Canonicity for 2-Dimensional Type Theory

Daniel R. Licata
with Robert Harper

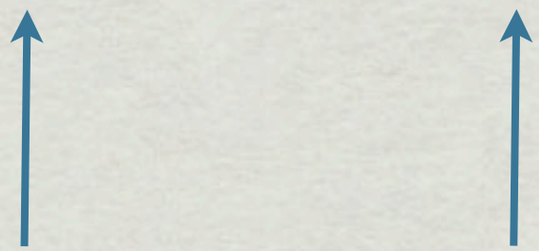
Carnegie Mellon University

Dependent Type Theory

Basis of proof assistants (Agda, Coq, NuPRL)

- Formalization of math
- Certified programming

sort : Π xs:int list. Σ ys:int list.Sorted(ys)



Identity Types

Usually called *propositional equality*

$$\frac{}{\mathsf{refl} : \mathsf{Id}_A(M,M)}$$

$$\frac{C : A \to \mathsf{type} \quad \alpha : \mathsf{Id}_A(M,N) \quad P : C[M]}{\mathsf{subst}_C\ \alpha\ P : C[N]}$$

more generally, J

Computation rule: $\mathsf{subst}_C\ \mathsf{refl}\ P \equiv P$

Identity Types

Usually called *propositional equality*

$$\frac{}{\mathsf{refl} : \mathsf{Id}_A(M,M)}$$

$$\frac{C : A \to \mathsf{type} \quad \alpha : \mathsf{Id}_A(M,N) \quad P : C[M]}{\mathsf{subst}_C\ \alpha\ P : C[N]}$$

more generally, J

Computation rule: $\mathsf{subst}_C\ \mathsf{refl}\ P \equiv P$

But we'll say *equivalence* because…

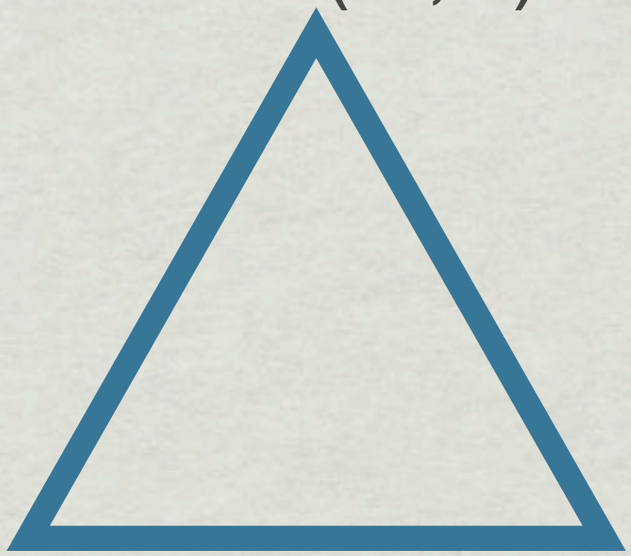
Homotopy Type Theory

dependent type theory

A : type

M, N : A

$\alpha : Id_A(M,N)$



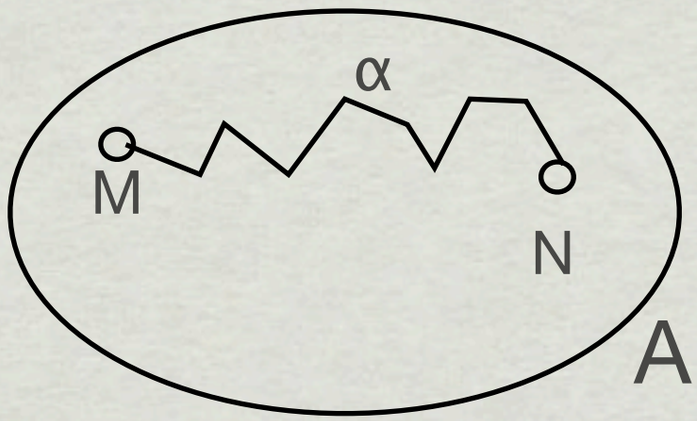
category theory

A is a groupoid

M,N objects

$\alpha : M \to N$ in A

homotopy theory

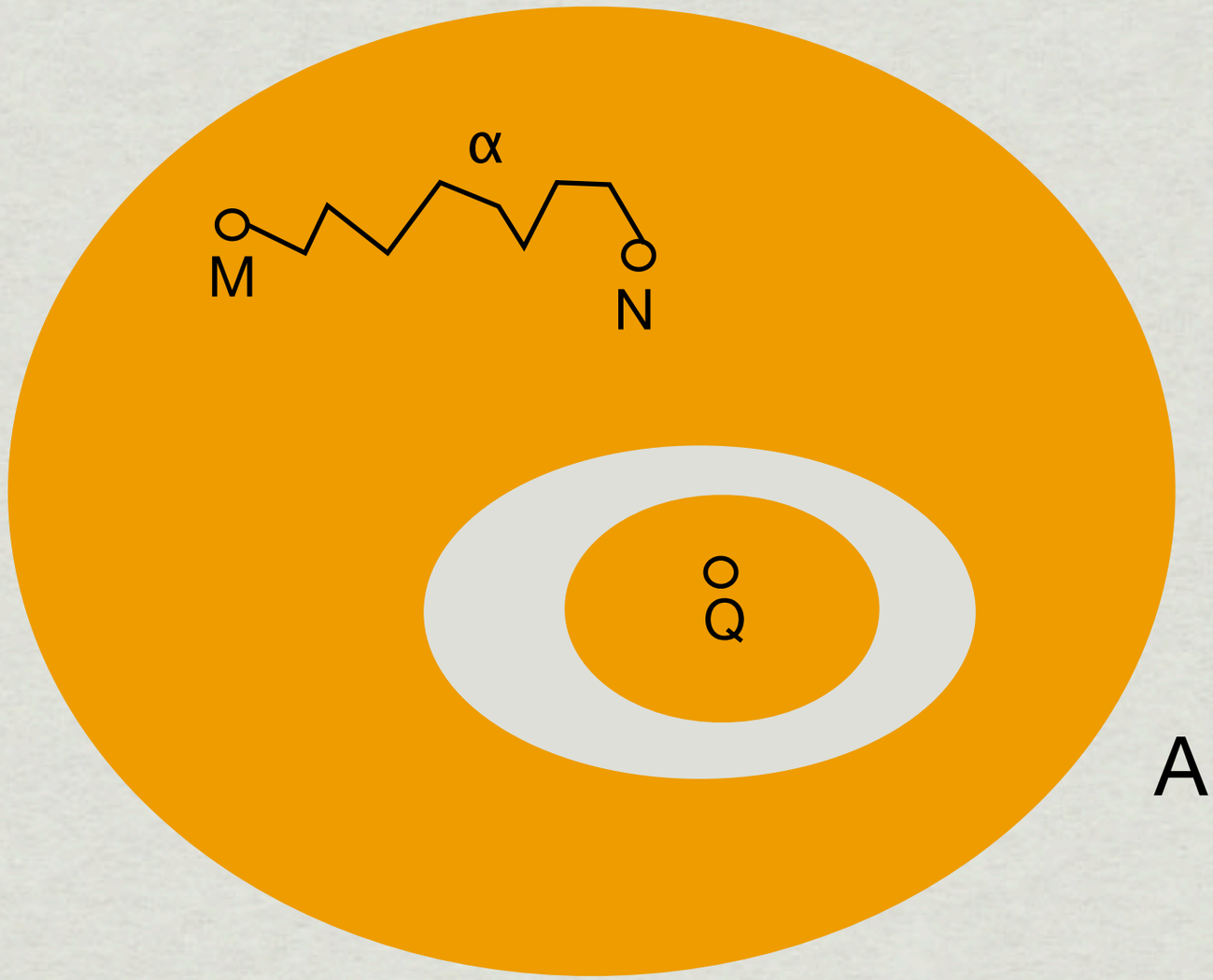


Homotopy Type Theory

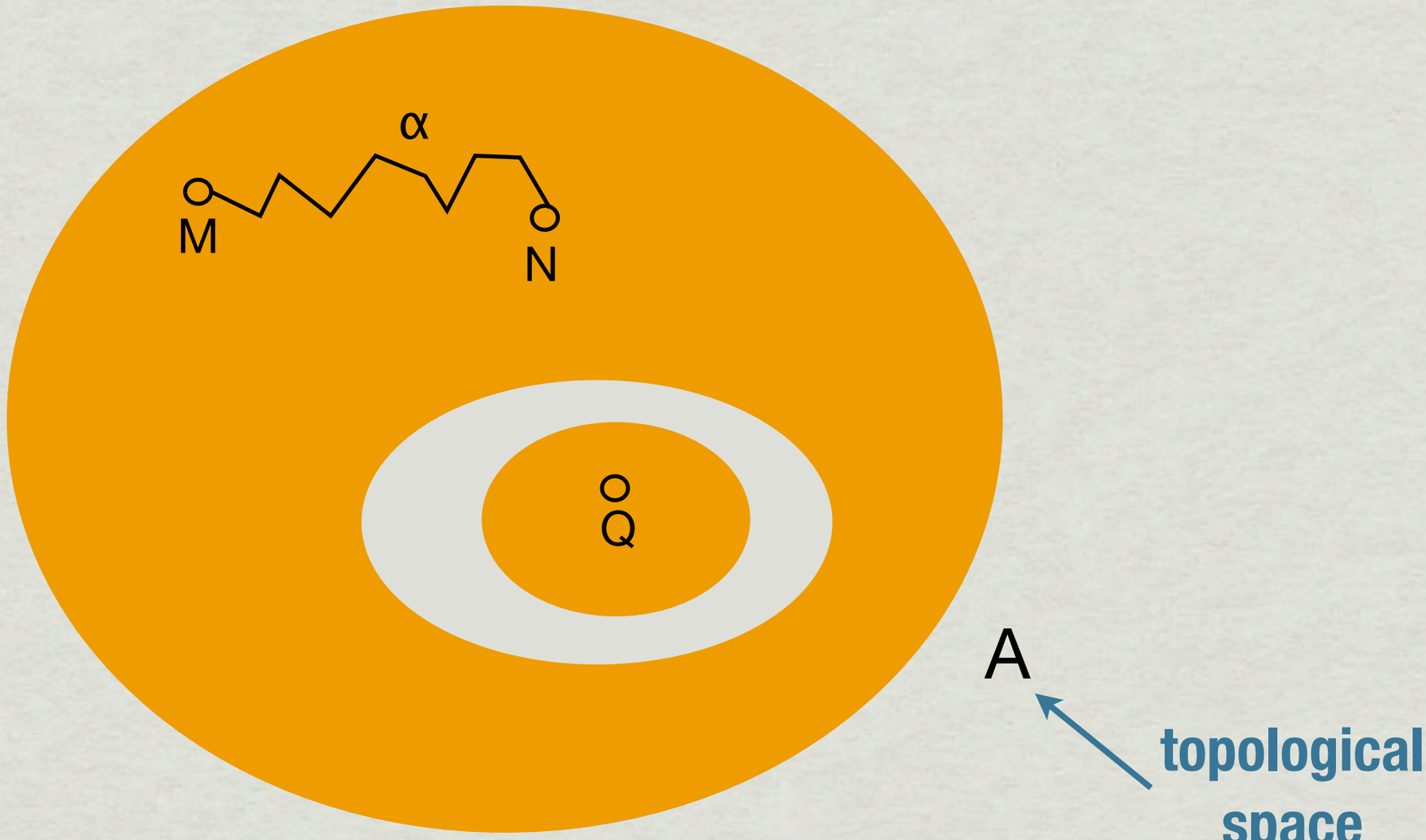
Theorem: Martin-Löf's intensional type theory has semantics in homotopy theory (spaces as types) and category theory (groupoids as types)

[Hofmann&Streicher,Awodey, Warren,Lumsdaine,Garner, Voevodsky, 1990's and 2000's]

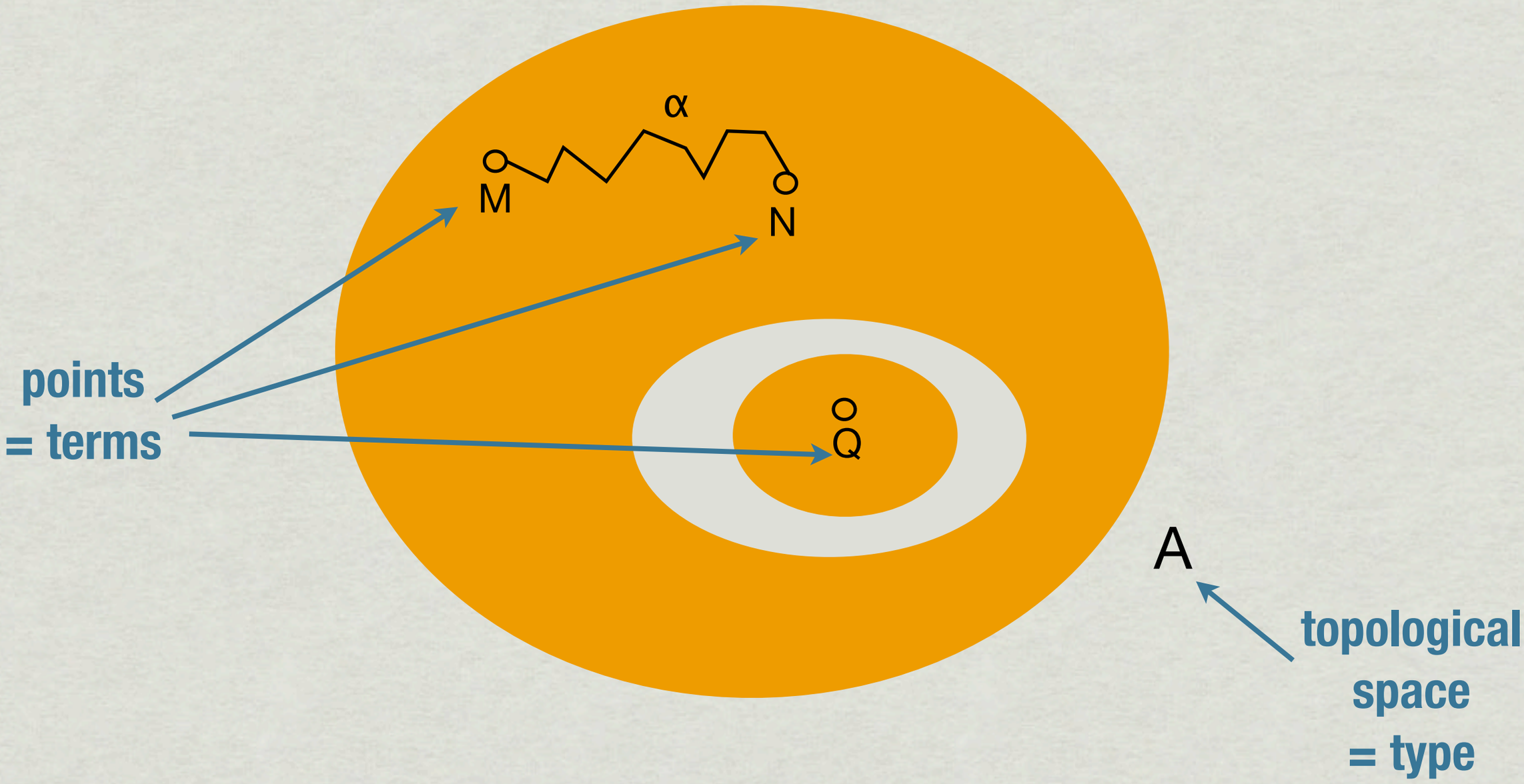
Types as Spaces



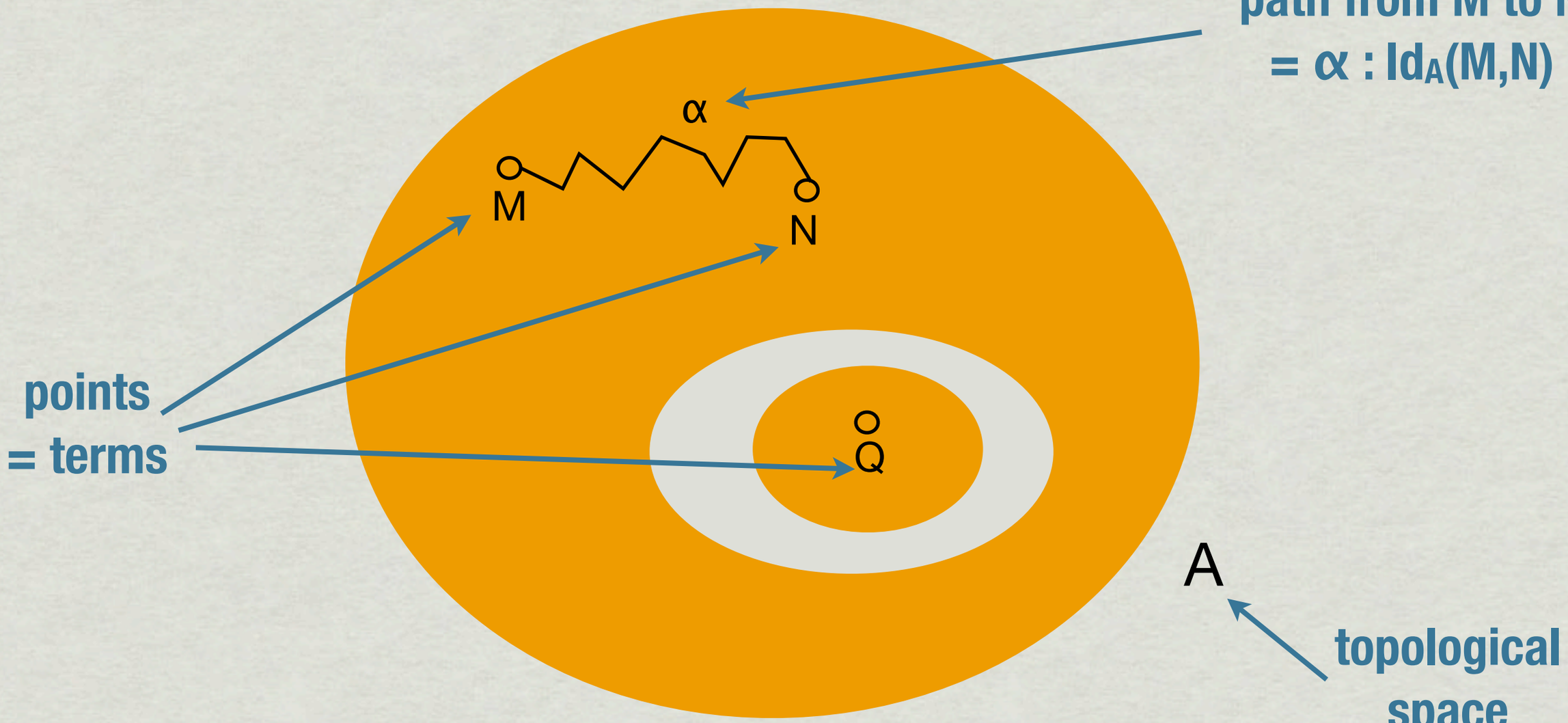
Types as Spaces



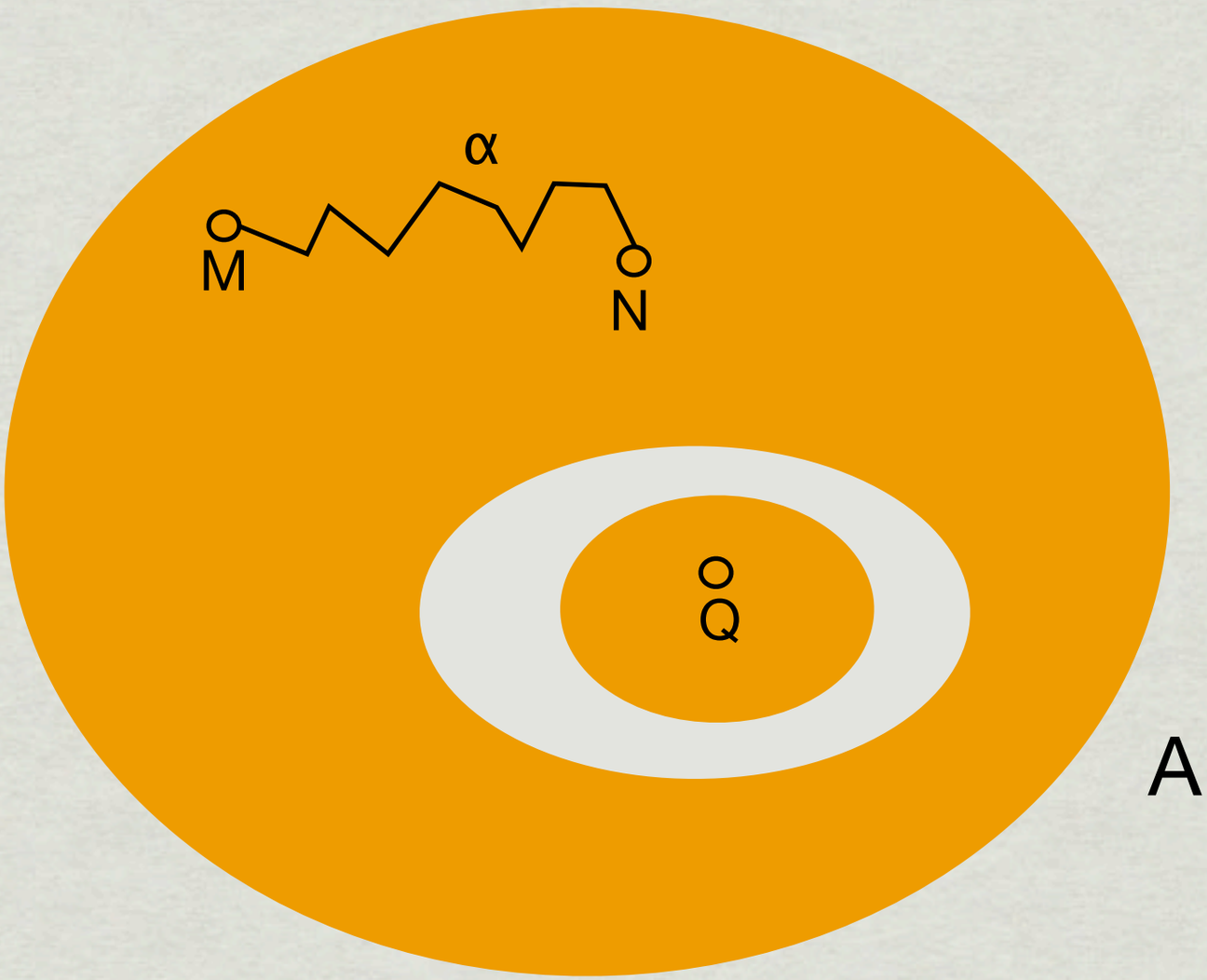
Types as Spaces



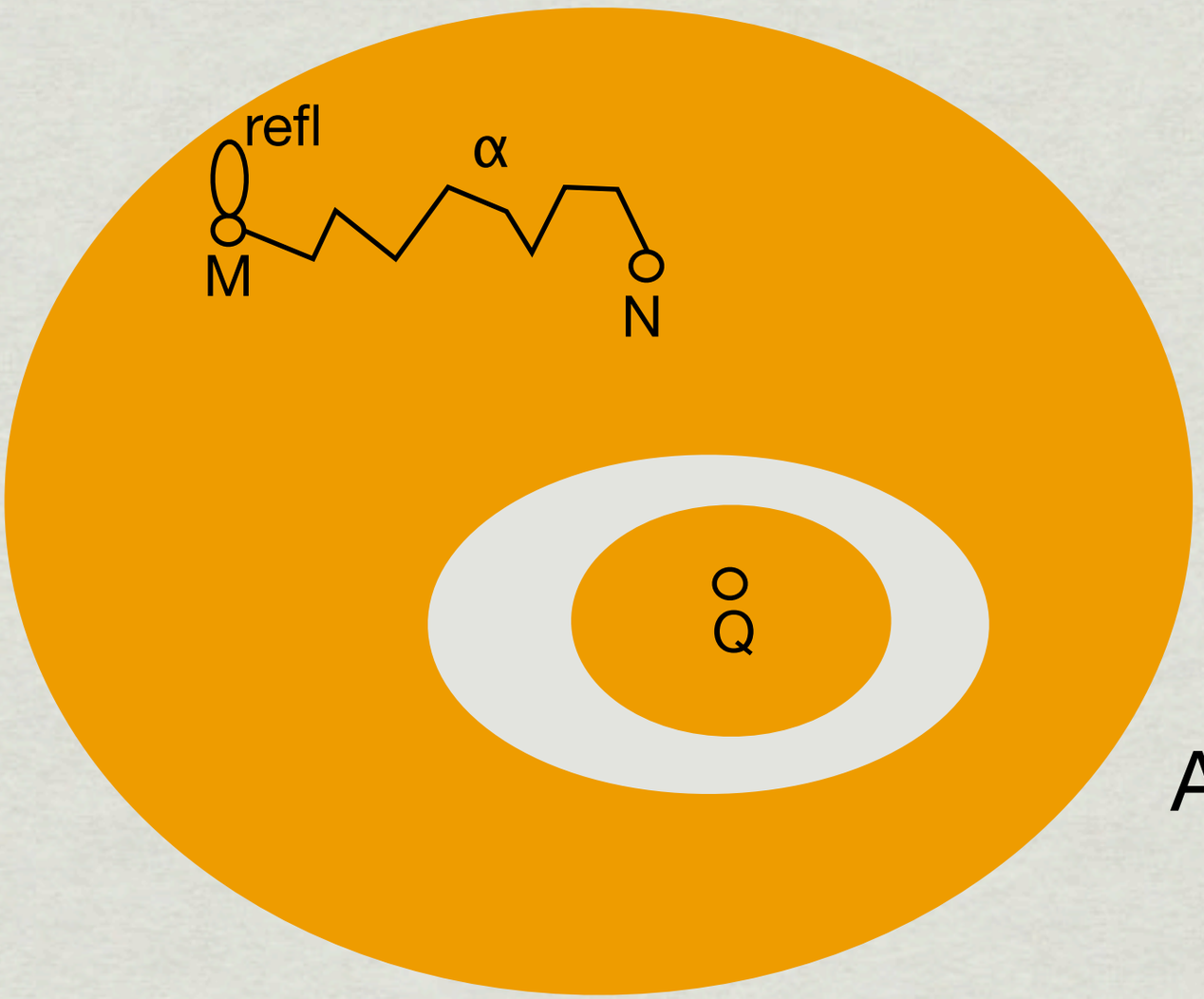
Types as Spaces



Operations on Paths

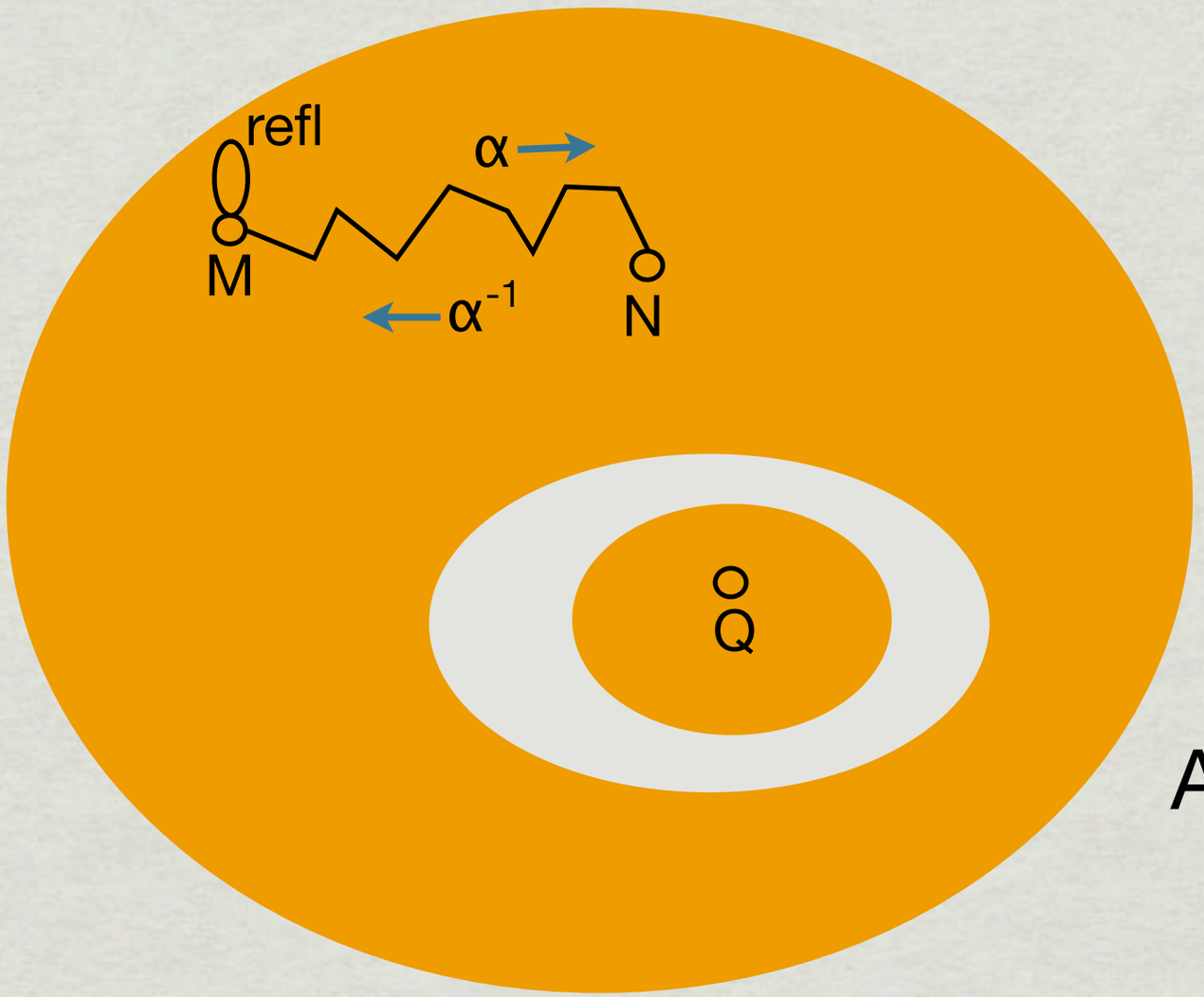


Operations on Paths



refl : $Id_A(M,M)$

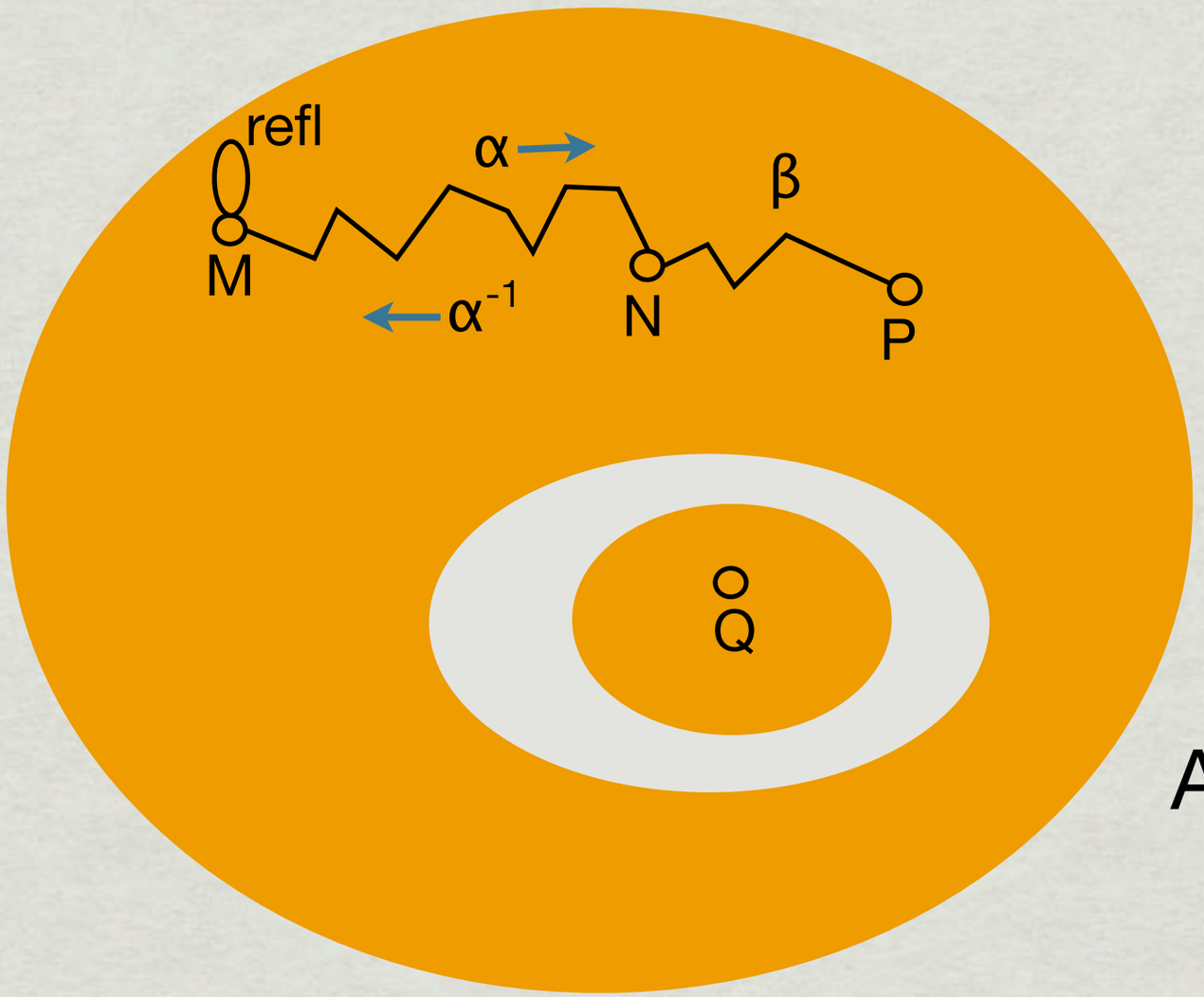
Operations on Paths



refl : $Id_A(M,M)$

$\alpha^{-1} : Id_A(N,M)$

Operations on Paths

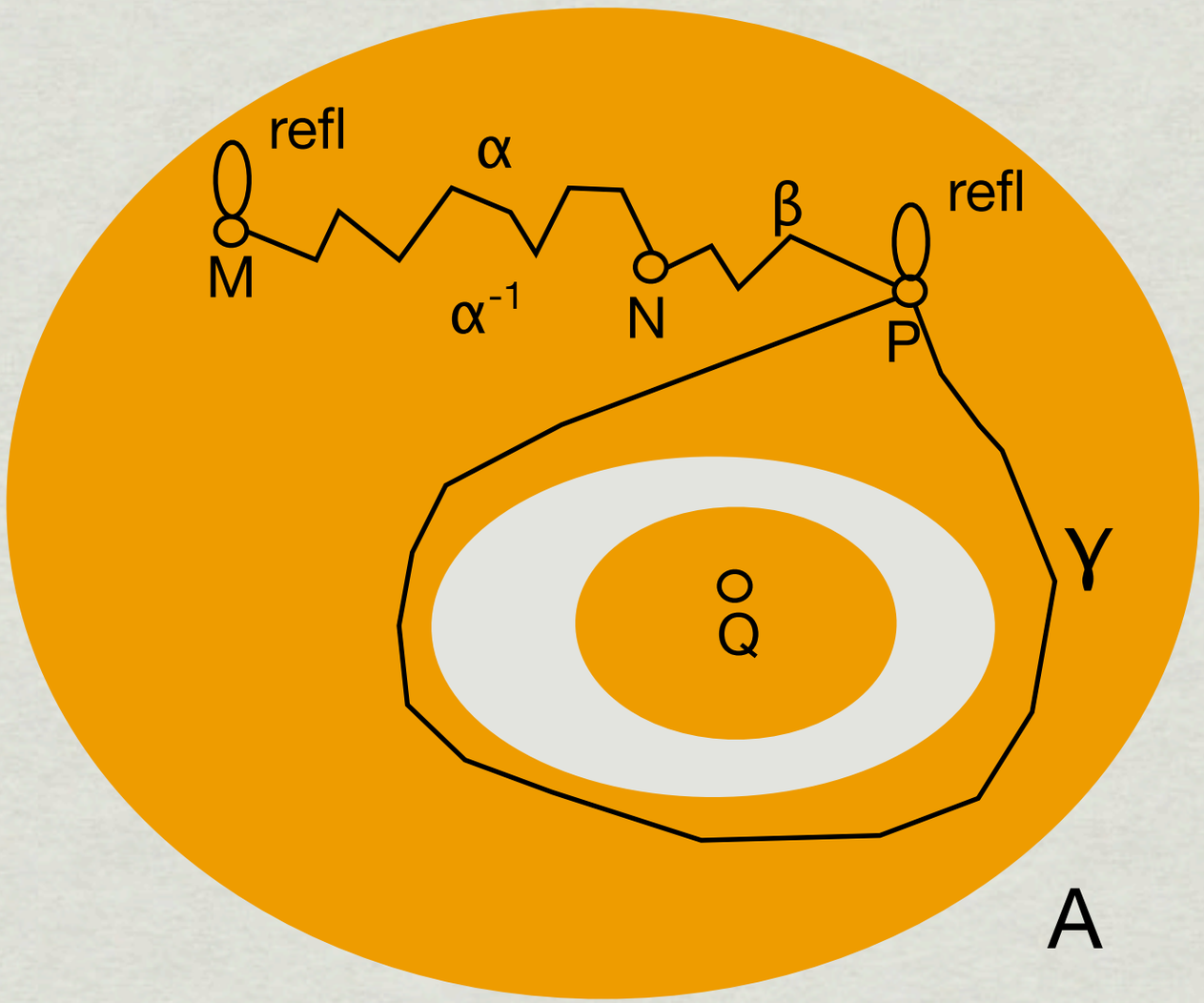


refl : $Id_A(M,M)$

$\alpha^{-1} : Id_A(N,M)$

$\beta \circ \alpha : Id_A(M,P)$

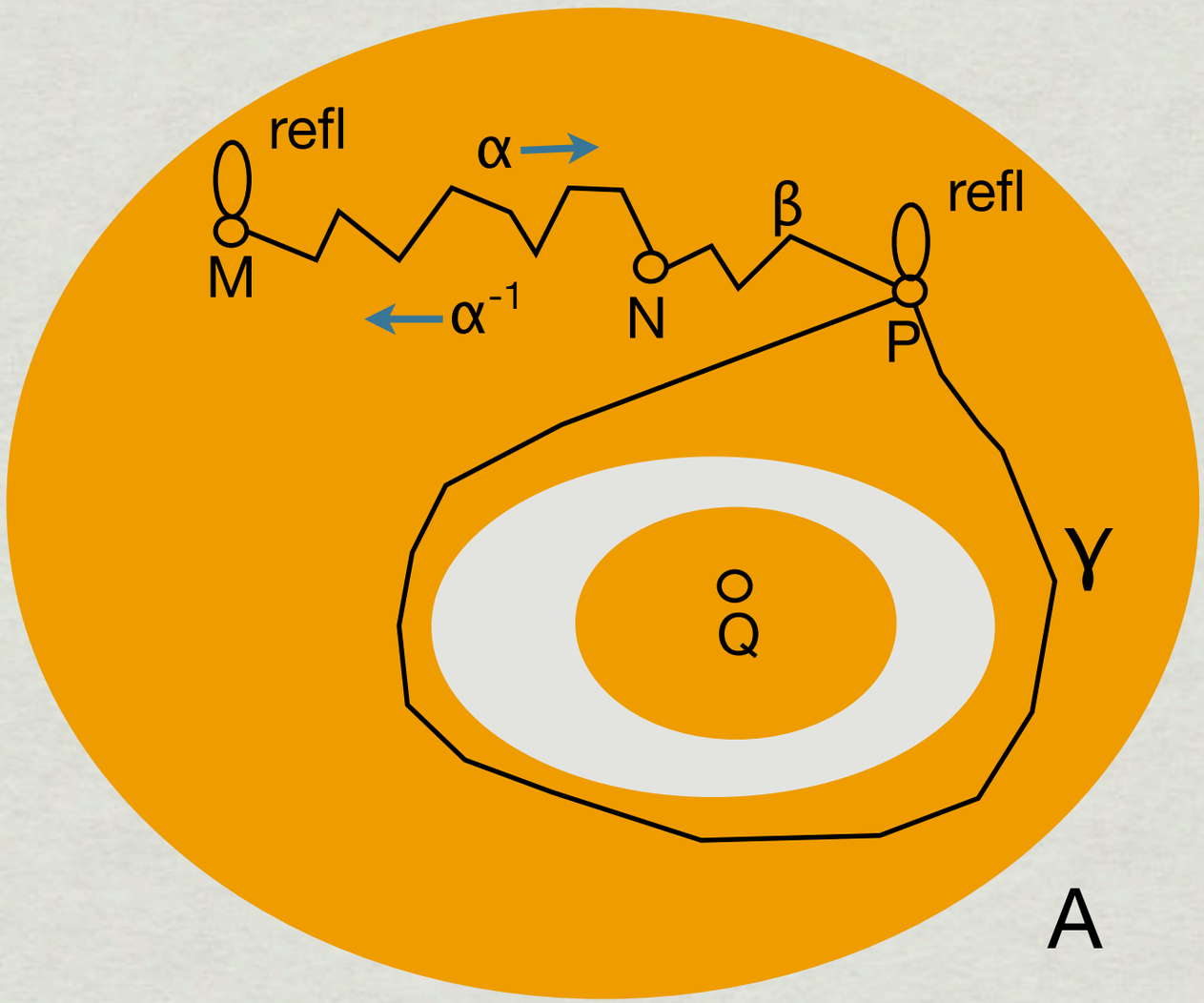
Equations on Paths



- associativity of o
- inverses cancel

γ != refl

Equations on Paths



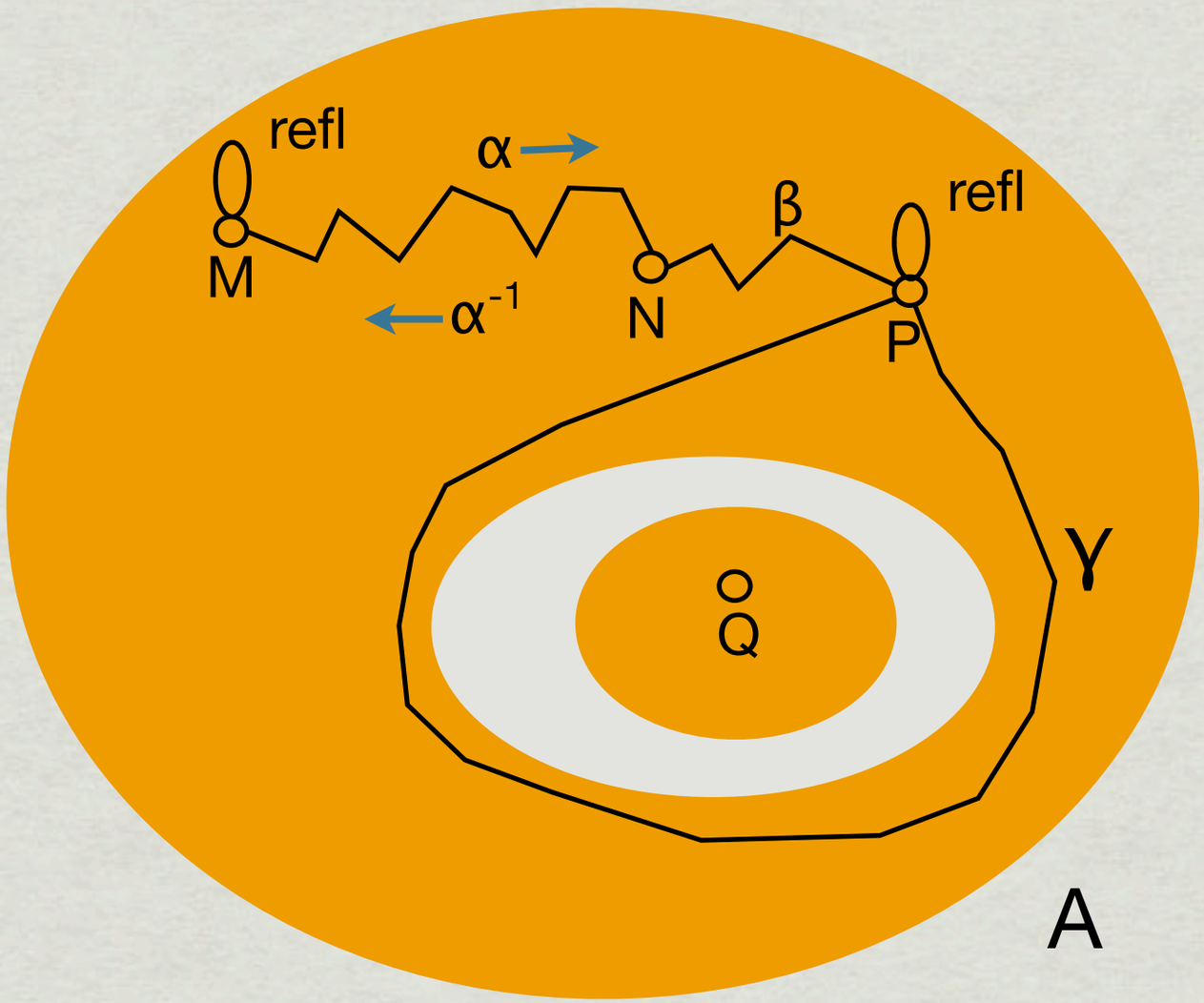
Yes:

- associativity of o
- inverses cancel

No:

γ != refl

Equations on Paths



Yes:

- associativity of o
- inverses cancel

No:

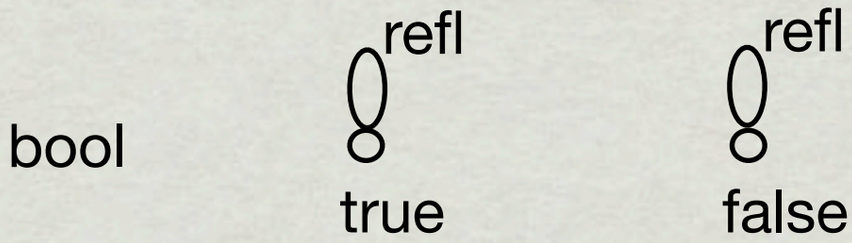
γ != refl

provable/not using identity types

Sets

Sets

Familiar base types are sets = discrete spaces

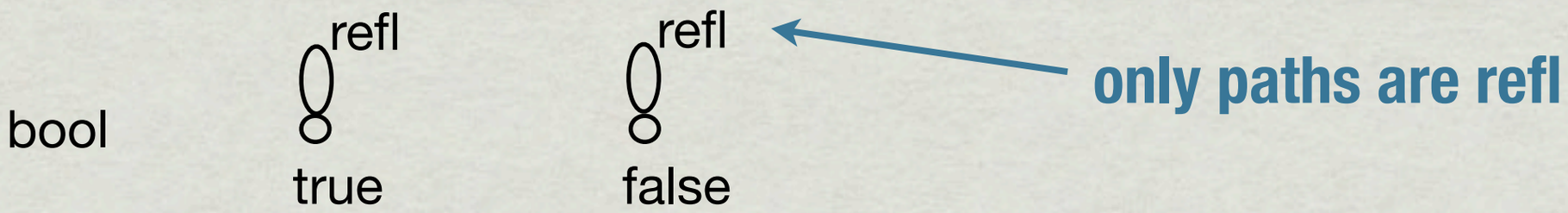


Π Σ Id W preserve set-hood

need a non-trivial base type to get started…

Sets

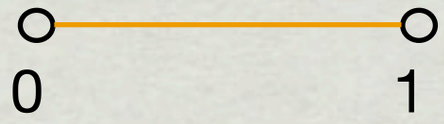
Familiar base types are sets = discrete spaces



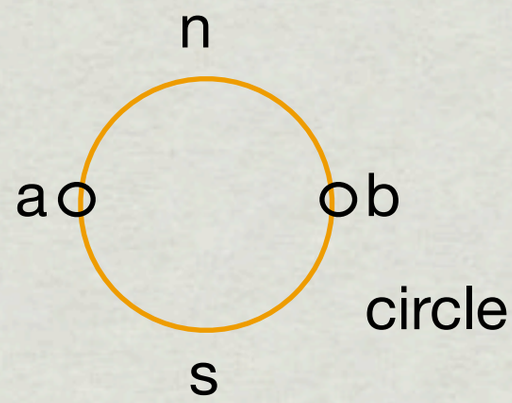
Π Σ Id W preserve set-hood

need a non-trivial base type to get started…

Higher-dimensional inductive types

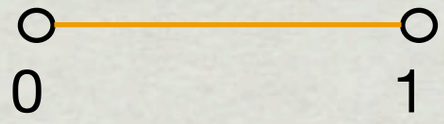


interval

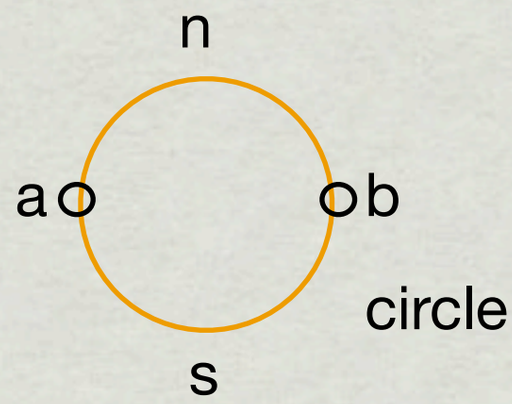


circle

Higher-dimensional inductive types



interval



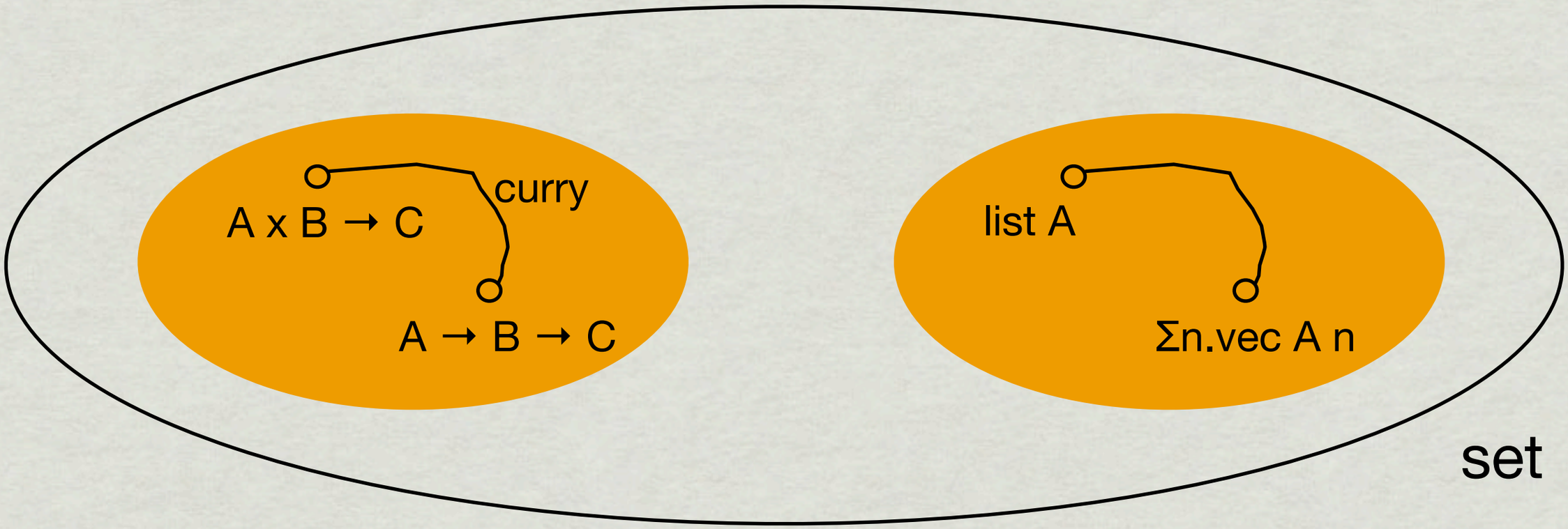
circle

0,1 : I
seg : Id(0,1)

a,b : C
n : Id(a,b)
s : Id(a,b)

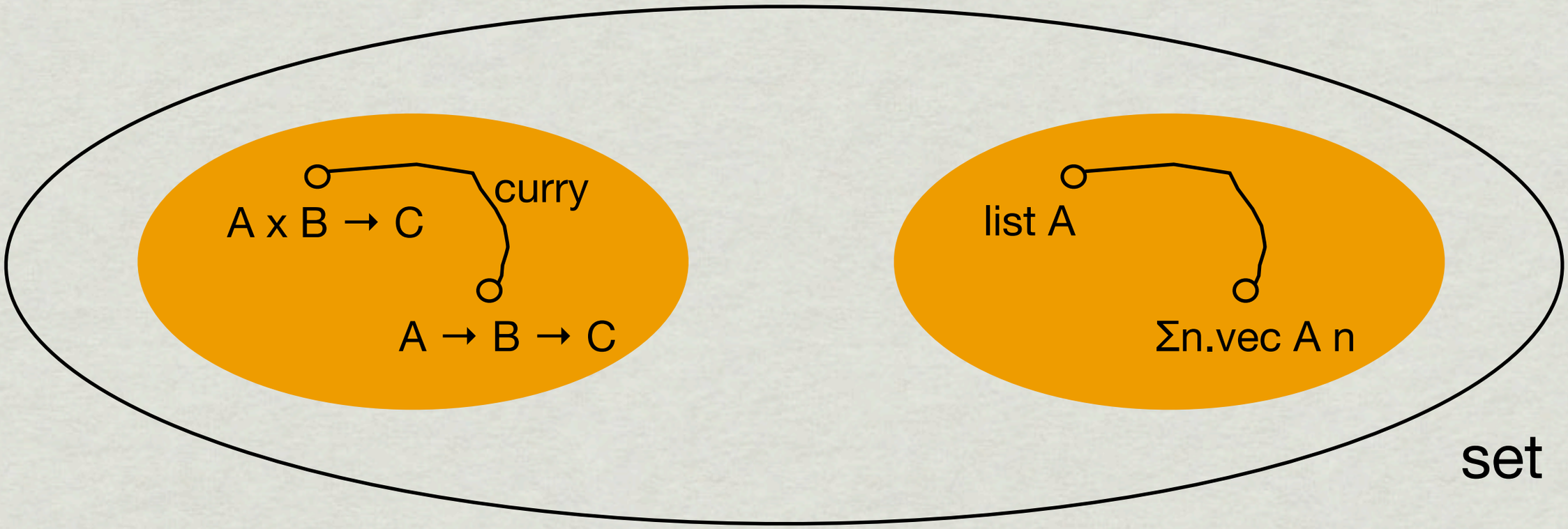
Univalence

[Hofmann&Streicher, Voevodsky]



Univalence

[Hofmann&Streicher, Voevodsky]

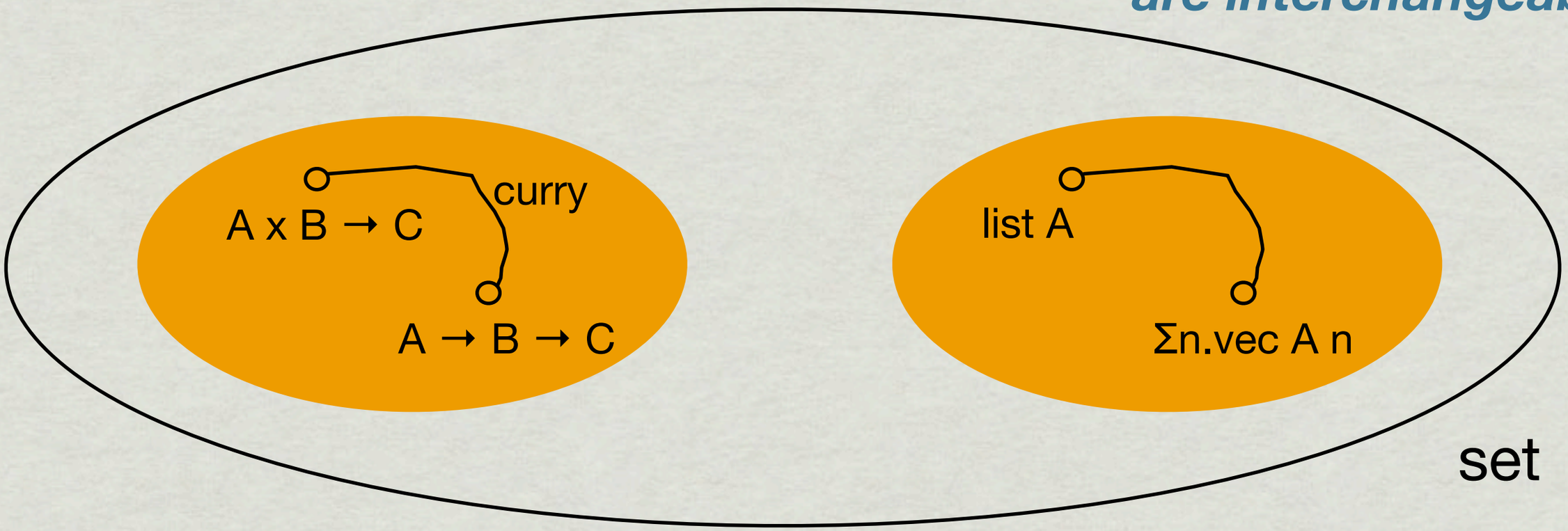


set = universe (type of types) of sets
where *paths are isomorphisms*

Univalence

[Hofmann&Streicher, Voevodsky]

isomorphic sets are interchangeable



set = universe (type of types) of sets
where *paths are isomorphisms*

Homotopy Type Theory

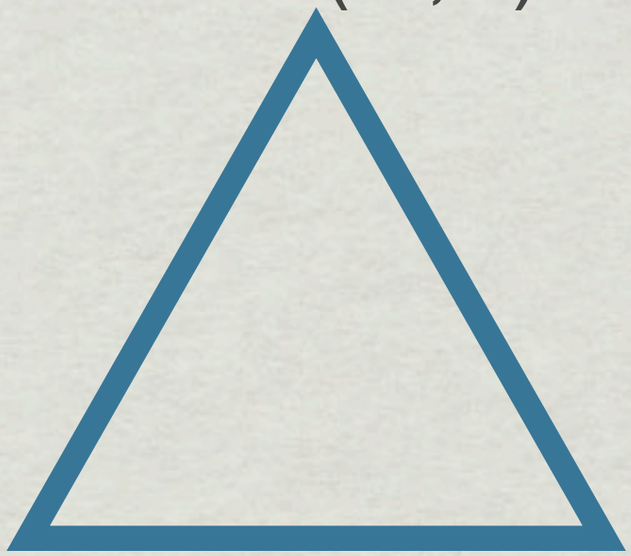
dependent type theory

A : type

M, N : A

$\alpha : Id_A(M,N)$

univalence is sound!



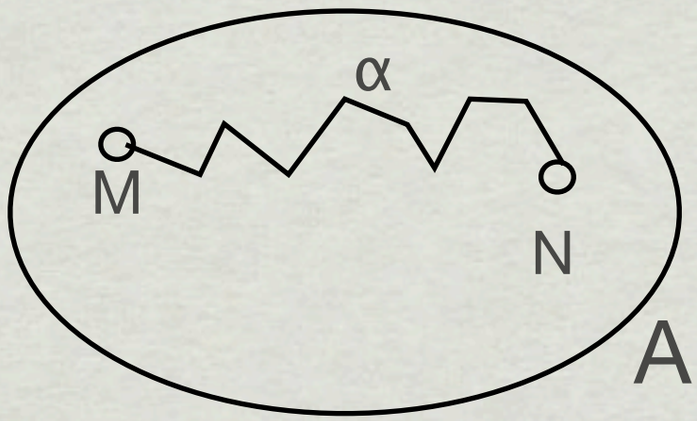
category theory

A is a groupoid

M,N objects

$\alpha : M \to N$ in A

homotopy theory



Homotopy Type Theory

Intensional identity types are compatible with rich notions of equivalence, where two terms can be equivalent in multiple different, ***computationally relevant****, ways.*

Applications:

- Formalization of math
- Code reuse/generic programming

Homotopy Type Theory

Intensional identity types are compatible with rich notions of equivalence, where two terms can be equivalent in multiple different, ***computationally relevant****, ways.*

Applications:

- Formalization of math
- Code reuse/generic programming

To put it to work, we need to draw out a generic program hiding inside of type theory.

Outline

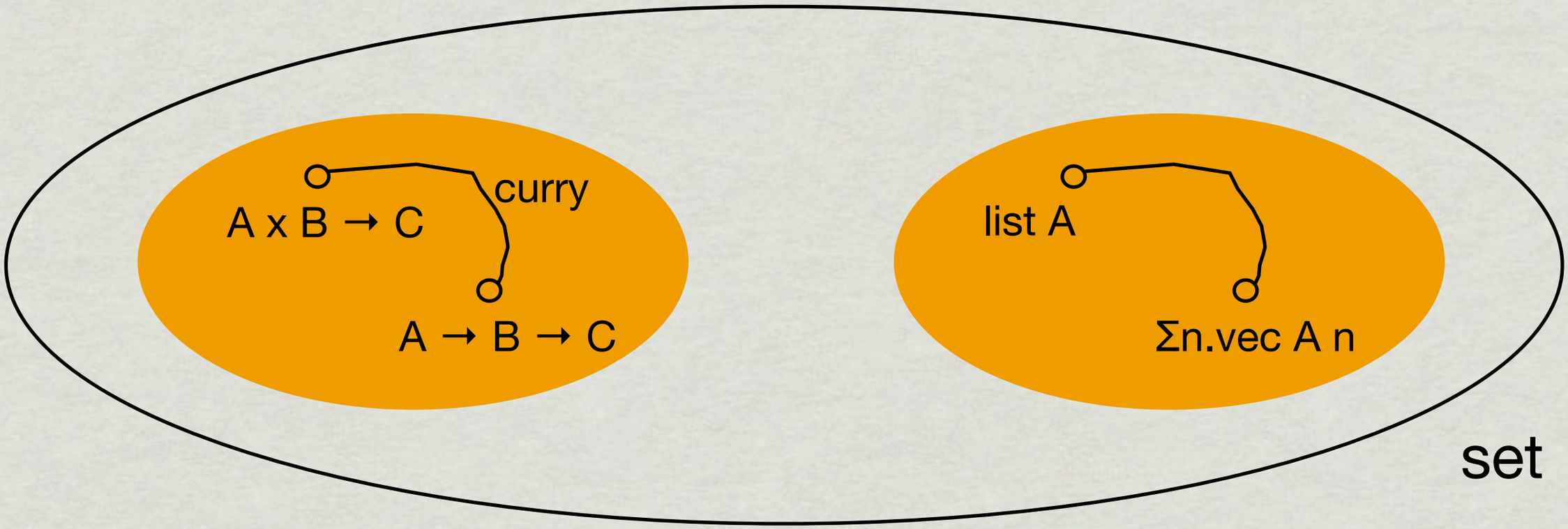
- What is homotopy type theory?
- Univalence axiom breaks canonicity
- Contributions: Judgemental 2-dimensional type theory and proof of canonicity

Outline

- What is homotopy type theory?
- **Univalence axiom breaks canonicity**
- Contributions: Judgemental 2-dimensional type theory and proof of canonicity

Univalence for **set**

subst: isomorphic sets are interchangeable



set = universe (type of types) of sets
where *paths are isomorphisms*

Type Isomorphisms

```
data list (A : set) : set =
   [] : list A
 | :: : A → list A → list A
```

```
data vec (A : set) (n : N) : set =
   Nil      : vec A 0
 | Cons : ∀n. A → vec A n → vec A (n+1)
```

$$\text{list } A \cong \Sigma\ n{:}\text{nat}.\ \text{vec } A\ n$$

Type Isomorphisms

list A ≅ Σ n:N. vec A n

fromlist : list A → Σn:N.vec A n
fromlist = ...

tolist : Σn:N.vec n A → list A
tolist = ...

and prove that they compose to the identity function

Type Isomorphisms

$f : A \to B$

$g : B \to A$

$\alpha : Id_{B \to B}(g \circ f, id)$

$\beta : Id_{A \to A}(f \circ g, id)$

$iso(f,g,\alpha,\beta) : Iso(A,B)$

Type Isomorphisms

fromlist : list A → Σn:N.vec A n

f : A → B
g : B → A
α : $Id_{B→B}$(g o f, id)
β : $Id_{A→A}$(f o g, id)

---

iso(f,g,α,β) : Iso(A,B)

Iso(list A,Σn:N.vec A n)

Type Isomorphisms

fromlist : list A → Σn:N.vec A n

f : A → B
g : B → A
α : $Id_{B→B}$(g o f, id)
β : $Id_{A→A}$(f o g, id)

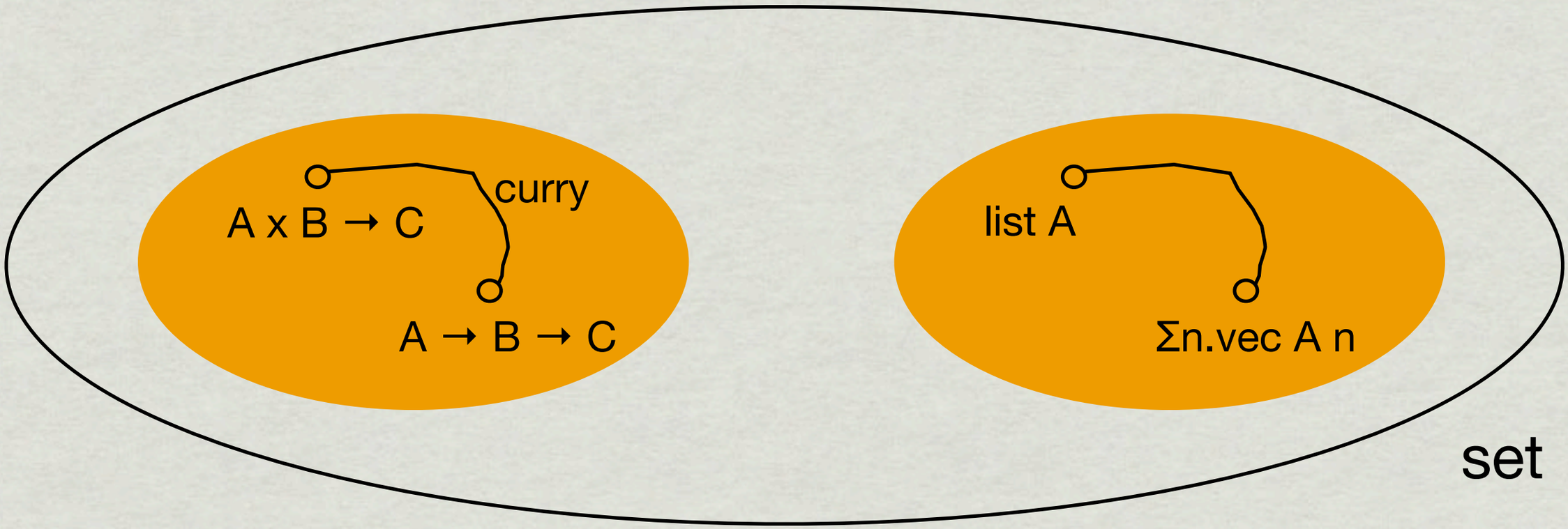
---

iso(f,g,α,β) : Iso(A,B)

Iso(list A,Σn:N.vec A n)

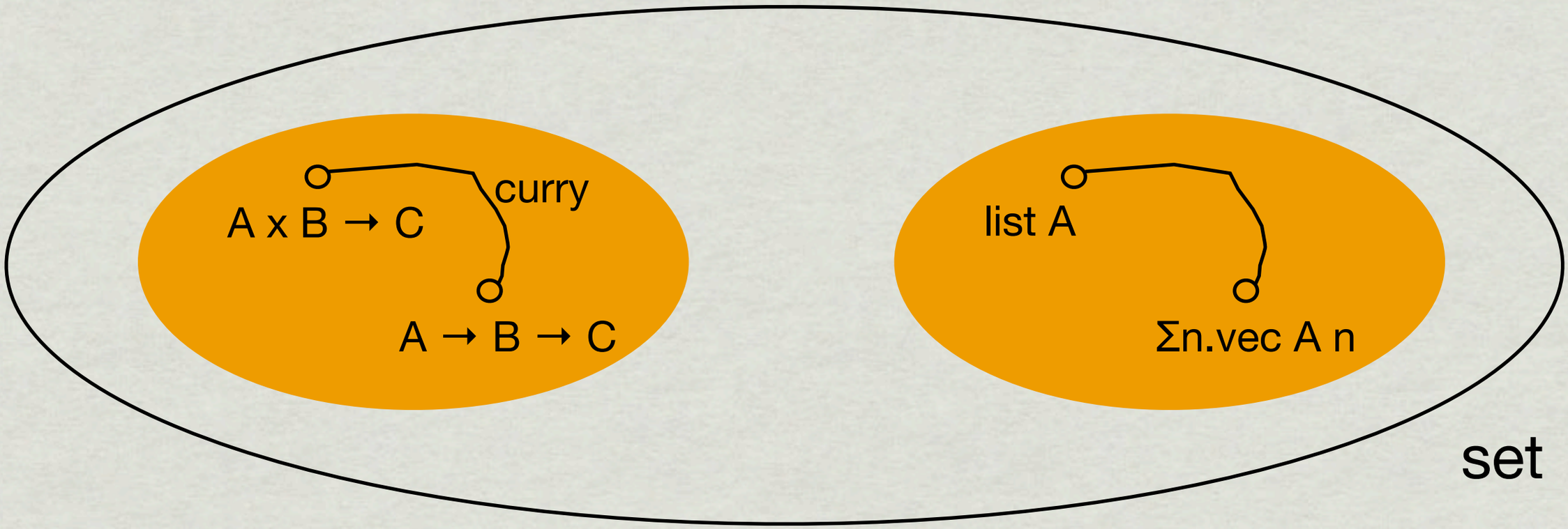
Iso has computational content!
(structure, not proposition)

Univalence Axiom



Univalence Axiom

univalence : Iso(A,B) → Id_{set}(A,B)



Isomorphic Sets are Interchangeable

$$\frac{C : \mathrm{set} \to \mathrm{set} \quad \alpha : \mathrm{Id}_{\mathrm{set}}(A,B) \quad P : C[A]}{\mathrm{subst}_C\ \alpha\ P : C[B]}$$

$C[\Sigma n{:}N.\mathrm{vec}\ A\ n]$

Isomorphic Sets are Interchangeable

$$\text{univalence} : \text{Iso}(A,B) \to \text{Id}_{\text{set}}(A,B)$$

$$\frac{C : \text{set} \to \text{set} \qquad \alpha : \text{Id}_{\text{set}}(A,B) \qquad P : C[A]}{\text{subst}_C\ \alpha\ P : C[B]}$$

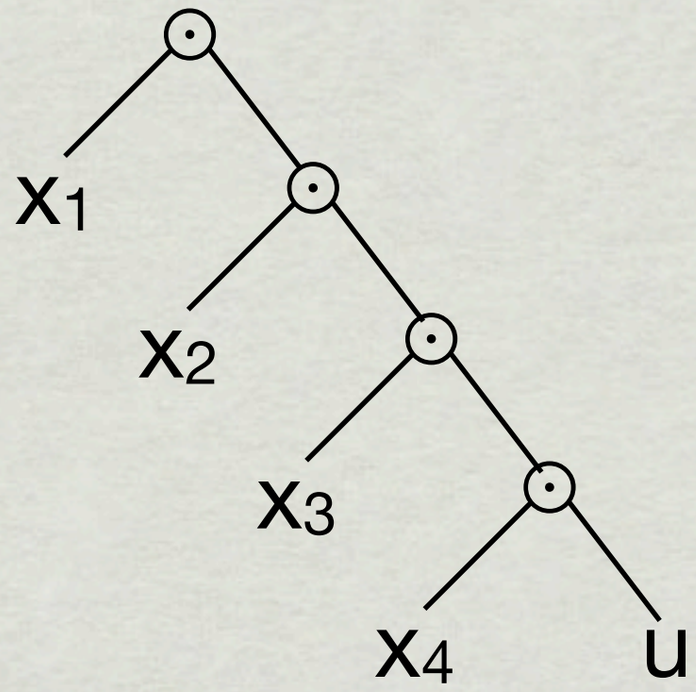
C[list A]

C[Σn:N.vec A n]

Parallel Programming

reduce : (A → A → A) → A → A sequence → A

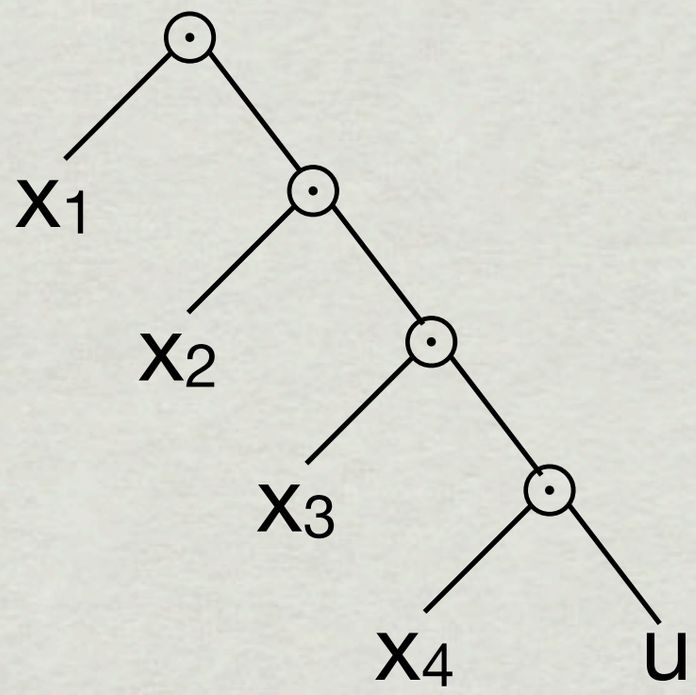
reduce ⊙ u [x_1,x_2,x_3,x_4] =



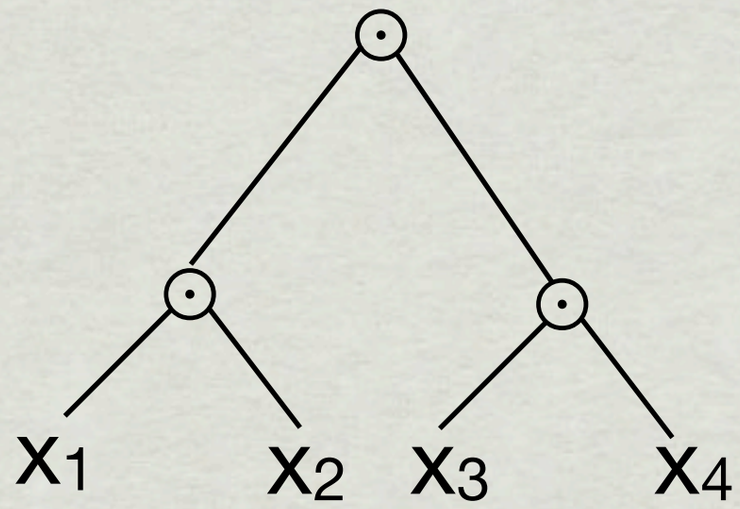
Parallel Programming

reduce : (A → A → A) → A → A sequence → A

reduce ⊙ u [x_1,x_2,x_3,x_4] =



if ⊙ is associative with unit u, can evaluate in parallel with same result:

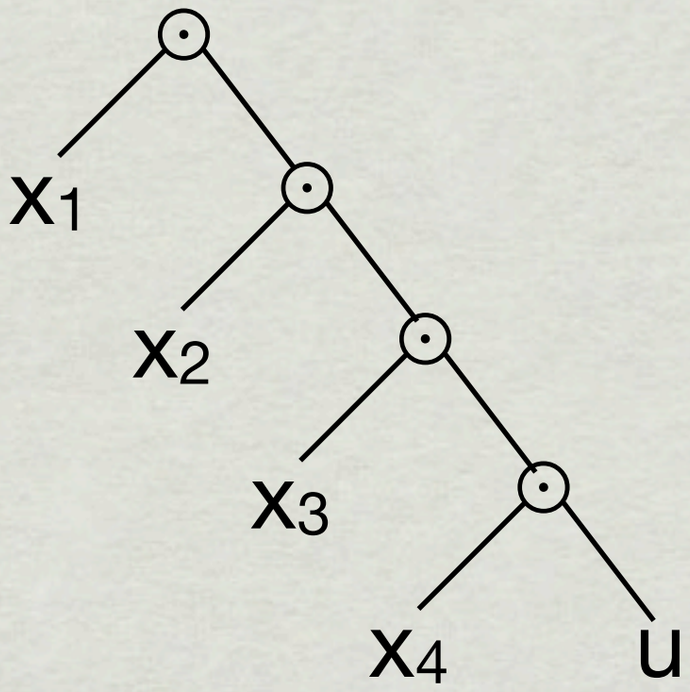


Parallel Programming

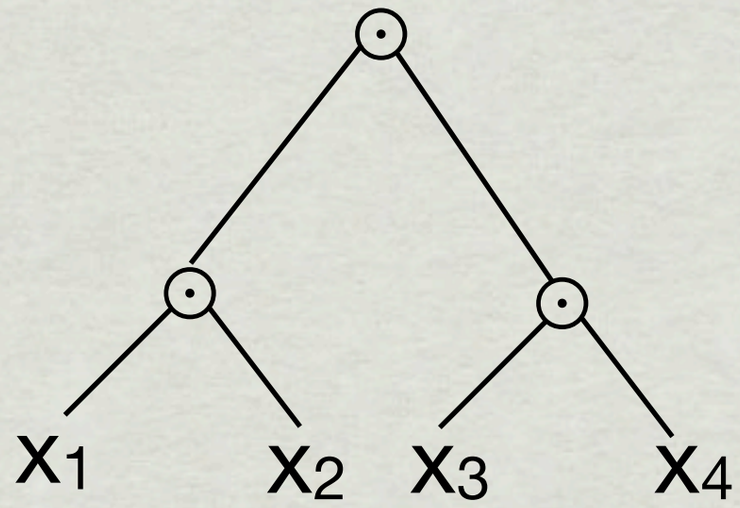
monoid

reduce : (A → A → A) → A → A sequence → A

reduce ⊙ u [x_1,x_2,x_3,x_4] =



if ⊙ is associative with unit u, can evaluate in parallel with same result:



Monoid

Monoid : set → set
Monoid X = Σ ⊙ : X → X → X.
Σ u : X.
(Πx,y,z:X. Id (x ⊙ (y ⊙ z)) ((x ⊙ y) ⊙ z))
* (Πx.Id (x ⊙ u) x)
* (Πx.Id (u ⊙ x) x)

Monoid

E.g. (append,[],…) : Monoid (list A)

Monoid : set → set

Monoid X = Σ ⊙ : X → X → X.

Σ u : X.

(Πx,y,z:X. Id (x ⊙ (y ⊙ z)) ((x ⊙ y) ⊙ z))

* (Πx.Id (x ⊙ u) x)

* (Πx.Id (u ⊙ x) x)

Monoid

E.g. (append,[],…) : Monoid (list A)

Monoid : set → set

Monoid X = Σ ⊙ : X → X → X.

Σ u : X.

(Πx,y,z:X. Id (x ⊙ (y ⊙ z)) ((x ⊙ y) ⊙ z))

* (Πx.Id (x ⊙ u) x)

* (Πx.Id (u ⊙ x) x)

Code reuse:

Given Iso(list A, Σn:N.vec A n)

make Monoid(list A) → Monoid(Σ n:nat. vec A n)

Interchange iso. sets

Make Monoid(list A) → Monoid(Σ n:nat. vec A n)

From

Interchange iso. sets

Make Monoid(list A) → Monoid(Σ n:nat. vec A n)

From (fromlist,tolist,…) : Iso(list A, Σn:N.vec n)

Interchange iso. sets

Make Monoid(list A) → Monoid(Σ n:nat. vec A n)

From (fromlist,tolist,…) : Iso(list A, Σn:N.vec n)

$$\frac{C : A \to \mathrm{set} \qquad \alpha : \mathrm{Id}_A(M,N) \qquad P : C[M]}{\mathrm{subst}_C\ \alpha\ P : C[N]}$$

Interchange iso. sets

Make Monoid(list A) → Monoid(Σ n:nat. vec A n)

From (fromlist,tolist,…) : Iso(list A, Σn:N.vec n)

$$\frac{C : A \to \mathsf{set} \quad \alpha : \mathsf{Id}_A(M,N) \quad P : C[M]}{\mathsf{subst}_C\ \alpha\ P : C[N]}$$

$$\mathsf{univalence} : \mathsf{Iso}(A,B) \to \mathsf{Id}_{\mathsf{set}}(A,B)$$

Interchange iso. sets

Make Monoid(list A) → Monoid(Σ n:nat. vec A n)

$\mathbf{subst}_{\mathbf{Monoid}}$(univalence (fromlist, tolist, …))

From (fromlist,tolist,…) : Iso(list A, Σn:N.vec n)

$$\frac{C : A \to \mathrm{set} \qquad \alpha : \mathrm{Id}_A(M,N) \qquad P : C[M]}{\mathrm{subst}_C\ \alpha\ P : C[N]}$$

univalence : Iso(A,B) → $\mathrm{Id}_{\mathrm{set}}$(A,B)

Interchange iso. sets

Make Monoid(list A) → Monoid(Σ n:nat. vec A n)

$\mathbf{subst}_{\mathbf{Monoid}}$(univalence (fromlist, tolist, …))

type-generic lifting of isos

From (fromlist,tolist,…) : Iso(list A, Σn:N.vec n)

$$\frac{C : A \to set \qquad \alpha : Id_A(M,N) \qquad P : C[M]}{subst_C\ \alpha\ P : C[N]}$$

univalence : Iso(A,B) → $Id_{set}(A,B)$

Failure of Canonicity

Recall that the only computation rule for subst is:

$$\mathsf{subst}_C\ \mathsf{refl}\ P \equiv P$$

Therefore

$$\mathsf{subst}_{\mathsf{Monoid}}(\mathsf{univalence}\ (\mathsf{fromlist}, \mathsf{tolist}, \ldots))$$

is well-typed but *stuck*:
violates Progress

Failure of Canonicity

Recall that the only computation rule for subst is:

$$\mathrm{subst}_C\ \mathrm{refl}\ P \equiv P$$

Therefore

$$\mathrm{subst}_{\mathrm{Monoid}}(\mathrm{univalence}\ (\mathrm{fromlist}, \mathrm{tolist}, \ldots))$$

is well-typed but *stuck*:
violates Progress

How do you compute with univalence?

Outline

- What is homotopy type theory?
- Univalence axiom breaks canonicity
- **Contributions: Judgemental 2-dimensional type theory and proof of canonicity**

Monoid : type → type
Monoid X = Σ ⊙:X→X→X. Σ u:X. …

Given iso(f,f^{-1},α,β) : Iso(A, B)
and (⊙,u,…) : Monoid(A)
make (⊙’,u’,…) : Monoid(B)

Monoid : type → type
Monoid X = Σ ⊙:X→X→X. Σ u:X. ...

Given iso(f,f^{-1},α,β) : Iso(A, B)
and (⊙,u,…) : Monoid(A)
make (⊙',u',…) : Monoid(B)

$\odot' = \lambda\ y_1,y_2{:}B\ .\ f\ ((f^{-1}\ y_1) \odot (f^{-1}\ y_2))$
$u' = f\ u$

Monoid : type → type
Monoid X = Σ ⊙:X→X→X. Σ u:X. ...

Given iso(f,f^{-1},α,β) : Iso(A, B)
and (⊙,u,...) : Monoid(A)
make (⊙',u',...) : Monoid(B)

⊙' = λ y_1,y_2:B . f ((f^{-1} y_1) ⊙ (f^{-1} y_2))
u' = f u

(y ⊙' u') = f ((f^{-1} y) ⊙ (f^{-1} (f u)))
= f (f^{-1} y ⊙ u) by α
= f (f^{-1} y) by unit law for ⊙ and u
= y by β

subst as a generic program

$$i = \text{univalence}\ (\text{iso}\ (f, f^{-1}, \alpha, \beta))$$

subst as a generic program

$$i = \text{univalence}\ (\text{iso}\ (f, f^{-1}, \alpha, \beta))$$

$$\text{subst}_{\mathbf{X:set.Monoid(X)}}\ i\ (\odot, u, \ldots)$$

subst as a generic program

$$i = \text{univalence}\ (\text{iso}\ (f, f^{-1}, \alpha, \beta))$$

$$\text{subst}_{\mathbf{X:set.Monoid(X)}}\ i\ (\odot, u, \ldots)$$

subst as a generic program

$$i = \text{univalence}\ (\text{iso}\ (f, f^{-1}, \alpha, \beta))$$

$$\text{subst}_{\mathbf{X:set.Monoid(X)}}\ i\ (\odot, u, \ldots)$$

$$\equiv (\lambda y_1, y_2.\ f\ (\ (f^{-1}\ y_1) \odot (f^{-1}\ y_2)\ )\ f\ u,\ \ldots)$$

subst as a generic program

$$i = \text{univalence}\ (\text{iso}\ (f, f^{-1}, \alpha, \beta))$$

$$\text{subst}_{X:\text{set}.\text{Monoid}(X)}\ i\ (\odot, u, \ldots)$$

$$\equiv \text{subst}_{X:\text{set}.\Sigma \odot:X\to X\to X.\Sigma u:X.\ldots}\ i\ (\odot, u, \ldots)$$

$$\equiv (\lambda y_1, y_2.\ f\ (\ (f^{-1}\ y_1) \odot (f^{-1}\ y_2)\ )\ f\ u,\ \ldots)$$

subst as a generic program

$$i = \text{univalence}\ (\text{iso}\ (f, f^{-1}, \alpha, \beta))$$

$\text{subst}_{X:\text{set}.\text{Monoid}(X)}\ i\ (\odot, u, \ldots)$

$\equiv \text{subst}_{X:\text{set}.\Sigma \odot:X\to X\to X.\Sigma u:X.\ldots}\ i\ (\odot, u, \ldots)$

$\equiv (\text{subst}_{x.x\to x\to x}\ i\ \odot,$
$\text{subst}_{x.x}\ i\ u, \ldots)$

$\equiv (\lambda y_1, y_2.\ f\ (\ (f^{-1}\ y_1) \odot (f^{-1}\ y_2)\ )$
$f\ u, \ldots)$

subst as a generic program

$$i = \text{univalence}\ (\text{iso}\ (f, f^{-1}, \alpha, \beta))$$

$\text{subst}_{X:\text{set}.\text{Monoid}(X)}\ i\ (\odot, u, \ldots)$

$\equiv \text{subst}_{X:\text{set}.\Sigma \odot:X\to X\to X.\Sigma u:X.\ldots}\ i\ (\odot, u, \ldots)$

$\equiv (\text{subst}_{x.x\to x\to x}\ i\ \odot,$
$\text{subst}_{x.x}\ i\ u, \ldots)$

$\equiv (\lambda y_1, y_2.\ \text{subst}_{x.x}\ i\ (\ (\text{subst}_{x.x}\ i^{-1}\ y_1)\ \odot\ (\text{subst}_{x.x}\ i^{-1}\ y_2)\ )$
$\text{subst}_{x.x}\ i\ u, \ldots)$

$\equiv (\lambda y_1, y_2.\ f\ (\ (f^{-1}\ y_1)\ \odot\ (f^{-1}\ y_2)\ )$
$f\ u, \ldots)$

Contributions

1. Define a 2-dimensional type theory
   - judgemental formulation of equivalence
   - **Key idea:** define subst/resp as generic programs
2. Prove canonicity:
   every closed M : bool is ≡ true or false
   - extend NuPRL semantics
     from setoids to groupoids
3. Internalize ≃ as Id-type and derive J

Contributions

Id_A can have comp. content but $Id_{Id(M,N)}$ is discrete

1. Define a 2-dimensional type theory
   - judgemental formulation of equivalence
   - **Key idea:** define subst/resp as generic programs

2. Prove canonicity:
   every closed M : bool is ≡ true or false
   - extend NuPRL semantics
     from setoids to groupoids

3. Internalize ≃ as Id-type and derive J

2-Dimensional Type Theory

Types $\Gamma \vdash A\ \mathrm{type}$

Terms $\Gamma \vdash M : A$

Equivalence $\Gamma \vdash \alpha : M \simeq_A N$

≡ of M's and α's

2D means no equivalence of equivalences

Families respect equivalence: generic programs

$$\frac{x{:}A \vdash C : \mathrm{type} \qquad \alpha : M \simeq_A N \qquad P : C[M/x]}{\mathrm{subst}_{x.C}\ \alpha\ P : C[N/x]}$$

$$\frac{x{:}A \vdash P : B \qquad \alpha : M \simeq_A N}{\mathrm{resp}\ (x.P)\ \alpha : \mathrm{map}_{x.B}\ \alpha\ P[M/x] \simeq_{B[N]} P[N/x]}$$

Contributions

1. Define a 2-dimensional type theory
   - judgemental formulation of equivalence
   - subst/resp as generic programs

2. **Prove canonicity:**
   every closed M : bool is ≡ true or false
   - **extend NuPRL semantics**
     from setoids to groupoids

3. Internalize ≃ as Id-type and derive J

Canonicity Theorem

No stuck closed terms, up to judgemental equality:

If $\cdot \vdash$ ***M:bool then either M*** $\equiv$ ***true or M*** $\equiv$ ***false***

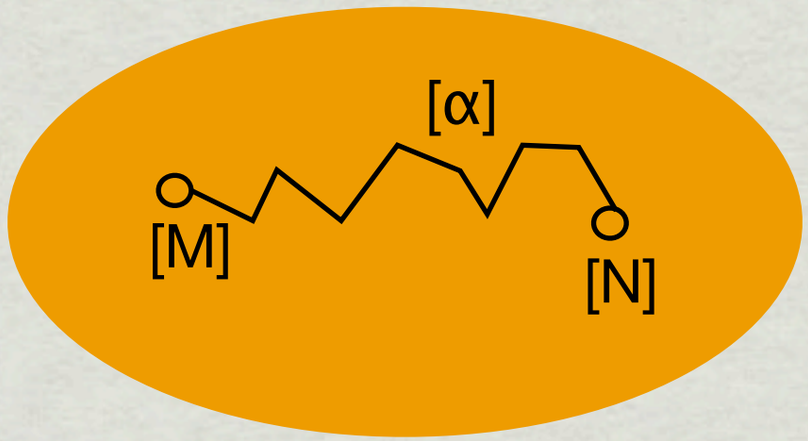
Proof extends reducibility from sets to spaces!

Canonicity: If $\cdot \vdash$ ***M:bool then***

either M $\equiv$ ***true or M*** $\equiv$ ***false***

Associate a reducibility groupoid with each type

[[A]] =

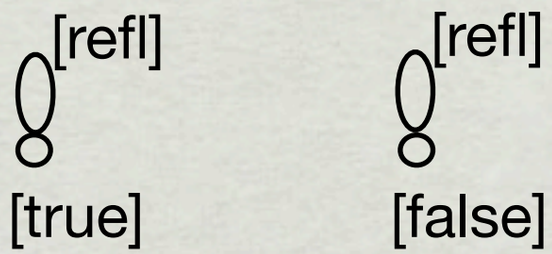


[M] means

≡ equiv. class

Reducibility groupoid for bool is discrete:

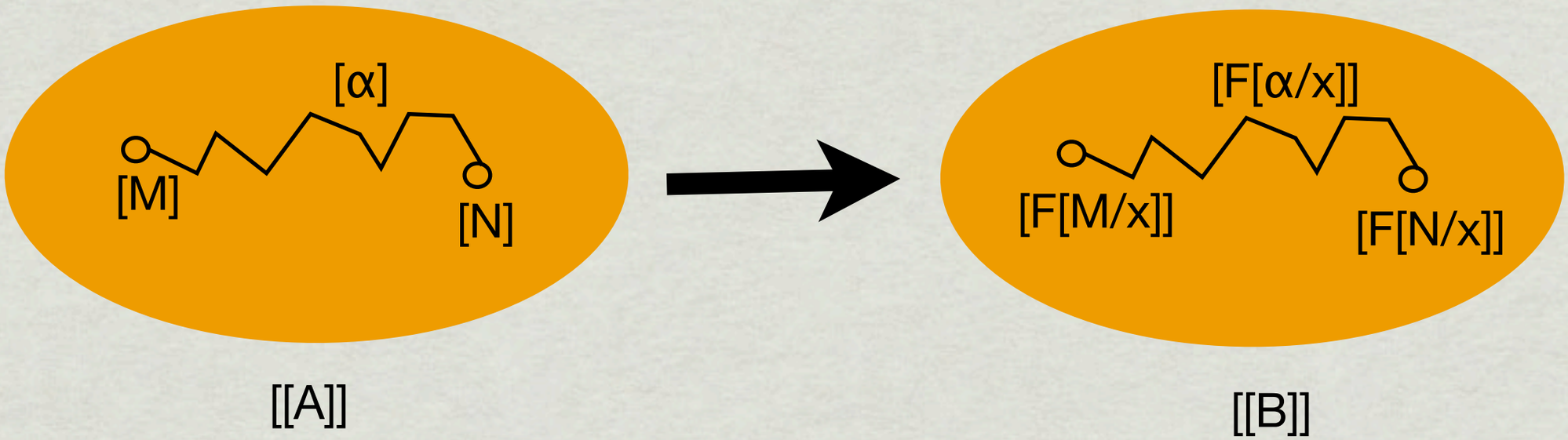
bool



Canonicity: If $\cdot \vdash$ ***M:bool then***

either M $\equiv$ ***true or M*** $\equiv$ ***false***

Reducible open terms are functors:
x:A $\vdash$ F : B is reducible iff



Fund. Lemma: *Well-formed types/terms are reducible*

Contributions

1. Define a 2-dimensional type theory
   - judgemental formulation of equivalence
2. Prove canonicity:
   every closed M : bool is ≡ true or false
   - extend NuPRL semantics
     from setoids to groupoids
3. Internalize ≃ as Id-type and derive J ← **validates judgemental approach**

Related Work

- Categorical/homotopy semantics of dependent types
- Type-directed equality for 1-dim type theory in OTT
- Coercive subtyping
- Explicit coercions
- Generic programming via functors

This work: computational interpretation of computationally-relevant equivalence for all of MLTT, in syntax

Future Work

$$\frac{\alpha : P \simeq_{\mathbf{Id}(M,N)} Q}{P \equiv Q}$$

Canonicity is necessary for an operational semantics, but not sufficient

- ❋ ≡ has more than β:
  - equality reflection for 2-dimensionality
- ❋ equational deduction, not reduction

Future work: find deterministic operational semantics for 2TT, and for fully higher-dimensional type theory

Homotopy Type Theory

Intensional identity types are compatible with rich notions of equivalence, where two terms can be equivalent in multiple different, ***computationally relevant****, ways.*

This work:

- **computational interpretation of 2D univalence by generic programming**
- **extend reducibility from relations to groupoids**

Reducibility

Reducibility groupoid $\langle \Gamma \rangle$ for Γ

- Objects: a set of eq. cls. of substitutions $\ll\theta\gg : \Gamma$
- Maps: a set of $\ll\delta\gg : \ll\theta_1\gg \to_{\langle\Gamma\rangle} \ll\theta_2\gg$ s.t. $\delta : \theta_1 \simeq \theta_2 : \Gamma$

Similarly for closed types, A

Reducibility functor $F : \langle A \rangle \to \langle B \rangle$ for $x : A \vdash M : B$

- Objects: $F(\ll N\gg) = \ll[N/x]M\gg$ for all objects $\ll N\gg$ in $\langle A\rangle$
- Maps: $F(\ll\alpha\gg) = \ll \mathsf{resp}_B(\alpha)(M)\gg$ for every $\ll\alpha\gg : \ll N_1\gg \to_{\langle A\rangle} \ll N_2\gg$

Reducibility

Reducibility family $F : \langle \Gamma \rangle \to \mathbf{GPD}$ for $\Gamma \vdash A$

- $F(\ll\theta_1\gg)$ is a red. gpd. for $A[\theta_1]$
- $F(\ll\delta\gg)$ is a red. func. for $x : A[\theta_1] \vdash \mathsf{map}_A(\delta)(x) : A[\theta_2]$, for each $\ll\delta\gg : \ll\theta_1\gg \to_{\langle A \rangle} \ll\theta_2\gg$

Interpretation of Types

Define $[\![\Gamma]\!]$ by induction:

- $[\![\bullet]\!] = \mathbf{1}$;
- $[\![\Gamma, x{:}A]\!] = \int_{[\![\Gamma]\!]} [\![A]\!]$: formal Grothendieck construction

Define family over $[\![\Gamma]\!]$ for $[\![\Gamma \vdash A]\!]$ by induction:

- $[\![\Gamma \vdash \mathsf{set}]\!] = \mathbf{const}(\mathbf{set})$: a universe of sets
- $[\![\Gamma \vdash \mathsf{El}(M)]\!] = \mathbf{elt}[M_*]$: discrete groupoids
- $[\![\Gamma \vdash \Pi x{:}A.B]\!] = \mathbf{pi}_{[\![\Gamma \vdash A]\!]}[\![\Gamma, x{:}A \vdash B]\!]$: dep. exponential

Canonicity

Everything in sight is reducible and preserves reducible equivalences. For example,

If Γ ctx, then $[\![\Gamma]\!]$ is a red. gpd. for Γ.

If $\Gamma \vdash A$ type, then $[\![\Gamma \vdash A]\!]$ is a red. fam. for A over $[\![\Gamma]\!]$.

If $\Gamma \vdash M : A$, then

- $\text{«}M[\theta_1]\text{»} \in \mathsf{Ob}\,[\![\Gamma \vdash A]\!](\text{«}\theta_1\text{»})$ for all $\text{«}\theta_1\text{»} \in \mathsf{Ob}\,[\![\Gamma]\!]$.
- $\text{«}M[\delta]\text{»} : \text{«}\,\mathsf{map}_A(\delta)(M[\theta_1])\text{»} \to_{[\![\Gamma \vdash A]\!](\theta_2)} \text{«}M[\theta_2]\text{»}$, whenever $\text{«}\delta\text{»} : \text{«}\theta_1\text{»} \to_{[\![\Gamma]\!]} \text{«}\theta_2\text{»}$.