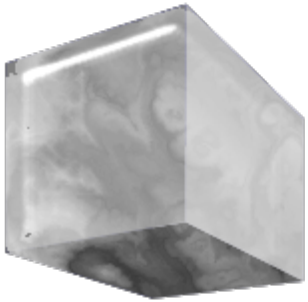
2018.07.07 LFMTP

Cubical Computational Type Theory & RedPRL

>> redprl.org >>



Carlo Angiuli
Evan Cavallo
(*) Favonia
Robert Harper
Jonathan Sterling
Todd Wilson

Cubical

features of homotopy type theory

univalence, higher inductive types

+

Computational

features of Nuprl and PVS

strict equality, strict quotients,
predicative subtypes...

Cartesian Cubical

features of homotopy type theory

univalence, higher inductive types

+

Computational

features of Nuprl and PVS

strict equality, strict quotients,
predicative subtypes...

Computational Types

programs/
realizers

computation

Computational Types

programs/ realizers <------ **computational type theory**

computation — theory of computation

Computational Types

programs/ realizers <----- **computational type theory**

computation — theory of computation

meaning explanation <----- Martin-Löf type theory

pre-mathematical in M-L's work

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)
```

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)

bool val       if(M,Mt,Mf) ↦ if(M',Mt,Mf)
true val       if(true,M,_) ↦ M
false val      if(false,_,M) ↦ M
```

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)

bool val        if(M,Mt,Mf) ↦ if(M',Mt,Mf)
true val        if(true,M,_) ↦ M
false val       if(false,_,M) ↦ M
```

The Language

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)

bool val        if(M,Mt,Mf) ↦ if(M',Mt,Mf)
true val        if(true,M,_) ↦ M
false val       if(false,_,M) ↦ M
```

The Language

What are the types in canonical forms? **{bool}**

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)

bool val        if(M,Mt,Mf) ↦ if(M',Mt,Mf)
true val        if(true,M,_) ↦ M
false val       if(false,_,M) ↦ M
```

The Language

What are the types in canonical forms? **{bool}**

What are the canonical forms of the types?

bool: {true, false}

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)

bool val        if(M,Mt,Mf) ↦ if(M',Mt,Mf)
true val        if(true,M,_) ↦ M
false val       if(false,_,M) ↦ M
```

The Language

What are the types in canonical forms? **{bool}**

What are the canonical forms of the types?

bool: {true, false}

How they are equal? **syntactic equality**

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)

bool val        if(M,Mt,Mf) ↦ if(M',Mt,Mf)
true val        if(true,M,_) ↦ M
false val       if(false,_,M) ↦ M
```

The Language

What are the types in canonical forms? **{bool}**

What are the canonical forms of the types?

bool: {true, false}

How they are equal? **syntactic equality**

One Theory

A Minimum Example

M := a | bool | true | false | if(M,M,M)

types: {bool} with syntactic equality $\approx$

bool: {true, false} with syntactic equality $\approx_{\text{bool}}$

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)
```

types: {bool} with syntactic equality $\approx$

bool: {true, false} with syntactic equality $\approx_{bool}$

$$A \doteq B \text{ type}$$

$$A \Downarrow A' \quad B \Downarrow B' \text{ and } A' \approx B'$$

A Minimum Example

M := a | bool | true | false | if(M,M,M)

types: {bool} with syntactic equality $\approx$

bool: {true, false} with syntactic equality $\approx_{bool}$

$$A \doteq B \text{ type}$$

$$A \Downarrow A' \quad B \Downarrow B' \quad \text{and} \quad A' \approx B'$$

$$\text{bool} \doteq \text{bool type}$$

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)
```

types: {bool} with syntactic equality ≈

bool: {true, false} with syntactic equality $\approx_{\text{bool}}$

A ≑ B type

A⇓A' B⇓B' and A'≈B'

---

bool ≑ bool type

if(true,bool,bool) ≑ bool type
⇓bool

A Minimum Example

M := a | bool | true | false | if(M,M,M)

types: {bool} with syntactic equality $\approx$

bool: {true, false} with syntactic equality $\approx_{\text{bool}}$

A ≐ B type

A⇓A' B⇓B' and A'≈B'

bool ≐ bool type

if(true,bool,bool) ≐ bool type
⇓bool

if(true,bool,*any closed term*) ≐ bool type

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)
```

types: {bool} with syntactic equality $\approx$

bool: {true, false} with syntactic equality $\approx_{bool}$

$$M \doteq N \in A$$

$A \doteq A$ type, $M \Downarrow M'$, $N \Downarrow N'$, $A \Downarrow A'$ and $M' \approx_{A'} N'$

A Minimum Example

M := a | bool | true | false | if(M,M,M)

types: {bool} with syntactic equality $\approx$

bool: {true, false} with syntactic equality $\approx_{bool}$

$$M \doteq N \in A$$

$A \doteq A$ type, $M \Downarrow M'$, $N \Downarrow N'$, $A \Downarrow A'$ and $M' \approx_{A'} N'$

$$\text{false} \doteq \text{false} \in \text{bool}$$

A Minimum Example

M := a | bool | true | false | if(M,M,M)

types: {bool} with syntactic equality ≈

bool: {true, false} with syntactic equality $\approx_{\mathrm{bool}}$

$$M \doteq N \in A$$

$A \doteq A$ type, $M \Downarrow M'$, $N \Downarrow N'$, $A \Downarrow A'$ and $M' \approx_{A'} N'$

$$\mathrm{false} \doteq \mathrm{false} \in \mathrm{bool}$$

$$\underset{\Downarrow \mathrm{true}}{\mathrm{if(true,true,bool)}} \doteq \mathrm{true} \in \underset{\Downarrow \mathrm{bool}}{\mathrm{if(true,bool,bool)}}$$

A Minimum Example

`M := a | bool | true | false | if(M,M,M)`

types: {bool} with syntactic equality $\approx$

bool: {true, false} with syntactic equality $\approx_{\text{bool}}$

$$a{:}A \gg M \doteq N \in B$$

$P \doteq Q \in A$ implies $M[P/a] \doteq N[Q/a] \in B[P/a]$

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)
```

types: {bool} with syntactic equality $\approx$

bool: {true, false} with syntactic equality $\approx_{\mathrm{bool}}$

a:A >> M ≐ N ∈ B

P ≐ Q ∈ A implies M[P/a] ≐ N[Q/a] ∈ B[P/a]

---

b:bool >> b ≐ if(b,true,false) ∈ bool?

A Functional Example

```
M := a | M1⟶M2 | \a.M | M1 M2 | ...

(M1⟶M2) val  \a.M val  (\a.M1)M2 ↦ M1[M2/a]
```

Another Language

A Functional Example

```
M := a | M1⟶M2 | \a.M | M1 M2 | ...

(M1⟶M2) val   \a.M val   (\a.M1)M2 ↦ M1[M2/a]
```

Another Language

What are the types in canonical forms?

```
  the least fixed point of
  S ↦ {M⟶N | M⇓, N⇓ in S} union ...
```

What are the canonical forms of the types?

```
  A⟶B: {\a.M}
```

How they are equal?

```
  A1⟶B1 ≈ A2⟶B2 if A1 ≐ A2 and B1 ≐ B2
```

$\backslash a.M1 \approx_{A\longrightarrow B} \backslash a.M2$ if $a{:}A \gg M1 \doteq M2 \in B$

Variables

Nuprl/...	Coq/Agda/...
Vars range over closed terms	Vars are indet.
Defined by transition b/w closed terms	Defined by conversion b/w open terms

Open-endedness

Proof theory/tactics/editors

↓

Computational type theory

↓

Programming language

Open-endedness

Proof theory/tactics/editors

↓

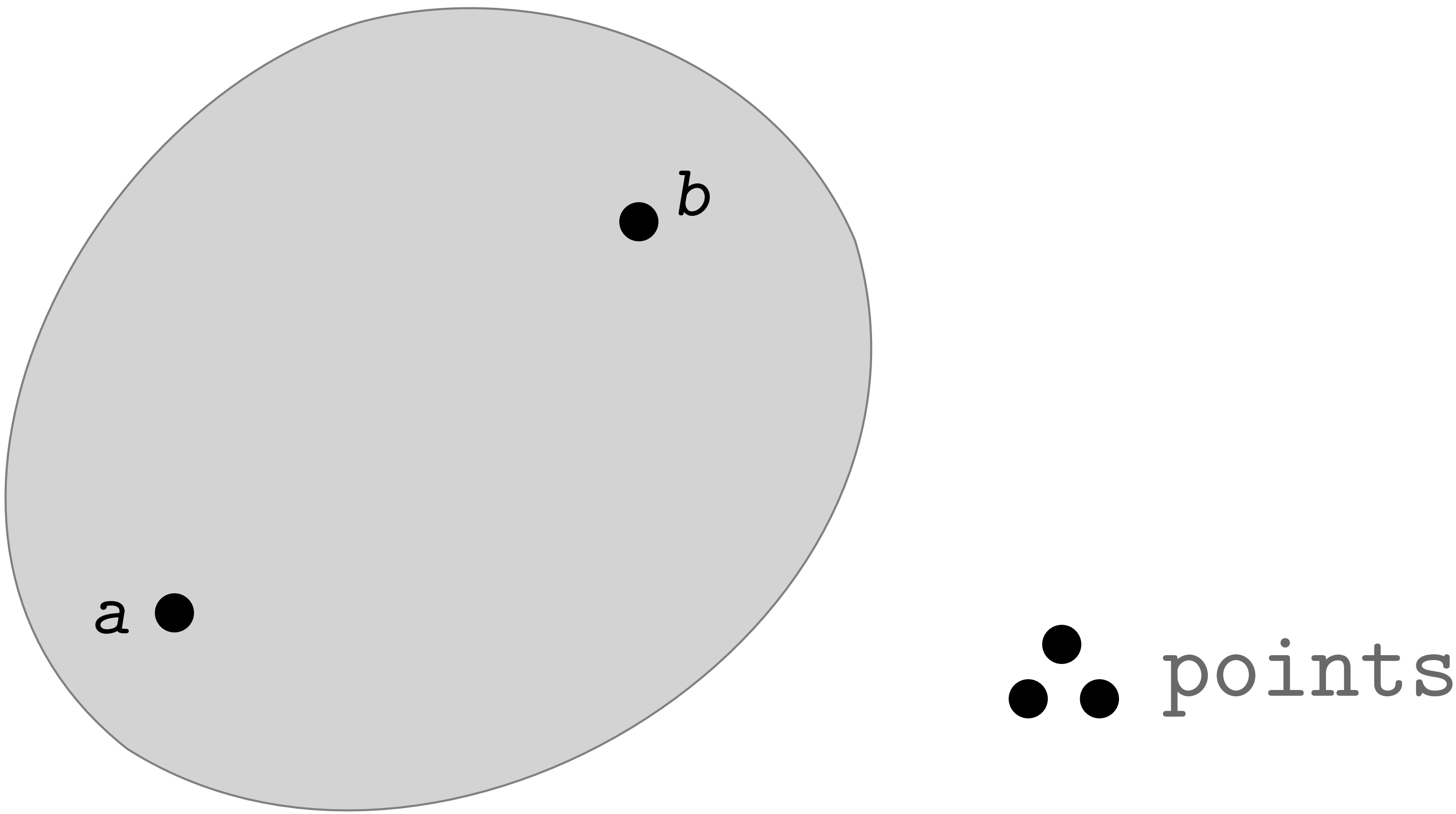
Computational type theory

↓

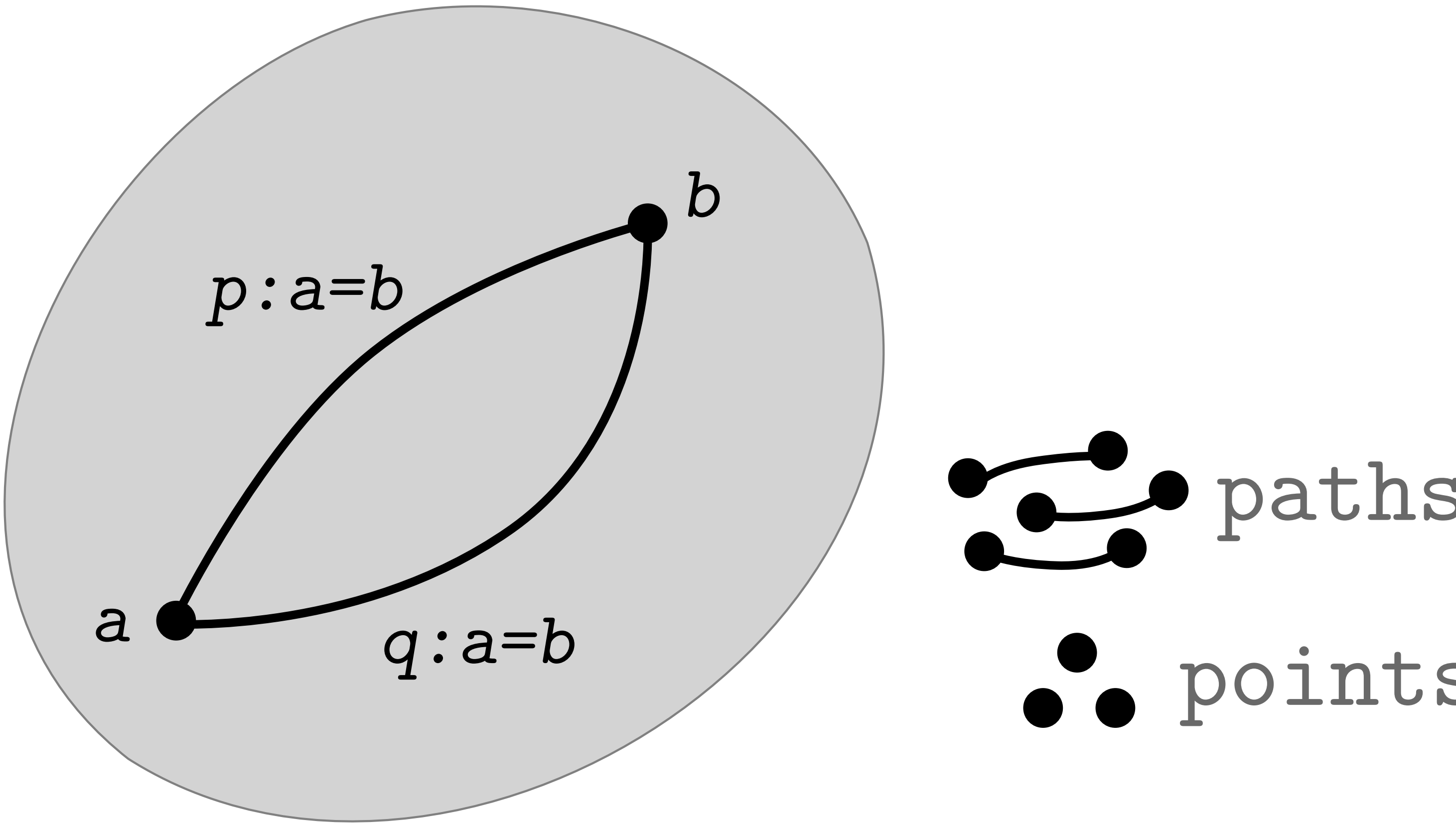
Programming language

Canonicity always holds

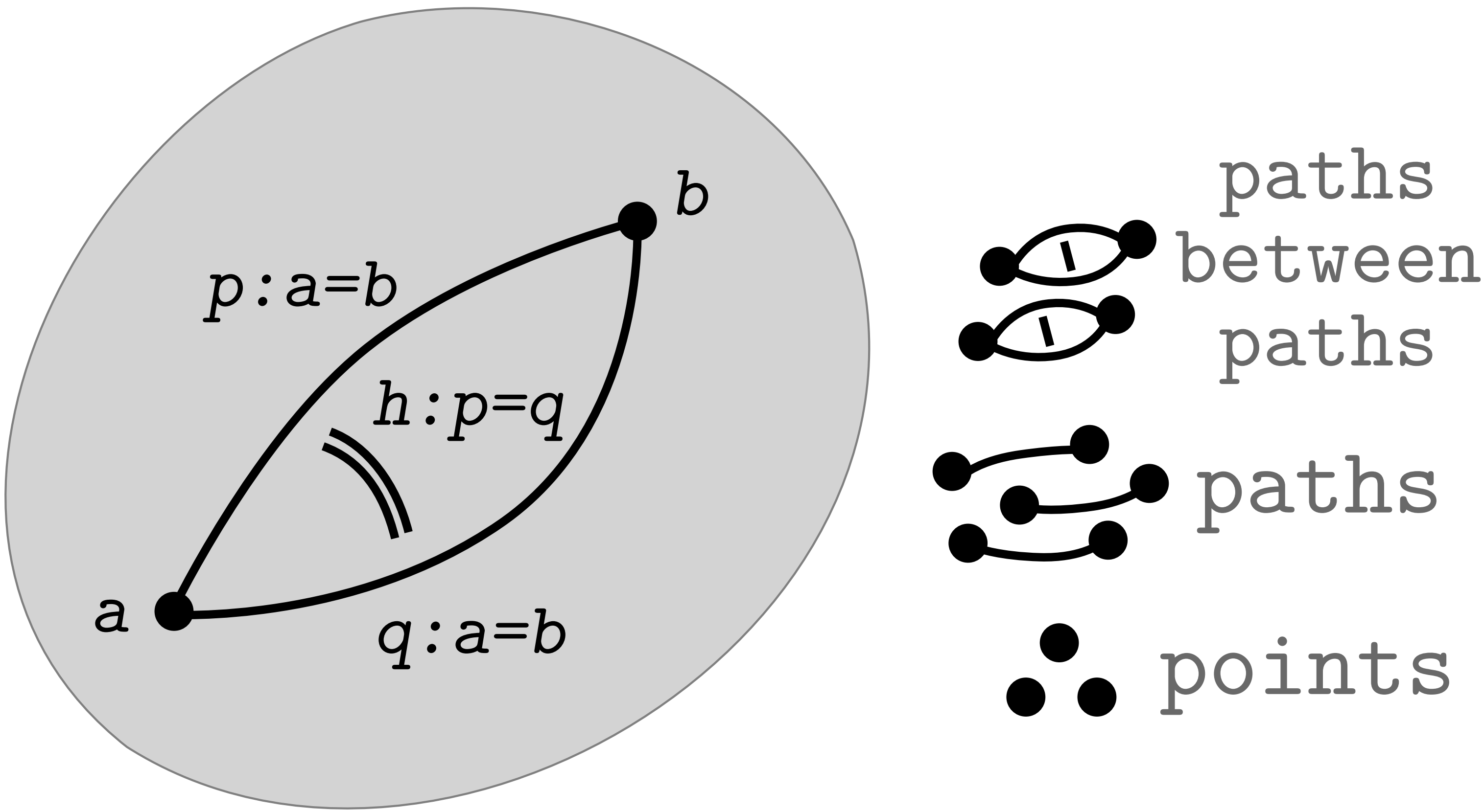
Homotopy Type Theory



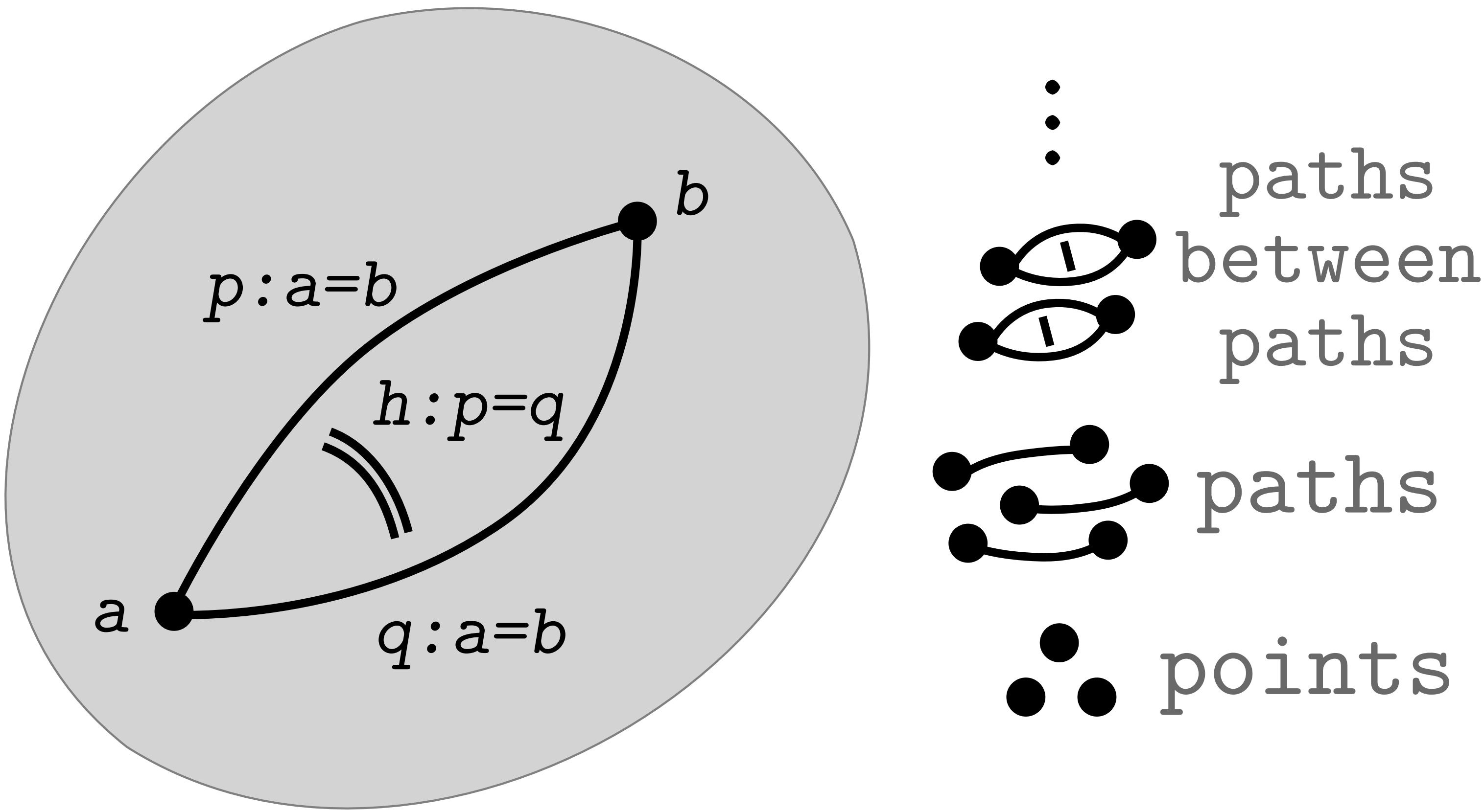
Homotopy Type Theory



Homotopy Type Theory



Homotopy Type Theory



Equality and Paths

Equality ($\equiv$)

Silent in theory

$$2 + 3 \equiv 5$$

$$fst\ \langle M, N \rangle \equiv M$$

Equality and Paths

Equality ($\equiv$)

Silent in theory

$2 + 3 \equiv 5$

$fst\ \langle M, N \rangle \equiv M$

If $A \equiv B$ and $M : A$ then $M : B$

Equality and Paths

Equality (≡)

Silent in theory

$2 + 3 \equiv 5$

$fst\ \langle M,N \rangle \equiv M$

If $A \equiv B$ and $M : A$ then $M : B$

Paths (=)

Visible in theory

If $P : A = B$ and $M : A$ then $transport(M,P) : B$

Homotopy Type Theory

[Awodey and Warren] [Voevodsky *et al*] [van den Berg and Garner]

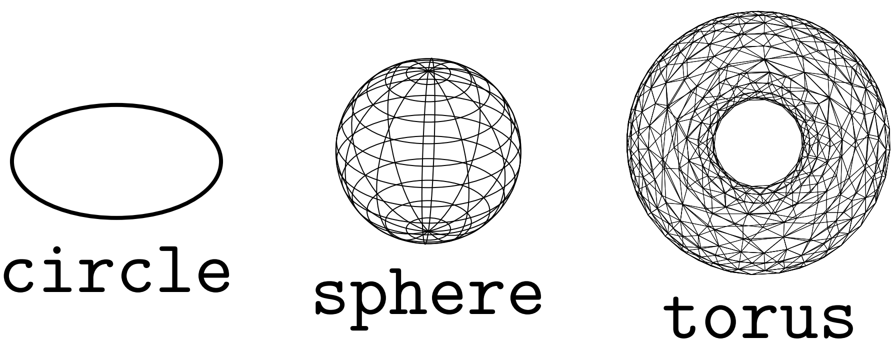
A	Type	Space
$a : A$	Element	Point
$f : A \to B$	Function	Continuous Mapping
$C : A \to Type$	Dependent Type	Fibration
$a =_A b$	Identification	Path

Features of HoTT

Univalence

If E is an equivalence between types A and B, then $ua(E) : A = B$

Higher Inductive Types



Canonicity?

Canonicity broken by
new features stated as axioms!

Canonicity?

Canonicity broken by
new features stated as axioms!

Canonicity

For any $M : bool$, either
$M \equiv true : bool$ or $M \equiv false : bool$

Canonicity?

Canonicity broken by
new features stated as axioms!

Canonicity

For any $M : bool$, either
$M \equiv true : bool$ or $M \equiv false : bool$

$ua(not) : bool = bool$

$transport(ua(not), true) \not\equiv false$

Canonicity for All

Canonicity for bool means
canonicity for *everyone*

Canonicity for All

Canonicity for bool means
canonicity for *everyone*

$$M : bool \times A$$

$$fst(M) \equiv ??? : bool$$

Canonicity for All

Canonicity for bool means
canonicity for *everyone*

$$M : bool \times A$$

$$fst(M) \equiv ??? : bool$$

Wants $M \equiv \langle P,Q \rangle$ and then
$fst(M) \equiv fst\langle P,Q \rangle \equiv P \equiv$ true or false

Canonicity for Paths?

$$\frac{M : A}{\mathit{refl}(M) : M =_A M}$$

Canonicity for Paths?

$$\frac{M : A}{\mathit{refl}(M) : M =_A M}$$

$$\frac{a{:}A \vdash R : C(a,a,\mathit{refl}(a)) \qquad P : M = N}{\mathit{path\text{-}ind}[C](a.R,P) : C(M,N,P)}$$

Canonicity for Paths?

$$\frac{M : A}{refl(M) : M =_A M}$$

$$\frac{a{:}A \vdash R : C(a,a,refl(a)) \qquad P : M = N}{path\text{-}ind[C](a.R,P) : C(M,N,P)}$$

$$\frac{a{:}A \vdash R : C(a,a,refl(a)) \qquad M : A}{\begin{array}{c} path\text{-}ind[C](a.R,refl(M)) \equiv R[M/a] \\ : C(M,M,refl(M)) \end{array}}$$

Canonicity for Paths?

$$\frac{M : A}{\mathtt{refl}(M) : M =_A M}$$

$$\frac{a{:}A \vdash R : C(a,a,\mathtt{refl}(a)) \qquad P : M = N}{\mathtt{path\text{-}ind}[C](a.R,P) : C(M,N,P)}$$

$$\frac{a{:}A \vdash R : C(a,a,\mathtt{refl}(a)) \qquad M : A}{\begin{array}{c}\mathtt{path\text{-}ind}[C](a.R,\mathtt{refl}(M)) \equiv R[M/a] \\ : C(M,M,\mathtt{refl}(M))\end{array}}$$

$$\mathtt{path\text{-}ind}[C](a.R,\mathtt{ua}(E)) \equiv \;???$$

Restore Canonicity

Can we have a new TT with canonicity + univalence?

Yes with De Morgan cubes [CCHM 2016]
Yes with Cartesian cubes [AFH 2017]

... and higher inductive types?

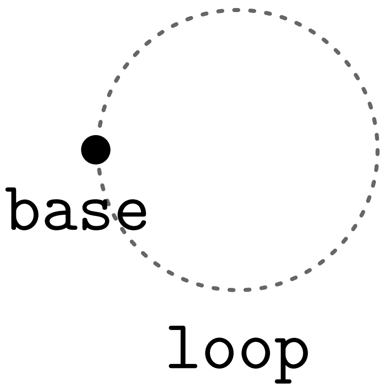
Examples with De Morgan cubes [CHM 2018]
Yes with Cartesian cubes [CH 2018]

Restore Canonicity

Idea: each type manages its own paths

Restore Canonicity

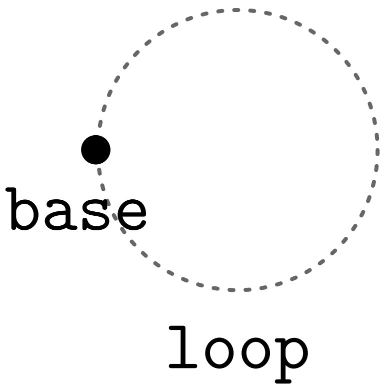
Idea: each type manages its own paths




```
base : S1
```

Restore Canonicity

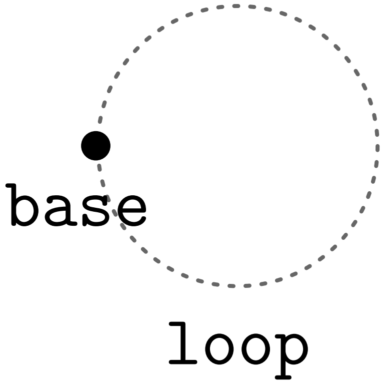
Idea: each type manages its own paths




```
base : S1
loop : base = base
```

Restore Canonicity

Idea: each type manages its own paths

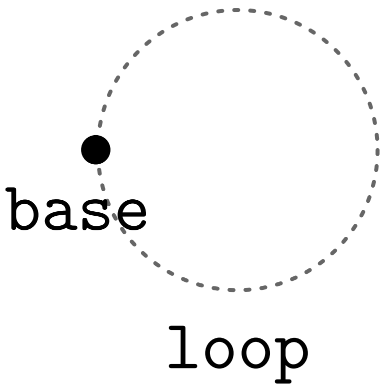


base : S1

~~loop : base = base~~

Restore Canonicity

Idea: each type manages its own paths



base : S1

~~loop : base = base~~

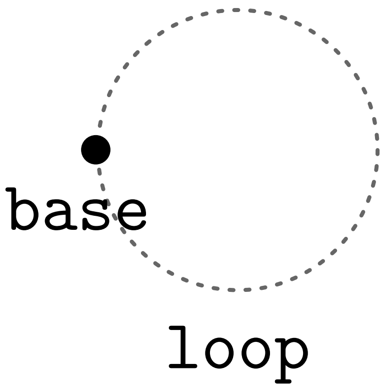
x:𝕀 ⊢ loop{x} : S1

loop{0} ≡ base : S1

loop{1} ≡ base : S1

Restore Canonicity

Idea: each type manages its own paths



base : S1

~~loop : base = base~~

x:𝕀 ⊢ loop{x} : S1

loop{0} ≡ base : S1

loop{1} ≡ base : S1

Kan structure:
sufficient to implement path-ind

Kan types: types with Kan structure

Cartesian Cubes

Introducing $\mathbb{I}$ the formal interval

Cartesian Cubes

Introducing $\mathbb{I}$ the formal interval

$$\Gamma \vdash 0 : \mathbb{I} \qquad \Gamma \vdash 1 : \mathbb{I}$$

$$\Gamma,\ x : \mathbb{I},\ \Gamma' \vdash x : \mathbb{I}$$

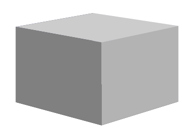
Cartesian Cubes

Introducing $\mathbb{I}$ the formal interval

$$\Gamma \vdash 0 : \mathbb{I} \qquad\qquad \Gamma \vdash 1 : \mathbb{I}$$

$$\Gamma,\ x : \mathbb{I},\ \Gamma' \vdash x : \mathbb{I}$$

$$x_1 : \mathbb{I},\ x_2 : \mathbb{I},\ \ldots,\ x_n : \mathbb{I} \vdash M : A$$

$\Leftrightarrow$ M is an n-cube in A 

Cartesian Cubes

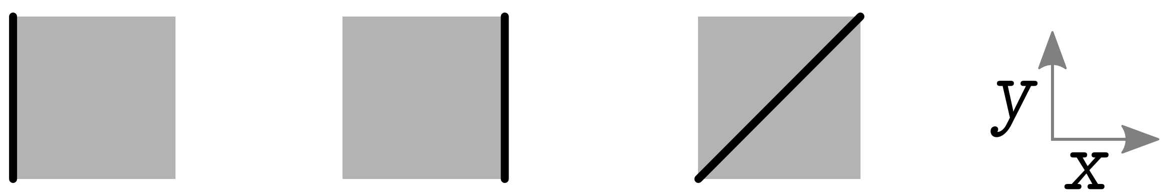
Introducing $\mathbb{I}$ the formal interval

$$\Gamma \vdash 0 : \mathbb{I} \qquad \Gamma \vdash 1 : \mathbb{I}$$

$$\Gamma,\ x : \mathbb{I},\ \Gamma' \vdash x : \mathbb{I}$$

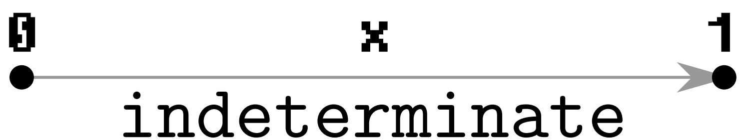
Cartesian: works as normal contexts

$$M\langle 0/x \rangle \quad M\langle 1/x \rangle \quad M\langle y/x \rangle$$

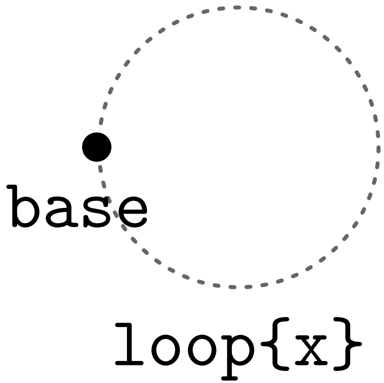


Cubical Programming

```
dim expr r := 0 | 1 | x
```



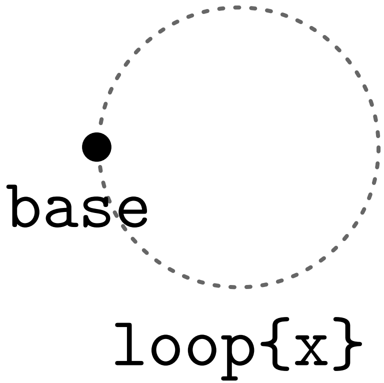
Circle



Circle

```
M := S1 | base | loop{r}
   | S1elim(a.M, M, M, x.M) | ...
```

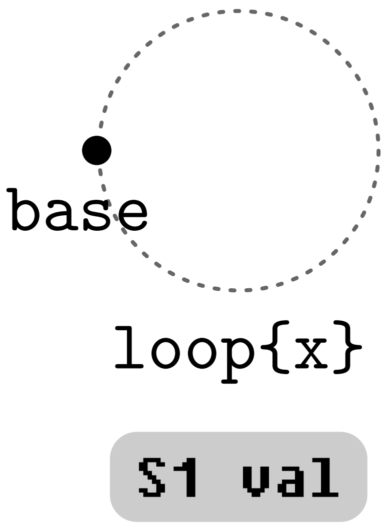
dim expr



Circle

```
M := S1 | base | loop{r}
   | S1elim(a.M, M, M, x.M) | ...
```

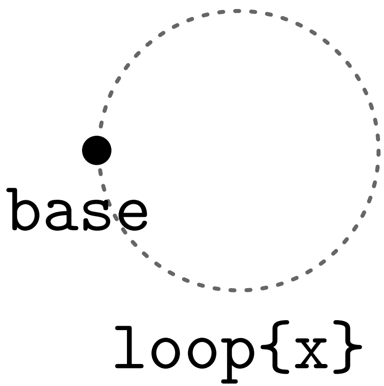
dim expr



Circle

```
M := S1 | base | loop{r}
   | S1elim(a.M, M, M, x.M) | ...
```

(r → dim expr)



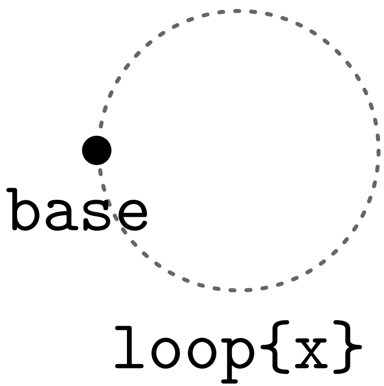
`S1 val`

`base val`

Circle

```
M := S1 | base | loop{r}
   | S1elim(a.M, M, M, x.M) | ...
```

dim expr



`S1 val`

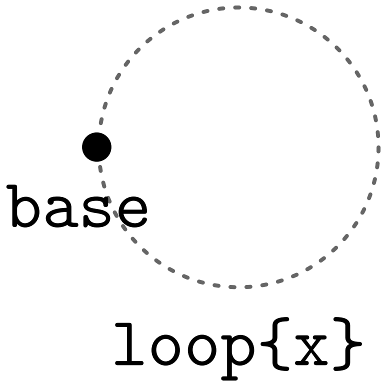
`base val`

`loop{x} val`

`loop{0} ↦ base`

`loop{1} ↦ base`

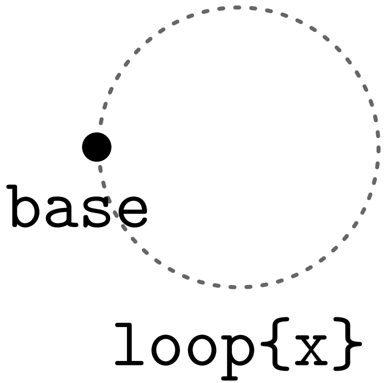
Circle



S1 val

```
M ↦ M'
──────────────────────────────
S1elim(a.A, M, B, x.L)
  ↦ S1elim(a.A, M', B, x.L)
```

Circle

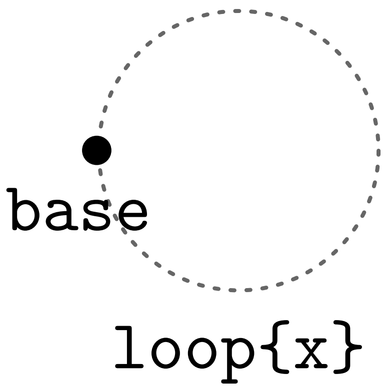



```
S1 val
```

```
M ↦ M'
──────────────────────────────
S1elim(a.A, M, B, x.L)
  ↦ S1elim(a.A, M', B, x.L)
```

```
S1elim(a.A, base, B, x._)
  ↦ B
```

Circle



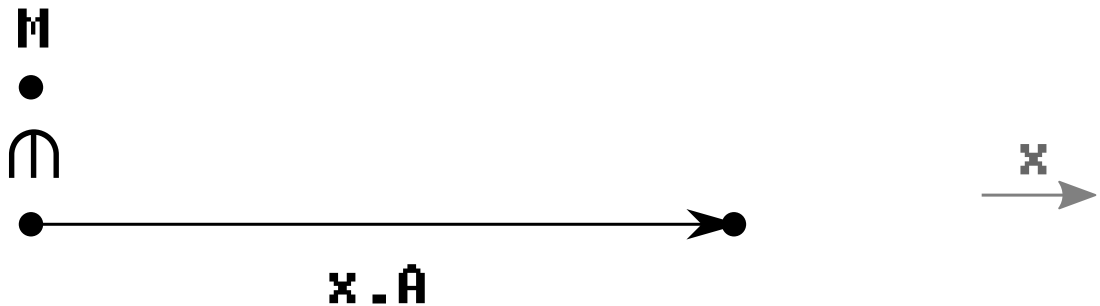

```
S1 val
```

```
M ↦ M'
────────────────────────────
S1elim(a.A, M, B, x.L)
  ↦ S1elim(a.A, M', B, x.L)
```

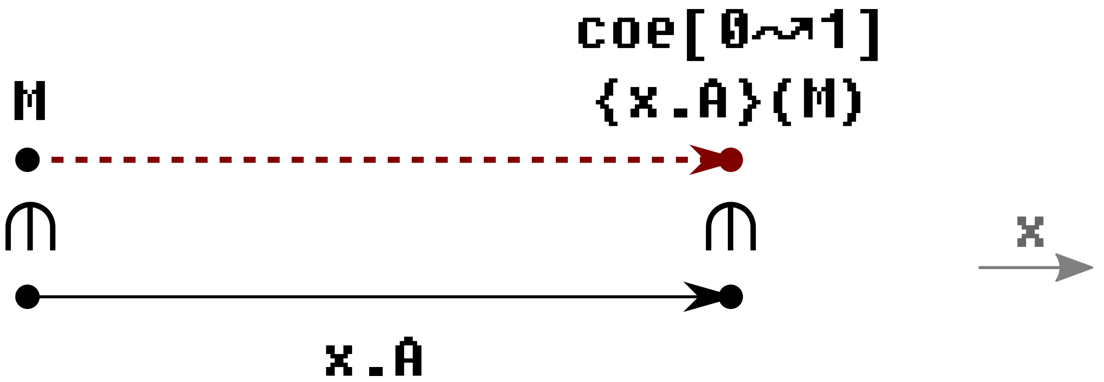
```
S1elim(a.A, base, B, x._)
  ↦ B
```

```
S1elim(a.A, loop{x}, _, y.L)
  ↦ L<x/y>
```

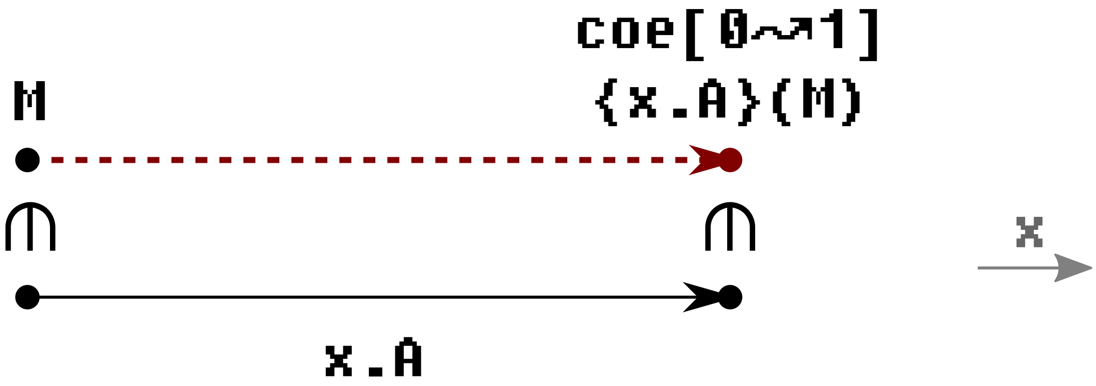
Kan 1/2: Coercion



Kan 1/2: Coercion



Kan 1/2: Coercion

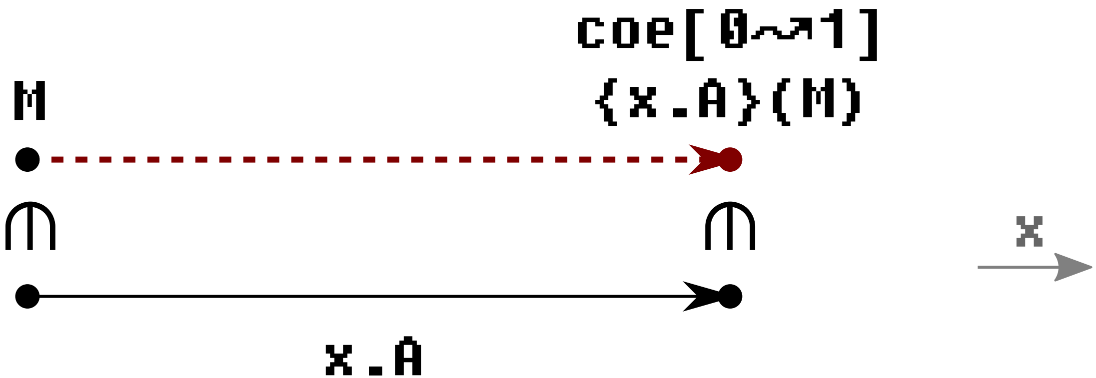


coe[r⇝r']{x.A}(M) ∈ A<r'/x>

⋔

A<r/x>

Kan 1/2: Coercion



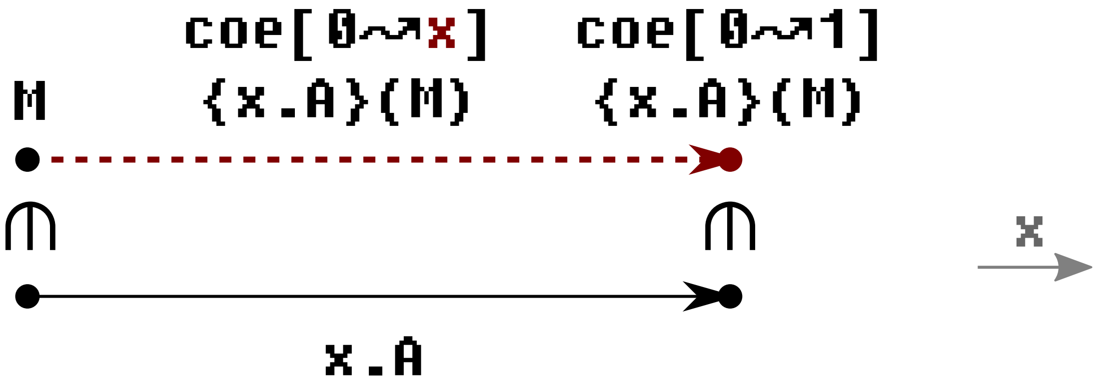
coe[r↝r']{x.A}(M) ∈ A<r'/x>

⋒

A<r/x>

coe[r↝r]{x.A}(M) ≐ M ∈ A<r/x>

Kan 1/2: Coercion



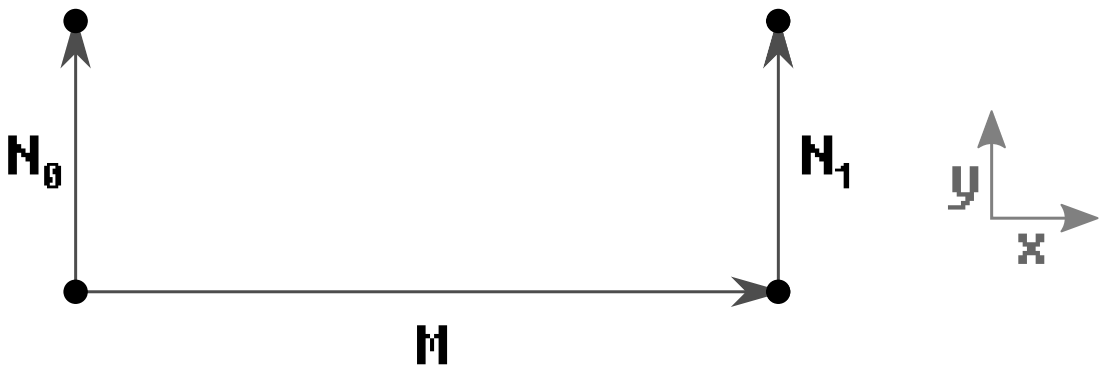
coe[r⤳r']{x.A}(M) ∈ A<r'/x>

⋔

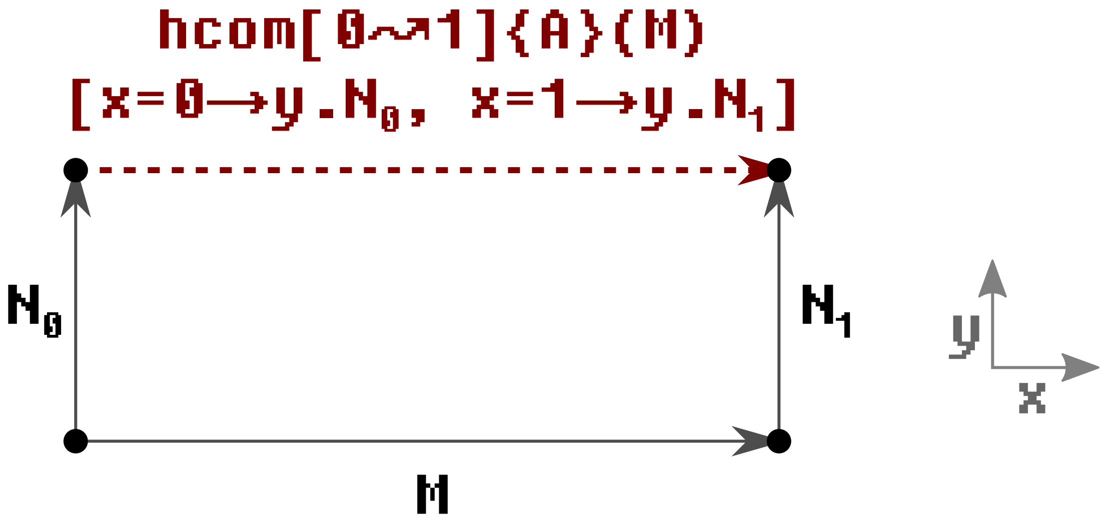
A<r/x>

coe[r⤳r]{x.A}(M) ≐ M ∈ A<r/x>

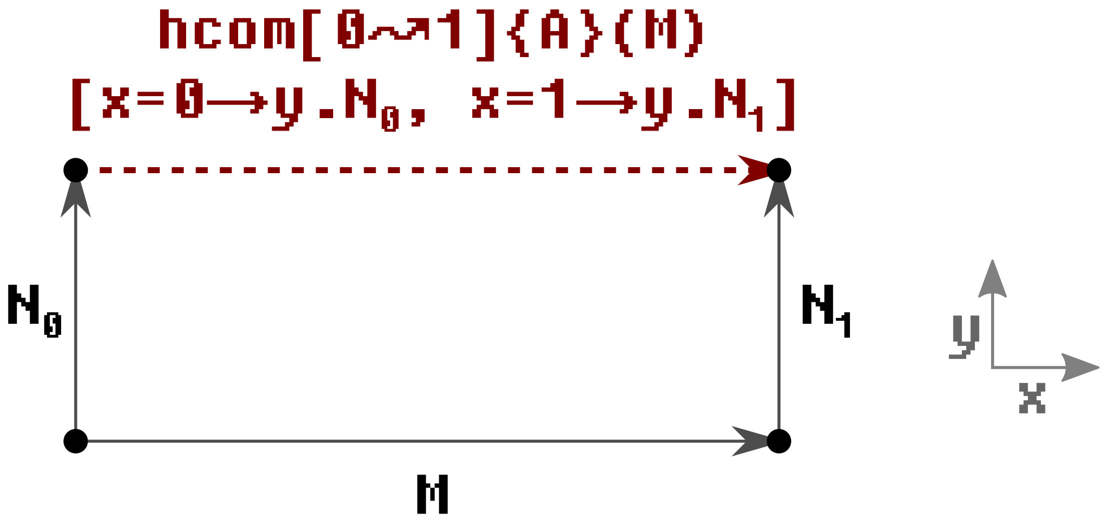
Kan 2/2: Homogeneous Comp.



Kan 2/2: Homogeneous Comp.

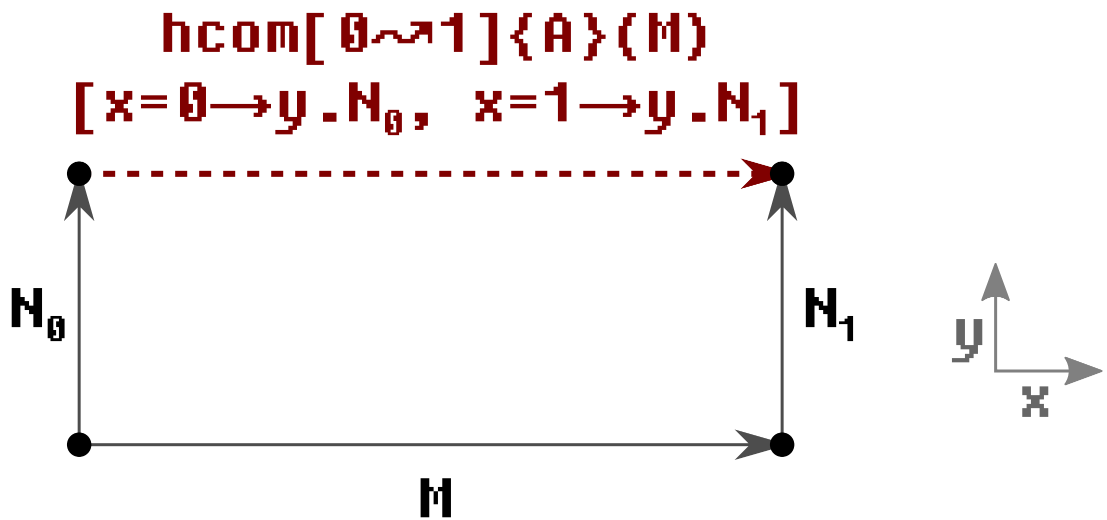


Kan 2/2: Homogeneous Comp.



hcom[r⤳r']{A}(M) [..., r_i=r'_i⟶y.N_i, ...] ∈ A

Kan 2/2: Homogeneous Comp.

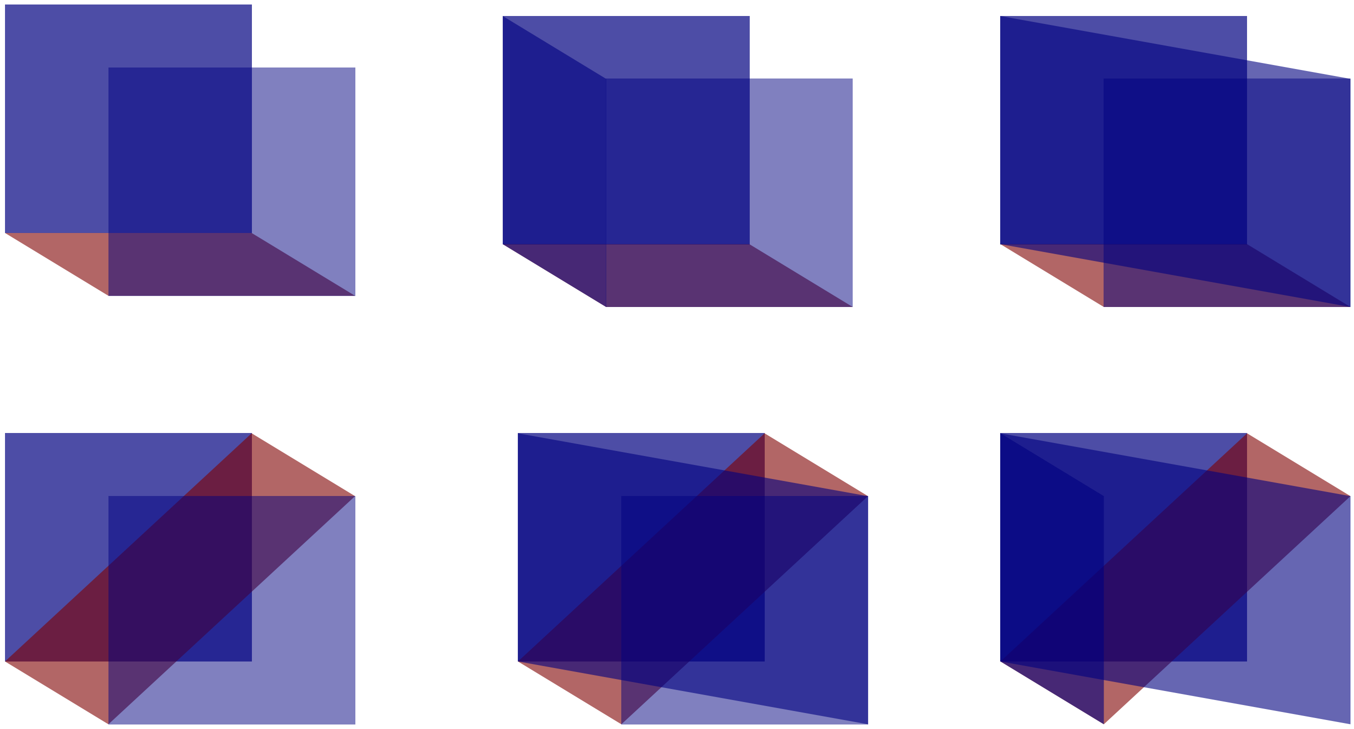


hcom[r⤳r']{A}(M) [..., r_i=r'_i⟶y.N_i, ...] ∈ A

hcom[r⤳r]{A}(M) ≐ M ∈ A

hcom[r⤳r']{A}(M)[..., r_i=r_i⟶y.N_i, ...]
  ≐ N_i⟨r'/y⟩ ∈ A

Kan 2/2: Homogeneous Comp.



Kan Circle

```
coe[r~>r'](_.S1)(M) ↦ M
```

Kan Circle

```
coe[r~>r'](_.S1)(M) ↦ M
```

```
hcom[r~>r'](S1)(M)[…] ↦ fhcom[r~>r'](M)[…]
```

formal homo.
composition

Kan Circle

```
coe[r~>r']{_.S1}(M) ↦ M
```

```
hcom[r~>r']{S1}(M)[…] ↦ fhcom[r~>r'](M)[…]
```

formal homo. composition

```
fhcom[r~>r](M)[…] ↦ M
```

Kan Circle

```
coe[r⤳r']{_.S1}(M) ↦ M
```

```
hcom[r⤳r']{S1}(M)[…] ↦ fhcom[r⤳r'](M)[…]
```

formal homo.
composition

```
fhcom[r⤳r](M)[…] ↦ M
```

```
r≠r'   rᵢ=r'ᵢ (the first i)
──────────────────────────────────────────────
fhcom[r⤳r'](M)[…, rᵢ=r'ᵢ→y.Nᵢ, …] ↦ Nᵢ<r'/y>
```

Kan Circle

```
coe[r⇝r'](_.S1}(M) ↦ M
```

```
hcom[r⇝r']{S1}(M)[…] ↦ fhcom[r⇝r'](M)[…]
```

formal homo. composition

```
fhcom[r⇝r](M)[…] ↦ M
```

```
r!=r'   r_i=r'_i (the first i)
--------------------------------------------
fhcom[r⇝r'](M)[…, r_i=r'_i→y.N_i, …] ↦ N_i<r'/y>
```

```
r!=r'   r_i!=r'_i for all i
----------------------------
fhcom[r⇝r'](M)[…] val
```

Kan Circle

`S1elim` needs to handle `fcom`

Kan Circle

S1elim needs to handle **fcom**

```
r≠r'  rᵢ≠r'ᵢ
─────────────────────────────────────────
S1elim(a.A, fhcom[r⇝r'](M)[…], B, x.L)
  ↦ com[r⇝r']{y.A[fhcom[r⇝y](M)[…]/a}
        (S1elim(M, B, x.L))[…]
```

S1elim(composition) ↦ composition(S1elim)

Cubical Stability

Dimension substs. do not commute with evaluation!

Cubical Stability

Dimension substs. do not commute with evaluation!

```
S1elim(a.A,
 loop{x}, B, y.L)   |————————→  L<x/y>
                                   |
                                   |  <0/x>
                                   ↓

                                 L<0/y>
```

Cubical Stability

Dimension substs. do not commute with evaluation!

```
S1elim(a.A,
 loop{x}, B, y.L)    |-------->  L<x/y>

        |                          |
        |  <0/x>                   |  <0/x>
        v                          v

S1elim(a.A,
 base, B, y.L)    |--> B   <=??=>  L<0/y>
```

Cubical Stability

Dimension substs. do not commute with evaluation!

```
S1elim(a.A,
 loop{x}, B, y.L)   |-----------> L<x/y>

        |                            |
        | <0/x>                      | <0/x>
        ↓                            ↓

S1elim(a.A,
 base, B, y.L)   |--> B   <=??=> L<0/y>
```

Restrict our theory to only cubically stable parts

Cubical Type Theory

stability: consider every substitution

Cubical Type Theory

stability: consider every substitution

$$A \doteq B \text{ type } [\Psi]$$

dim context

A and B stably recognize the same stable values and have stably equal Kan structures

(see our arXiv papers)

Cubical Type Theory

stability: consider every substitution

$$A \doteq B \ \mathsf{type}\ [\Psi]$$

dim context

A and B stably recognize the same stable values and have stably equal Kan structures

$$M \doteq N \in A\ [\Psi]$$

$A \doteq A$ type $[\Psi]$,
M and N stably eval to M' and N',
A stably treats M' and N' as the same

(see our arXiv papers)

Variables

Nuprl/...	Coq/Agda/...
Vars range over closed terms	Vars are indet.
Defined by transition b/w closed terms	Defined by conversion b/w open terms

exp vars dim vars

cubical computational TT

arXiv papers

CHTT Part I [AHW 2016]

Cartesian cubical + computational

CHTT Part II [AH 2017]

Dependent types

CHTT Part III [AFH 2017]

Univalent Kan universes

Strict equality

CHTT Part IV [AFH 2017]

Higher inductive types

Proof Assistants

RedPRL

In Nuprl style

redprl.org

redtt

(Work in progress)

github.com/RedPRL/redtt

---

yacctt

Proof of concept

modified from cubicaltt

github.com/mortberg/yacctt

Conclusion

We extended Nuprl semantics
by cubical structure which
justifies key features of HoTT

Conclusion

We extended Nuprl semantics
by cubical structure which
justifies key features of HoTT

Best of the two worlds!

Conclusion

We extended Nuprl semantics
by cubical structure which
justifies key features of HoTT

Best of the two worlds!

We also built proof assistants

redprl.org
github.com/RedPRL/redtt
github.com/mortberg/yacctt