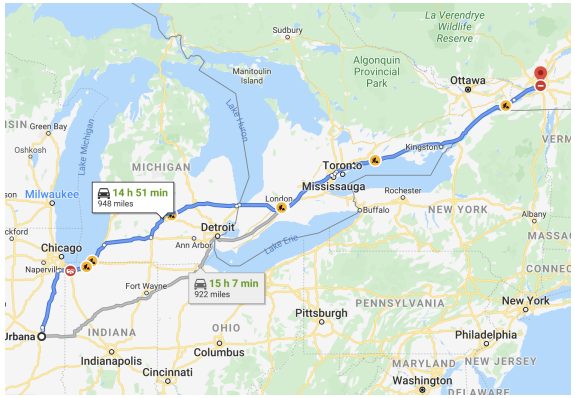
LFMTP
June 30, 2020

Those Were the Days...

LFMTP 2009, Montreal, Canada. Theory support for weak higher order abstract syntax in Isabelle/HOL. Elsa Gunter, Chris Osborn and Andrei Popescu.

Those Were the Days...

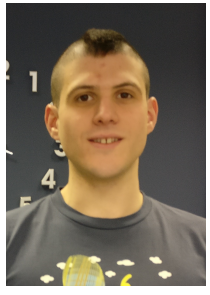
LFMTP 2009, Montreal, Canada. Theory support for weak higher order abstract syntax in Isabelle/HOL. Elsa Gunter, Chris Osborn and Andrei Popescu.

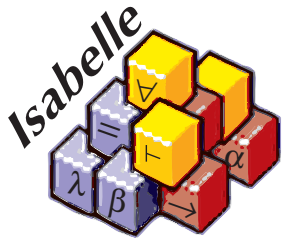


Supernominal

Joint work with...

Jasmin Blanchette, Lorenzo Gheri, Dmitriy Traytel, Isabelle/HOL





Ideology

How do most mathematicians, logicians and computer scientists view syntax with bindings?

How do most mathematicians, logicians and computer scientists view syntax with bindings?

Syntax with Bindings

=

Bureaucracy

Domain-Specific Results

=

Interesting Bits

How do most mathematicians, logicians and computer scientists view syntax with bindings?

Syntax with Bindings

=

Bureaucracy

Domain-Specific Results

=

Interesting Bits

Keep buraucracy low, offer high-level definition and proof principles.

Different Schools of Thought / Paradigms / Dogmas

De Bruijn

HOAS (Higher-Order Abstract Syntax)

Nominal/Nameful

Different Schools of Thought / Paradigms / Dogmas

De Bruijn nameless pointers

HOAS (Higher-Order Abstract Syntax) (meta-level) functional bindings

Nominal/Nameful explicit bound names, α-quotienting

They define terms with bindings in different ways.

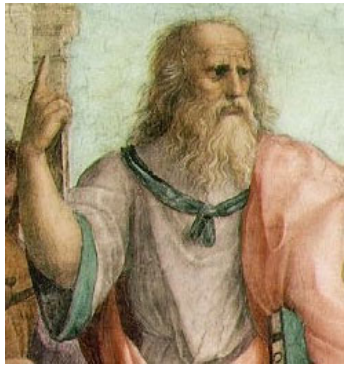
Different Schools of Thought / Paradigms / Dogmas

De Bruijn nameless pointers

HOAS (Higher-Order Abstract Syntax) (meta-level) functional bindings

Nominal/Nameful explicit bound names, α-quotienting

They define terms with bindings in different ways.



They talk about the same datatype

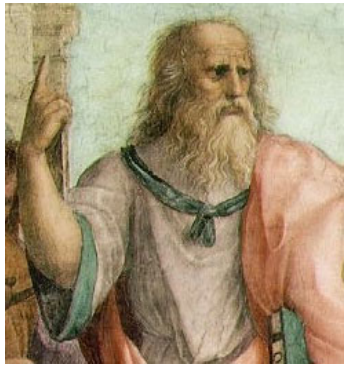
Different Schools of Thought / Paradigms / Dogmas

De Bruijn nameless pointers

HOAS (Higher-Order Abstract Syntax) (meta-level) functional bindings

Nominal/Nameful explicit bound names, α-quotienting

They define terms with bindings in different ways.



They talk about the same datatype
... but offer different ways to manipulate it

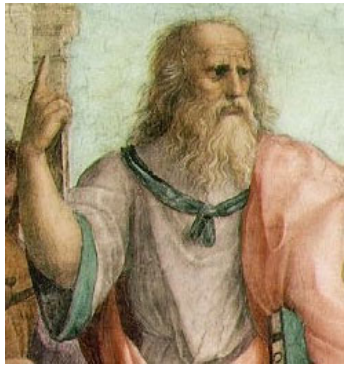
Different Schools of Thought / Paradigms / Dogmas

De Bruijn nameless pointers

HOAS (Higher-Order Abstract Syntax) (meta-level) functional bindings

Nominal/Nameful explicit bound names, α-quotienting

They define terms with bindings in different ways.



They talk about the same datatype
... but offer different ways to manipulate it

What's important: How to manipulate this datatype.

Different Schools of Thought / Paradigms / Dogmas

De Bruijn nameless pointers

$\lambda : \mathrm{Term} \to \mathrm{Term}$ $\lambda_n : \mathrm{Term}_{n+1} \to \mathrm{Term}_n$ (well-scoped)

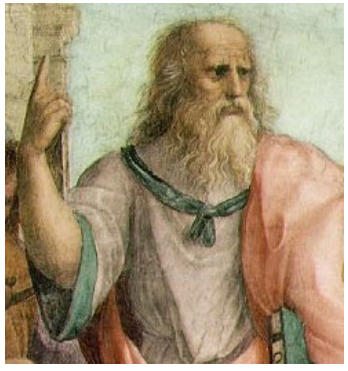
HOAS (Higher-Order Abstract Syntax) (meta-level) functional bindings

$\lambda : (\mathrm{Term} \to \mathrm{Term}) \to \mathrm{Term}$ (strong) $\lambda : (\mathrm{Var} \to \mathrm{Term}) \to \mathrm{Term}$ (weak)

Nominal/Nameful explicit bound names, α-quotienting

$\lambda : \mathrm{Var} \to \mathrm{Term} \to \mathrm{Term}$

They define terms with bindings in different ways.



They talk about the same datatype
... but offer different ways to manipulate it

What's important: How to manipulate this datatype.

Different Schools of Thought / Paradigms / Dogmas

De Bruijn nameless pointers

$\lambda : \mathrm{Term} \to \mathrm{Term}$ $\lambda_n : \mathrm{Term}_{n+1} \to \mathrm{Term}_n$ (well-scoped)

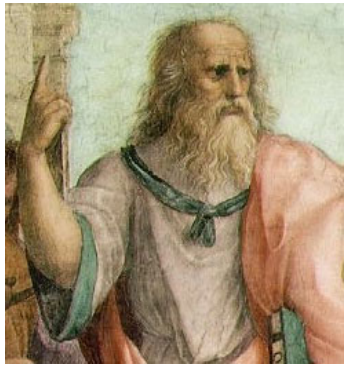
HOAS (Higher-Order Abstract Syntax) (meta-level) functional bindings

$\lambda : (\mathrm{Term} \to \mathrm{Term}) \to \mathrm{Term}$ (strong) $\lambda : (\mathrm{Var} \to \mathrm{Term}) \to \mathrm{Term}$ (weak)

Nominal/Nameful explicit bound names, α-quotienting

$\lambda : \mathrm{Var} \to \mathrm{Term} \to \mathrm{Term}$

They define terms with bindings in different ways.



They talk about the same datatype
... but offer different ways to manipulate it

What's important: How to manipulate this datatype.

Different schools have different insights and can learn from each other.

Combining and Sharing Knowledge across Paradigms

Johan van der
Brunomhoas
1471–1530

Combining and Sharing Knowledge across Paradigms

Johan van der
Brunomhoas
1471–1530

Gordon & Melham, 5 Axioms of Alpha-Conversion, 1996
Discovers a weak HOAS recursor

Norrish, Recursion for Types with Binders, 2004
Adjusts the above into a Nominal-style recursor

Hofmann, Semantical Analysis of HOAS, 1999
Topos where well-scoped De Bruijn = weak HOAS

Felty & Momigliano, Hybrid, 2008
HOAS on a De Bruijn substratum

Popescu et al., HOAS on top of FOAS, 2010
HOAS on a Nominal substratum

Berghofer & Urban, De Bruijn & Names head-to-head, 2007
Compares Nominal and "raw" De Bruijn

Abel et al., POPLMark reloaded, 2019
Compares strong HOAS with well-scoped De Bruijn

Combining and Sharing Knowledge across Paradigms

Johan van der
Brunomhoas
1471–1530

Gordon & Melham, 5 Axioms of Alpha-Conversion, 1996
Discovers a weak HOAS recursor

Norrish, Recursion for Types with Binders, 2004
Adjusts the above into a Nominal-style recursor

Hofmann, Semantical Analysis of HOAS, 1999
Topos where well-scoped De Bruijn = weak HOAS

Felty & Momigliano, Hybrid, 2008
HOAS on a De Bruijn substratum

Popescu et al., HOAS on top of FOAS, 2010
HOAS on a Nominal substratum

Berghofer & Urban, De Bruijn & Names head-to-head, 2007
Compares Nominal and "raw" De Bruijn

Abel et al., POPLMark reloaded, 2019
Compares strong HOAS with well-scoped De Bruijn

Next: Paradigm-agnostic description of general binders
Nominal-style reasoning infrastructure for them
Could employ De Bruijn or HOAS views of the same datatypes!

Supernominal 

Theory of binding-aware datatypes and codatatypes

Supports modular specification of complex binding mechanisms

(Co)datatypes come with definition and reasoning principles

Generalizes Nominal techniques: no finite support restriction

Has been formalized in Isabelle (implementation under way)

Supernominal 

Theory of binding-aware datatypes and codatatypes

Supports modular specification of complex binding mechanisms

(Co)datatypes come with definition and reasoning principles

Generalizes Nominal techniques: no finite support restriction

Has been formalized in Isabelle (implementation under way)

Let's talk bindings without commiting to any syntax

Supernominal 

Theory of binding-aware datatypes and codatatypes

Supports modular specification of complex binding mechanisms

(Co)datatypes come with definition and reasoning principles

Generalizes Nominal techniques: no finite support restriction

Has been formalized in Isabelle (implementation under way)

Let's talk bindings without commiting to any syntax
Then can have any syntax with any bindings we want

Supernominal 

Theory of binding-aware datatypes and codatatypes

Supports modular specification of complex binding mechanisms

(Co)datatypes come with definition and reasoning principles

Generalizes Nominal techniques: no finite support restriction

Has been formalized in Isabelle (implementation under way)

Let's talk bindings without commiting to any syntax
Then can have any syntax with any bindings we want

... which does not mean we lose any visual intuition

Bounded Natural Functors (BNFs)

Container Intuition: Shape Filled In With Content

$$\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$$

$$\mathsf{F}\,(A)$$

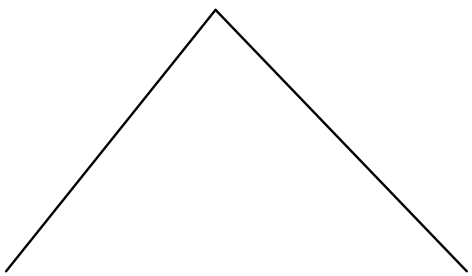
$$\biguplus$$

Container Intuition: Shape Filled In With Content

$$\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$$

$$\mathsf{F}\,(A)$$

$$\Psi$$

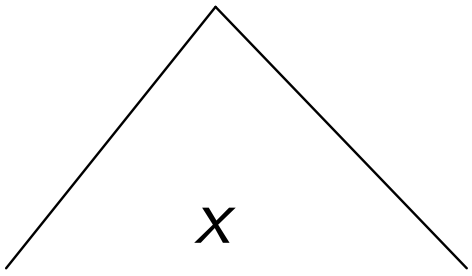


Container Intuition: Shape Filled In With Content

$\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

$\mathsf{F}\,(A)$

Ψ



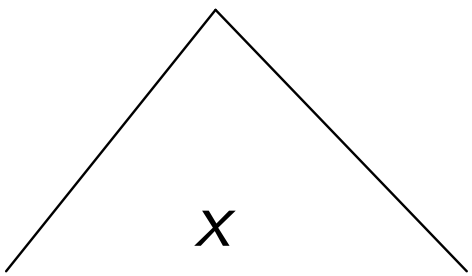
where $x \in A$

Container Intuition: Shape Filled In With Content

$\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

$\mathsf{F}\,(A)$

Ψ



where $x \in A$

Example

$\mathsf{F}\,(A) = \mathbb{N} \times A$

$\mathsf{F}\,(A)$

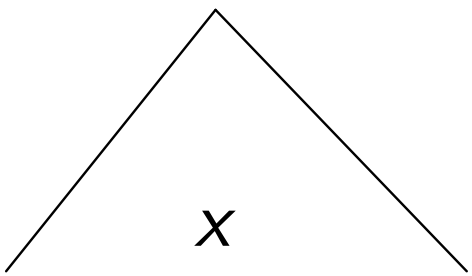
Ψ

Container Intuition: Shape Filled In With Content

$\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

$\mathsf{F}\,(A)$

$\Downarrow$



where $x \in A$

Example

$\mathsf{F}\,(A) = \mathbb{N} \times A$

$\mathsf{F}\,(A)$

$\Downarrow$

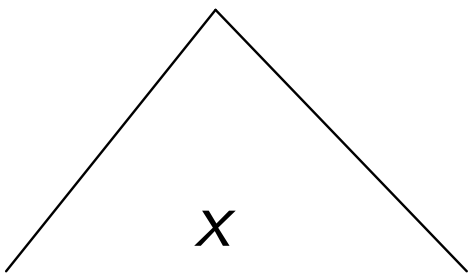
$(n, \_)$

Container Intuition: Shape Filled In With Content

$\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

$\mathsf{F}\,(A)$

Ψ



where $x \in A$

Example

$\mathsf{F}\,(A) = \mathbb{N} \times A$

$\mathsf{F}\,(A)$

Ψ

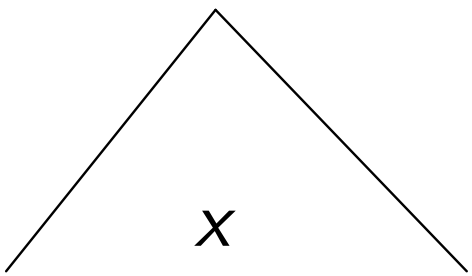
(n, x)

Container Intuition: Shape Filled In With Content

$\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

$\mathsf{F}\,(A)$

Ψ



where $x \in A$

Example

$\mathsf{F}\,(A) = \mathsf{List}\,(A)$

$\mathsf{F}\,(A)$

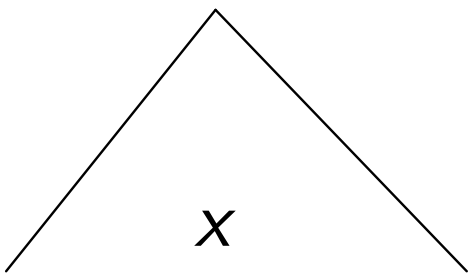
Ψ

Container Intuition: Shape Filled In With Content

$\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

$\mathsf{F}\,(A)$

$\Downarrow$



where $x \in A$

Example

$\mathsf{F}\,(A) = \mathsf{List}\,(A)$

$\mathsf{F}\,(A)$

$\Downarrow$

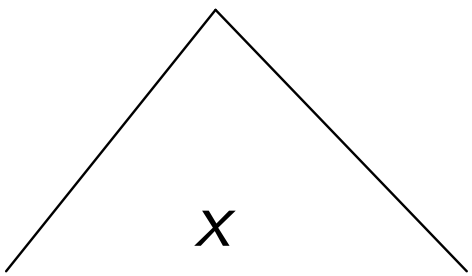
$[\_, \_, \_]$

Container Intuition: Shape Filled In With Content

$\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$

$\mathsf{F}\,(A)$

Ψ



where $x \in A$

Example

$\mathsf{F}\,(A) = \mathsf{List}\,(A)$

$\mathsf{F}\,(A)$

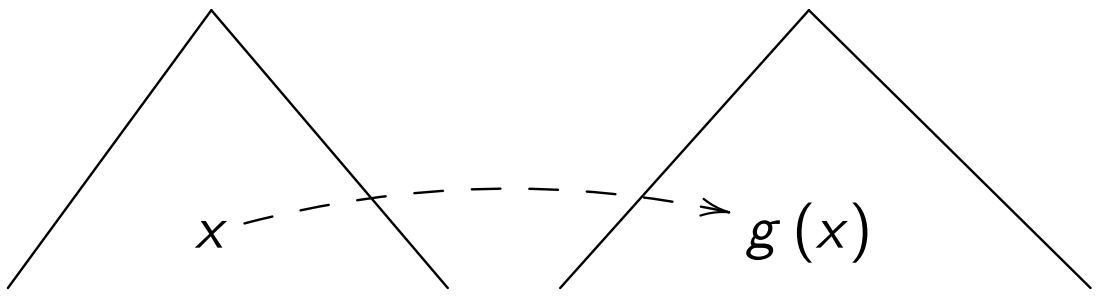
Ψ

$[x_1, x_2, x_3]$

3 Pillars: Boundedness, Naturality, **Functoriality** (BNF)

$$\begin{aligned} \mathsf{F} &: \mathsf{Set} \to \mathsf{Set} \\ \mathsf{map}_{\mathsf{F}} &: \prod_{A,B \in \mathsf{Set}} (A \to B) \to \mathsf{F}(A) \to \mathsf{F}(B) \end{aligned}$$

$$\begin{array}{ccc} A & \xrightarrow{\quad g \quad} & B \\ & & \\ \mathsf{F}(A) & \xrightarrow{\mathsf{map}_{\mathsf{F}}(g)} & \mathsf{F}(B) \\ \cup & & \cup \end{array}$$



3 Pillars: Boundedness, **Naturality**, Functoriality (BNF)

$$\begin{aligned} \mathsf{F} &: \mathsf{Set} \to \mathsf{Set} \\ \mathsf{supp}_{\mathsf{F}} &: \textstyle\prod_{A \in \mathsf{Set}} \mathsf{F}\,(A) \to \mathcal{P}\,(A) \end{aligned}$$

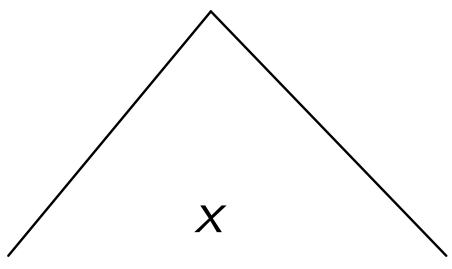
$$\begin{array}{ccc} \mathsf{F}\,(A) & \xrightarrow{\quad \mathsf{supp}_{\mathsf{F}} \quad} & \mathcal{P}\,(A) \\ \cup & & \cup \\ & & \{x, \ldots\} \\ & x & \end{array}$$

3 Pillars: **Boundedness**, Naturality, Functoriality (BNF)

$$\begin{array}{rcl} \mathsf{F} & : & \mathsf{Set} \to \mathsf{Set} \\ \mathsf{supp}_{\mathsf{F}} & : & \prod_{A \in \mathsf{Set}} \mathsf{F}\,(A) \to \mathcal{P}\,(A) \end{array}$$

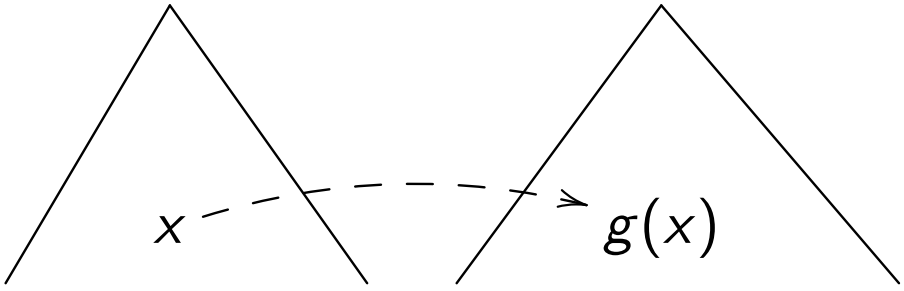
bd_{F} cardinal number

$$\begin{array}{ccc} \mathsf{F}\,(A) & \xrightarrow{\quad \mathsf{supp}_{\mathsf{F}} \quad} & \mathcal{P}\,(A) \\ \cup & & \cup \\ & & \mathsf{card}\,\{x, \ldots\} < \mathsf{bd}_{\mathsf{F}} \end{array}$$

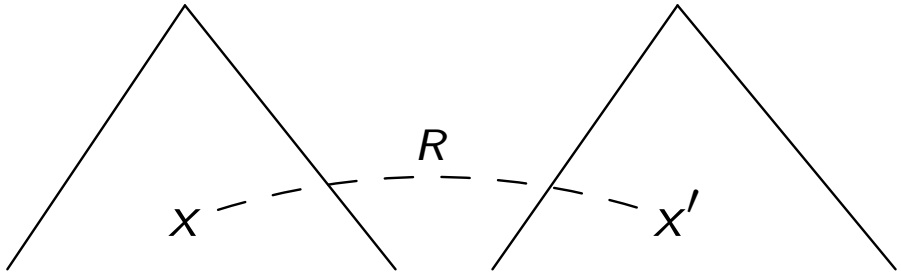


4th Pillar: Relator Structure

$$
\begin{array}{rrcl}
 & \mathsf{F} & : & \mathsf{Set} \to \mathsf{Set} \\
\text{Functor} & \mathsf{map}_{\mathsf{F}} & : & \prod_{A,B \in \mathsf{Set}} (A \to B) \to \mathsf{F}(A) \to \mathsf{F}(B) \\
\text{Relator} & \mathsf{rel}_{\mathsf{F}} & : & \prod_{A,B \in \mathsf{Set}} \mathcal{P}(A \times B) \to \mathcal{P}(\mathsf{F}(A) \times \mathsf{F}(B))
\end{array}
$$



$\mathsf{map}_{\mathsf{F}}(g)$
(slot-wise application of g)



$\mathsf{rel}_{\mathsf{F}}(R)$
(slot-wise lifting of R)

Example BNF: Lists

$$\mathsf{List} : \mathsf{Set} \to \mathsf{Set}$$

$\mathsf{map}_{\mathsf{List}}\ g\ [x_0, \ldots, x_{n-1}] = [g(x_0), \ldots, g(x_{n-1})]$

$\mathsf{supp}_{\mathsf{List}}\ [x_0, \ldots, x_{n-1}] = \{x_1, \ldots, x_{n-1}\}$

$\mathsf{bd}_{\mathsf{List}} = \aleph_0$

$([x_0, \ldots, x_{m-1}], [y_0, \ldots, y_{n-1}]) \in \mathsf{rel}_{\mathsf{List}}\ R$ iff
$m = n$ and $\forall i < m.\ (x_i, y_i) \in R$

Example BNF: Streams

$$\mathsf{Stream} : \mathsf{Set} \to \mathsf{Set}$$

$$\mathsf{map}_{\mathsf{Stream}}\ g\ [x_0, x_1, \ldots] = [g(x_0), g(x_1), \ldots]$$

$$\mathsf{supp}_{\mathsf{List}}\ [x_0, x_1, \ldots] = \{x_0, x_1, \ldots\}$$

$$\mathsf{bd}_{\mathsf{List}} = \aleph_1$$

$$([x_0, x_1, \ldots], [y_0, y_1, \ldots]) \in \mathsf{rel}_{\mathsf{List}}\ R \text{ iff}$$
$$\quad \forall i \in \mathbb{N}.\ (x_i, y_i) \in R$$

Other Examples of BNFs

Trees – finitely/infinitely branching, finite/infinite depth

Finite sets, countable sets, k-bounded sets

Multisets

Fuzzy sets

Probability distributions

...

Bounded Natural Functors (BNFs) in Isabelle

- BNFs
  - Include many useful container types
  - Closed under composition
  - Closed under least fixpoints (initial algebras)
  - Closed under greatest fixpoints (final coalgebra)

Bounded Natural Functors (BNFs) in Isabelle

BNFs

- Include many useful container types
- Closed under composition
- Closed under least fixpoints (initial algebras)
- Closed under greatest fixpoints (final coalgebra)

Isabelle/HOL's (co)datatype package

"One of the greatest engineering projects since Stonehenge!"

Jasmin Blanchette

Dmitriy Traytel

Datatypes and Codatatypes Based on BNFs

Example: Interactive processes with final states in A, making queries in B and acting upon responses in C.

$$\begin{array}{l}\mathsf{datatype}\ \mathsf{Proc}(A, B, C) = \\ \quad \mathsf{Step}\ (A + B \times (C \to \mathsf{Proc}(A, B, C)))\end{array}$$

Datatypes and Codatatypes Based on BNFs

Example: Interactive processes with final states in A, making queries in B and acting upon responses in C.

$$\begin{aligned}&\text{codatatype } \mathsf{Proc}(A, B, C) = \\ &\qquad \mathsf{Step}\ (A + B \times (C \to \mathsf{Proc}(A, B, C)))\end{aligned}$$

Possibly nonterminating.

Datatypes and Codatatypes Based on BNFs

Example: Interactive processes with final states in A, making queries in B and acting upon responses in C.

$$\begin{aligned}&\text{co}\mathsf{datatype}\ \mathsf{Proc}(A, B, C) =\\ &\qquad \mathsf{Step}\ (A + B \times (C \to \mathsf{Proc}(A, B, C)))\end{aligned}$$

Possibly nonterminating.

Finitely nondeterministic?

Datatypes and Codatatypes Based on BNFs

Example: Interactive processes with final states in A, making queries in B and acting upon responses in C.

$$\begin{aligned}&\mathsf{co}\mathsf{datatype}\ \mathsf{Proc}(A, B, C) = \\ &\qquad \mathsf{Step}\ (\mathsf{FPow}\ (A + B \times (C \to \mathsf{Proc}(A, B, C))))\end{aligned}$$

Possibly nonterminating.

Finitely nondeterministic? Plug in the finite powerset BNF.

Datatypes and Codatatypes Based on BNFs

Example: Interactive processes with final states in A, making queries in B and acting upon responses in C.

$$\begin{aligned}&\mathsf{co}\mathsf{datatype}\ \mathsf{Proc}(A, B, C) = \\ &\qquad \mathsf{Step}\ (\mathsf{CPow}\ (A + B \times (C \to \mathsf{Proc}(A, B, C))))\end{aligned}$$

Possibly nonterminating.

Finitely nondeterministic? Plug in the finite powerset BNF.

Countably nondeterministic? Plug in the countable powerset BNF.

Datatypes and Codatatypes Based on BNFs

Example: Interactive processes with final states in A, making queries in B and acting upon responses in C.

$$\begin{aligned}&\text{codatatype } \mathrm{Proc}(A, B, C) = \\ &\qquad \mathrm{Step}\ (\mathrm{PDist}\ (A + B \times (C \to \mathrm{Proc}(A, B, C))))\end{aligned}$$

Possibly nonterminating.

Finitely nondeterministic? Plug in the finite powerset BNF.

Countably nondeterministic? Plug in the countable powerset BNF.

Probabilistic? Plug in the probability distributions BNF.

Datatypes and Codatatypes Based on BNFs

Example: Interactive processes with final states in A, making queries in B and acting upon responses in C.

$$\text{codatatype } \mathrm{Proc}(A, B, C) = \\ \quad \mathrm{Step}\,(\mathrm{CPow}\,(\mathrm{PDist}\,(A + B \times (C \to \mathrm{Proc}(A, B, C)))))$$

Possibly nonterminating.

Finitely nondeterministic? Plug in the finite powerset BNF.

Countably nondeterministic? Plug in the countable powerset BNF.

Probabilistic? Plug in the probability distributions BNF.

Nondeterminism plus probability? Plug in both BNFs.

Datatypes and Codatatypes Based on BNFs

For each defined (co)datatype, Isabelle provides

- operators: constructor, map, relator, support, etc.
- lemmas about these operators: injectiveness, functoriality, naturality ("free" theorems), etc.
- (co)recursion definition principles
- (co)induction proof principles

Datatypes and Codatatypes Based on BNFs

For each defined (co)datatype, Isabelle provides

- operators: constructor, map, relator, support, etc.
- lemmas about these operators: injectiveness, functoriality, naturality ("free" theorems), etc.
- (co)recursion definition principles
- (co)induction proof principles

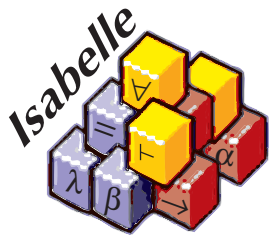
... typically not provided automatically by proof assistants

Datatypes and Codatatypes Based on BNFs

For each defined (co)datatype, Isabelle provides

- operators: constructor, map, relator, support, etc.
- lemmas about these operators: injectiveness, functoriality, naturality ("free" theorems), etc.
- (co)recursion definition principles
- (co)induction proof principles

... typically not provided automatically by proof assistants



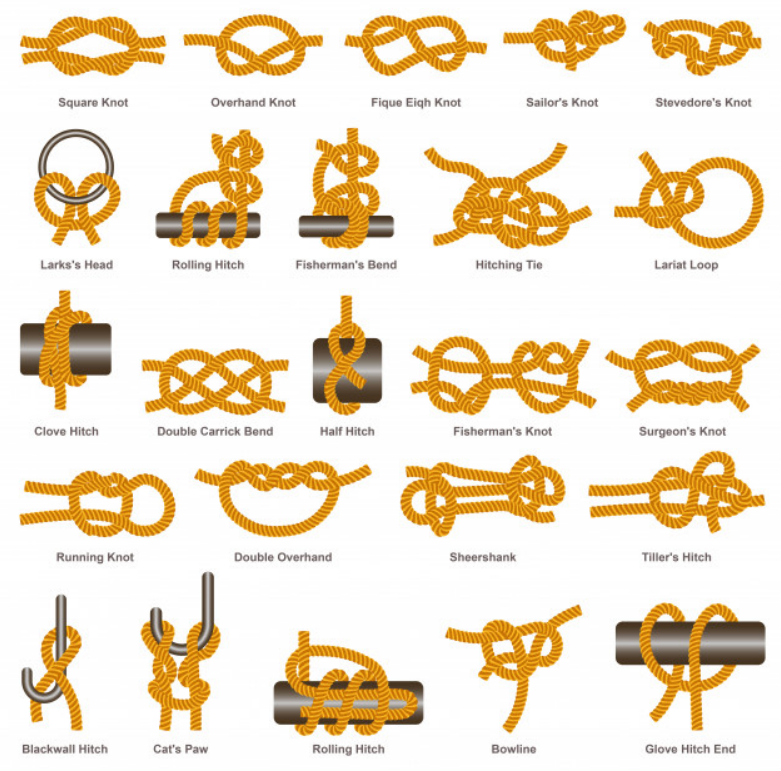
BNFs lurking in the background
without the users knowing of them

A Foundation for Binders

What is a Binder?

Several sophisticated syntactic formats:

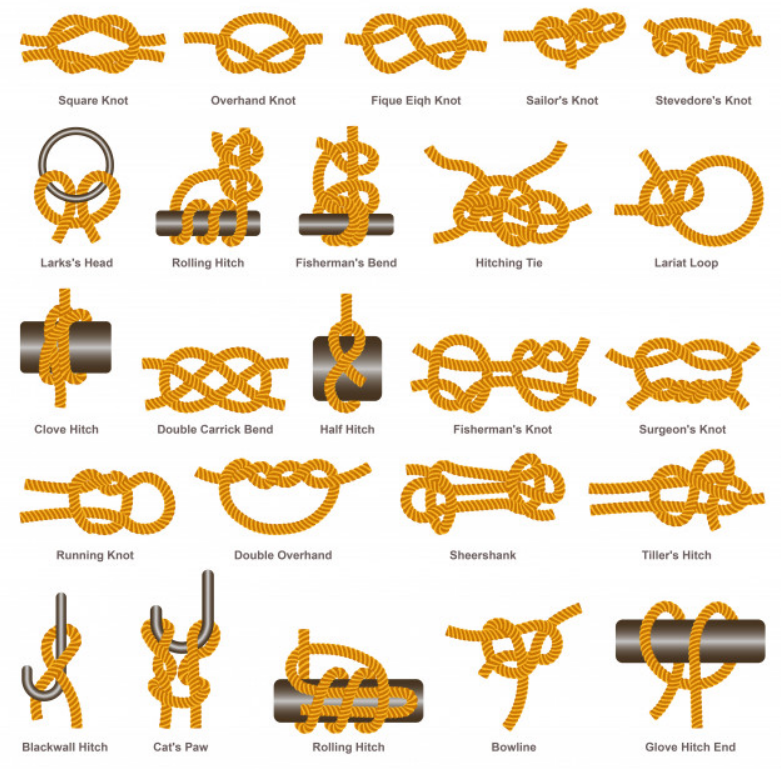
- CαML [Pottier 2006], Ott [Sewell et al. 2010], Unbound [Weirich et al. 2011], Isabelle Nominal2 [Urban and Kaliszyk 2012], Needle & Knot [Keuchel et al. 2016]



What is a Binder?

Several sophisticated syntactic formats:

- CαML [Pottier 2006], Ott [Sewell et al. 2010], Unbound [Weirich et al. 2011], Isabelle Nominal2 [Urban and Kaliszyk 2012], Needle & Knot [Keuchel et al. 2016]



Capture essence without committing to a particular syntax?

What is a Binder?

Binder = Mechanism for combining any variables with any terms.

What is a Binder?

Binder = Mechanism for combining any variables with any terms.

$\lambda v.\, t$

let $v = t_1$ in t_2

let rec $v_1 = t_1$ and $\ldots$ and $v_k = t_k$ in t

What is a Binder?

Binder = Mechanism for combining any variables with any terms.

Proposal: Binder = Operator on sets $F : \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$
plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Think: $F\,(V_1, \ldots, V_m, T_1, \ldots, T_n)$ combines variables $v_i \in V_i$ and terms $t_j \in T_j$ such that $v_i \in V_i$ binds in $t_j \in T_j$ if $(i, j) \in \theta$.

$$\lambda v.\, t$$

$$\text{let } v = t_1 \text{ in } t_2$$

$$\text{let rec } v_1 = t_1 \text{ and } \ldots \text{ and } v_k = t_k \text{ in } t$$

What is a Binder?

Binder = Mechanism for combining any variables with any terms.

Proposal: Binder = Operator on sets $F : \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$
plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Think: $F\,(V_1, \ldots, V_m, T_1, \ldots, T_n)$ combines variables $v_i \in V_i$ and terms $t_j \in T_j$ such that $v_i \in V_i$ binds in $t_j \in T_j$ if $(i, j) \in \theta$.

$\lambda v.\, t$

$m = n = 1$
$\mathsf{F}\,(V, T) = V \times T$
$\theta = \{(1, 1)\}$

let $v = t_1$ in t_2

let rec $v_1 = t_1$ and $\ldots$ and $v_k = t_k$ in t

What is a Binder?

Binder = Mechanism for combining any variables with any terms.

Proposal: Binder = Operator on sets $F : \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$
plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Think: $F\,(V_1, \ldots, V_m, T_1, \ldots, T_n)$ combines variables $v_i \in V_i$ and terms $t_j \in T_j$ such that $v_i \in V_i$ binds in $t_j \in T_j$ if $(i, j) \in \theta$.

$\lambda v.\, t$	$m = n = 1$ $\mathrm{F}\,(V, T) = V \times T$ $\theta = \{(1, 1)\}$
$\mathrm{let}\ v = t_1\ \mathrm{in}\ t_2$	$m = 1,\ n = 2$ $\mathrm{F}\,(V, T_1, T_2) = V \times T_1 \times T_2$ $\theta = \{(1, 2)\}$

$\mathrm{let\ rec}\ v_1 = t_1\ \mathrm{and}\ \ldots\ \mathrm{and}\ v_k = t_k\ \mathrm{in}\ t$

What is a Binder?

Binder = Mechanism for combining any variables with any terms.

Proposal: Binder = Operator on sets $F : \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$
plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Think: $F\,(V_1, \ldots, V_m, T_1, \ldots, T_n)$ combines variables $v_i \in V_i$ and terms $t_j \in T_j$ such that $v_i \in V_i$ binds in $t_j \in T_j$ if $(i, j) \in \theta$.

$\lambda v.\, t$	$m = n = 1$ $\mathsf{F}\,(V, T) = V \times T$ $\theta = \{(1, 1)\}$
let $v = t_1$ in t_2	$m = 1,\ n = 2$ $\mathsf{F}\,(V, T_1, T_2) = V \times T_1 \times T_2$ $\theta = \{(1, 2)\}$
let rec $v_1 = t_1$ and $\ldots$ and $v_k = t_k$ in t	$m = n = 1$ $\mathsf{F}\,(V, T) = \mathrm{List}\,(V \times T) \times T$ $\theta = \{(1, 1)\}$

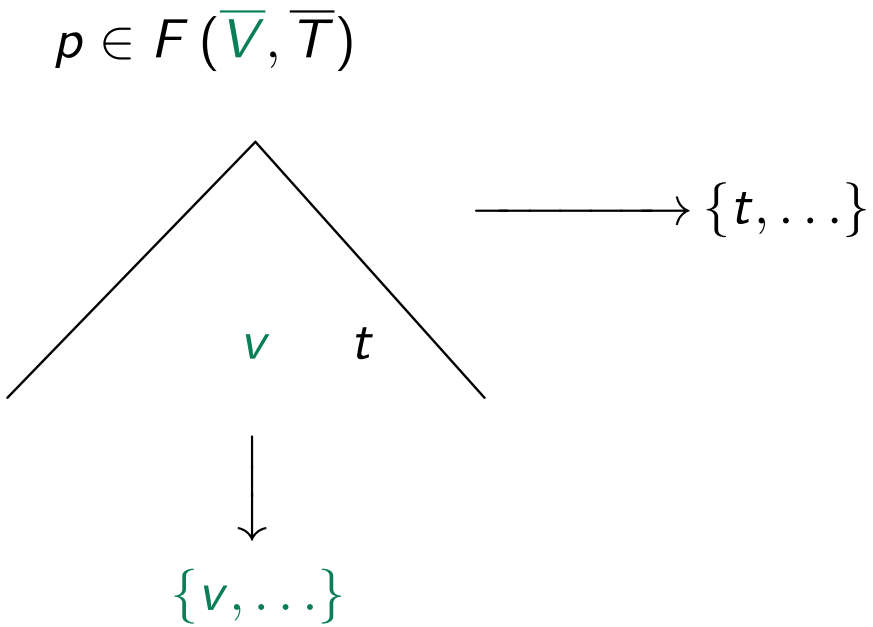
Structure of Binders

Proposal: Binder = Operator on sets $F : \mathsf{Set}^m \times \mathsf{Set}^n \to \mathsf{Set}$ plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Structure of Binders

Proposal: Binder = Operator on sets $F : \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$ plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

F "Natural" (Container-like)

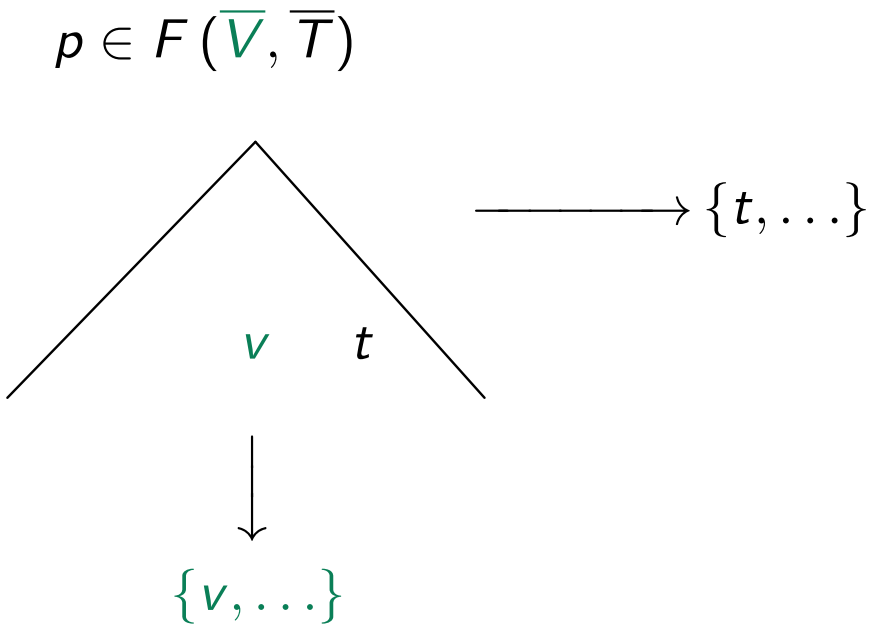


Structure of Binders

Proposal: Binder = Operator on sets $F : \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$ plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Finitary?

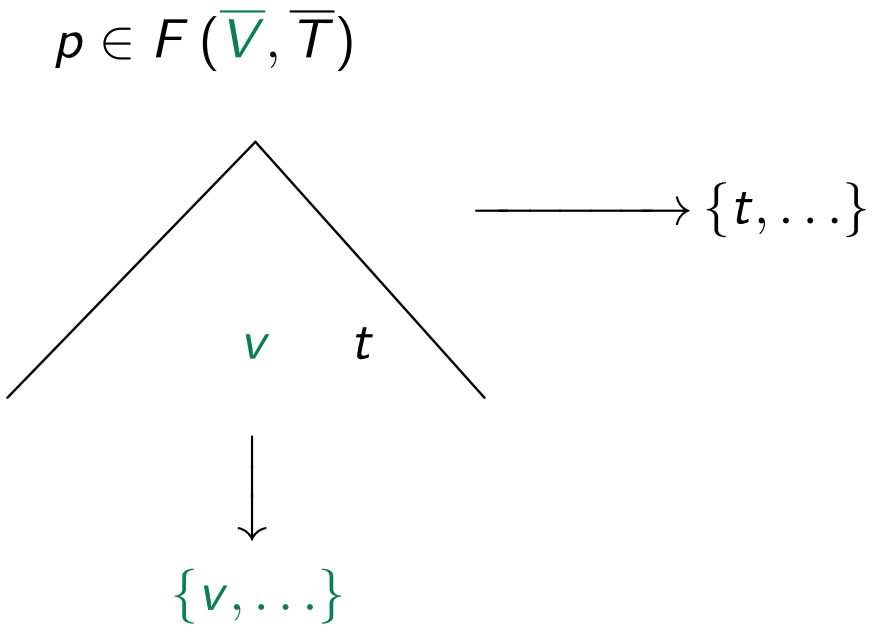
F "Natural" (Container-like)



Structure of Binders

Proposal: Binder = Operator on sets $F : \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$
plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

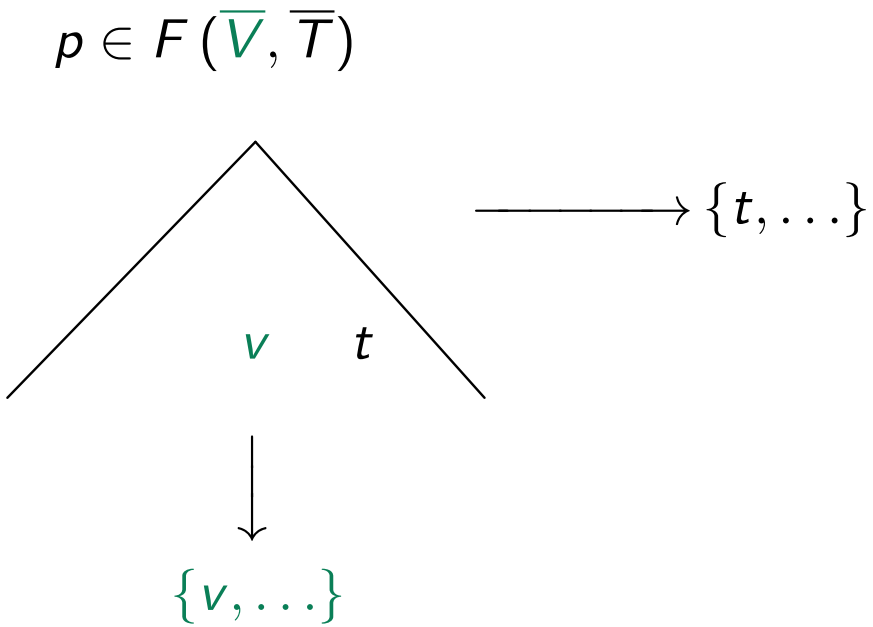
Bounded
F "Natural" (Container-like)



Structure of Binders

Proposal: Binder = Operator on sets $F : \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$ plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Bounded
F "Natural" (Container-like)
Functor?



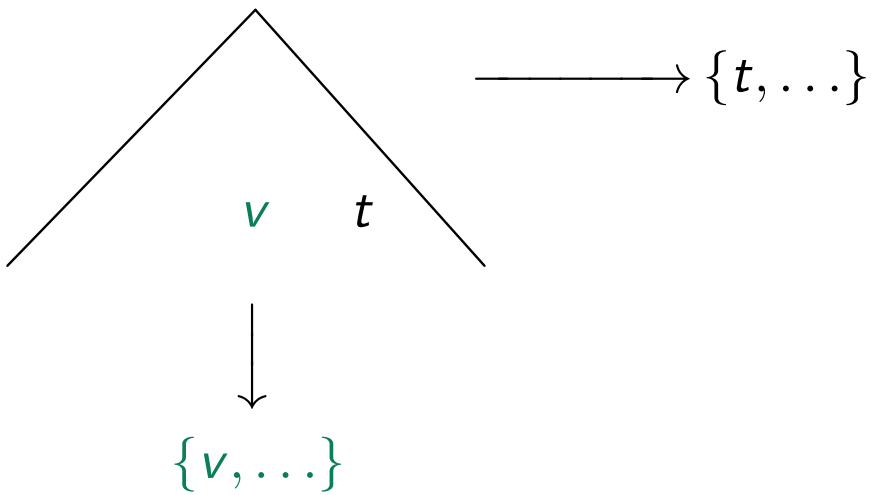
Structure of Binders

Proposal: Binder = Operator on sets $F : \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$
plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Bounded
F "Natural" (Container-like)
Functor?

$\mathsf{let\ rec}\ (v = t)^*\ \mathsf{in}\ t$	$\mathsf{F}\,(V, T) =$ $\mathsf{List}\,(V \times T) \times T$ $\theta = \{(1,1)\}$	$\mathsf{let\ rec}\ v = t_1\ \mathsf{and}\ v = t_2\ \mathsf{in}\ t$ $[(v, t_1), (v, t_2)] \in \mathsf{List}\,(V \times T)$

$$p \in F\,(\overline{V}, \overline{T})$$



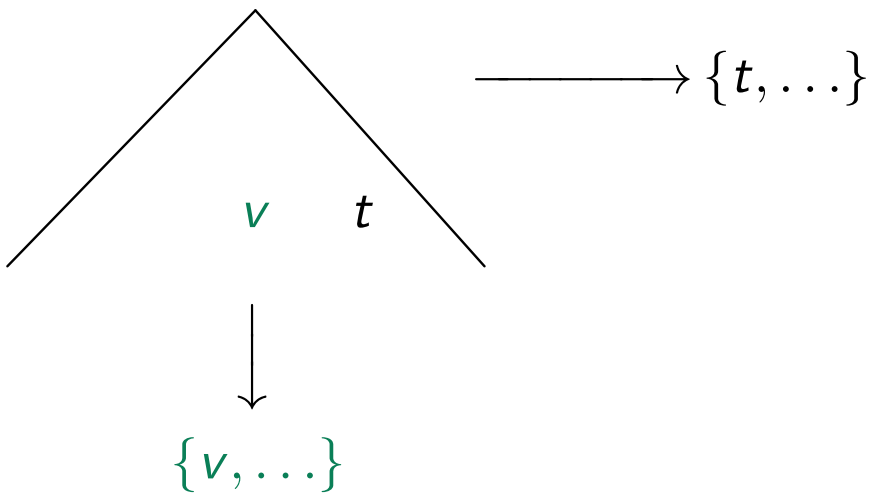
Structure of Binders

Proposal: Binder = Operator on sets $F : \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$
plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Bounded
F "Natural" (Container-like)
Functor?

let rec $(v = t)^*$ in t	$\mathsf{F}(V, T) =$ $\mathrm{List}(V \times T)^{@V} \times T$ $\theta = \{(1,1)\}$	let rec $v = t_1$ and $v = t_2$ in t $[(v, t_1), (v, t_2)] \in \mathrm{List}(V \times T)$

$$p \in F(\overline{V}, \overline{T})$$



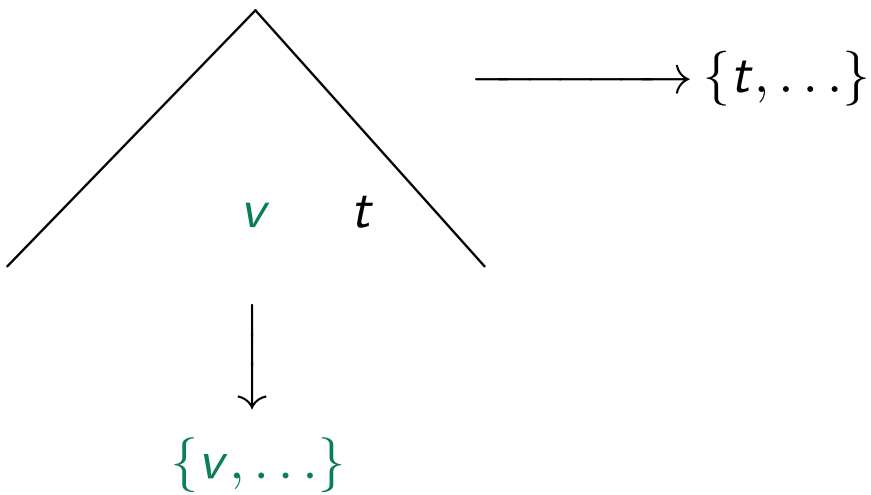
Structure of Binders

Proposal: Binder = Operator on sets $F : \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$
plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Bounded
F "Natural" (Container-like)
Functor on (binding) variable arguments only w.r.t. injections

let rec $(v = t)^*$ in t	$\mathsf{F}(V, T) =$ List $(V \times T)^{@V} \times T$ $\theta = \{(1, 1)\}$	let rec $v = t_1$ and $v = t_2$ in t $[(v, t_1), (v, t_2)] \in \mathrm{List}(V \times T)$

$$p \in F(\overline{V}, \overline{T})$$



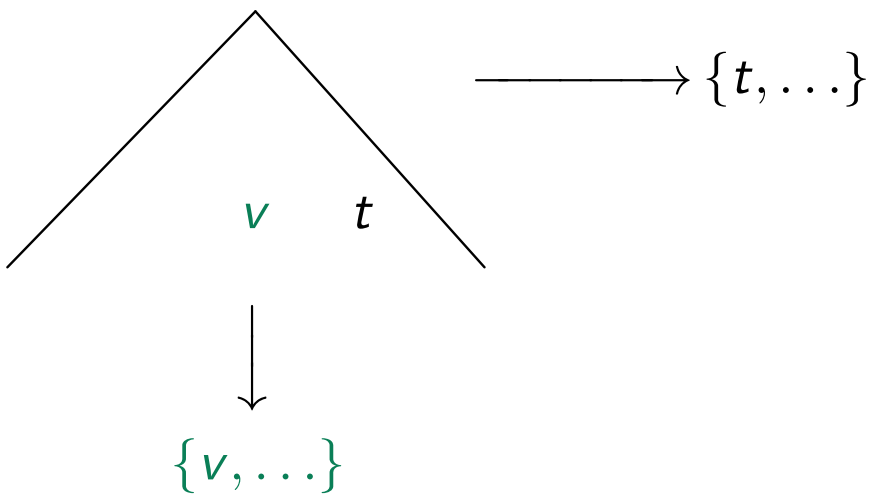
Structure of Binders

Proposal: Binder = Operator on sets $F : \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$
plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Bounded
F "Natural" (Container-like)
Functor on (binding) variable arguments only w.r.t. injections

$w\,(v).\ p$

$p \in F\,(\overline{V}, \overline{T})$



Structure of Binders

Proposal: Binder = Operator on sets $F : \mathrm{Set}^p \times \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$ plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

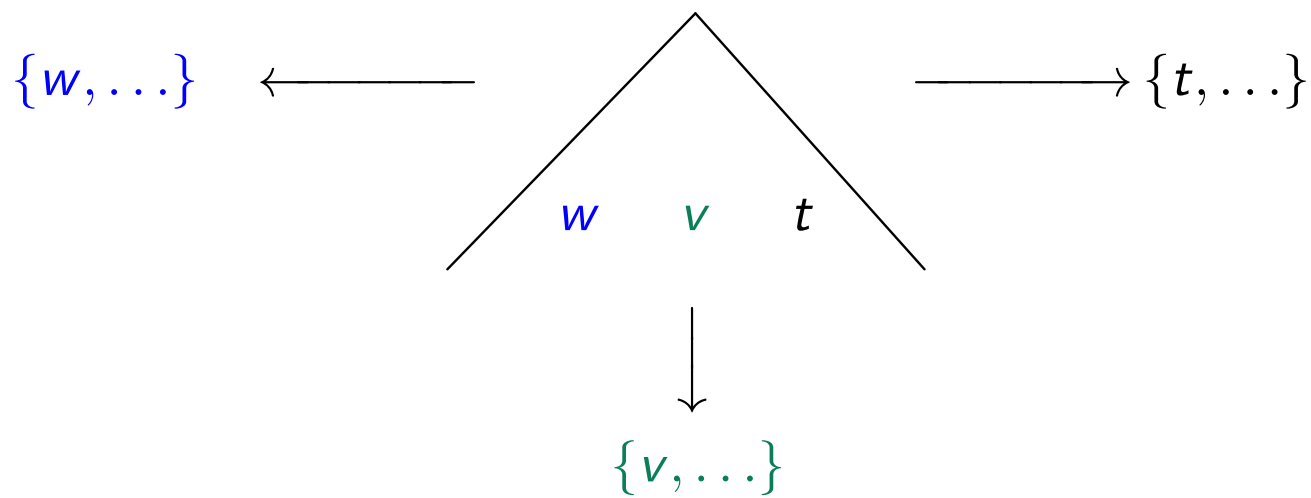
Bounded

F "Natural" (Container-like)

Functor on (binding) variable arguments only w.r.t. injections

$w\,(v).\ p$	$p = m = n = 1$ $\mathrm{F}\,(W, V, T) = W \times V \times T$ $\theta = \{(1, 1)\}$

$$p \in F\,(\overline{V}, \overline{T})$$



Parenthesis: Linearization Modifier $^{@}$

$$\mathsf{List}\,(A)^{@A} = \{xs \in \mathsf{List}\,(A) \mid \forall i, j.\ i \neq j \longrightarrow xs_i \neq xs_j\}$$

Parenthesis: Linearization Modifier $^{@}$

$$\mathsf{List}\,(A)^{@A} = \{xs \in \mathsf{List}\,(A) \mid \forall i, j.\ i \neq j \longrightarrow xs_i \neq xs_j\}$$

Linearity for an arbitrary BNF, F : Set $\rightarrow$ Set?

Parenthesis: Linearization Modifier $^{@}$

$$\mathsf{List}\,(A)^{@A} = \{xs \in \mathsf{List}\,(A) \mid \forall i, j.\ i \neq j \longrightarrow xs_i \neq xs_j\}$$

Linearity for an arbitrary BNF, F : Set $\rightarrow$ Set?

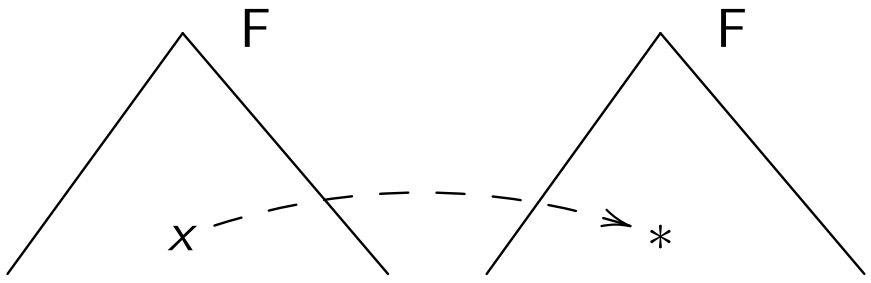
How about: $p \in F\,(A)$ linear
... if $\forall q.\ \mathsf{shape}\,(q) = \mathsf{shape}\,(p) \longrightarrow |\mathsf{supp}_{\mathsf{F}}\,(q)| \leq |\mathsf{supp}_{\mathsf{F}}\,(p)|$

Parenthesis: Linearization Modifier $^{@}$

$$\mathsf{List}\,(A)^{@A} = \{xs \in \mathsf{List}\,(A) \mid \forall i, j.\ i \neq j \longrightarrow xs_i \neq xs_j\}$$

Linearity for an arbitrary BNF, $\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$?

$$\begin{aligned} !_A &: A \to \mathsf{Unit} = \{*\} \\ \mathsf{shape} = \mathsf{map}_{\mathsf{F}}\, !_A &: \mathsf{F}\,(A) \to \mathsf{F}\,(\mathsf{Unit}) \end{aligned}$$



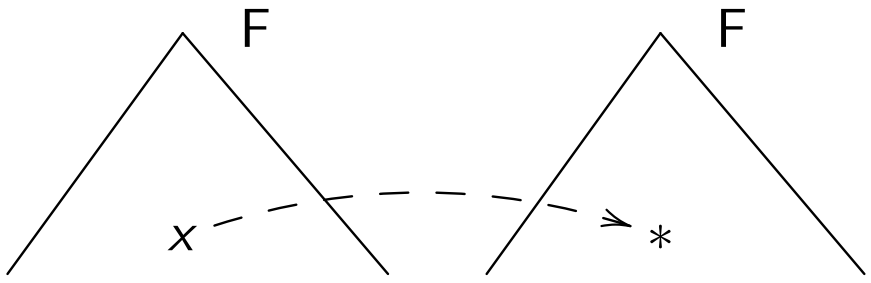
How about: $p \in F\,(A)$ linear
... if $\forall q.\ \mathsf{shape}\,(q) = \mathsf{shape}\,(p) \longrightarrow |\mathsf{supp}_{\mathsf{F}}\,(q)| \leq |\mathsf{supp}_{\mathsf{F}}\,(p)|$

Parenthesis: Linearization Modifier $^{@}$

$$\mathsf{List}\,(A)^{@A} = \{xs \in \mathsf{List}\,(A) \mid \forall i, j.\ i \neq j \longrightarrow xs_i \neq xs_j\}$$

Linearity for an arbitrary BNF, $\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$?

$$\begin{aligned} !_A &: A \to \mathsf{Unit} = \{*\} \\ \mathsf{shape} = \mathsf{map}_{\mathsf{F}}\, !_A &: \mathsf{F}\,(A) \to \mathsf{F}\,(\mathsf{Unit}) \end{aligned}$$



How about: $p \in F\,(A)$ linear
... if $\forall q.\ \mathsf{shape}\,(q) = \mathsf{shape}\,(p) \longrightarrow |\mathsf{supp}_{\mathsf{F}}\,(q)| \leq |\mathsf{supp}_{\mathsf{F}}\,(p)|$

Works for finitary functors.

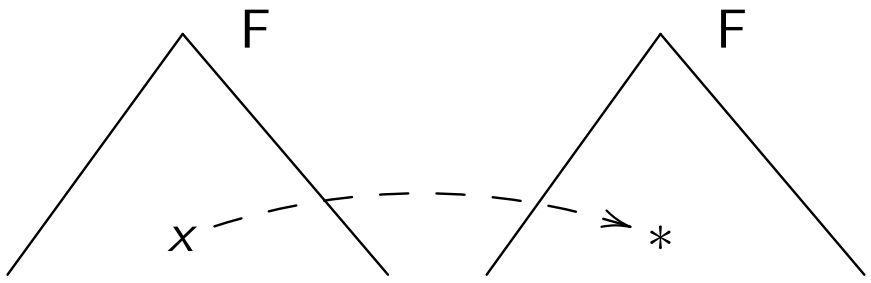
Fails in general: For $\mathsf{F} = \mathsf{Stream}$, $[0, 0, 1, 2, 3, \ldots] \in \mathsf{F}\,(\mathbb{N})$ linear.

Parenthesis: Linearization Modifier $^{@}$

$$\mathsf{List}\,(A)^{@A} = \{xs \in \mathsf{List}\,(A) \mid \forall i, j.\ i \neq j \longrightarrow xs_i \neq xs_j\}$$

Linearity for an arbitrary BNF, $\mathsf{F} : \mathsf{Set} \to \mathsf{Set}$?

$$\begin{array}{rcl} !_A & : & A \to \mathsf{Unit} = \{*\} \\ \mathsf{shape} = \mathsf{map}_{\mathsf{F}}\, !_A & : & \mathsf{F}\,(A) \to \mathsf{F}\,(\mathsf{Unit}) \end{array}$$



Better: $p \in F\,(A)$ linear
... if $\forall q.\ \mathsf{shape}\,(q) = \mathsf{shape}\,(p) \longrightarrow \exists f : A \to A.\ \mathsf{map}_{\mathsf{F}}\,(f)\,(p) = q$

Works in general.

Gives us back a sub-functor, $\mathsf{F}^{@}$, of F's restriction to bijections.

Structure of Binders (Summary)

Proposal: Binder = Operator on sets $F : \mathrm{Set}^p \times \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$ that is a Map-Restricted Bounded Natural Functor (MRBNF):

- w.r.t. arbitrary functions on the p free-variable arguments
- w.r.t. injections on the m binding-variable arguments
- w.r.t. arbitrary functions on the n "term" arguments

plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

Structure of Binders (Summary)

Proposal: Binder = Operator on sets $F : \mathrm{Set}^p \times \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$ that is a Map-Restricted Bounded Natural Functor (MRBNF):

- w.r.t. arbitrary functions on the p free-variable arguments
- w.r.t. injections on the m binding-variable arguments
- w.r.t. arbitrary functions on the n "term" arguments

plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

So far, term-agnostic: can bind in any *hypothetical* terms.

Structure of Binders (Summary)

Proposal: Binder = Operator on sets $F : \mathrm{Set}^p \times \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$ that is a Map-Restricted Bounded Natural Functor (MRBNF):

- w.r.t. arbitrary functions on the p free-variable arguments
- w.r.t. injections on the m binding-variable arguments
- w.r.t. arbitrary functions on the n "term" arguments

plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

So far, term-agnostic: can bind in any *hypothetical* terms.

Actual terms?

Structure of Binders (Summary)

Proposal: Binder = Operator on sets $F : \mathrm{Set}^p \times \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$ that is a Map-Restricted Bounded Natural Functor (MRBNF):

- w.r.t. arbitrary functions on the p free-variable arguments
- w.r.t. injections on the m binding-variable arguments
- w.r.t. arbitrary functions on the n "term" arguments

plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

So far, term-agnostic: can bind in any *hypothetical* terms.

Actual terms? Built by composing and iterating binders.

Structure of Binders (Summary)

Proposal: Binder = Operator on sets $F : \mathrm{Set}^p \times \mathrm{Set}^m \times \mathrm{Set}^n \to \mathrm{Set}$ that is a Map-Restricted Bounded Natural Functor (MRBNF):

- w.r.t. small-support functions on the p free-variable arguments
- w.r.t. small-support bijections on the m binding-vars. arguments
- w.r.t. arbitrary functions on the n "term" arguments

plus binding dispatcher relation $\theta \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$.

So far, term-agnostic: can bind in any *hypothetical* terms.

Actual terms? Built by composing and iterating binders.

Example: POPLmark Syntax Fragment

Type-variable α, term-variables x, labels l

$$
\begin{array}{llll}
\text{Types} & \sigma & ::= & \alpha \mid \ldots \\
\text{Patterns} & p & ::= & x : \sigma \mid \{l_i = p_i \,^{i \in 1 \ldots n}\} \\
\text{Terms} & t & ::= & x \mid \Lambda\alpha.\, t \mid \text{let } p = t_1 \text{ in } t_2
\end{array}
$$

Assumptions: Term-variables are pairwise distinct in any pattern.
In terms, term-variables coming from patterns and type-variables near Λ's are binding.

Example: POPLmark Syntax Fragment

Type-variable α, term-variables x, labels l

$$
\begin{array}{lrcl}
\text{Types} & \sigma & ::= & \alpha \mid \ldots \\
\text{Patterns} & p & ::= & x : \sigma \mid \{l_i = p_i \,^{i \in 1 \ldots n}\} \\
\text{Terms} & t & ::= & x \mid \Lambda\alpha.\, t \mid \text{let } p = t_1 \text{ in } t_2
\end{array}
$$

Assumptions: Term-variables are pairwise distinct in any pattern.
In terms, term-variables coming from patterns and type-variables near Λ's are binding.

$$
\begin{array}{rcl}
\text{Type}\,(A) & = & \ldots \\
\text{Pattern}\,(A, X) & = & (\mu P.\, X \times \text{Type}\,(A) + \text{FinPFunc}\,(\text{Label}, P))^{@X}
\end{array}
$$

Example: POPLmark Syntax Fragment

Type-variable α, term-variables x, labels l

$$
\begin{array}{lrcl}
\text{Types} & \sigma & ::= & \alpha \mid \ldots \\
\text{Patterns} & p & ::= & x : \sigma \mid \{l_i = p_i \; {}^{i \in 1 \ldots n}\} \\
\text{Terms} & t & ::= & x \mid \Lambda\alpha.\, t \mid \text{let } p = t_1 \text{ in } t_2
\end{array}
$$

Assumptions: Term-variables are pairwise distinct in any pattern.
In terms, term-variables coming from patterns and type-variables near Λ's are binding.

$$
\begin{array}{rcl}
\text{Type}(A) & = & \ldots \\
\text{Pattern}(A, X) & = & (\mu P.\, X \times \text{Type}(A) + \text{FinPFunc}(\text{Label}, P))^{@X} \\
\text{Term}(A, X) & = & \mu_\theta T.\, X + A \times T + \text{Pattern}(A, X) \times T^2
\end{array}
$$

Example: POPLmark Syntax Fragment

Type-variable α, term-variables x, labels l

$$\begin{array}{lrcl} \text{Types} & \sigma & ::= & \alpha \mid \ldots \\ \text{Patterns} & p & ::= & x : \sigma \mid \{l_i = p_i \,^{i \in 1 \ldots n}\} \\ \text{Terms} & t & ::= & x \mid \Lambda\alpha.\, t \mid \text{let } p = t_1 \text{ in } t_2 \end{array}$$

Assumptions: Term-variables are pairwise distinct in any pattern.
In terms, term-variables coming from patterns and type-variables near Λ's are binding.

$$\begin{aligned} \text{Type}(A) &= \ldots \\ \text{Pattern}(A, X) &= (\mu P.\, X \times \text{Type}(A) + \text{FinPFunc}(\text{Label}, P))^{@X} \\ \text{Term}(A, X) &= \mu_\theta T.\, X + A \times T + \text{Pattern}(A, X) \times T^2 \\ &= \mu_\theta T.\, F(A, X, A, X, T) \end{aligned}$$

where:

$$F(A', X', A, X, T) = X' + A \times T + \text{Pattern}(A', X) \times T^2$$

$$\theta = \{(1,1), (2,1)\}$$

Example: POPLmark Syntax Fragment

Type-variable α, term-variables x, labels l

$$\begin{array}{lrcl} \text{Types} & \sigma & ::= & \alpha \mid \ldots \\ \text{Patterns} & p & ::= & x : \sigma \mid \{l_i = p_i \,^{i \in 1 \ldots n}\} \\ \text{Terms} & t & ::= & x \mid \Lambda\alpha.\, t \mid \text{let } p = t_1 \text{ in } t_2 \end{array}$$

Assumptions: Term-variables are pairwise distinct in any pattern.
In terms, term-variables coming from patterns and type-variables near Λ's are binding.

$$\begin{aligned} \text{Type}(A) &= \ldots \\ \text{Pattern}(A, X) &= (\mu P.\, X \times \text{Type}(A) + \text{FinPFunc}(\text{Label}, P))^{@X} \\ \text{Term}(A, X) &= \mu_\theta T.\, X + A \times T + \text{Pattern}(A, X) \times T^2 \\ &= \mu_\theta T.\, F(A, X, A, X, T) \end{aligned}$$

where:

$$F(A', X', A, X, T) = X' + A \times T + \text{Pattern}(A', X) \times T^2$$

$$\theta = \{(1,1), (2,1)\}$$

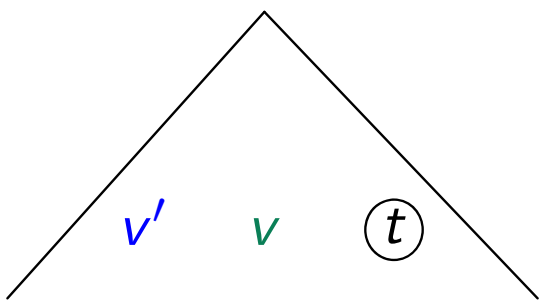
From Binders to Terms with Bindings

Constructing Terms from Binders

$$F : \mathsf{Set}^{p} \times \mathsf{Set}^{m} \times \mathsf{Set}^{n} \to \mathsf{Set}$$

Assume $p = m$.

$$\mathsf{T}\,(\overline{V}) \;=\; \mu\overline{A}.\; \mathsf{F}(\overline{V}, \overline{V}, \overline{A})$$

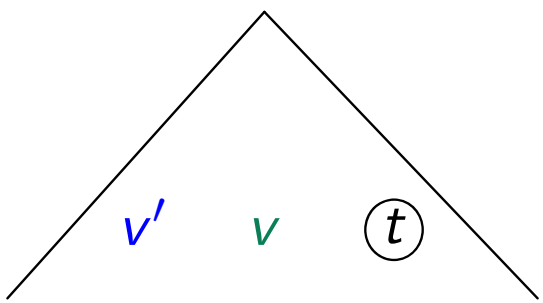


Constructing Terms from Binders

$$F : \mathsf{Set}^{p} \times \mathsf{Set}^{m} \times \mathsf{Set}^{n} \to \mathsf{Set}$$

Assume $p = m$.

Raw terms: $\mathsf{T}\,(\overline{V}) \;=\; \mu\overline{A}.\; \mathsf{F}(\overline{V}, \overline{V}, \overline{A})$

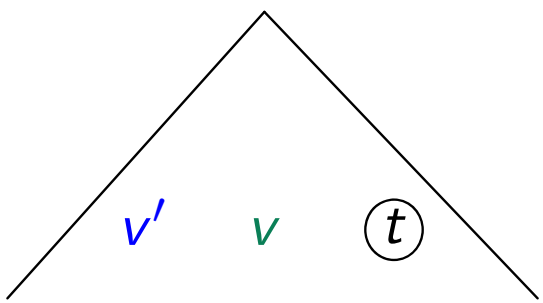


Constructing Terms from Binders

$$F : \mathsf{Set}^{p} \times \mathsf{Set}^{m} \times \mathsf{Set}^{n} \to \mathsf{Set}$$

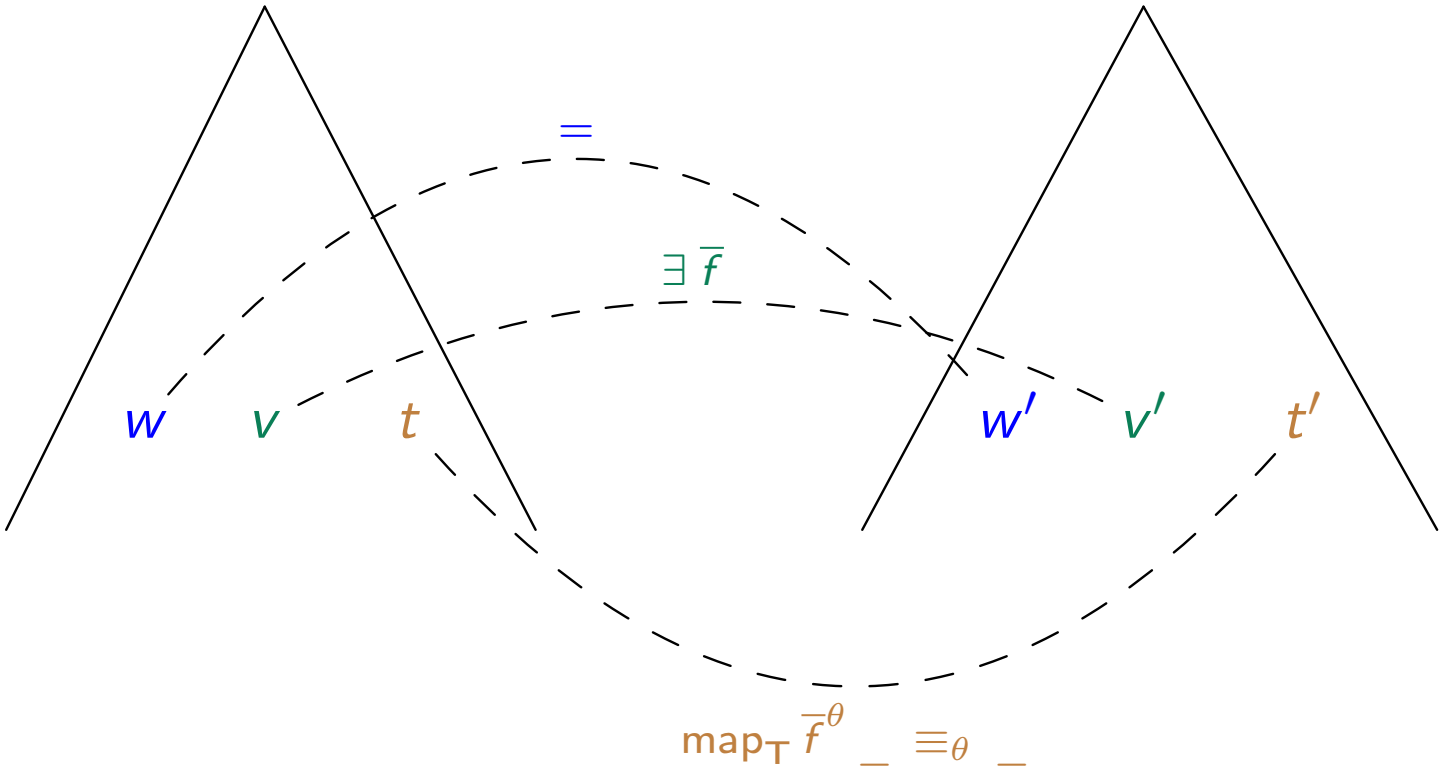
Assume $p = m$.

Raw terms: $\mathsf{T}\,(\overline{V}) \;=\; \mu\overline{A}.\; \mathsf{F}(\overline{V}, \overline{V}, \overline{A})$



Alpha-quotiented terms: $\overline{\mathbf{T}\,(\overline{V})} \;=\; \overline{\mathsf{T}\,(\overline{V})\,/{\equiv_{\theta}}}$

Inductive Definition of Alpha-Equivalence

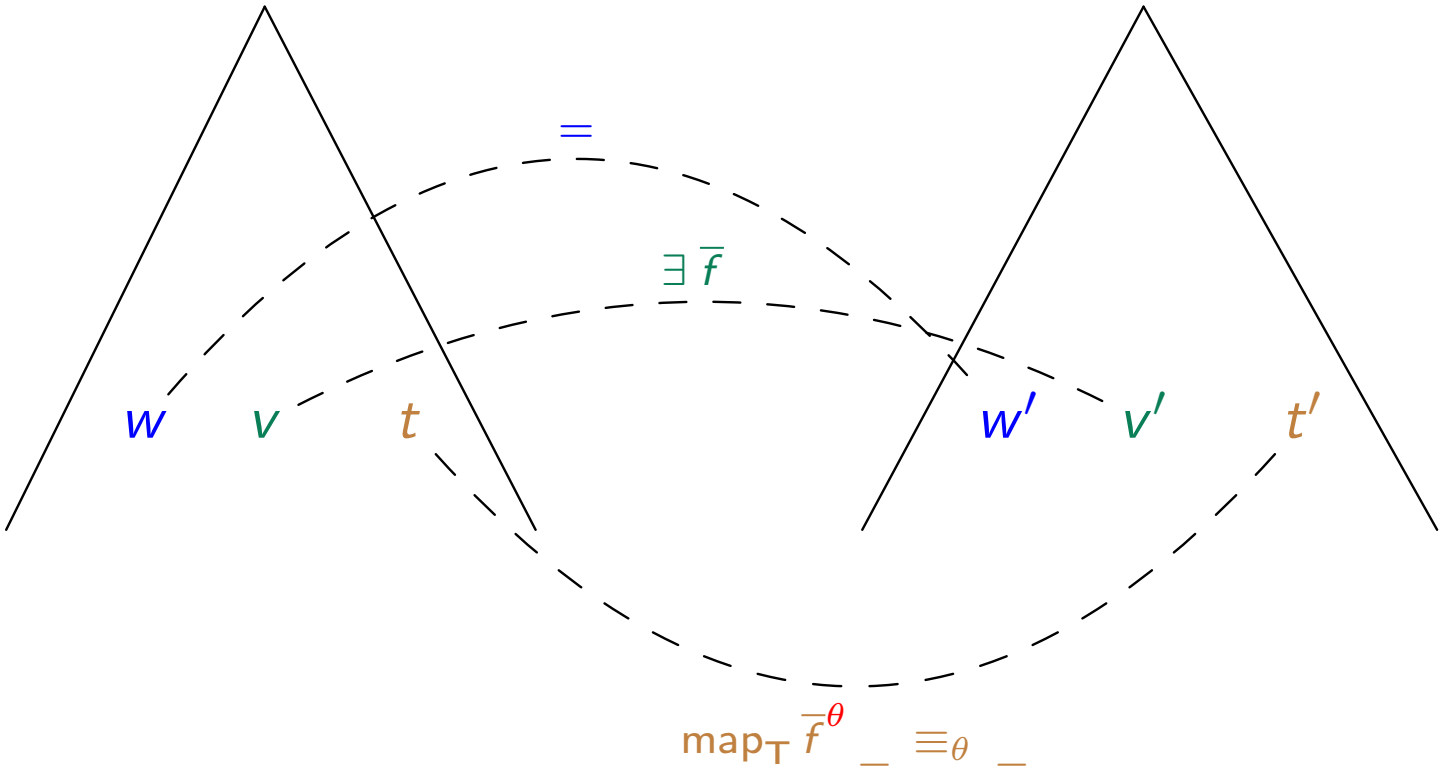


Equality on the top free variables

Possible bijective renamings of top binding variables

Recursive call factoring in the renamings

Inductive Definition of Alpha-Equivalence



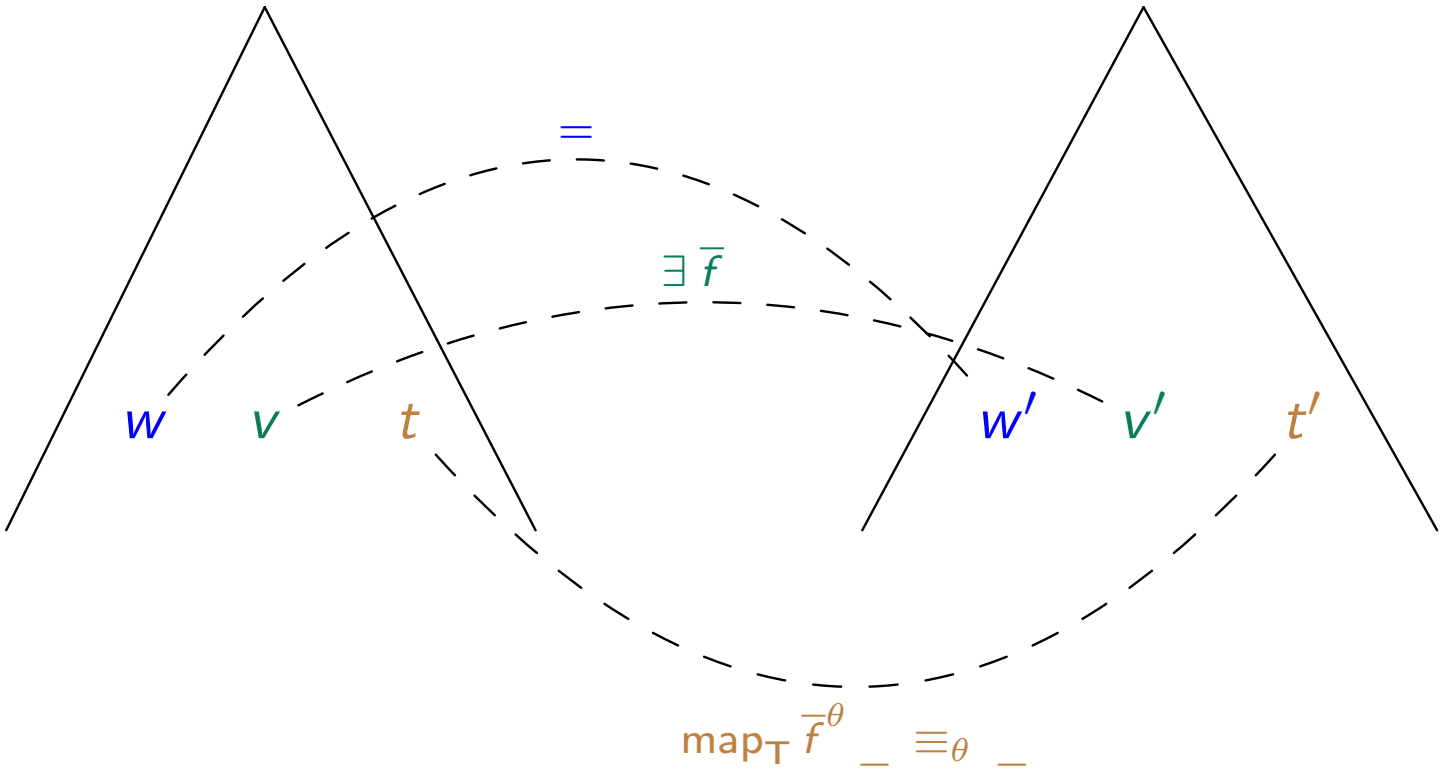
Renaming via f_i of v_i in t_j only if $(i, j) \in \theta$

Equality on the top free variables

Possible bijective renamings of top binding variables

Recursive call factoring in the renamings

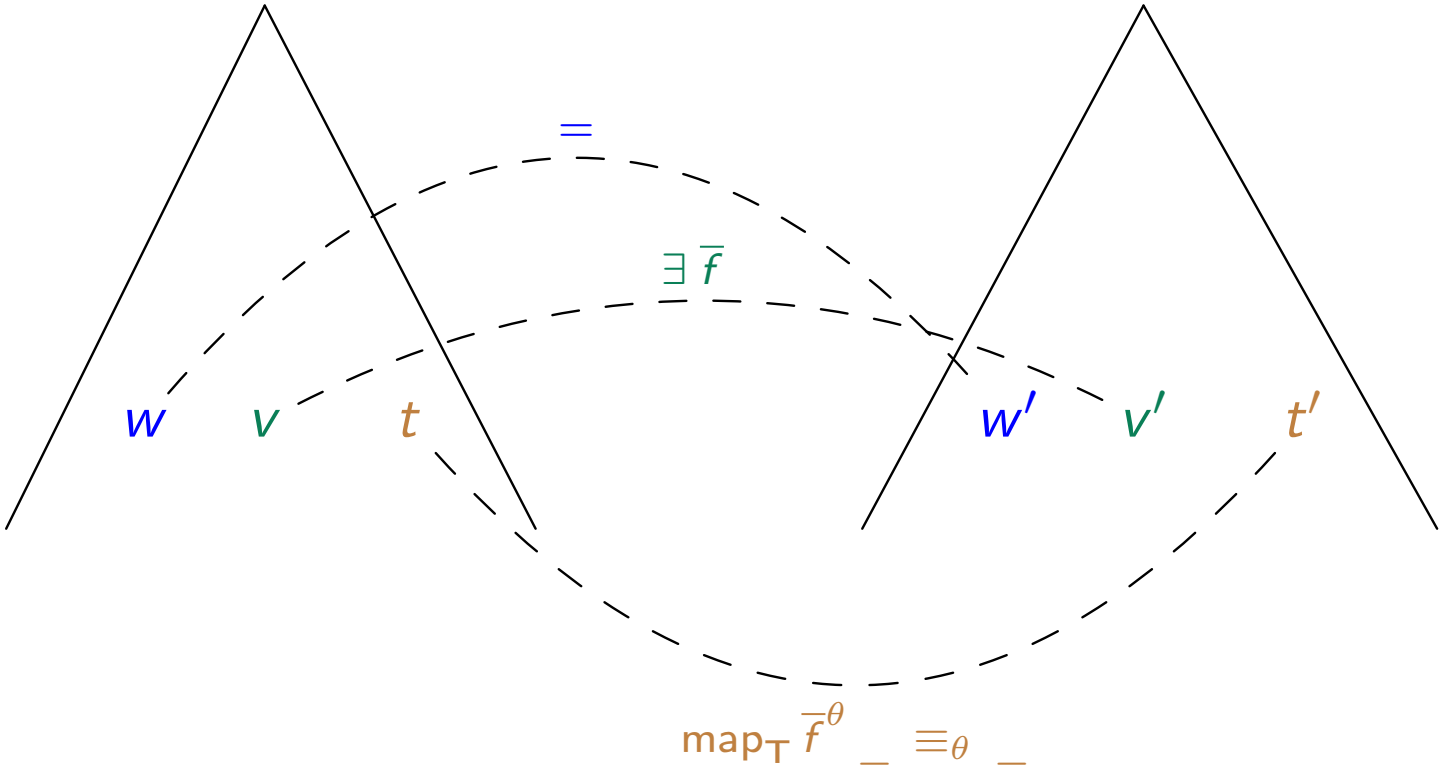
Inductive Definition of Alpha-Equivalence



F being a relator is crucial:

- Equality on the top free variables
- Possible bijective renamings of top binding variables
- Recursive call factoring in the renamings

Inductive Definition of Alpha-Equivalence



F being a relator is crucial:

$$\frac{(\mathsf{unf}(t), \mathsf{unf}(t')) \in \mathsf{rel}_{\mathsf{F}}\ (=)\ \{(\overline{v}, \overline{f\ v}) \mid \ldots\}\ \{(\overline{t}, \overline{t'}) \mid \mathsf{map}_{\mathsf{T}}\ \overline{f}^{\theta}\ \overline{t} \equiv_{\Theta} \overline{t'}\}}{t \equiv_{\theta} t'}$$

Abstract Characterization of Alpha-Quotinented Terms?

$$
\begin{array}{lcll}
\mathsf{T}\,(\overline{V}) & = & \mu A.\ \mathsf{F}(\overline{V}, \overline{V}, A) & \text{OK} \\
\mathbf{T}\,(\overline{V}) & = & \mathsf{T}\,(\overline{V}) \,/\, \equiv_\theta & \text{too low-level}
\end{array}
$$

Abstract Characterization of Alpha-Quotinented Terms?

$$\begin{array}{lcll} \mathsf{T}\,(\overline{V}) & = & \mu A.\ \mathsf{F}(\overline{V}, \overline{V}, A) & \text{OK} \\ \mathbf{T}\,(\overline{V}) & = & \mathsf{T}\,(\overline{V}) \,/\, \equiv_\theta & \text{too low-level} \end{array}$$

Operators on $\mathbf{T}$:

- ctor : $\mathsf{F}\,(\overline{V}, \overline{V}, \mathbf{T}\,(\overline{V})) \to \mathbf{T}\,(\overline{V})$ non-injective constructor
- FVars_i : $\mathbf{T}\,(\overline{V}) \to V_i$
- $\mathsf{map}_{\mathbf{T}}$ functorial action on $\mathbf{T}$ w.r.t. bijections

Theorem: $(\mathbf{T}, \overline{\mathsf{FVars}}, \mathsf{map}_{\mathbf{T}}, \mathsf{ctor})$ is the initial object in a category of models $\mathcal{U} = (\mathsf{U}, \overline{\mathsf{UFVars}}, \mathsf{Umap}, \mathsf{Uctor})$ satisfying:

- Umap functorial on bijections
- Umap and UFVars_i distribute over Uctor
- Umap satisfies congruence w.r.t. UFVars_i

Abstract Characterization of Alpha-Quotinented Terms?

$$\begin{array}{lcll} \mathsf{T}\,(\overline{V}) & = & \mu A.\ \mathsf{F}(\overline{V}, \overline{V}, A) & \text{OK} \\ \mathbf{T}\,(\overline{V}) & = & \mathsf{T}\,(\overline{V}) \,/\, \equiv_\theta & \text{too low-level} \end{array}$$

Operators on $\mathbf{T}$:

- ctor : $\mathsf{F}\,(\overline{V}, \overline{V}, \mathbf{T}\,(\overline{V})) \to \mathbf{T}\,(\overline{V})$ non-injective constructor
- FVars_i : $\mathbf{T}\,(\overline{V}) \to V_i$
- $\mathsf{map}_{\mathbf{T}}$ functorial action on $\mathbf{T}$ w.r.t. bijections

Theorem: $(\mathbf{T}, \overline{\mathsf{FVars}}, \mathsf{map}_{\mathbf{T}}, \mathsf{ctor})$ is the initial object in a category of models $\mathcal{U} = (\mathsf{U}, \overline{\mathsf{UFVars}}, \mathsf{Umap}, \mathsf{Uctor})$ satisfying:

- Umap functorial on bijections
- Umap and UFVars_i distribute over Uctor
- Umap satisfies congruence w.r.t. UFVars_i

$$\Downarrow$$

Recursor generalizing the state-of-the-art nominal recursors (Norrish 2004, Pitts 2006, Urban and Berghofer 2006, GP 2017)

Abstract Characterization of Alpha-Quotinented Terms?

$$\boxed{\text{Notation: } \mathbf{T}\,(\overline{V}) \;=\; \mu_\theta A.\; \mathsf{F}(\overline{V}, \overline{V}, A)}$$

Operators on $\mathbf{T}$:

- ctor : $\mathsf{F}\,(\overline{V}, \overline{V}, \mathbf{T}\,(\overline{V})) \to \mathbf{T}\,(\overline{V})$ non-injective constructor
- FVars_i : $\mathbf{T}\,(\overline{V}) \to V_i$
- $\mathsf{map}_{\mathbf{T}}$ functorial action on $\mathbf{T}$ w.r.t. bijections

Theorem: $(\mathbf{T}, \overline{\mathsf{FVars}}, \mathsf{map}_{\mathbf{T}}, \mathsf{ctor})$ is the initial object in a category of models $\mathcal{U} = (\mathsf{U}, \overline{\mathsf{UFVars}}, \mathsf{Umap}, \mathsf{Uctor})$ satisfying:

- Umap functorial on bijections
- Umap and UFVars_i distribute over Uctor
- Umap satisfies congruence w.r.t. UFVars_i

$$\Downarrow$$

Recursor generalizing the state-of-the-art nominal recursors (Norrish 2004, Pitts 2006, Urban and Berghofer 2006, GP 2017)

Parenthesis: How the Recursors Work

Parenthesis: How the Recursors Work

Example – HOAS encoding

$\# : \mathsf{Term}_\lambda \to \mathsf{Term}_\lambda(\mathsf{app}, \mathsf{lam})$

(1) $x^\# = x$

(2) $(s\,t)^\# = \mathsf{app}\, s^\#\, t^\#$

(3) $(\lambda x.\ t)^\# = \mathsf{lam}\,(\lambda x.\ t^\#)$

Why is this a correct recursive definition?

Parenthesis: How the Recursors Work

Example – HOAS encoding

$\# : \mathsf{Term}_\lambda \to \mathsf{Term}_\lambda(\mathsf{app}, \mathsf{lam})$

(1) $x^\# = x$

(2) $(s\,t)^\# = \mathsf{app}\, s^\#\, t^\#$

(3) $(\lambda x.\ t)^\# = \mathsf{lam}\,(\lambda x.\ t^\#)$

Why is this a correct recursive definition?

To answer this, we must also ask:

How should # behave w.r.t. free variables?

Parenthesis: How the Recursors Work

Example – HOAS encoding

$\# : \mathsf{Term}_\lambda \to \mathsf{Term}_\lambda(\mathsf{app}, \mathsf{lam})$

(1) $x^\# = x$

(2) $(s\ t)^\# = \mathsf{app}\ s^\#\ t^\#$

(3) $(\lambda x.\ t)^\# = \mathsf{lam}\ (\lambda x.\ t^\#)$

Why is this a correct recursive definition?

To answer this, we must also ask:

How should # behave w.r.t. free variables?

(4) $\mathsf{FV}(t^\#) \subseteq \mathsf{FV}(t)$

Parenthesis: How the Recursors Work

Example – HOAS encoding

$\# : \mathsf{Term}_\lambda \to \mathsf{Term}_\lambda(\mathsf{app}, \mathsf{lam})$

(1) $x^\# = x$

(2) $(s\,t)^\# = \mathsf{app}\, s^\#\, t^\#$

(3) $(\lambda x.\ t)^\# = \mathsf{lam}\,(\lambda x.\ t^\#)$

Why is this a correct recursive definition?

To answer this, we must also ask:

How should # behave w.r.t. free variables?

(4) $\mathsf{FV}(t^\#) \subseteq \mathsf{FV}(t)$

How should # behave w.r.t. swapping?

Parenthesis: How the Recursors Work

Example – HOAS encoding

$\# : \mathsf{Term}_\lambda \to \mathsf{Term}_\lambda(\mathsf{app}, \mathsf{lam})$

(1) $x^\# = x$

(2) $(s\,t)^\# = \mathsf{app}\, s^\#\, t^\#$

(3) $(\lambda x.\ t)^\# = \mathsf{lam}\,(\lambda x.\ t^\#)$

Why is this a correct recursive definition?

To answer this, we must also ask:

How should $\#$ behave w.r.t. free variables?

(4) $\mathsf{FV}(t^\#) \subseteq \mathsf{FV}(t)$

How should $\#$ behave w.r.t. swapping?

(5) $t[x \leftrightarrow y]^\# = t^\#[x \leftrightarrow y]$

Parenthesis: How the Recursors Work

Example – HOAS encoding

$\# : \mathsf{Term}_\lambda \to \mathsf{Term}_\lambda(\mathsf{app}, \mathsf{lam})$

(1) $x^\# = x$

(2) $(s\,t)^\# = \mathsf{app}\, s^\#\, t^\#$

(3) $(\lambda x.\ t)^\# = \mathsf{lam}\,(\lambda x.\ t^\#)$

Why is this a correct recursive definition?

To answer this, we must also ask:

How should # behave w.r.t. free variables?

(4) $\mathsf{FV}(t^\#) \subseteq \mathsf{FV}(t)$

How should # behave w.r.t. swapping?

(5) $t[x \leftrightarrow y]^\# = t^\#[x \leftrightarrow y]$

Alternatively: How should # behave w.r.t. substitution?

(5') $t[s/x]^\# = t^\#[s^\#/x]$ (btw, this part of adequacy)

Parenthesis: How the Recursors Work

Example – HOAS encoding

$\# : \mathsf{Term}_\lambda \to \mathsf{Term}_\lambda(\mathsf{app}, \mathsf{lam})$

(1) $x^\# = x$

(2) $(s\ t)^\# = \mathsf{app}\ s^\#\ t^\#$

(3) $(\lambda x.\ t)^\# = \mathsf{lam}\ (\lambda x.\ t^\#)$

Why is this a correct recursive definition?

To answer this, we must also ask:

How should # behave w.r.t. free variables?

(4) $\mathsf{FV}(t^\#) \subseteq \mathsf{FV}(t)$

How should # behave w.r.t. swapping?

(5) $t[x \leftrightarrow y]^\# = t^\#[x \leftrightarrow y]$

Alternatively: How should # behave w.r.t. substitution?

(5') $t[s/x]^\# = t^\#[s^\#/x]$ (btw, this part of adequacy)

Must check some conditions – here, easy, as in Isabelle's "by auto".

Parenthesis: How the Recursors Work

Example – HOAS encoding

$\# : \mathsf{Term}_\lambda \to \mathsf{Term}_\lambda(\mathsf{app}, \mathsf{lam})$

(1) $x^\# = x$

(2) $(s\ t)^\# = \mathsf{app}\ s^\#\ t^\#$

(3) $(\lambda x.\ t)^\# = \mathsf{lam}\ (\lambda x.\ t^\#)$

Why is this a correct recursive definition?

To answer this, we must also ask:

How should # behave w.r.t. free variables?

(4) $\mathsf{FV}(t^\#) \subseteq \mathsf{FV}(t)$

How should # behave w.r.t. swapping?

(5) $t[x \leftrightarrow y]^\# = t^\#[x \leftrightarrow y]$

Alternatively: How should # behave w.r.t. substitution?

(5') $t[s/x]^\# = t^\#[s^\#/x]$ (btw, this part of adequacy)

Must check some conditions – here, easy, as in Isabelle's "by auto".

End product: $\exists!$ # satisfying (1-4), and (5) or (5') or both.

Parenthesis: How the Recursors Work

Example: Interpretation of FOL formulas

sem : Fmla $\to$ (Var $\to$ M) $\to$ Bool

sem $(\forall x.\ \varphi)$ (ξ) defined as

sem (φ) $(\xi[a \leftarrow x])$ for all $a \in M$

Parenthesis: How the Recursors Work

Example: Interpretation of FOL formulas

sem : Fmla $\to$ (Var $\to$ M) $\to$ Bool

sem $(\forall x.\ \varphi)\ (\xi)$ defined as

sem $(\varphi)\ (\xi[a \leftarrow x])$ for all $a \in M$

To "convince" the recursor this is correct, we declare:

If $x \notin \mathrm{FV}(\varphi)$ and $\xi =_{-x} \xi'$, then sem $(\varphi)\ (\xi) =$ sem $(\varphi)\ (\xi')$

sem $(\varphi[t/x])\ \xi =$ sem $(\varphi)\ (\xi[x \leftarrow \mathrm{sem}(t)])$

Again, the necessary checks are trivial.

Parenthesis: How the Recursors Work

Example: Interpretation of FOL formulas

sem : Fmla $\to$ (Var $\to$ M) $\to$ Bool

sem $(\forall x.\ \varphi)$ (ξ) defined as

sem (φ) $(\xi[a \leftarrow x])$ for all $a \in M$

To "convince" the recursor this is correct, we declare:

If $x \notin \mathrm{FV}(\varphi)$ and $\xi =_{-x} \xi'$, then sem (φ) $(\xi) =$ sem (φ) (ξ')

sem $(\varphi[t/x])$ $\xi =$ sem (φ) $(\xi[x \leftarrow \mathrm{sem}(t)])$

Again, the necessary checks are trivial.

"That was not too hard: I have my (α-preserving) semantic interpretation defined, its dependence on free vars proved, and its substitution lemma proved, all in one go. Now I can move on and do interesting things."

Infinitary Terms with Bindings?

Occasionally useful

- Infinitely branching process sums in Milner's CSS: $\sum_{i \in I} P_i$
- Infinitary logics
- Böhm trees: λ-terms with possibly infinite depth
- Fully abstract π-calculus trees (via "unfolding" process terms)

Infinitary Terms with Bindings?

Occasionally useful

- Infinitely branching process sums in Milner's CSS: $\sum_{i \in I} P_i$
- Infinitary logics
- Böhm trees: λ-terms with possibly infinite depth
- Fully abstract π-calculus trees (via "unfolding" process terms)

Already in the scope of what I've shown

Infinitary Terms with Bindings?

Occasionally useful

- Infinitely branching process sums in Milner's CSS: $\sum_{i \in I} P_i$
- Infinitary logics
- Böhm trees: λ-terms with possibly infinite depth
- Fully abstract π-calculus trees (via "unfolding" process terms)

Already in the scope of what I've shown
Covered by binding-aware greatest fixpoints

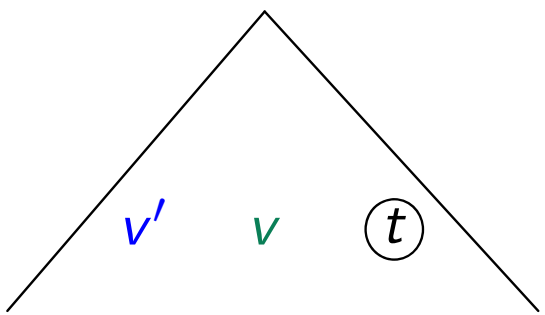
Binding-Aware Greatest Fixpoints?

Recall the Binding-Aware Least Fixpoints

$$F : \mathrm{Set}^p \times \mathrm{Set}^m \times \mathrm{Set} \to \mathrm{Set}$$

Assume $p = m$.

Raw terms: $\overline{\mathrm{T}\,(\overline{V})} = \mu\overline{A}.\ \mathrm{F}(\overline{V}, \overline{V}, \overline{A})$



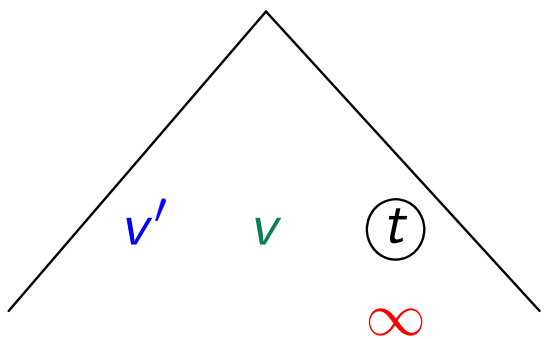
Alpha-quotiented terms: $\overline{\mathbf{T}\,(\overline{V})} = \overline{\mathrm{T}\,(\overline{V})\,/{\equiv_\theta}}$

Binding-Aware Greatest Fixpoints?

$$F : \mathrm{Set}^{p} \times \mathrm{Set}^{m} \times \mathrm{Set} \to \mathrm{Set}$$

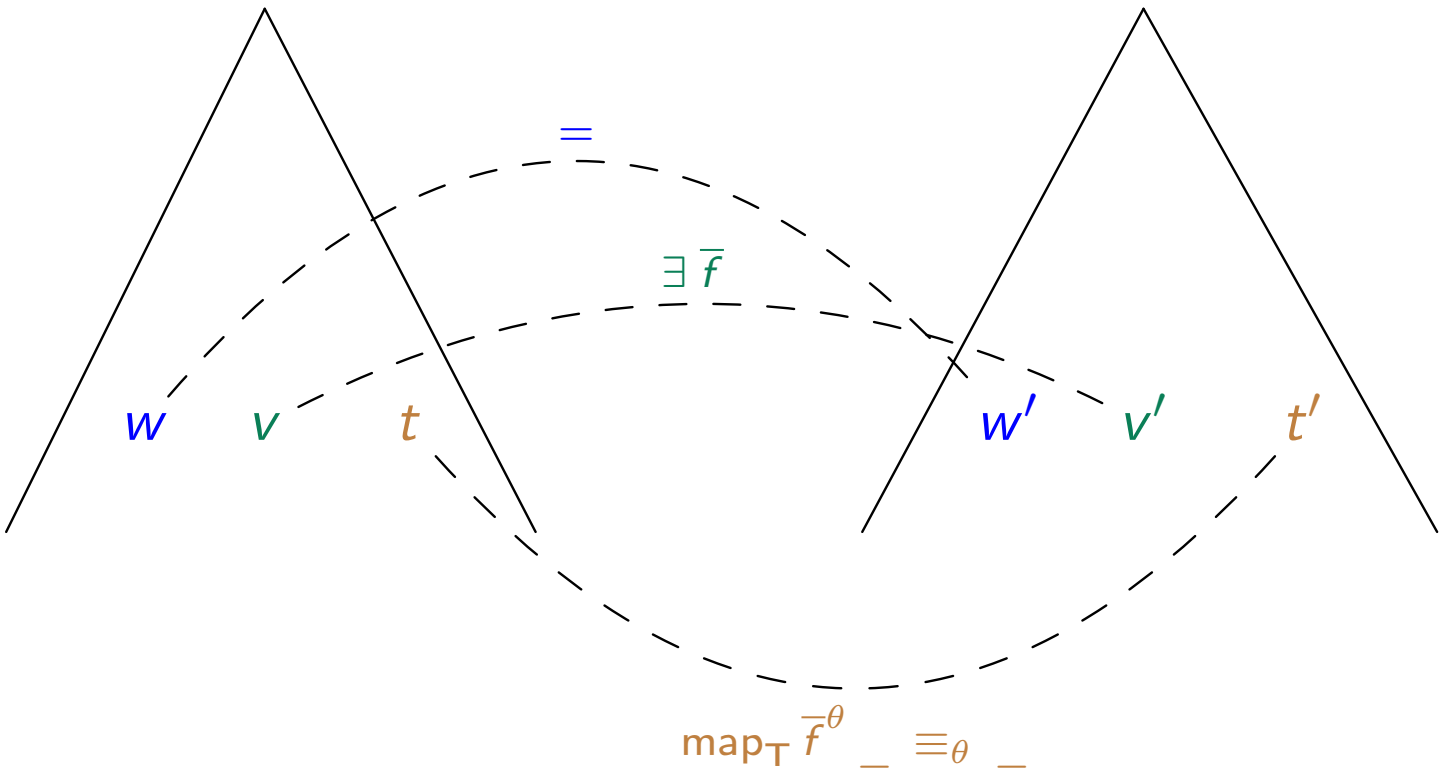
Assume $p = m$.

Raw terms: $\overline{\mathrm{T}\,(\overline{V})} = \nu \overline{A}.\ \mathrm{F}(\overline{V}, \overline{V}, \overline{A})$



Alpha-quotiented terms: $\overline{\mathbf{T}\,(\overline{V})} = \overline{\mathrm{T}\,(\overline{V})\,/{\equiv_\theta}}$

Inductive Definition of Alpha-Equivalence



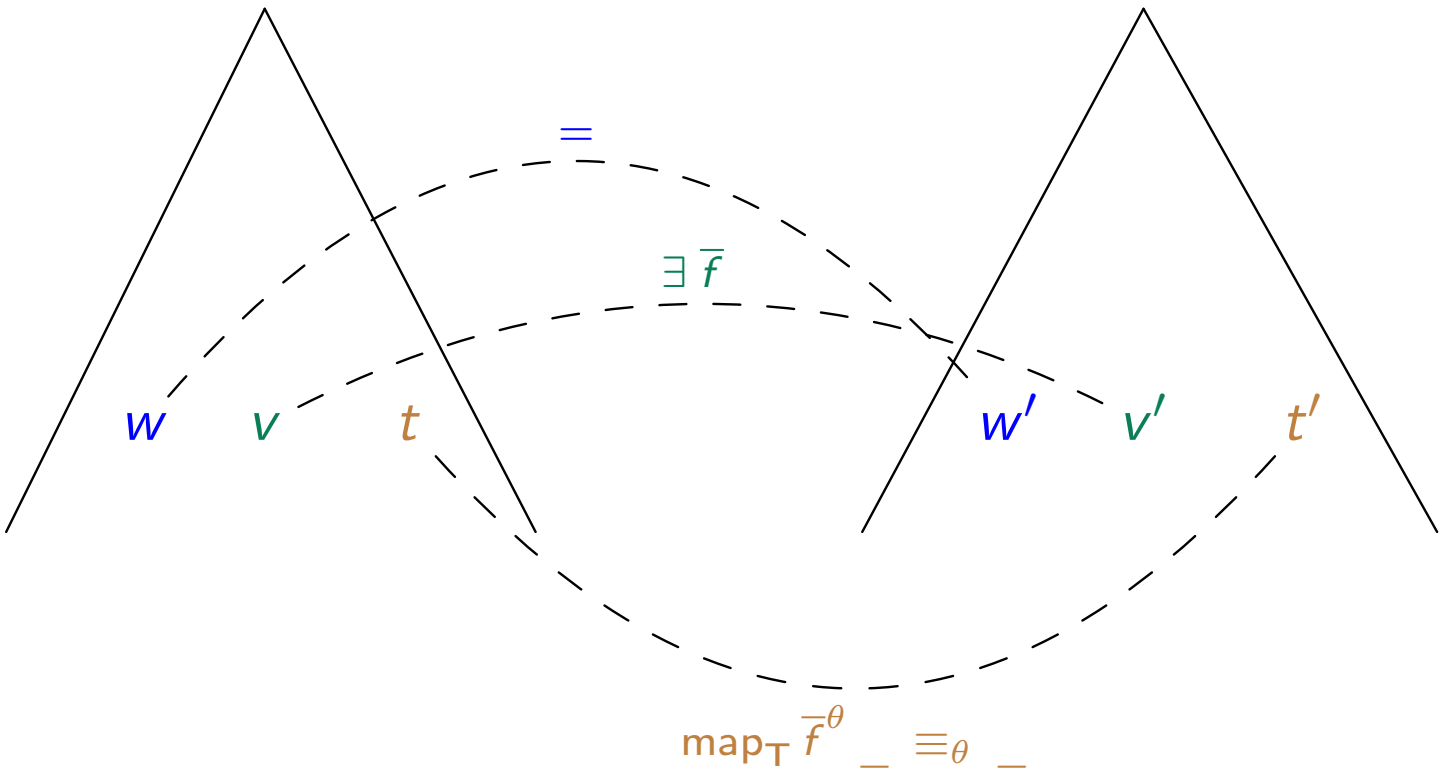
Equality on the top free variables

Possible bijective renamings of top binding variables

Recursive call factoring in the renamings

Coinductive Definition of Alpha-Equivalence

Same characteristic clause, same intuition,
... but GFP instead of LFP in Knaster-Tarski



Equality on the top free variables
Possible bijective renamings of top binding variables
Recursive call factoring in the renamings

MRBNFs = BNFs with Binding Awareness

BNFs

- Include many useful container types
- Closed under composition
- Closed under least fixpoints
- Closed under greatest fixpoints

Compositional (co)datatypes
Implemented in Isabelle: user-friendly, hides category theory

MRBNFs = BNFs with Binding Awareness

MRBNFs

- Include many useful container types
- Closed under composition
- Closed under least fixpoints
- Closed under greatest fixpoints
- Closed under binding-aware least fixpoints
- Closed under binding-aware greatest fixpoints
- Closed under linearization

Compositional binding-aware (co)datatypes

Isabelle: worked out category theory, not yet user-friendly

References

Expressive datatypes and codatatypes:

- Foundational, Compositional (Co)datatypes for HOL. LICS'12
- Truly Modular (Co)datatypes for Isabelle/HOL. ITP'14
- Foundational nonuniform (Co)datatypes for HOL. LICS'17
- Relational Parametricity and Quotient Preservation for Modular (Co)datatypes. ITP'18
- Quotients of Bounded Natural Functors. IJCAR'20
  (TBP tomorrow)

Ensuring non-emptiness of types:

- Witnessing (Co)datatypes. ESOP'15

Expressive function definition mechanisms:

- Foundational extensible corecursion. ICFP'15
- Corecursion in Foundational Proof Assistants. ESOP'17

Overview of entire line of work

- Foundational (Co)datatypes and (Co)recursion for HOL. FroCoS'17

Supernominal (MRBNF extension of BNF)

- Bindings as Bounded Natural Functors. POPL'19

Related Work

Nominal: Logic, Techniques, Datatypes

Murdoch J. Gabbay'

Michael Norrish

Andrew Pitts

Christian Urban

The Super in Supernominal

	Nominal Datatypes	MRBNFs (Binders as Functors)
BA-Induction	●	●
BA-Recursion	◒	●
Infinite Branching		●
Coinductive Types	◔	●
BA-Coinduction		●
BA-Corecursion	◔	●
Complex Binders	◒	●
Modularity	◔	●

BA = binding-aware

The Super in Supernominal

Also, Transnominal: Beyond Finite Support

	Nominal Datatypes	MRBNFs (Binders as Functors)
BA-Induction	●	●
BA-Recursion	◒	●
Infinite Branching		
Coinductive Types		
BA-Coinduction		
BA-Corecursion		
Complex Binders	◒	●
Modularity	◔	●

BA = binding-aware

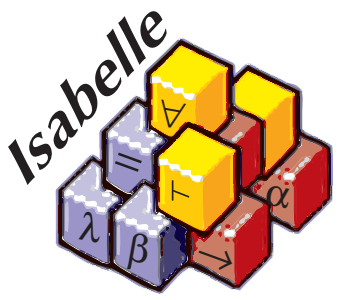
Binding-Aware Induction and Recursion

Supernominal Recursors

Urban'08 Norrish'04 Gheri&Popescu'17
Pitts'05 Gordon&Melham'96 Popescu&Gunter'11

Prior work on nominal corecursion: Kurz et al. 2013
Supernominal lifts their restriction to finite support

Syntax with Bindings in Isabelle



Isabelle Nominal2 [Urban and Kaliszyk 2012]

- Good user support
- Complex binders via syntactic format

Supernominal (not yet fully implemented)

- Will boost expressiveness and compositionality
- Will stay backwards-compatiblish with Nominal/Nominal2

1999: The Year the Earth Stood Still

Much category theory on De Bruijn style, starting with

- Fiore et al. (LICS'99)
- Hofmann (LICS'99)
- Bird and Paterson (J. Func. Prog. '99)
- Altenkirch and Reus (CSL'99)

The same year: Nominal Logic – Gabbay and Pitts (LICS'99)

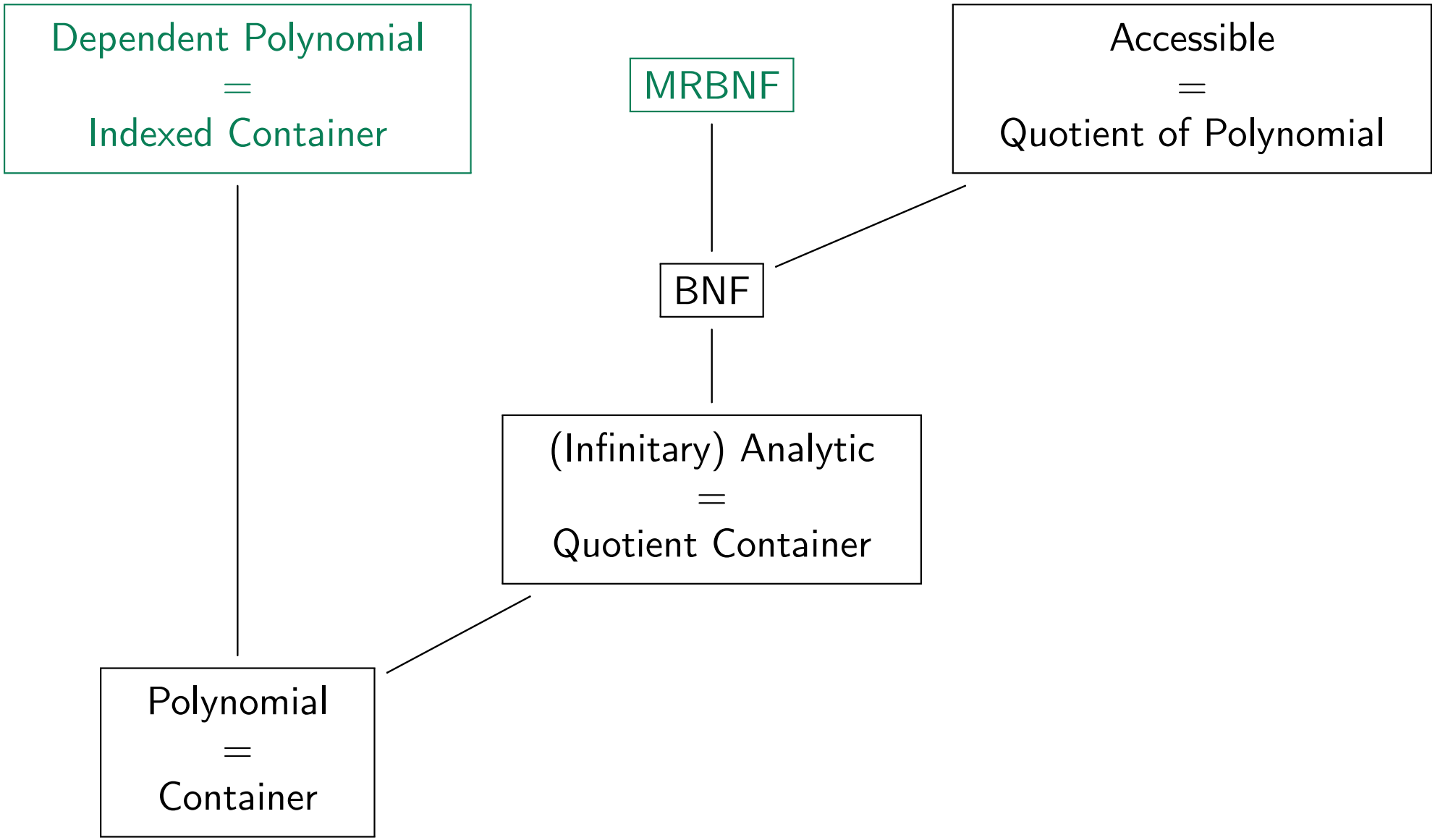
Precursors of BNFs

BNFs = subclass of k-accessible functors

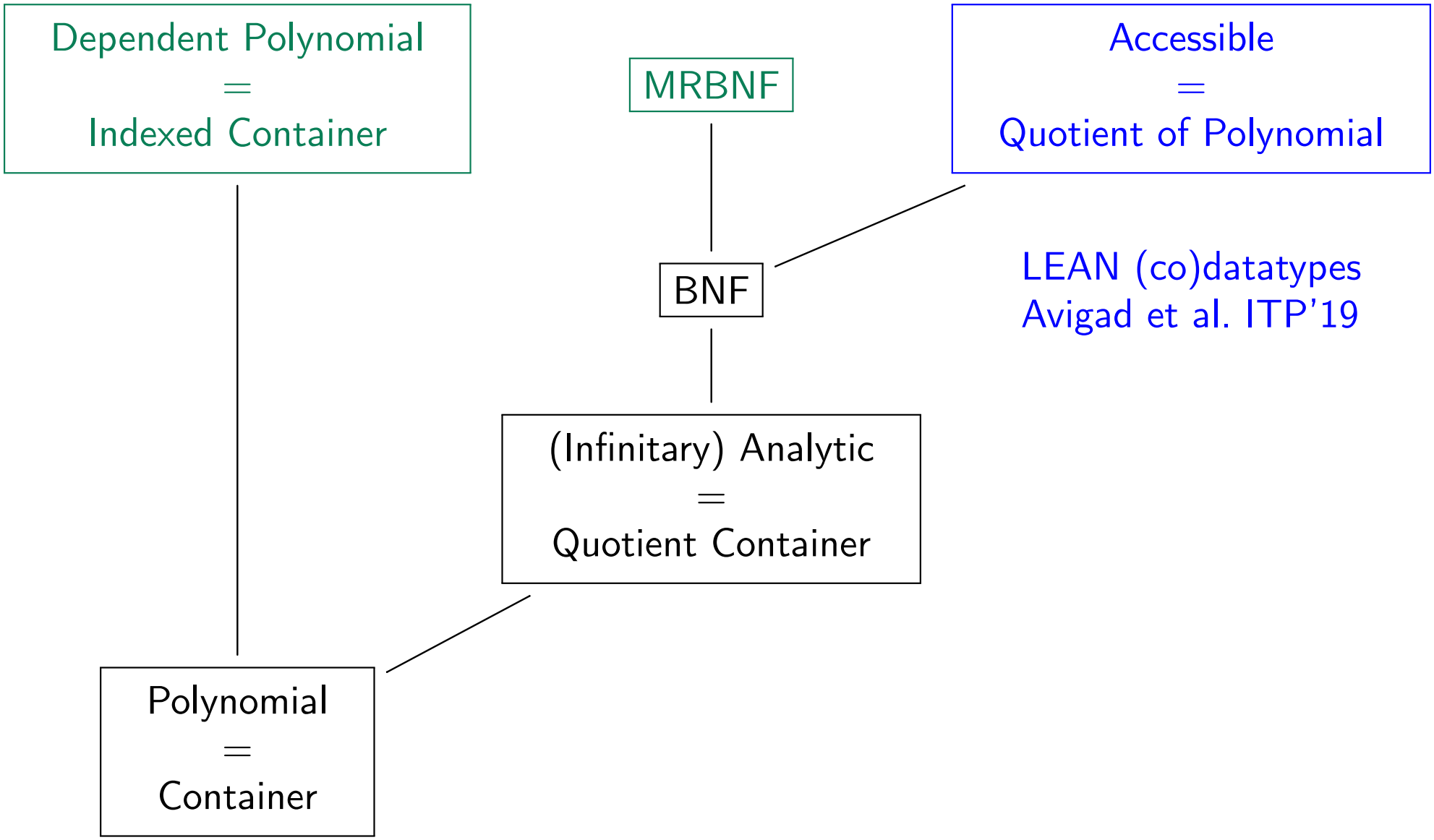
Container types [Hoogendijk and de Moor 2000]

Containers [Abbott et al. 2005]

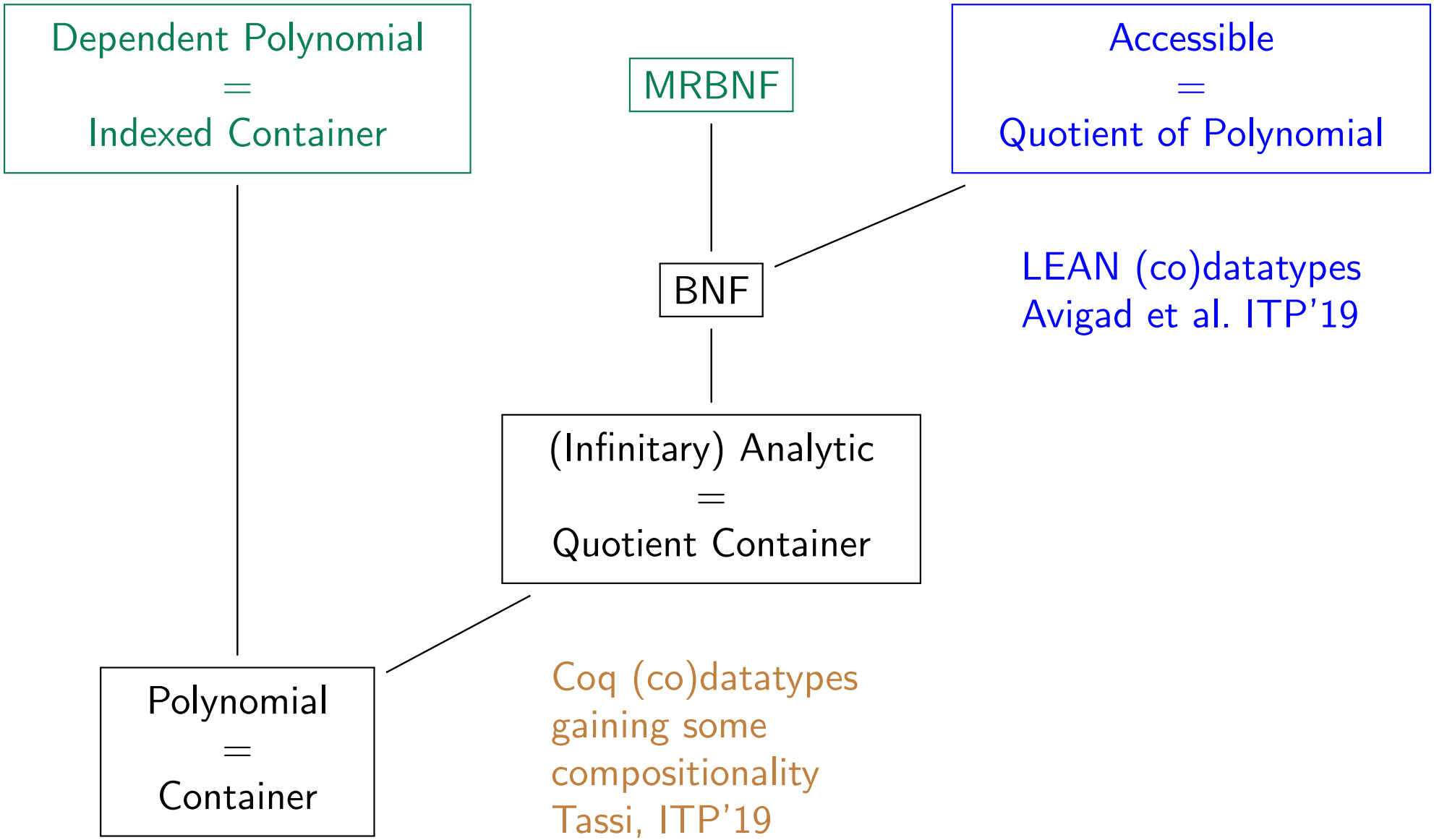
Relevant Classes of Functors



Relevant Classes of Functors



Relevant Classes of Functors



Bindings Are Functors

Bindings Are Functors

Andrei Popescu
University of Sheffield, UK
LFMTP
June 30, 2020

Joint work with

Jasmin Blanchette, Lorenzo Gheri, Dmitriy Traytel, Isabelle/HOL