Nominal Type Theory by Nullary Internal Parametricity

Variable binding by nullary parametricity

Antoine Van Muylder, Andreas Nuyts, Dominique Devriese

HoTT/UF 2026
(full paper accepted at: TYPES 2025 post-proceedings)
(arxiv)

KU Leuven

Variable binding in nominal frameworks

- **Goal:** convenient representation of binders in DTT

Variable binding in nominal frameworks

- **Goal:** convenient representation of binders in DTT
- **Idea:** extend DTT with
  - a name abstraction type former NAbs
  - a type of names Nm

Variable binding in nominal frameworks

- **Goal:** convenient representation of binders in DTT
- **Idea:** extend DTT with
  - a name abstraction type former NAbs
  - a type of names Nm
- Then: nominal data types, e.g.

```
data Ltm : Type where
  var : Nm → Ltm
  app : Ltm → Ltm → Ltm
  lam : NAbs Ltm → Ltm      -- binder
```

Variable binding in nominal frameworks

- **Goal:** convenient representation of binders in DTT
- **Idea:** extend DTT with
  - a name abstraction type former NAbs
  - a type of names Nm
- Then: nominal data types, e.g.

```
data Ltm : Type where
  var : Nm → Ltm
  app : Ltm → Ltm → Ltm
  lam : NAbs Ltm → Ltm      -- binder
```

- Several axiomatizations of NAbs exist

Variable binding in nominal frameworks

- **Goal:** convenient representation of binders in DTT
- **Idea:** extend DTT with
  - a name abstraction type former NAbs
  - a type of names Nm
- Then: nominal data types, e.g.

```
data Ltm : Type where
  var : Nm → Ltm
  app : Ltm → Ltm → Ltm
  lam : NAbs Ltm → Ltm      -- binder
```

- Several axiomatizations of NAbs exist
- The way you write terms in Ltm depends on the used axiomatization

Pros and cons of nominal data types

```
data Ltm : Type where
  var : Nm → Ltm;        app : Ltm → Ltm → Ltm;        lam : NAbs Ltm → Ltm
```

Pros and cons of nominal data types

```
data Ltm : Type where
  var : Nm → Ltm;      app : Ltm → Ltm → Ltm;      lam : NAbs Ltm → Ltm
```

Intended pros (sometimes depend on axiomatization):

- α-equivalent terms are equal cf. **names-as-strings**

Pros and cons of nominal data types

```
data Ltm : Type where
  var : Nm → Ltm;      app : Ltm → Ltm → Ltm;      lam : NAbs Ltm → Ltm
```

Intended pros (sometimes depend on axiomatization):

- α-equivalent terms are equal — cf. **names-as-strings**
- Closed induction principle — cf. **PHOAS**, **HOAS**

Pros and cons of nominal data types

```
data Ltm : Type where
  var : Nm → Ltm;      app : Ltm → Ltm → Ltm;      lam : NAbs Ltm → Ltm
```

Intended pros (sometimes depend on axiomatization):

- α-equivalent terms are equal — cf. names-as-strings
- Closed induction principle — cf. PHOAS, HOAS
- Proofs stable under weakening — cf. De Bruijn

Pros and cons of nominal data types

```
data Ltm : Type where
  var : Nm → Ltm;      app : Ltm → Ltm → Ltm;      lam : NAbs Ltm → Ltm
```

Intended pros (sometimes depend on axiomatization):

- α-equivalent terms are equal — cf. names-as-strings
- Closed induction principle — cf. PHOAS, HOAS
- Proofs stable under weakening — cf. De Bruijn
- Sometimes matches informal reasoning

Pros and cons of nominal data types

```
data Ltm : Type where
  var : Nm → Ltm;      app : Ltm → Ltm → Ltm;      lam : NAbs Ltm → Ltm
```

Intended pros (sometimes depend on axiomatization):

- α-equivalent terms are equal — cf. names-as-strings
- Closed induction principle — cf. PHOAS, HOAS
- Proofs stable under weakening — cf. De Bruijn
- Sometimes matches informal reasoning

Cons:

- Not plain DTT

Pros and cons of nominal data types

```
data Ltm : Type where
  var : Nm → Ltm;      app : Ltm → Ltm → Ltm;      lam : NAbs Ltm → Ltm
```

Intended pros (sometimes depend on axiomatization):

- α-equivalent terms are equal — cf. names-as-strings
- Closed induction principle — cf. PHOAS, HOAS
- Proofs stable under weakening — cf. De Bruijn
- Sometimes matches informal reasoning

Cons:

- Not plain DTT
- `Ltm` = closed terms only in a name-free context

Axiomatization 1: pairs

In some frameworks, NAbs $\approx$ type of **pairs** $\langle n, t \rangle$.

Axiomatization 1: pairs

In some frameworks, NAbs $\approx$ type of **pairs** $\langle n, t \rangle$.

Rules enforce:

- t can mention n; pair is up to α; n is fresh in $\langle n, t \rangle$

Axiomatization 1: pairs

In some frameworks, NAbs $\approx$ type of **pairs** $\langle n, t \rangle$.

Rules enforce:

- t can mention n; pair is up to α; n is fresh in $\langle n, t \rangle$
- Can **pattern match** ✓

Axiomatization 1: pairs

In some frameworks, NAbs $\approx$ type of **pairs** $\langle n, t \rangle$.

Rules enforce:

- t can mention n; pair is up to α; n is fresh in $\langle n, t \rangle$
- Can **pattern match** ✓

```
f : Ltm → X
f (lam <n,t>) = something(n, t)
```

Axiomatization 1: pairs

In some frameworks, NAbs $\approx$ type of **pairs** $\langle n, t \rangle$.

Rules enforce:

- t can mention n; pair is up to α; n is fresh in $\langle n, t \rangle$
- Can **pattern match** ✓

```
f : Ltm → X
f (lam <n,t>) = something(n, t)
```

Issue: eliminating $\langle n, t \rangle$ requires n to be fresh in the target
$\Rightarrow$ annoying typing rules

Axiomatization 1: pairs

In some frameworks, NAbs $\approx$ type of **pairs** $\langle n, t \rangle$.

Rules enforce:

- t can mention n; pair is up to α; n is fresh in $\langle n, t \rangle$
- Can **pattern match** ✓

```
f : Ltm → X
f (lam <n,t>) = something(n, t)
```

Issue: eliminating $\langle n, t \rangle$ requires n to be fresh in the target
$\Rightarrow$ annoying typing rules

```
f (lam <n,t>) = n   -- breaks α-equivalence
```

Axiomatization 2: functions

In some frameworks, NAbs $\approx$ type of **functions** $\lambda n.\, t(n)$
Our notation: NAbs Ltm $=$ @N $\multimap$ Ltm

Axiomatization 2: functions

In some frameworks, NAbs $\approx$ type of **functions** $\lambda n.\, t(n)$
Our notation: NAbs Ltm $=$ @N $\multimap$ Ltm

- Application has a *freshness side condition*:
  $t\ n$ typechecks only if n is fresh for t

Axiomatization 2: functions

In some frameworks, NAbs $\approx$ type of **functions** $\lambda n.\, t(n)$
Our notation: NAbs Ltm $=$ @N $\multimap$ Ltm

- Application has a *freshness side condition*:
  $t\ n$ typechecks only if n is fresh for t
- **Issue:** cannot match on functions

Axiomatization 2: functions

In some frameworks, NAbs $\approx$ type of **functions** $\lambda n.\, t(n)$
Our notation: NAbs Ltm $=$ @N $\multimap$ Ltm

- Application has a *freshness side condition*:
  $t\ n$ typechecks only if n is fresh for t
- **Issue:** cannot match on functions

```
freeNames : Ltm → List Nm
freeNames (var n)   = [ n ]
freeNames (app a b) = freeNames a ++ freeNames b
freeNames (lam g)   = ???   -- g : @N ⊸ Ltm
```

Axiomatization 2: functions

In some frameworks, NAbs $\approx$ type of **functions** $\lambda n.\, t(n)$
Our notation: NAbs Ltm $=$ @N $\multimap$ Ltm

- Application has a *freshness side condition*:
  $t\ n$ typechecks only if n is fresh for t
- **Issue:** cannot match on functions

```
freeNames : Ltm → List Nm
freeNames (var n)   = [ n ]
freeNames (app a b) = freeNames a ++ freeNames b
freeNames (lam g)   = ???   -- g : @N ⊸ Ltm
```

Our solution to both issues:
Use a type theory with *nullary internal parametricity*

Internal (binary) parametric type theory?

Cubical Type Theory

- Paths: $I \to A$, with endpoints eqs

(Binary) Parametric TT

- Bridges: $@\mathsf{N} \multimap A$, with endpoints eqs

Internal (binary) parametric type theory?

Cubical Type Theory

- Paths: $I \to A$, with endpoints eqs
- Functions preserve paths

(Binary) Parametric TT

- Bridges: $@\mathsf{N} \multimap A$, with endpoints eqs
- Functions preserve bridges

Internal (binary) parametric type theory?

Cubical Type Theory

- Paths: $I \to A$, with endpoints eqs
- Functions preserve paths
- Paths $\simeq$ isomorphisms (SIP)

(Binary) Parametric TT

- Bridges: $@\mathsf{N} \multimap A$, with endpoints eqs
- Functions preserve bridges
- Bridges $\simeq$ structural relations (SRP)

Internal (binary) parametric type theory?

Cubical Type Theory

- Paths: $I \to A$, with endpoints eqs
- Functions preserve paths
- Paths $\simeq$ isomorphisms (SIP)
- $\Rightarrow$ functions preserve isomorphisms (representation independence)

(Binary) Parametric TT

- Bridges: $@\mathsf{N} \multimap A$, with endpoints eqs
- Functions preserve bridges
- Bridges $\simeq$ structural relations (SRP)
- $\Rightarrow$ functions preserve structural relations (parametricity)

Internal (binary) parametric type theory?

Cubical Type Theory

- Paths: $I \to A$, with endpoints eqs
- Functions preserve paths
- Paths $\simeq$ isomorphisms (SIP)
- $\Rightarrow$ functions preserve isomorphisms (representation independence)
- Semantics: higher groupoids

(Binary) Parametric TT

- Bridges: $@\mathbb{N} \multimap A$, with endpoints eqs
- Functions preserve bridges
- Bridges $\simeq$ structural relations (SRP)
- $\Rightarrow$ functions preserve structural relations (parametricity)
- Semantics: higher reflexive graphs

Internal (binary) parametric type theory?

Cubical Type Theory

- Paths: $I \to A$, with endpoints eqs
- Functions preserve paths
- Paths $\simeq$ isomorphisms (SIP)
- $\Rightarrow$ functions preserve isomorphisms (representation independence)
- Semantics: higher groupoids
- in H.O.T.T.: no interval, first-class observational eq.

(Binary) Parametric TT

- Bridges: $@\mathbb{N} \multimap A$, with endpoints eqs
- Functions preserve bridges
- Bridges $\simeq$ structural relations (SRP)
- $\Rightarrow$ functions preserve structural relations (parametricity)
- Semantics: higher reflexive graphs
- in Obs. PTT: no interval, first-class param. tranlsation

Internal (binary) parametric type theory?

Cubical Type Theory

- Paths: $I \to A$, with endpoints eqs
- Functions preserve paths
- Paths $\simeq$ isomorphisms (SIP)
- $\Rightarrow$ functions preserve isomorphisms (representation independence)
- Semantics: higher groupoids
- in H.O.T.T.: no interval, first-class observational eq.

This work: Cavallo Harper theory (interval based) at arity 0

(Binary) Parametric TT

- Bridges: $@\mathbb{N} \multimap A$, with endpoints eqs
- Functions preserve bridges
- Bridges $\simeq$ structural relations (SRP)
- $\Rightarrow$ functions preserve structural relations (parametricity)
- Semantics: higher reflexive graphs
- in Obs. PTT: no interval, first-class param. tranlsation

The CH theory with arity 0

Parametricity primitives:

- **Nullary Bridge type:** $@\mathsf{N} \multimap A$ (same notation...)

The CH theory with arity 0

Parametricity primitives:

- **Nullary Bridge type:** $@\mathsf{N} \multimap A$ (same notation...)
- Just functions from the interval pretype $@\mathsf{N}$

The CH theory with arity 0

Parametricity primitives:

- **Nullary Bridge type:** $@\mathsf{N} \multimap A$ (same notation...)
- Just functions from the interval pretype $@\mathsf{N}$
- but can only be applied to *fresh* interval variables

The CH theory with arity 0

Parametricity primitives:

- **Nullary Bridge type:** $@\mathsf{N} \multimap A$ (same notation...)
- Just functions from the interval pretype $@\mathsf{N}$
- but can only be applied to *fresh* interval variables

- **Extent primitive:**

$$\mathsf{extent} : \quad ((@\mathsf{N} \multimap \mathsf{A}) \to (@\mathsf{N} \multimap \mathsf{B})) \quad \simeq \quad (@\mathsf{N} \multimap (\mathsf{A} \to \mathsf{B}))$$

The CH theory with arity 0

Parametricity primitives:

- **Nullary Bridge type:** $@\mathsf{N} \multimap A$ (same notation...)
- Just functions from the interval pretype $@\mathsf{N}$
- but can only be applied to *fresh* interval variables

- **Extent primitive:**

$$\mathsf{extent} : \quad ((@\mathsf{N} \multimap \mathsf{A}) \to (@\mathsf{N} \multimap \mathsf{B})) \quad \simeq \quad (@\mathsf{N} \multimap (\mathsf{A} \to \mathsf{B}))$$

- **Gel primitive:**

$$\mathsf{Gel} : \mathsf{Type} \simeq (@\mathsf{N} \multimap \mathsf{Type})$$

Contribution: Parametric Nominal Type Theory (PNTT)

Core idea *(observation existed in the community):*

@N $\multimap$ Ltm from nominal TT is the nullary Bridge type

Contribution: Parametric Nominal Type Theory (PNTT)

Core idea *(observation existed in the community):*

@N $\multimap$ Ltm from nominal TT is the nullary Bridge type

Our recipe:

1. Take the CH type theory at arity 0

Contribution: Parametric Nominal Type Theory (PNTT)

Core idea *(observation existed in the community):*

@N $\multimap$ Ltm from nominal TT is the nullary Bridge type

Our recipe:

1. Take the CH type theory at arity 0
2. CH primitives have nominal content:
   e.g. `Gel` is a freshness type former $(\dots)$

Contribution: Parametric Nominal Type Theory (PNTT)

Core idea *(observation existed in the community):*

@N $\multimap$ Ltm from nominal TT is the nullary Bridge type

Our recipe:

1. Take the CH type theory at arity 0
2. CH primitives have nominal content:
   e.g. `Gel` is a freshness type former $(\dots)$
3. Need Nm **type** for nominal data types

Contribution: Parametric Nominal Type Theory (PNTT)

Core idea *(observation existed in the community):*

@N $\multimap$ Ltm from nominal TT is the nullary Bridge type

Our recipe:

1. Take the CH type theory at arity 0
2. CH primitives have nominal content:
   e.g. `Gel` is a freshness type former $(\dots)$
3. Need Nm **type** for nominal data types
4. So $\neq$ @N (only a pretype)

Contribution: Parametric Nominal Type Theory (PNTT)

Core idea *(observation existed in the community):*

@N $\multimap$ Ltm from nominal TT is the nullary Bridge type

Our recipe:

1. Take the CH type theory at arity 0
2. CH primitives have nominal content:
   e.g. `Gel` is a freshness type former $(\dots)$
3. Need Nm **type** for nominal data types
4. So $\neq$ @N (only a pretype)
5. $\Rightarrow$ postulate Nm and *name induction*

Contribution: Parametric Nominal Type Theory (PNTT)

Core idea *(observation existed in the community):*

@N $\multimap$ Ltm from nominal TT is the nullary Bridge type

Our recipe:

1. Take the CH type theory at arity 0
2. CH primitives have nominal content:
   e.g. `Gel` is a freshness type former $(\dots)$
3. Need Nm **type** for nominal data types
4. So $\neq$ @N (only a pretype)
5. $\Rightarrow$ postulate Nm and *name induction*
6. given $\dots (x : @\mathsf{N}) \dots \vdash n : \mathsf{Nm}$, either $n = x$ or x is fresh in n (via `Gel`)

Contribution: Parametric Nominal Type Theory (PNTT)

Core idea *(observation existed in the community):*

@N ⊸ Ltm from nominal TT is the nullary Bridge type

Our recipe:

1. Take the CH type theory at arity 0
2. CH primitives have nominal content:
   e.g. `Gel` is a freshness type former $(\dots)$
3. Need Nm **type** for nominal data types
4. So $\neq$ @N (only a pretype)
5. $\Rightarrow$ postulate Nm and *name induction*
6. given $\dots (x : @\mathsf{N}) \dots \vdash n : \mathsf{Nm}$, either $n = x$ or x is fresh in n (via `Gel`)
7. Add rules for nominal data types

Contribution: Parametric Nominal Type Theory (PNTT)

Core idea *(observation existed in the community):*

@N $\multimap$ Ltm from nominal TT is the nullary Bridge type

Our recipe:

1. Take the CH type theory at arity 0
2. CH primitives have nominal content:
   e.g. `Gel` is a freshness type former $(\dots)$
3. Need Nm **type** for nominal data types
4. So $\neq$ @N (only a pretype)
5. $\Rightarrow$ postulate Nm and *name induction*
6. given $\dots (x : @\mathsf{N}) \dots \vdash n : \mathsf{Nm}$, either $n = x$ or x is fresh in n (via `Gel`)
7. Add rules for nominal data types

Summary

$\mathbf{PNTT}$ = nullary CH theory + Nm and name induction + nominal data types

Nominal pattern matching recovered (1)

PNTT validates the **Structure Abstraction Principle (SAP)** (cf. SIP, SRP, DSIP)

Nominal pattern matching recovered (1)

PNTT validates the **Structure Abstraction Principle (SAP)** (cf. SIP, SRP, DSIP)

For* each type former K:

$$[K]_0 \;\simeq\; (@\mathsf{N} \multimap K)$$

Nominal pattern matching recovered (1)

PNTT validates the **Structure Abstraction Principle (SAP)** (cf. SIP, SRP, DSIP)

For* each type former K:

$$[K]_0 \simeq (@\mathsf{N} \multimap K)$$

Some instances:

extent : $((@\mathsf{N} \multimap \mathsf{A}) \to (@\mathsf{N} \multimap \mathsf{B})) \simeq (@\mathsf{N} \multimap (\mathsf{A} \to \mathsf{B}))$
Gel : $\mathsf{Type} \simeq (@\mathsf{N} \multimap \mathsf{Type})$
SAP-Nm : $1 + \mathsf{Nm} \simeq (@\mathsf{N} \multimap \mathsf{Nm})$ -- new

Nominal pattern matching recovered (1)

Also: $\mathsf{Ltm}_1 \simeq (@\mathsf{N} \multimap \mathsf{Ltm})$, where

data Ltm_1 : Type **where**
  $[\mathsf{var}]_0$: $1 + \mathsf{Nm} \to \mathsf{Ltm}_1$
  $[\mathsf{app}]_0$: $\mathsf{Ltm}_1 \to \mathsf{Ltm}_1 \to \mathsf{Ltm}_1$
  $[\mathsf{lam}]_0$: $(@\mathsf{N} \multimap \mathsf{Ltm}_1) \to \mathsf{Ltm}_1$

Nominal pattern matching recovered (1)

Also: $\mathsf{Ltm}_1 \simeq (@\mathsf{N} \multimap \mathsf{Ltm})$, where

data Ltm_1 : Type **where**
  $[\mathsf{var}]_0$: $1 + \mathsf{Nm} \to \mathsf{Ltm}_1$
  $[\mathsf{app}]_0$: $\mathsf{Ltm}_1 \to \mathsf{Ltm}_1 \to \mathsf{Ltm}_1$
  $[\mathsf{lam}]_0$: $(@\mathsf{N} \multimap \mathsf{Ltm}_1) \to \mathsf{Ltm}_1$

Summary

$[-]_0$ traverses the syntax, *except* on Nm, where $[\mathsf{Nm}]_0 = 1 + \mathsf{Nm}$

Nominal pattern matching recovered (2)

Thanks to the SAP, can “chase”

Nominal pattern matching recovered (2)

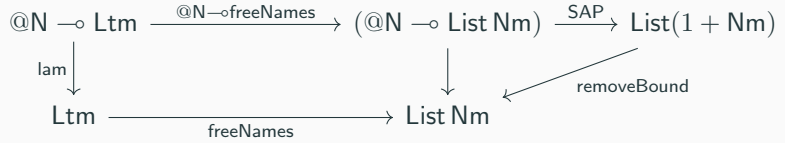
Thanks to the SAP, can "chase"

```
freeNames : Ltm → List Nm
freeNames (var n)   = [ n ]
freeNames (app a b) = freeNames a ++ freeNames b
freeNames (lam g)   = ?
```

Nominal pattern matching recovered (2)

Thanks to the SAP, can "chase"

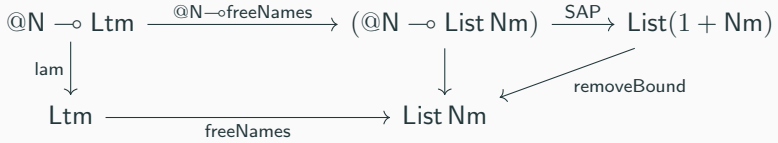
```
freeNames : Ltm → List Nm
freeNames (var n)   = [ n ]
freeNames (app a b) = freeNames a ++ freeNames b
freeNames (lam g)   = ?
```



Nominal pattern matching recovered (2)

Thanks to the SAP, can "chase"

freeNames : Ltm → List Nm
freeNames (var n) = [n]
freeNames (app a b) = freeNames a ++ freeNames b
freeNames (lam g) = ?



Thus PNTT legitimate nominal framework to study languages with binders

(Contrib.: Synthetic Kripke parametricity)

Nullary parametricity can emulate a type of **"future worlds"** reasoning.

See paper.

Future perspectives: observational PNTT

- **Interval-based theory (current work):** SAP holds *up to equivalence*
  $\Rightarrow$ nominal pattern matching recovered, but *indirectly*

Future perspectives: observational PNTT

- **Interval-based theory (current work):** SAP holds *up to equivalence*
  $\Rightarrow$ nominal pattern matching recovered, but *indirectly*
- **Observational PTT:** cleaner nominal pm. Want to look at Narya

Future perspectives: observational PNTT

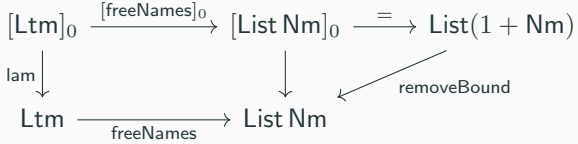
- **Interval-based theory (current work):** SAP holds *up to equivalence*
  $\Rightarrow$ nominal pattern matching recovered, but *indirectly*
- **Observational PTT:** cleaner nominal pm. Want to look at Narya

```
data Ltm : Type where
  var  : Nm → Ltm
  app  : Ltm → Ltm → Ltm
  lam  : [Ltm]₀ → Ltm

freeNames (lam g) = (removeBound ∘ [freeNames]₀) g
...
```

Future perspectives: observational PNTT

- **Interval-based theory (current work):** SAP holds *up to equivalence*
  $\Rightarrow$ nominal pattern matching recovered, but *indirectly*
- **Observational PTT:** cleaner nominal pm. Want to look at Narya

```
data Ltm : Type where
  var  : Nm → Ltm
  app  : Ltm → Ltm → Ltm
  lam  : [Ltm]₀ → Ltm

freeNames (lam g) = (removeBound ∘ [freeNames]₀) g
...
```



Future perspectives: assessing PNTT

PNTT has untyped names. Is it possible to represent e.g. the syntax of DTT in a good way still?