Computing with Univalence

Dan Licata

Dependent Type Theory

Logical system underlying *proof assistants:* tools for

- Computer-checked math
- Programming

Dependent Type Theory

Logical system underlying *proof assistants:* tools for

- Computer-checked math
- Programming

even-or-odd : Π n:nat. (Σk. Id n 2*k) ∨ (Σk. Id n (2*k+1))

Dependent Type Theory

Logical system underlying *proof assistants:* tools for

- Computer-checked math
- Programming

even-or-odd : Π n:nat. (Σk. Id n 2*k) ∨ (Σk. Id n (2*k+1))

dependent function type (∀)

Dependent Type Theory

Logical system underlying *proof assistants:* tools for

- Computer-checked math
- Programming

disjoint union

even-or-odd : Π n:nat. (Σk. Id n 2*k) ∨ (Σk. Id n (2*k+1))

dependent function type (∀)

Dependent Type Theory

Logical system underlying *proof assistants:* tools for

- Computer-checked math
- Programming

disjoint union

even-or-odd : Π n:nat. (Σk. Id n 2*k) ∨ (Σk. Id n (2*k+1))

dependent function type (∀)

dependent pair type (∃)

Dependent Type Theory

Logical system underlying *proof assistants:* tools for

- Computer-checked math
- Programming

disjoint union

even-or-odd : Π n:nat. (Σk. Id n 2*k) ∨ (Σk. Id n (2*k+1))

dependent function type (∀)

dependent pair type (∃)

identity type (=)
identical terms are interchangeable in all contexts

Dependent Type Theory

Logical system underlying *proof assistants:* tools for

- Computer-checked math
- Programming

disjoint union

even-or-odd : Π n:nat. (Σk. Id n 2*k) ∨ (Σk. Id n (2*k+1))

dependent function type (∀)

dependent pair type (∃)

identity type (=)
identical terms are interchangeable in all contexts

proofs = programs

Example

```
even-or-odd : Π n:nat. (Σk. Id n 2*k) ∨ (Σk. Id n (2*k+1))
even-or-odd 0         = Even even0
even-or-odd (1 + n) =
  case (even-or-odd n) of
    Even nIsEven → Odd (odd1+ nIsEven)
    Odd nIsOdd  → Even (even1+ nIsOdd)
```

Example

```
even-or-odd : Π n:nat. (Σk. Id n 2*k) ∨ (Σk. Id n (2*k+1))
even-or-odd 0         = Even even0
even-or-odd (1 + n) =
  case (even-or-odd n) of
    Even nIsEven → Odd (odd1+ nIsEven)
    Odd nIsOdd  → Even (even1+ nIsOdd)
```

Computing by calculation:

```
even-or-odd 3 = case (even-or-odd 2) of …
              = <steps elided>
              = case (Even … ) of …
              = Odd …
```

Canonicity Property

“programs don’t get stuck”

If M : nat
then
∃ a numeral k s.t. M $\longmapsto^*$ k

algorithm for =

guides the design of a type theory

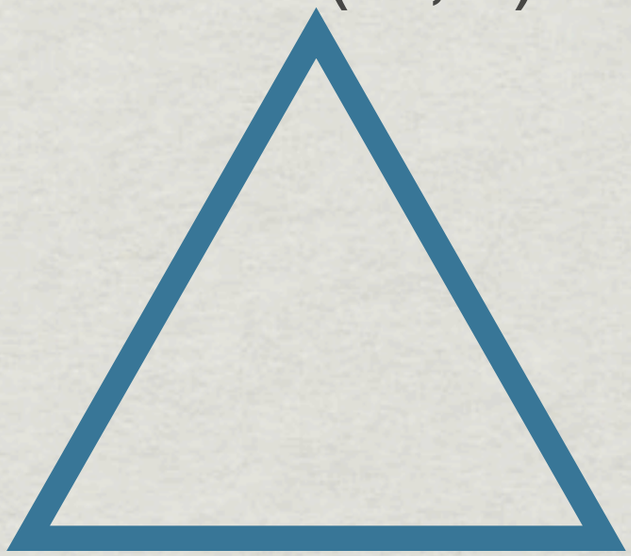
Homotopy Type Theory

dependent type theory

A : type

M : A

$\alpha : Id_A(M,N)$



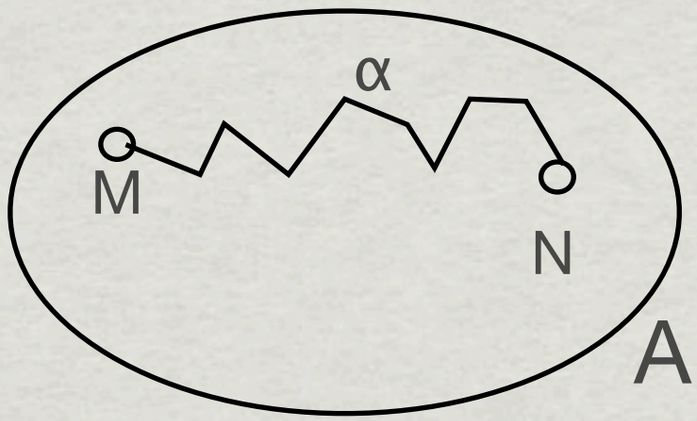
category theory

A is a groupoid

M is an object

$\alpha : M \to N$ in A

homotopy theory

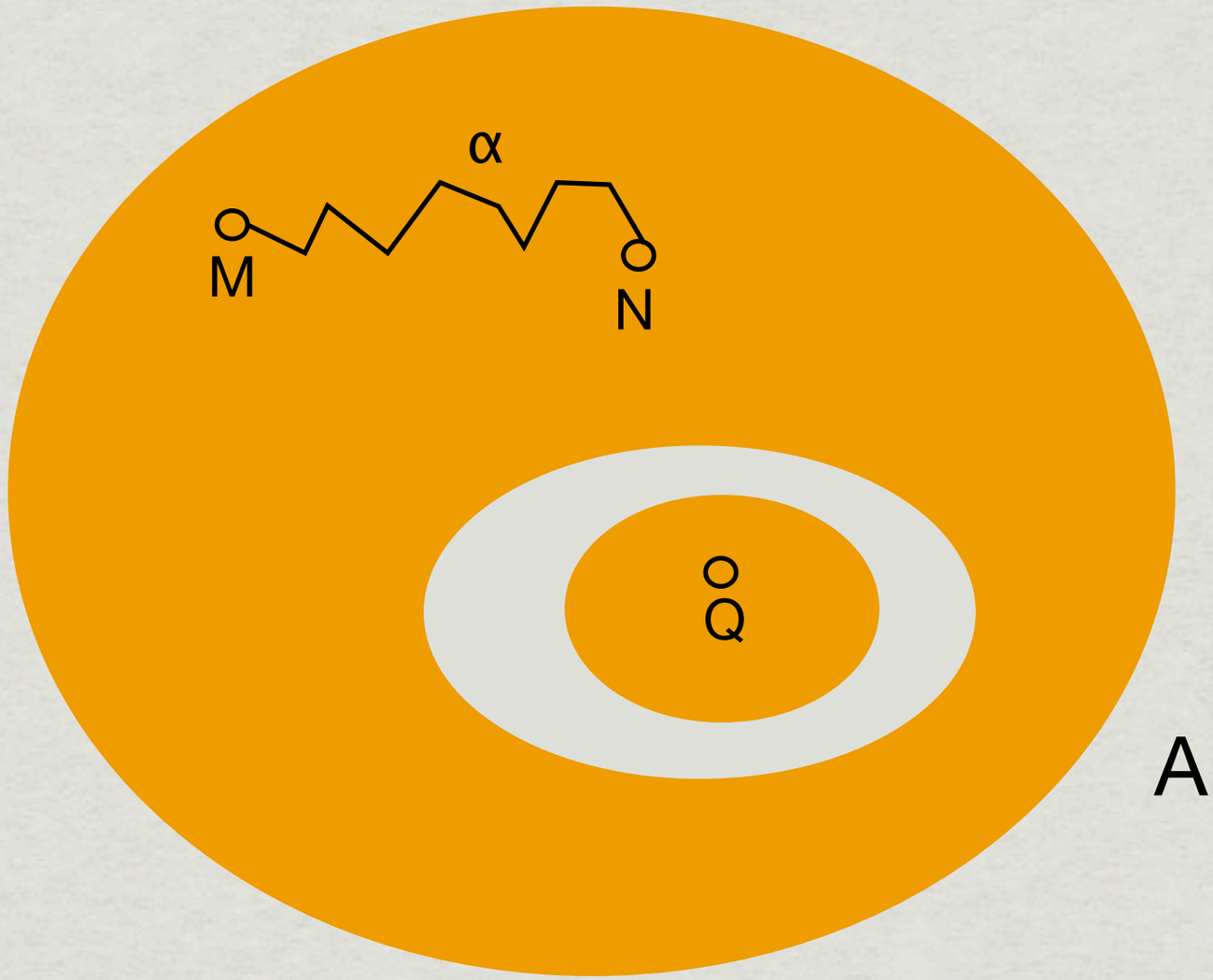


Homotopy Type Theory

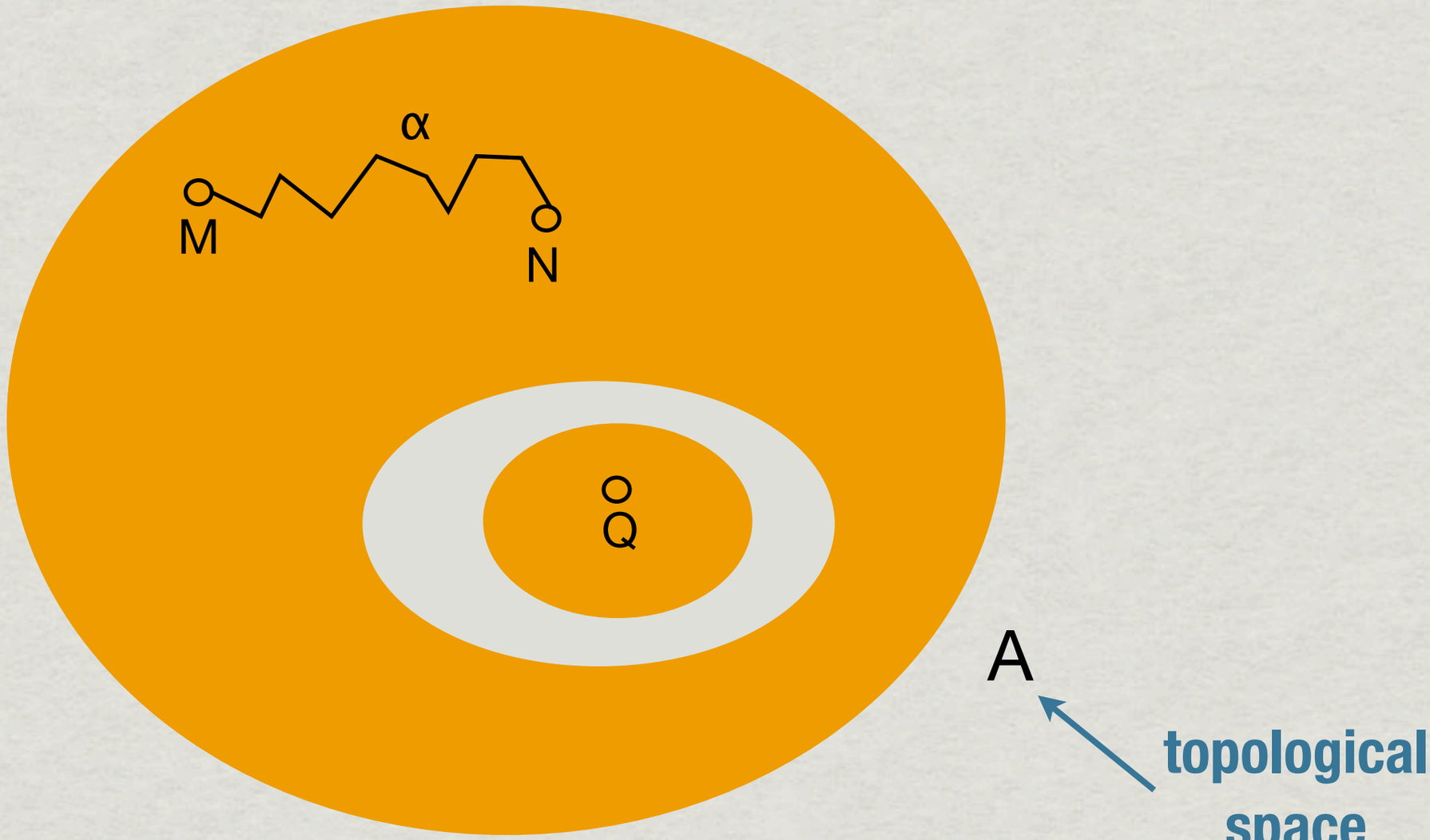
Theorem: Martin-Löf's intensional type theory has semantics in homotopy theory (spaces as types) and category theory (groupoids as types)

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

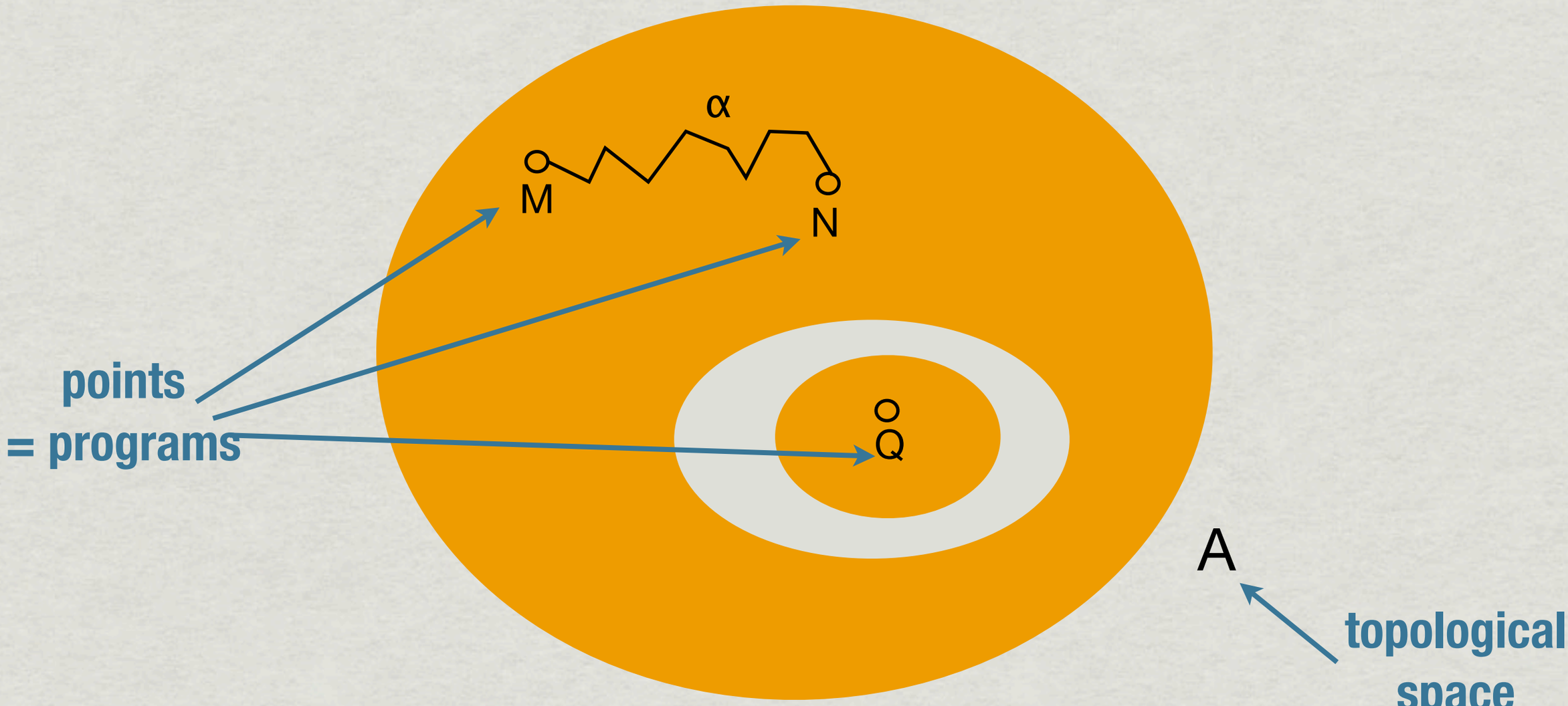
Types as Spaces



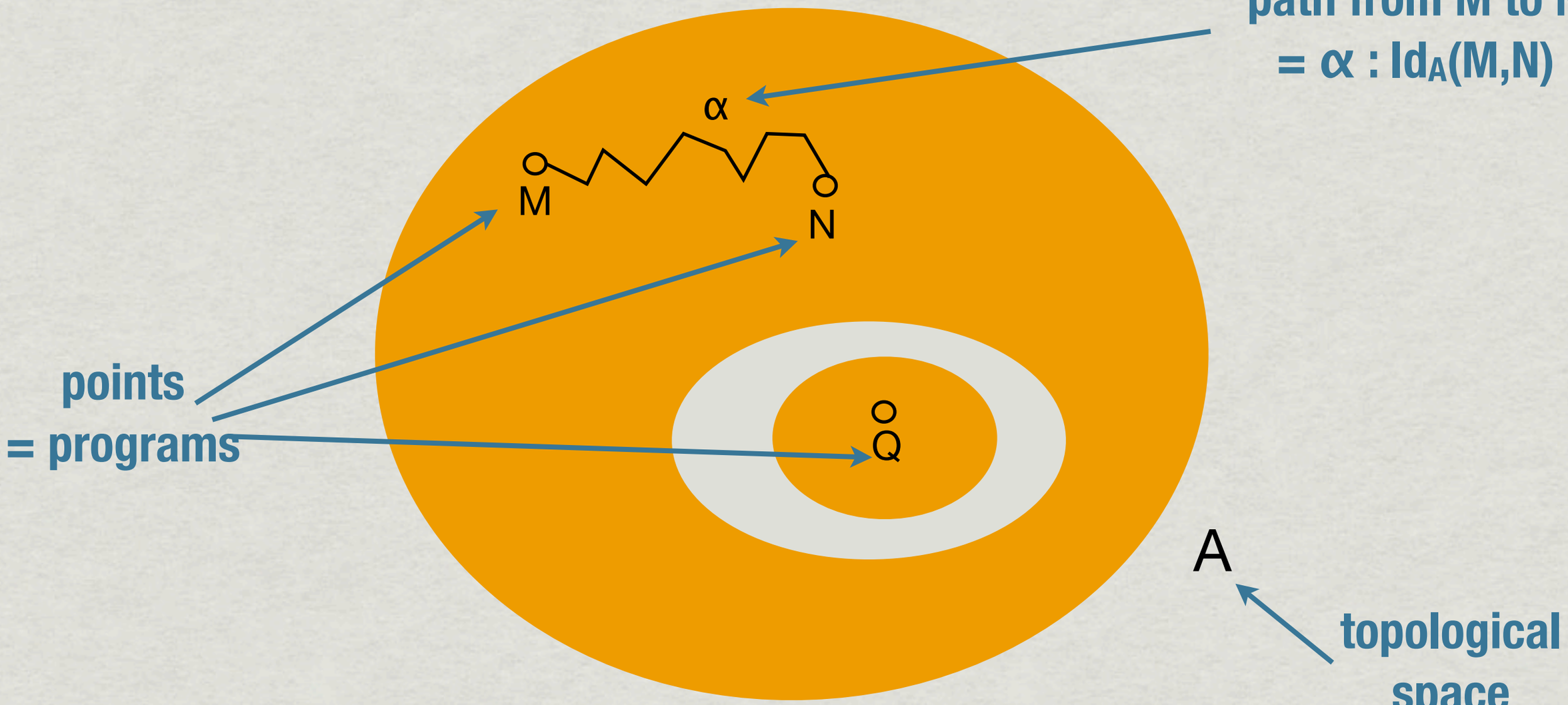
Types as Spaces



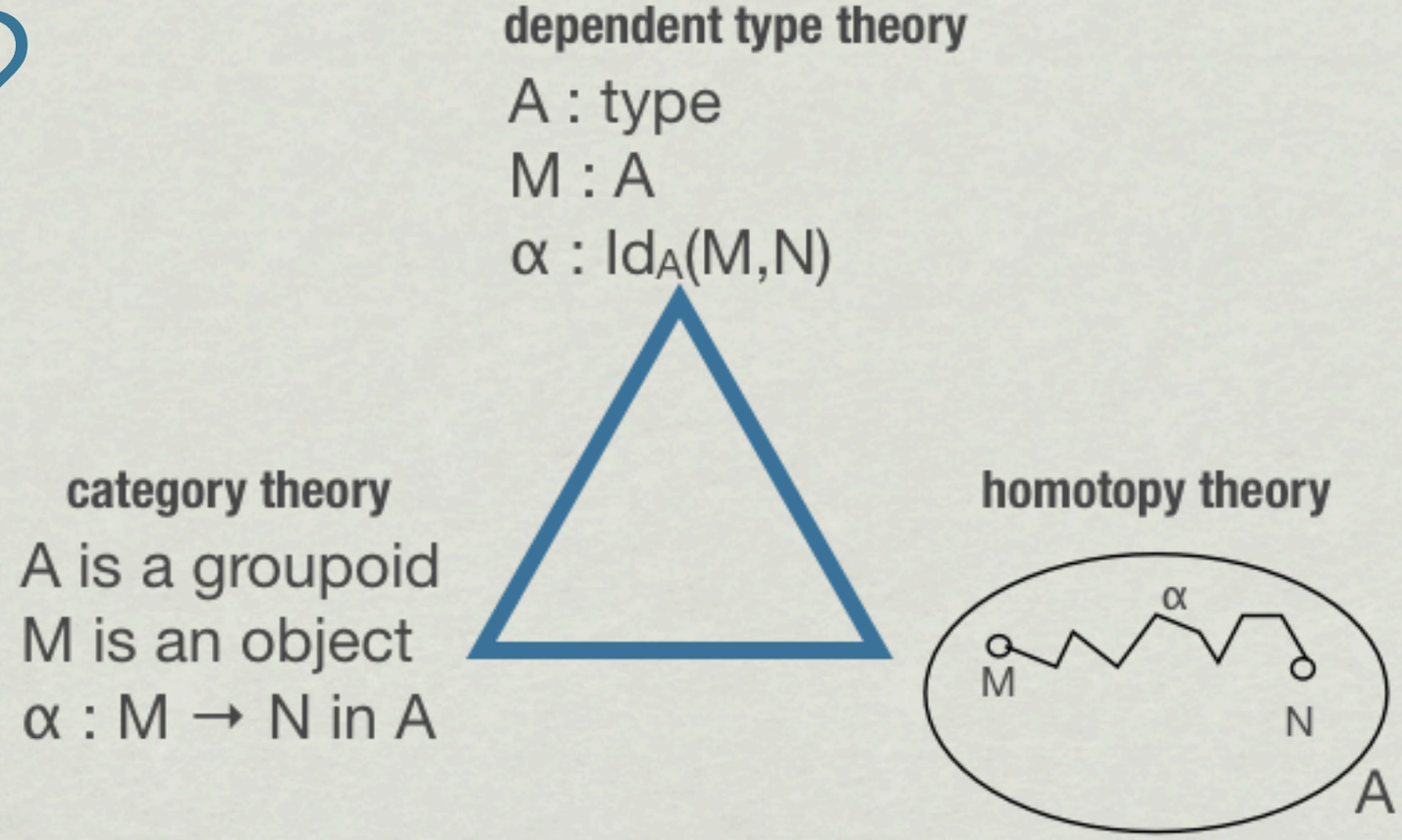
Types as Spaces



Types as Spaces



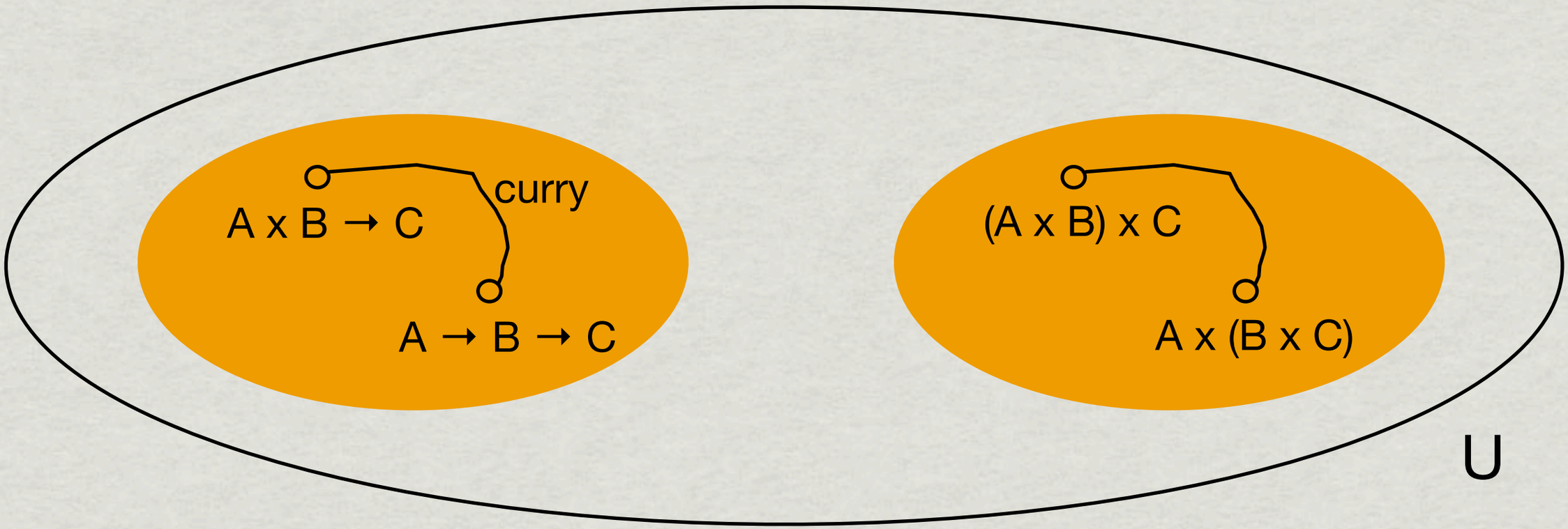
So what?



- Type theory can be used as a formal logical calculus for proving results in higher-category theory and homotopy theory
- Correspondence suggests new logical principles to add to type theory

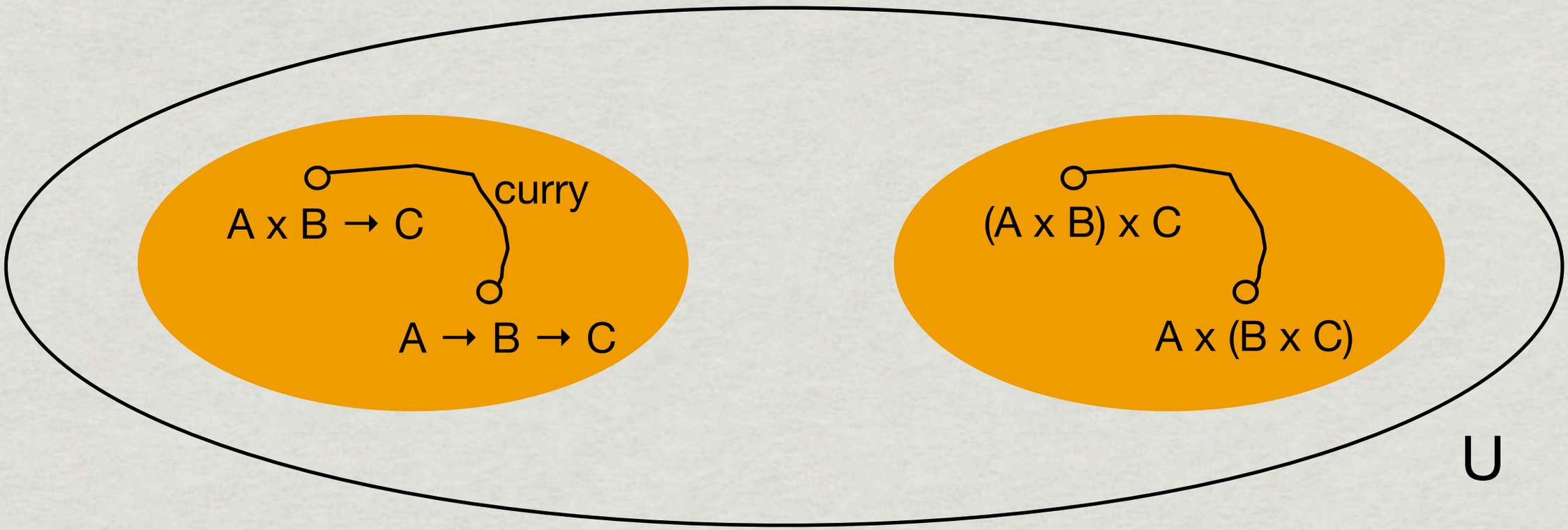
Univalence

[Hofmann&Streicher, Voevodsky]



Univalence

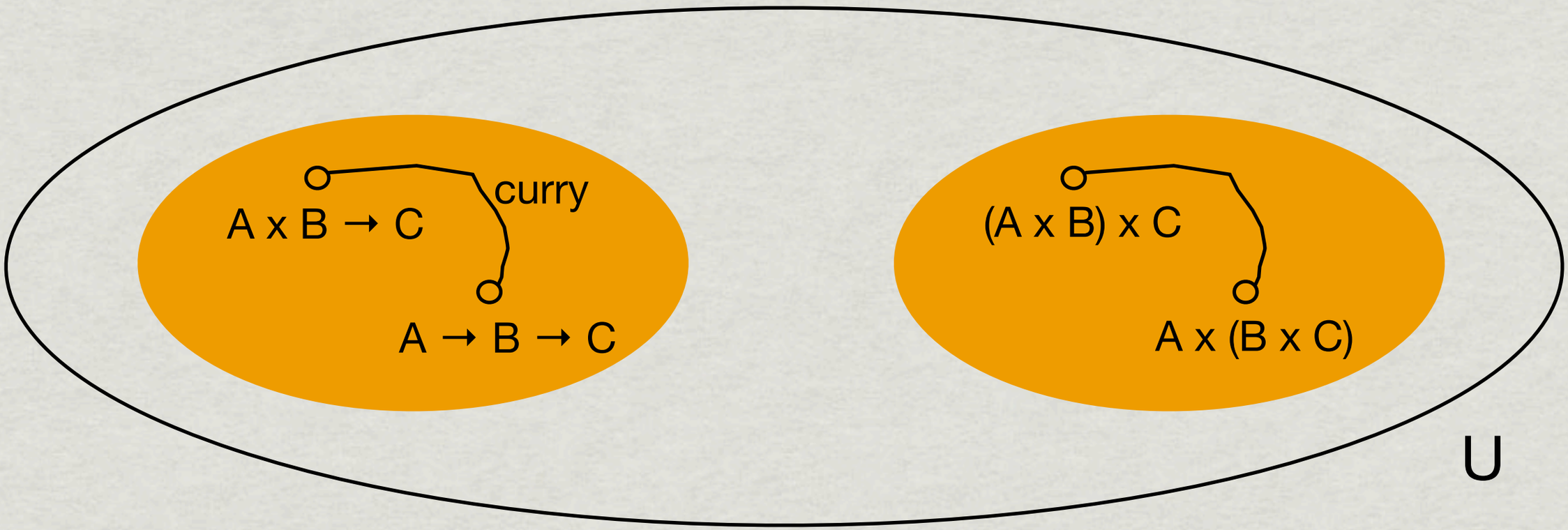
[Hofmann&Streicher, Voevodsky]



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

Univalence

[Hofmann&Streicher, Voevodsky]

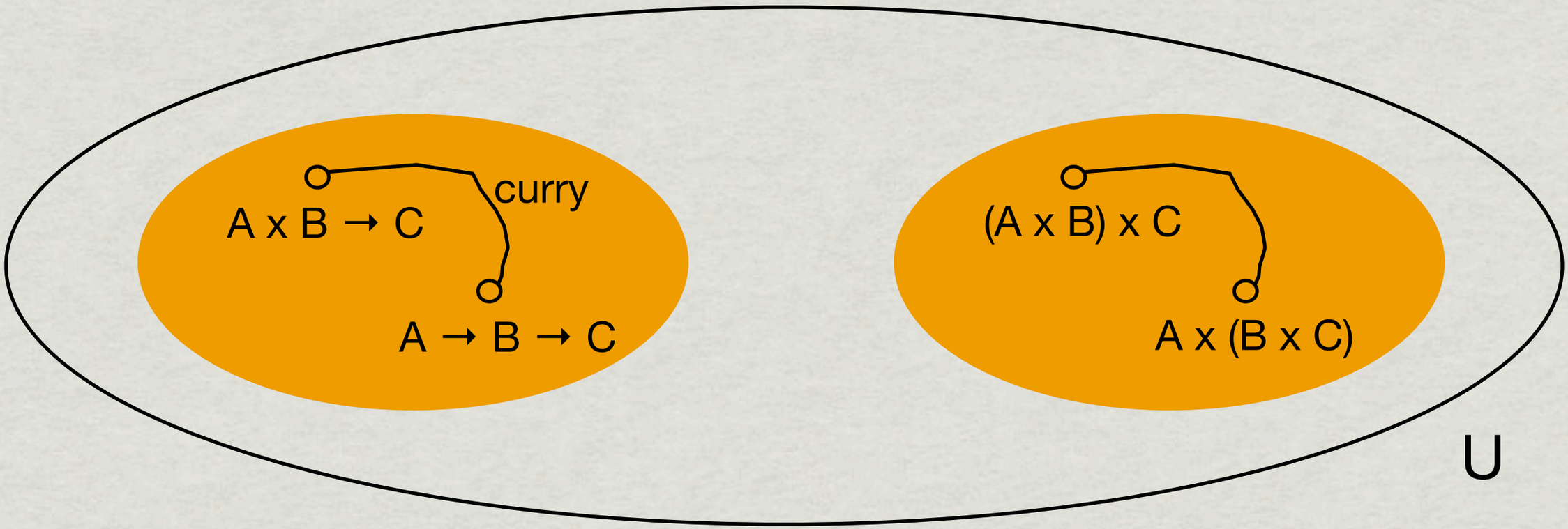


U = universe (type of types)
where *paths are isomorphisms*

Isomorphic types are interchangeable in all contexts; or, all constructions respect isomorphism

Univalence

[Hofmann&Streicher, Voevodsky]



U = universe (type of types)
where *paths are isomorphisms*
(really weak equivalences)

Isomorphic types are interchangeable in all contexts; or, all constructions respect isomorphism

Work “up-to-iso”

Monoid = Σ X:U. MonoidStr[X]
MonoidStr[X:U] : U =
Σ ⊙ : X → X → X.
Σ u : X.
(Πx,y,z. Id (x ⊙ (y ⊙ z)) ((x ⊙ y) ⊙ z))) ×
(Πx.Id (u ⊙ x) x) ×
(Πx.Id (x ⊙ u) x)

Work “up-to-iso”

Monoid = Σ X:U. MonoidStr[X]
MonoidStr[X:U] : U =
Σ ⊙ : X → X → X.
Σ u : X.
(Πx,y,z. Id (x ⊙ (y ⊙ z)) ((x ⊙ y) ⊙ z))) ×
(Πx.Id (u ⊙ x) x) ×
(Πx.Id (x ⊙ u) x)

If A ≃ B then MonoidStr[A] ≃ MonoidStr[B] ?

“the hard way”

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

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

f : A → B
f^{-1} : B → A
α : Id (f o f^{-1}) id
β : Id (f^{-1} o f) id

“the hard way”

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

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

f : A → B
f^{-1} : B → A
α : Id (f o f^{-1}) id
β : Id (f^{-1} o f) id

$y_1 \odot' y_2 = f ((f^{-1} y_1) \odot (f^{-1} y_2))$
$u' = f\ u$

"the hard way"

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

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

f : A → B
f^{-1} : B → A
α : Id (f o f^{-1}) id
β : Id (f^{-1} o f) id

$y_1 \odot' y_2 = f ((f^{-1} y_1) \odot (f^{-1} y_2))$
$u' = f u$

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

“The easy way”

1. Each individual construction in type theory without univalence already respects isomorphism

2. Univalence gives generic name to this principle, and ensures that hypotheses/extensions still do.
   To make MonoidStr[A] → MonoidStr[B] just write

$$\mathrm{transport}_{\mathrm{MonoidStr}}(\mathrm{univalence}\ (f, f^{-1}, \alpha, \beta))$$

“The easy way”

1. Each individual construction in type theory without univalence already respects isomorphism

2. Univalence gives generic name to this principle, and ensures that hypotheses/extensions still do.
   To make MonoidStr[A] → MonoidStr[B] just write

$$\mathrm{transport}_{\mathrm{MonoidStr}}(\mathrm{univalence}\ (f, f^{-1}, \alpha, \beta))$$

Identical terms are interchangeable in all contexts

“The easy way”

1. Each individual construction in type theory without univalence already respects isomorphism

2. Univalence gives generic name to this principle, and ensures that hypotheses/extensions still do.
   To make MonoidStr[A] → MonoidStr[B] just write

$$\mathrm{transport}_{\mathrm{MonoidStr}}(\mathrm{univalence}\ (f, f^{-1}, \alpha, \beta))$$

Identical terms are interchangeable in all contexts

MonoidStr is one such context

“The easy way”

1. Each individual construction in type theory without univalence already respects isomorphism

2. Univalence gives generic name to this principle, and ensures that hypotheses/extensions still do.

To make MonoidStr[A] → MonoidStr[B] just write

$$\mathrm{transport}_{\mathrm{MonoidStr}}(\mathrm{univalence}\ (f, f^{-1}, \alpha, \beta))$$

Identical terms are interchangeable in all contexts

MonoidStr is one such context

iso determines Id

“The easy way”

$$\text{transport}_{\text{MonoidStr}}\,(\text{univalence}\,(f, f^{-1}, \alpha, \beta))$$

Identical terms are interchangeable in all contexts

MonoidStr is one such context

iso determines Id

- Formalized math: don’t need to prove this by hand for each type
- Programming: automatically compute monoid structure on B from monoid structure on A using a *generic program*

Computational Interpretation

- We can **prove** that
  $\mathrm{transport}_{\mathrm{MonoidStr}}(\mathrm{univalence}(f, f^{-1}, \alpha, \beta))$
  is Id-entical to what we wrote out by hand

- But for type theory + univalence, as currently formulated, it doesn't **compute**: programs get stuck

- Formally: canonicity fails

Canonicity Property

“programs don’t get stuck”

algorithm for =

If M : nat

then

∃ a numeral k s.t. M $\longmapsto^{*}$ ***k***

guides the design of a type theory

Status

- **Theorem (Licata&Harper,2012):**
  Canonicity for special case of a 2-dimensional type theory with univalent universe of h-sets
  … up to equational deduction
- In progress: extends to an algorithm for 2TT
- One the goals for this special year:
  canonicity for full higher-dimensional univalence

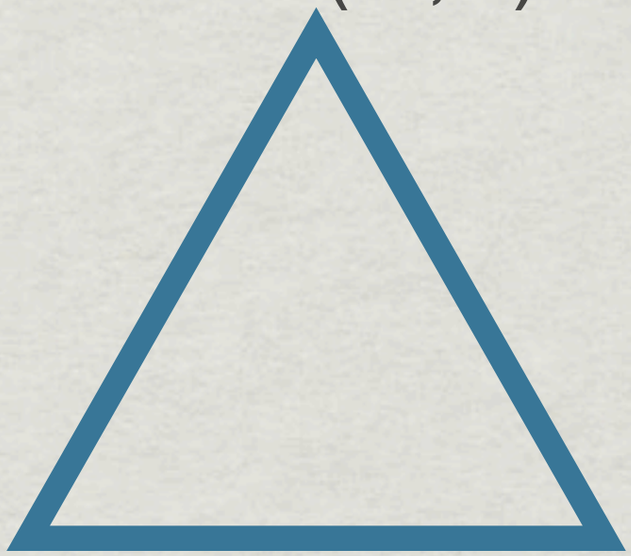
Homotopy Type Theory

dependent type theory

A : type

M : A

$\alpha : Id_A(M,N)$



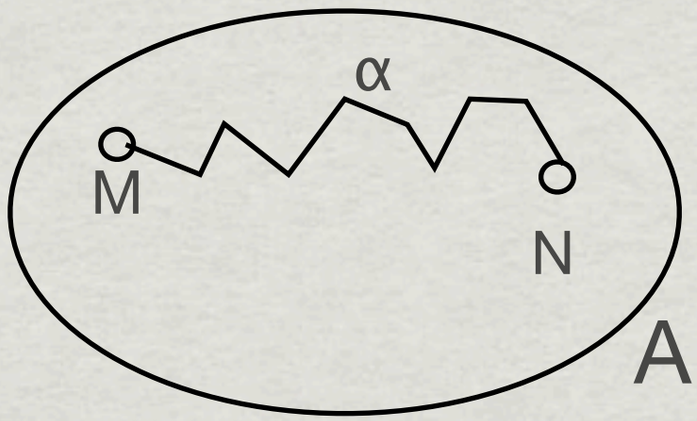
category theory

A is a groupoid

M is an object

$\alpha : M \to N$ in A

homotopy theory



Dimension Hierarchy

A type is an

- h-proposition: all terms are identical
  (in the sense of Id-type)
- h-set: unique identities
  e.g. nat
- h-groupoid: unique identities between identities
  e.g. universe of h-sets
- h-2-groupoid: and so on
  e.g. universe of h-groupoids

Status

types are at most h-groupoids

- **Theorem (Licata&Harper,2012):** Canonicity for a 2-dimensional type theory with univalent universe of h-sets … up to equational deduction

- In progress: extends to an algorithm for 2TT

- One the main goals for this special year: canonicity for full higher-dimensional univalence