

# **2-Dimensional Directed Type Theory**

**Dan Licata  
with Robert Harper**

**Carnegie Mellon University**



# Martin-Löf Type Theory

A type is specified by

Elements  $\Gamma \vdash M : A$

Equality  $\Gamma \vdash M \equiv N : A$

Families of types respect equality:

$x:A \vdash C : \text{type}$

$M \equiv N : A$

$P : C[M/x]$

$P : C[N/x]$



# Martin-Löf Type Theory

A type is specified by

Elements  $\Gamma \vdash M : A$

Equality  $\Gamma \vdash M \equiv N : A$

Families of types respect equality:

$$x:A \vdash C : \text{type}$$
$$M \equiv N : A$$
$$P : C[M/x]$$

---

$$P : C[N/x]$$



# Martin-Löf Type Theory

A type is specified by

Elements  $\Gamma \vdash M : A$

Equality  $\Gamma \vdash M \equiv N : A$

Families of types respect equality:

$x:A \vdash C : \text{type}$

$M \equiv N : A$

$P : C[M/x]$

---

$P : C[N/x]$

**universe :**  
**elements of type**  
**are classifiers**



# Type Isomorphisms

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



# Type Isomorphisms

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

Monoid : type  $\rightarrow$  type

Monoid  $X = \sum m : X \rightarrow X \rightarrow X.$

$\sum u : X.$

$(\prod x,y,z:X. \text{Id } (m \ x \ (m \ y \ z)) \ (m \ (m \ x \ y) \ z))$

\*  $(\prod x.\text{Id } (m \ x \ u) \ x)$

\*  $(\prod x.\text{Id } (m \ u \ x) \ x)$



# Type Isomorphisms

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

Monoid : type  $\rightarrow$  type

Monoid  $X = \sum m : X \rightarrow X \rightarrow X.$

$\sum u : X.$

$(\prod x,y,z:X. \text{Id } (m x (m y z)) (m (m x y) z))$

\*  $(\prod x.\text{Id } (m x u) x)$

\*  $(\prod x.\text{Id } (m u x) x)$

Want type families to respect iso, so

$$\text{Monoid}(\text{list } A) \cong \text{Monoid}(\sum n:\text{nat. } \text{vec } A n)$$



# Type Isomorphisms

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

Monoid : type → type

Monoid X =  $\sum m : X \rightarrow X \rightarrow X.$

$\sum u : X.$

$(\prod x,y,z:X. \text{Id } (m x (m y z)) (m (m x y) z))$

\*  $(\prod x.\text{Id } (m x u) x)$

\*  $(\prod x.\text{Id } (m u x) x)$

Want type families to respect iso, so

$$\text{Monoid}(\text{list } A) \cong \text{Monoid}(\sum n:\text{nat. } \text{vec } A n)$$

[cf. Voevodsky's univalence axiom]



# Type Isomorphisms

Cannot **equate** isomorphic types:

- \* Different representations!

- \* Types can be isomorphic in different ways:  
isomorphism is structure, not property

**Bool**  $\cong$  **Bool** by identity and by not



# 2-Dimensional Type Theory

Elements  $\Gamma \vdash M : A$

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

**+ equality of M's and  $\alpha$ 's**

All families of types respect equivalence,  
but this has computational content:

$x:A \vdash C : \text{type}$

$\alpha : M \approx_A N$

$P : C[M/x]$

---

$\text{map}_{x.C} \alpha P : C[N/x]$

**a generic  
proof/program  
defined for  
each family C  
(e.g. Monoid)**



# 2-Dimensional Type Theory

Elements  $\Gamma \vdash M : A$

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

+ equality of M's and  $\alpha$ 's

All families of types respect equivalence,  
but this has computational content:

$x:A \vdash C : \text{type}$

$\alpha : M \approx_A N$

$P : C[M/x]$

---

$\text{map}_{x.C} \alpha P : C[N/x]$

essentially  $C[\alpha/x]$

a generic  
proof/program  
defined for  
each family C  
(e.g. Monoid)



# Outline

1. 2-Dimensional Type Theory
2. 2-Dimensional Directed Type Theory
  - a. Application to functorial syntax
  - b. Semantics in Cat
3. Higher Dimensions



# Outline

- 1. 2-Dimensional Type Theory**
2. 2-Dimensional Directed Type Theory
  - a. Application to functorial syntax
  - b. Semantics in Cat
3. Higher Dimensions



# 2-Dimensional Type Theory

Define equivalence for each  $A$

$$\Gamma \vdash \alpha : M \approx_A N$$

Define map for each  $C$

$$x:A \vdash C : \text{type}$$

$$\alpha : M \approx_A N$$

$$P : C[M/x]$$

---

$$\text{map}_{x.c} \alpha P : C[N/x]$$



# Equivalence for types

$f : A \rightarrow B$

$g : B \rightarrow A$

$\alpha : g \circ f \simeq \text{id}$

$\beta : f \circ g \simeq \text{id}$

---

$\text{iso}(f, g, \alpha, \beta) : A \simeq_{\text{type}} B$



# map for type

$$\text{iso}(f, g, \alpha, \beta) : A \simeq_{\text{type}} B$$

---

$$\text{map}_{(\mathbf{a}:\text{type.a})} (\text{iso}(f, g, \alpha, \beta)) : A \rightarrow B$$

Computation rule: deploy the isomorphism!

$$\text{map}_{(\mathbf{a.a})} (\text{iso}(f, g, \alpha, \beta)) \equiv f$$



# map for Functions

$$\text{map}_{x:A.B \rightarrow C} (\alpha : M_1 \simeq_A M_2) f \equiv$$
$$\lambda x:B[M_2]. \text{map}_{x:A.C} \alpha (f (\text{map}_{x:A.B} (\text{sym } \alpha) x))$$



# map for Functions

$B[M_1] \rightarrow C[M_1]$



$\text{map}_{x:A.B \rightarrow C} (\alpha : M_1 \simeq_A M_2) f \equiv$

$\lambda x:B[M_2]. \text{map}_{x:A.C} \alpha (f (\text{map}_{x:A.B} (\text{sym } \alpha) x))$



# map for Functions

$B[M_1] \rightarrow C[M_1]$

$\text{map}_{x:A.B \rightarrow C} (\alpha : M_1 \simeq_A M_2) f \equiv$

**Goal:**  $B[M_2] \rightarrow C[M_2]$

$\lambda x:B[M_2]. \text{map}_{x:A.C} \alpha (f (\text{map}_{x:A.B} (\text{sym } \alpha) x))$



# map for Functions

$B[M_1] \rightarrow C[M_1]$

$\text{map}_{x:A.B \rightarrow C} (\alpha : M_1 \approx_A M_2) f \equiv$

Goal:  $B[M_2] \rightarrow C[M_2]$

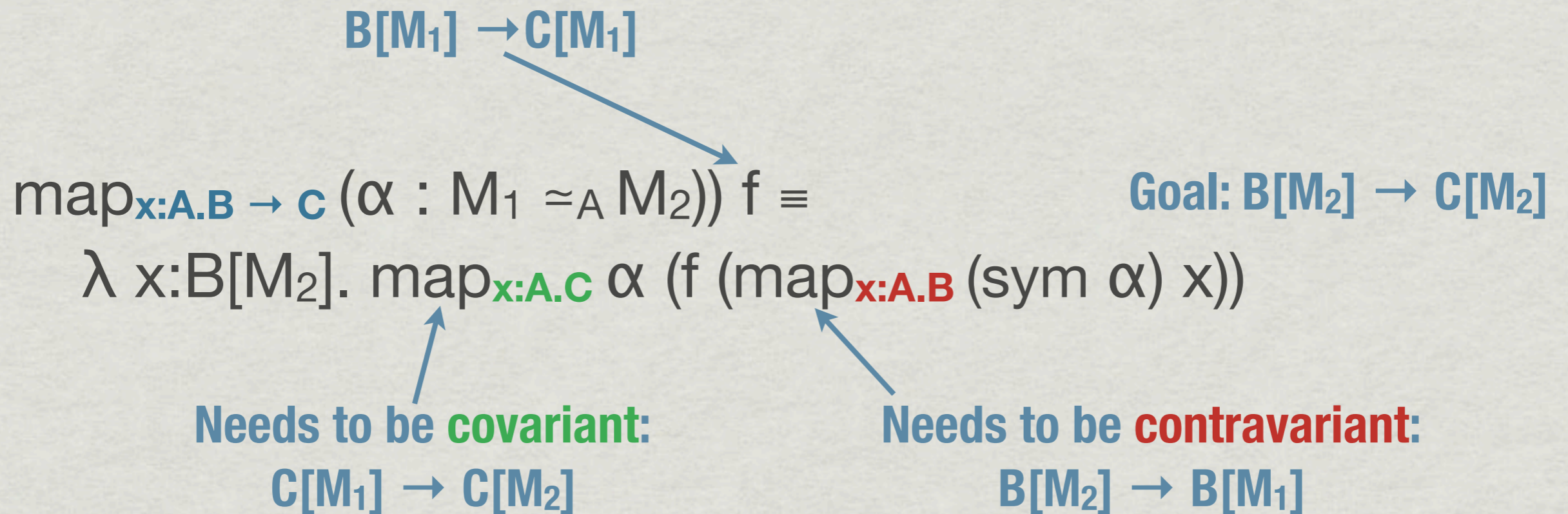
$\lambda x:B[M_2]. \text{map}_{x:A.C} \alpha (f (\text{map}_{x:A.B} (\text{sym } \alpha) x))$

Needs to be **contravariant**:

$B[M_2] \rightarrow B[M_1]$

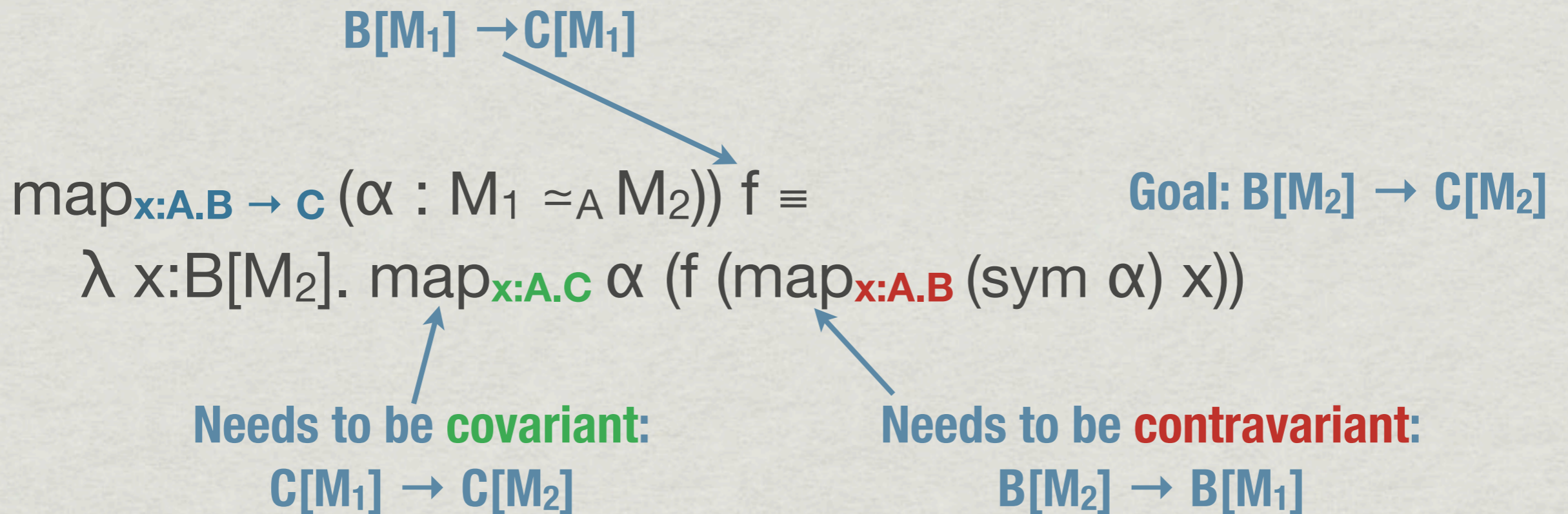


# map for Functions





# map for Functions



**Requires** 
$$\frac{\alpha : M_1 \simeq M_2}{\text{sym } \alpha : M_2 \simeq M_1}$$



# Symmetry for $\simeq_{\text{type}}$

$f : A \rightarrow B$

$g : B \rightarrow A$

$\alpha : g \circ f \simeq \text{id}$

$\beta : f \circ g \simeq \text{id}$

**symmetry requires  
backwards map!**

---

$\text{iso}(f, g, \alpha, \beta) : A \simeq_{\text{type}} B$

$\text{sym} (\text{iso}(f, g, \alpha, \beta)) \equiv \text{iso}(g, f, \beta, \alpha)$



# Semantics

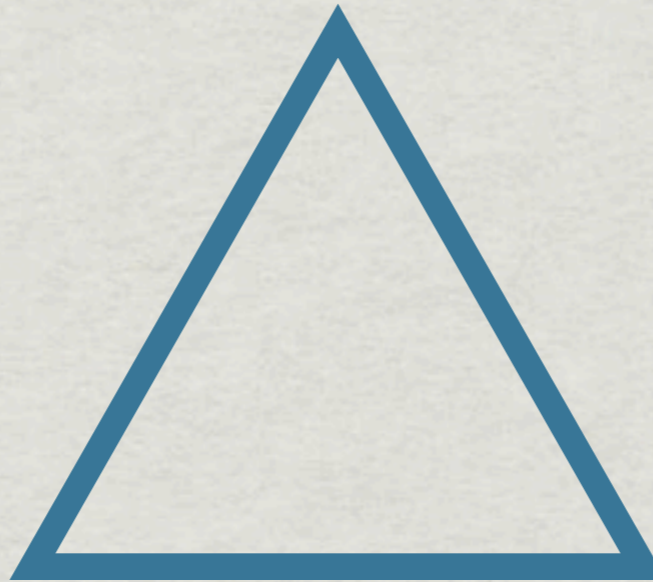
[Hofmann&Streicher, Awodey, Warren, Lumsdaine, Garner, Voevodsky]

## type theory

$A : \text{type}$

$M, N : A$

$\alpha : M \simeq_A N$



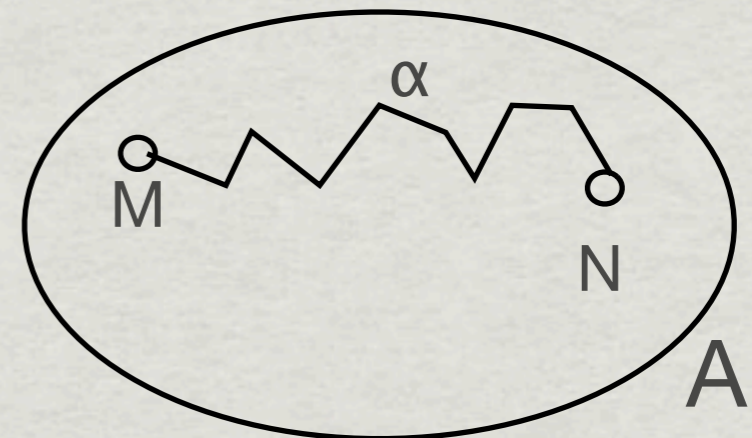
## category theory

$A$  is a groupoid

$M, N$  objects

$\alpha : M \rightarrow N$  in  $A$

## homotopy theory





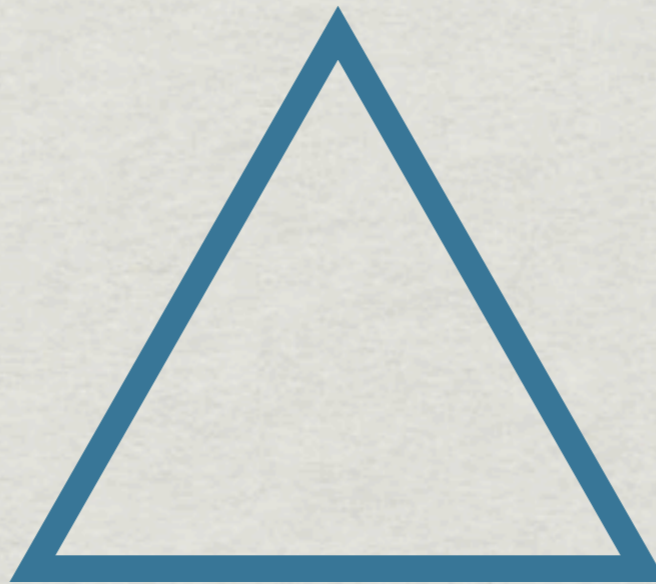
# Semantics

[Hofmann&Streicher,Awodey,  
Warren,Lumsdaine,Garner,  
Voevodsky]

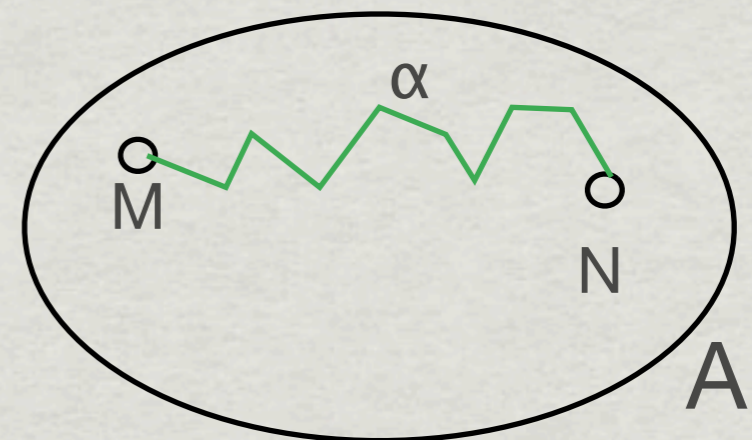
**type theory**  $\frac{\alpha : M \approx_A N}{\text{sym}(\alpha) : N \approx_A M}$   
 $A : \text{type}$   
 $M, N : A$   
 $\alpha : M \approx_A N$

**category theory**

$A$  is a groupoid  
 $M, N$  objects  
 $\alpha : M \rightarrow N$  in  $A$   
 $\alpha^{-1} : N \rightarrow M$  in  $A$



**homotopy theory**





# Summary

map is a generic program definable for all types:

- \* above implementation for  $\rightarrow$  extends to  $\Pi$
- \* can also be defined for  $\Sigma$ , (co)inductives, ...

*Let's put it to use!*



# Functorial Abstract Syntax

[FPT'99,AR'99,H'99]

A family of types  $\text{Form}[\Psi:\text{Ctx}]$  for formulas in  $\Psi$

Extends to a functor: 
$$\frac{\Psi' \vdash \sigma : \Psi}{\text{Form}[\Psi] \rightarrow \text{Form}[\Psi']}$$

that implements the structural properties:



# Functorial Abstract Syntax

[FPT'99,AR'99,H'99]

A family of types  $\text{Form}[\Psi:\text{Ctx}]$  for formulas in  $\Psi$

Extends to a functor: 
$$\frac{\Psi' \vdash \sigma : \Psi}{\text{Form}[\Psi] \rightarrow \text{Form}[\Psi']}$$

that implements the structural properties:

- \*  $\sigma$  is bijections: exchange



# Functorial Abstract Syntax

[FPT'99,AR'99,H'99]

A family of types  $\text{Form}[\Psi:\text{Ctx}]$  for formulas in  $\Psi$

Extends to a functor: 
$$\frac{\Psi' \vdash \sigma : \Psi}{\text{Form}[\Psi] \rightarrow \text{Form}[\Psi']}$$

that implements the structural properties:

- \*  $\sigma$  is bijections: exchange
- \*  $\sigma$  is var-for-var: weakening, exchange, contraction



# Functorial Abstract Syntax

[FPT'99,AR'99,H'99]

A family of types  $\text{Form}[\Psi:\text{Ctx}]$  for formulas in  $\Psi$

Extends to a functor: 
$$\frac{\Psi' \vdash \sigma : \Psi}{\text{Form}[\Psi] \rightarrow \text{Form}[\Psi']}$$

that implements the structural properties:

- \*  $\sigma$  is bijections: exchange
- \*  $\sigma$  is var-for-var: weakening, exchange, contraction
- \*  $\sigma$  is term-for-var: substitution, too



# Idea

Given a datatype for syntax/judgement:

data Form : Ctx → type where

all : Form[Ψ,i] → Form[Ψ]

...

data nd : (Ψ : Ctx) → Form[Ψ] → type

allR : ∀ {Ψ A} → nd (Ψ,i) A → nd Ψ (all A)

*automatically implement structural properties using map*



# Exchange

$x,y,z$  vs.  $y,z,x$

Take Ctx to be a 2D type:

Terms  $\Psi : \text{Ctx}$

Equivalences  $\Psi \approx_{\text{Ctx}} \Psi'$

bijection between vars

$$\alpha : \Psi \approx_{\text{Ctx}} \Psi'$$

---

$$\text{map}_{\Psi.\text{Form}[\Psi]} \alpha : \text{Form}[\Psi] \rightarrow \text{Form}[\Psi']$$



# Exchange

$x,y,z$  vs.  $y,z,x$

Take Ctx to be a 2D type:

Terms  $\Psi : \text{Ctx}$

Equivalences  $\Psi \approx_{\text{Ctx}} \Psi'$

bijection between vars

for de Bruijn form,  
not merely a relation


$$\alpha : \Psi \approx_{\text{Ctx}} \Psi'$$

---

$$\text{map}_{\Psi.\text{Form}[\Psi]} \alpha : \text{Form}[\Psi] \rightarrow \text{Form}[\Psi']$$



# Exchange

$x,y,z$  vs.  $y,z,x$

Take Ctx to be a 2D type:

Terms

$\Psi : \text{Ctx}$

Equivalences

$\Psi \approx_{\text{Ctx}} \Psi'$

bijection between vars

for de Bruijn form,  
not merely a relation



$\alpha : \Psi \approx_{\text{Ctx}} \Psi'$

---

$\text{map}_{\Psi.\text{Form}[\Psi]} \alpha : \text{Form}[\Psi] \rightarrow \text{Form}[\Psi']$

derived from map for  
definition of Form





# Weakening

$x,y$  vs.  $x,z,y$

Take Ctx to be a 2D type:

Terms  $\Psi : \text{Ctx}$

Equivalences  $\Psi \approx_{\text{Ctx}} \Psi'$

variable-for-variable substitutions

$x,z,y \vdash x/x, y/y : x,y$



# Weakening

$x,y$  vs.  $x,z,y$

Take Ctx to be a 2D type:

Terms  $\Psi : \text{Ctx}$

Equivalences  $\Psi \approx_{\text{Ctx}} \Psi'$

variable-for-variable substitutions

$x,z,y \vdash x/x, y/y : x,y$

***Problem: not symmetric!***



# Outline

1. 2-Dimensional Type Theory

**2. 2-Dimensional Directed Type Theory**

a. Application to functorial syntax

b. Semantics in Cat

3. Higher Dimensions

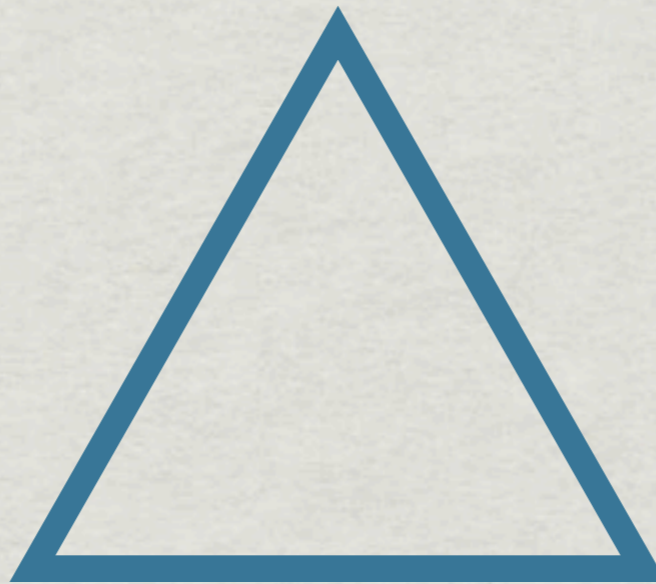


# 2-Dimensional Type Theory

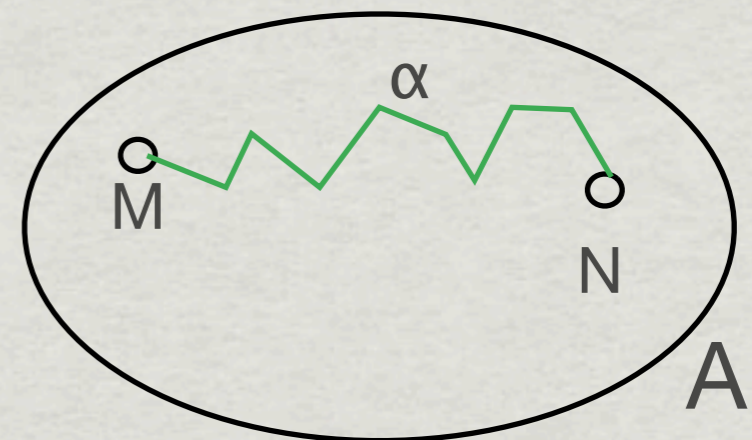
**type theory**  $\frac{\alpha : M \approx_A N}{\text{sym}(\alpha) : N \approx_A M}$   
 $A : \text{type}$   
 $M, N : A$   
 $\alpha : M \approx_A N$

**category theory**

$A$  is a groupoid  
 $M, N$  objects  
 $\alpha : M \rightarrow N$  in  $A$   
 $\alpha^{-1} : N \rightarrow M$  in  $A$

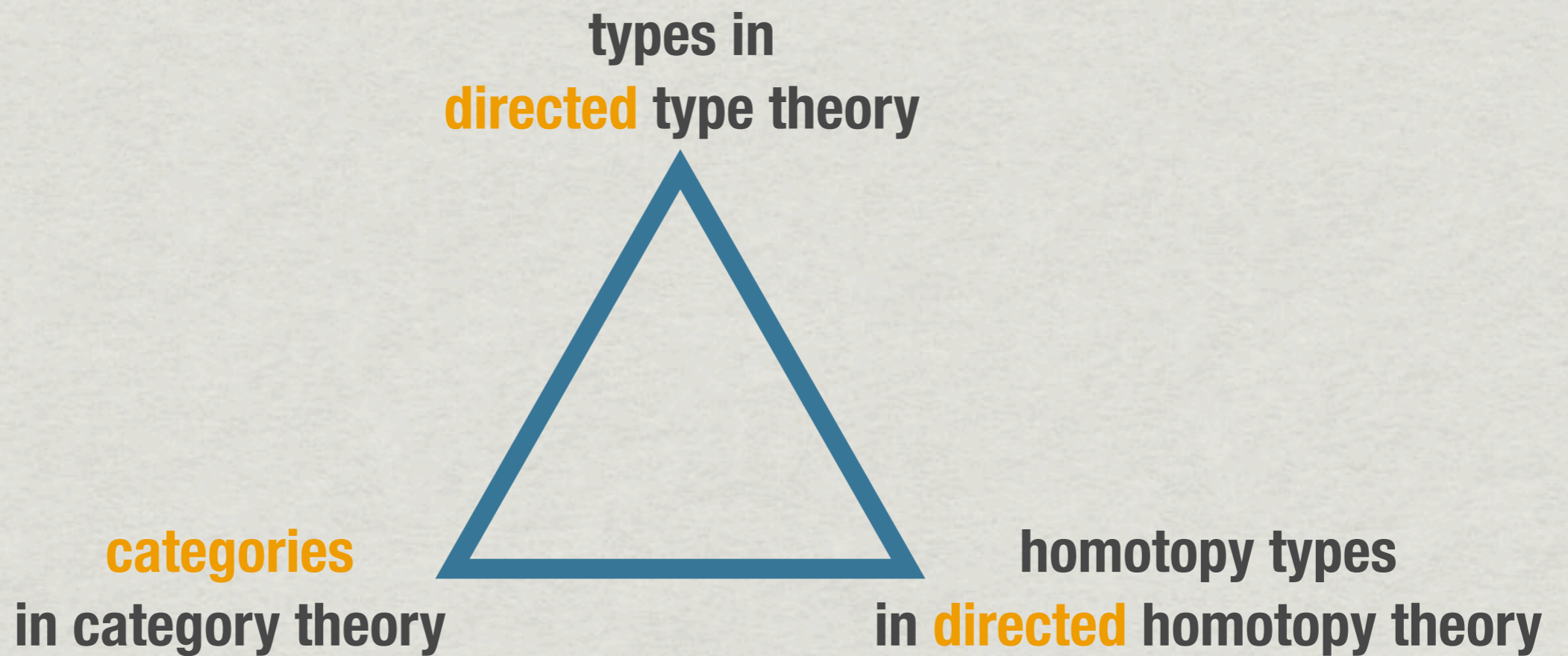


**homotopy theory**





# 2-Dimensional Directed Type Theory





# Directed Type Theory

What is a type?

Elements  $\Gamma \vdash M : A$

Transformation  $\Gamma \vdash \alpha : M \approx_A N$

**not necessarily symmetric!**





# Weakening

$x,y$  vs.  $x,z,y$

Take Ctx to be a **directed** type:

Terms  $\Psi : \text{Ctx}$

Transformations  $\Psi \approx_{\text{Ctx}} \Psi'$

variable for variable substitutions

$\Psi' \vdash \sigma : \Psi$

$$\frac{\alpha : \Psi \approx_{\text{Ctx}} \Psi'}{\text{map}_{\Psi.\text{Form}[\Psi]} \alpha : \text{Form}[\Psi] \rightarrow \text{Form}[\Psi']}$$



# What about map for $\rightarrow$ ?

$$\text{map}_{x:A.B \rightarrow C} (\alpha : M_1 \simeq_A M_2) f \equiv$$
$$\lambda x:B[M_2]. \text{map}_{x:A.C} \alpha (f (\text{map}_{x:A.B} (\text{sym } \alpha) x))$$



# Variations

$$\frac{\Gamma \text{ ctx}}{\Gamma^{\text{op}} \text{ ctx}}$$

**Contravariant**

**Covariant**

$$\frac{\Gamma^{\text{op}} \vdash A : \text{type} \quad \Gamma \vdash B : \text{type}}{\Gamma \vdash A \rightarrow B : \text{type}}$$



# Variations

$\Gamma ::= \cdot \quad | \quad \text{Covariant } \Gamma, x:A^+ \quad | \quad \text{Contravariant } \Gamma, x:A^-$

$\overline{\Gamma, x:A^+ \vdash x : A}$

**only + assumptions  
can be used in a term**

$\frac{\Gamma \text{ ctx}}{\Gamma^{\text{op}} \text{ ctx}}$

**interchanges + and -**



# Map

Covariant:

$$\frac{\Gamma, x:A^+ \vdash B \text{ type} \quad \Gamma \vdash \alpha : M_1 \lesssim_A M_2 \quad \Gamma \vdash M : B[M_1/x]}{\Gamma \vdash \text{map}_{x:A^+.B} \alpha M : B[M_2/x]}$$

Contravariant:

$$\frac{\Gamma, x:A^- \vdash B \text{ type} \quad \Gamma^{\text{op}} \vdash \alpha : M_2 \lesssim_A M_1 \quad \Gamma \vdash M : B[M_1/x]}{\Gamma \vdash \text{map}_{x:A^-.B} \alpha M : B[M_2/x]}$$



# Functions

$$\frac{\Gamma^{\text{op}} \vdash A : \text{type} \quad \Gamma \vdash B : \text{type}}{\Gamma \vdash A \rightarrow B : \text{type}}$$

$$\frac{\Gamma, x:A \vdash M : B}{\Gamma \vdash \lambda x.M : A \rightarrow B}$$

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma^{\text{op}} \vdash N : A}{\Gamma \vdash M N : B}$$



map ( $\rightarrow$ )

$$\frac{x:A^- \vdash B : \text{type} \quad x:A^+ \vdash C : \text{type}}{x:A^+ \vdash B \rightarrow C : \text{type}}$$

$B[M_1] \rightarrow C[M_1]$

$\text{map}_{x:A^+.B \rightarrow C} (\alpha : M_1 \approx_A M_2) f \equiv$

$\lambda x:B[M_2]^-. \text{map}_{x:A^+.C} \alpha (f (\text{map}_{x:A^-.B} \alpha x))$

Goal:  $B[M_2] \rightarrow C[M_2]$

Covariant:  
 $C[M_1] \rightarrow C[M_2]$

Contravariant:  
 $B[M_2] \rightarrow B[M_1]$



map ( $\rightarrow$ )

$$\frac{x:A^- \vdash B : \text{type} \quad x:A^+ \vdash C : \text{type}}{x:A^+ \vdash B \rightarrow C : \text{type}}$$

$B[M_1] \rightarrow C[M_1]$

$\text{map}_{x:A^+.B \rightarrow C} (\alpha : M_1 \approx_A M_2) f \equiv$

$\lambda x:B[M_2]^-. \text{map}_{x:A^+.C} \alpha (f (\text{map}_{x:A^-.B} \alpha x))$

Goal:  $B[M_2] \rightarrow C[M_2]$

Covariant:

$C[M_1] \rightarrow C[M_2]$

Contravariant:

$B[M_2] \rightarrow B[M_1]$

$\Gamma, x:B[M_2]^- \vdash f (\text{map}_{x:A^-.B} \alpha x) : C[M_1]$

if  $\Gamma^{\text{op}}, x:B[M_2]^+ \vdash (\text{map}_{x:A^-.B} \alpha x) : B[M_1]$

if  $(\Gamma^{\text{op}})^{\text{op}} \vdash \alpha : M_1 \approx_A M_2$



map ( $\rightarrow$ )

$$\frac{x:A^- \vdash B : \text{type} \quad x:A^+ \vdash C : \text{type}}{x:A^+ \vdash B \rightarrow C : \text{type}}$$

$B[M_1] \rightarrow C[M_1]$

$\text{map}_{x:A+.B \rightarrow C} (\alpha : M_1 \approx_A M_2) f \equiv$

$\lambda x:B[M_2]^-. \text{map}_{x:A+.C} \alpha (f (\text{map}_{x:A-.B} \alpha x))$

Goal:  $B[M_2] \rightarrow C[M_2]$

Covariant:

$C[M_1] \rightarrow C[M_2]$

Contravariant:

$B[M_2] \rightarrow B[M_1]$

involution

$\Gamma, x:B[M_2]^- \vdash f (\text{map}_{x:A-.B} \alpha x) : C[M_1]$

if  $\Gamma^{\text{op}}, x:B[M_2]^+ \vdash (\text{map}_{x:A-.B} \alpha x) : B[M_1]$

if  $(\Gamma^{\text{op}})^{\text{op}} \vdash \alpha : M_1 \approx_A M_2$



# Semantics in Cat

Context	$\Gamma \text{ ctx}$	category
Substitution	$\Gamma \vdash \theta : \Delta$	functor
Transformation	$\Gamma \vdash \delta : \theta \approx_{\Delta} \theta'$	natural transform.
Type	$\Gamma \vdash A \text{ type}$	functor $\Gamma \rightarrow \text{Cat}$
Term	$\Gamma \vdash M : A$	“dependent functor”
Term Trans.	$\Gamma \vdash \alpha : M \approx_A N$	“dependent n.t.”



# Identity and Composition

Transformation:  $\Gamma \vdash \delta : \theta \approx_{\Delta} \theta'$

$\Gamma \vdash \delta_1 : \theta_1 \approx_{\Delta} \theta_2$

$\Gamma \vdash \delta_2 : \theta_2 \approx_{\Delta} \theta_3$

---

$\Gamma \vdash \text{refl} : \theta \approx_{\Delta} \theta$

---

$\Gamma \vdash \delta_2 \circ \delta_1 : \theta_1 \approx_{\Delta} \theta_3$

**also**  
**horizontal composition:**  
**transformation**  
**respects**  
**transformation**



# Identity and Composition

Transformation:  $\Gamma \vdash \delta : \theta \approx_{\Delta} \theta'$

$$\frac{}{\Gamma \vdash \text{refl} : \theta \approx_{\Delta} \theta} \quad \frac{\Gamma \vdash \delta_1 : \theta_1 \approx_{\Delta} \theta_2 \quad \Gamma \vdash \delta_2 : \theta_2 \approx_{\Delta} \theta_3}{\Gamma \vdash \delta_2 \circ \delta_1 : \theta_1 \approx_{\Delta} \theta_3}$$

**also**  
**horizontal composition:**  
**transformation**  
**respects**  
**transformation**

Equations hold strictly:

$$\delta_3 \circ (\delta_2 \circ \delta_1) \equiv (\delta_3 \circ \delta_2) \circ \delta_1$$

$$\text{map}_C (\delta_2 \circ \delta_1) M \equiv \text{map}_C \delta_3 (\text{map}_C \delta_2 M)$$



# Application

Generalizes functorial syntax to

- \* judgements/dependent types, via **map** for  $\Sigma$  and  $\Pi$
- \* admissibility premises

(see paper/thesis for details)



# Outline

1. 2-Dimensional Type Theory
2. 2-Dimensional Directed Type Theory
  - a. Application to functorial syntax
  - b. Semantics in Cat
- 3. Higher Dimensions**



# Higher Dimensions

More generally, equations may hold weakly:

$$\delta_3 \circ (\delta_2 \circ \delta_1) \approx (\delta_3 \circ \delta_2) \circ \delta_1 : \theta_1 \approx \theta_4 : \Gamma \vdash \Delta$$

Example: **type**<sub>1</sub> where **type** : **type**<sub>1</sub>  
equivalence should be “iso up to iso”



# Higher Dimensions

More generally, equations may hold weakly:

$$\delta_3 \circ (\delta_2 \circ \delta_1) \simeq (\delta_3 \circ \delta_2) \circ \delta_1 : \theta_1 \simeq \theta_4 : \Gamma \vdash \Delta$$

Example: **type**<sub>1</sub> where **type** : **type**<sub>1</sub>  
equivalence should be “iso up to iso”

Elements  $\Gamma \vdash M : A$

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

Equivalence<sup>2</sup>  $\Gamma \vdash \varphi : \alpha \simeq \alpha'$

...



# Higher Dimensions

In symmetric type theory, can exploit the identity type

Elements  $M : A$

Equivalence  $\alpha : M \simeq N$        $\alpha : \text{Id}_A(M, N)$

Equivalence<sup>2</sup>  $\varphi : \alpha \simeq \alpha'$        $\varphi : \text{Id}_{\text{Id}(M, N)}(\alpha, \alpha')$

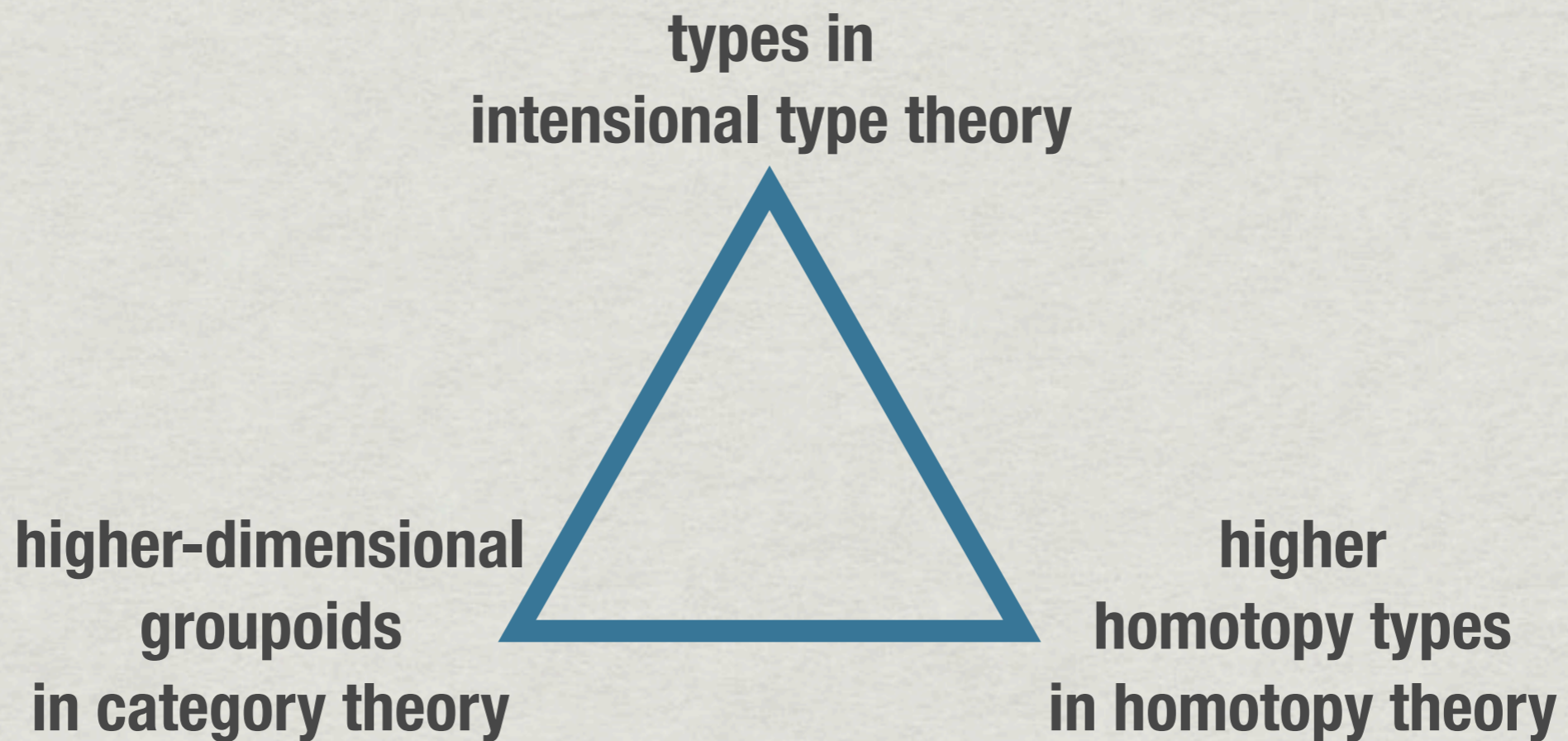
Interpret  $\text{Id}_A$  as hom-functor  $A \times A \rightarrow [\text{type}]$

Standard rules for  $\text{Id}$  are sound for higher-dimensions,  
but miss computation rules for **map**



# Homotopy Type Theory

[[homotopytypetheory.org](http://homotopytypetheory.org)]



[Hofmann&Streicher, Awodey,  
Warren, Lumsdaine, Garner, Voevodsky]



# Higher Dimensions

In symmetric type theory, can exploit the identity type

Elements  $M : A$

Equivalence  $\alpha : M \simeq N$        $\alpha : \text{Id}_A(M, N)$

Equivalence<sup>2</sup>  $\varphi : \alpha \simeq \alpha'$        $\varphi : \text{Id}_{\text{Id}(M, N)}(\alpha, \alpha')$

But in directed type theory, a Hom type is harder:

Mixed variance of Hom :  $C^{\text{op}} \times C \rightarrow [\text{type}]$



# Outline

1. 2-Dimensional Type Theory
2. 2-Dimensional Directed Type Theory
  - a. Application to functorial syntax
  - b. Semantics in Cat
3. Higher Dimensions

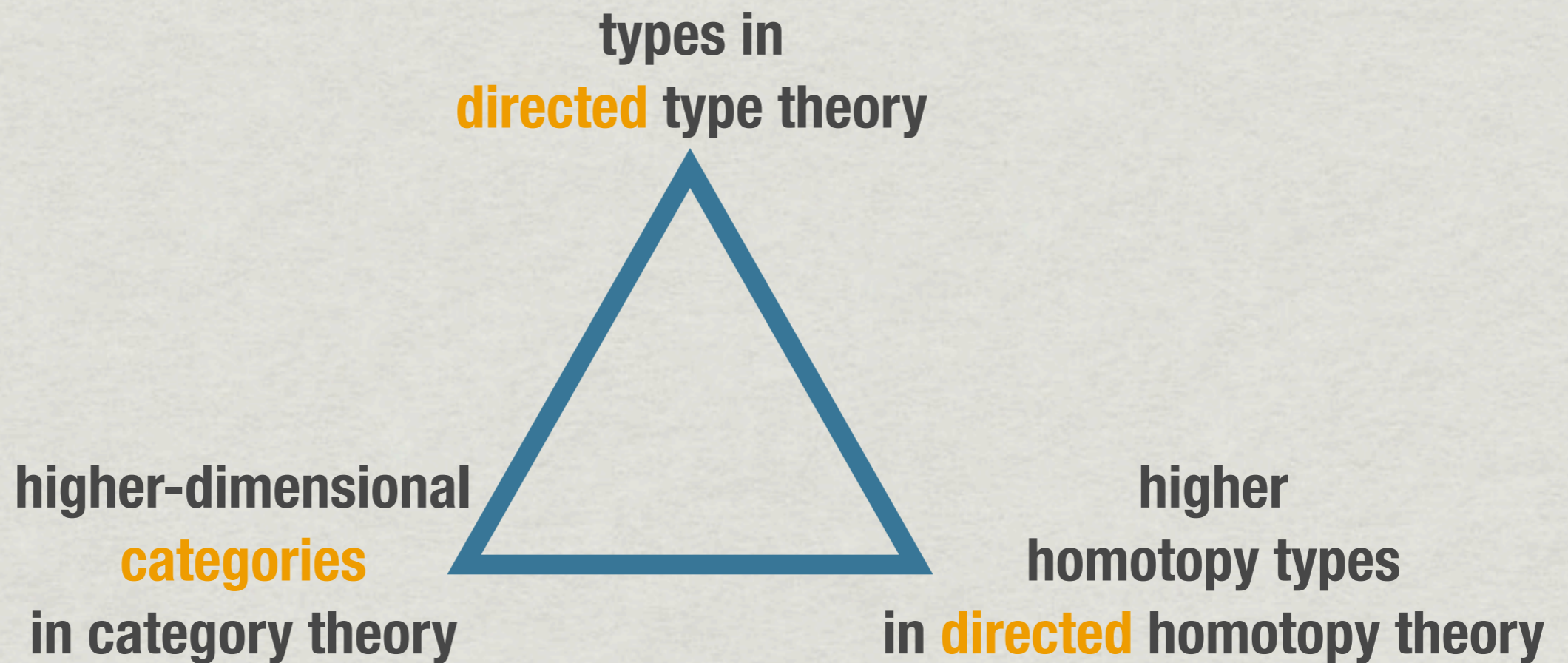


# Future Work

- \* Hom type, paralleling identity type  $\text{Id}$
- \* Inductive, co-inductive types
- \* Quotients by internal descriptions of categories
- \* Integration with symmetric type theory
- \* Semantics in more general 2-categories
- \* Higher dimensions and their semantics
- \* Computational interpretation:  
N-canonicity, decidable definitional equality



# Directed Type Theory



**Blog: [homotopytypetheory.org](http://homotopytypetheory.org)**