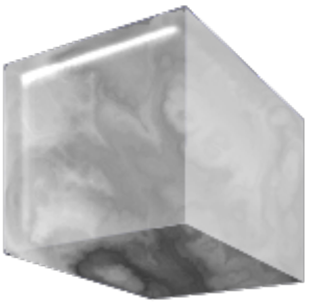
2018.02.23 Penn

Cubical Computational Type Theory & RedPRL

>> redprl.org >>



Carlo Angiuli
Evan Cavallo
(*) Favonia
Bob Harper
Dan Licata
Jon Sterling
Todd Wilson

Vladimir Voevodsky
1966-2017

Martin Hofmann
1965-2018

Cubical & Computational

features of homotopy type theory (HoTT)

features of computational type theory
(equality types, strict quotients, ...)

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    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    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    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    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    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    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

bool: {true, false} with syntactic equality

A Minimum Example

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

types: {bool} with syntactic equality
bool: {true, false} with syntactic equality
```

A ≐ B type

A⇓A' B⇓B' and A'≈B'

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)
types: {bool} with syntactic equality
bool: {true, false} with syntactic equality
```

$A \doteq B$ type

$A \Downarrow A'$ $B \Downarrow B'$ and $A' \approx B'$

$\mathrm{bool} \doteq \mathrm{bool}$ type

A Minimum Example

```
M := a | bool | true | false | if(M,M,M)
types: {bool} with syntactic equality
bool: {true, false} with syntactic equality
```

$$M \doteq N \in A$$

$A \doteq A$ 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

bool: {true, false} with syntactic equality

$$M \doteq N \in A$$

$A \doteq A$ 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
```

$$M \doteq N \in A$$

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

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

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

A Minimum Example

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

types: {bool} with syntactic equality
bool: {true, false} with syntactic equality
```

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

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

A Minimum Example

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

`types: {bool}` with syntactic equality

`bool: {true, false}` with syntactic equality

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

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

$b{:}bool \gg b \doteq if(b,true,false) \in bool$?

Variables

In Nuprl and friends

variables range over closed terms

Variables

In Nuprl and friends

variables range over closed terms

In Coq, Agda, and friends

variables are *not* subject to inductive analysis

Variables

In Nuprl and friends

variables range over closed terms

In Coq, Agda, and friends

variables are *not* subject to inductive analysis

---

closed reduction ⇔ vars over closed terms

open reduction ⇔ indeterminate vars

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
```

\a.M1 $\approx_{A\to B}$ \a.M2 if a:A >> M1 ≐ M2 ∈ B

Open-endedness

Open-endedness

Open to new constructs

Open-endedness

Open to new constructs

Open to new theories
for the same language

Open-endedness

Open to new constructs

Open to new theories
for the same language

Open to new proof theories
(rules in proof assistants)
for the same theory

Open-endedness

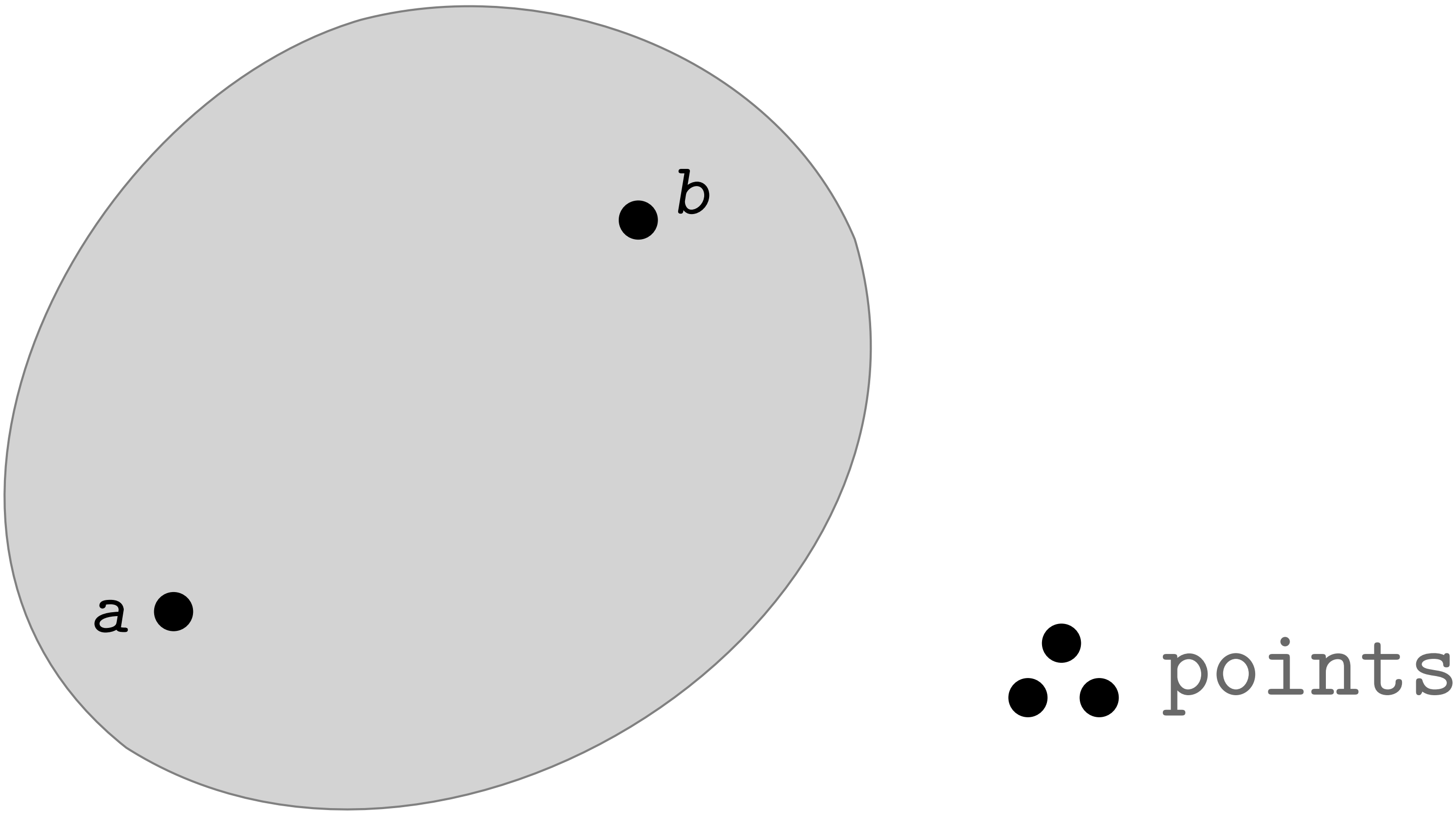
Open to new constructs

Open to new theories
for the same language

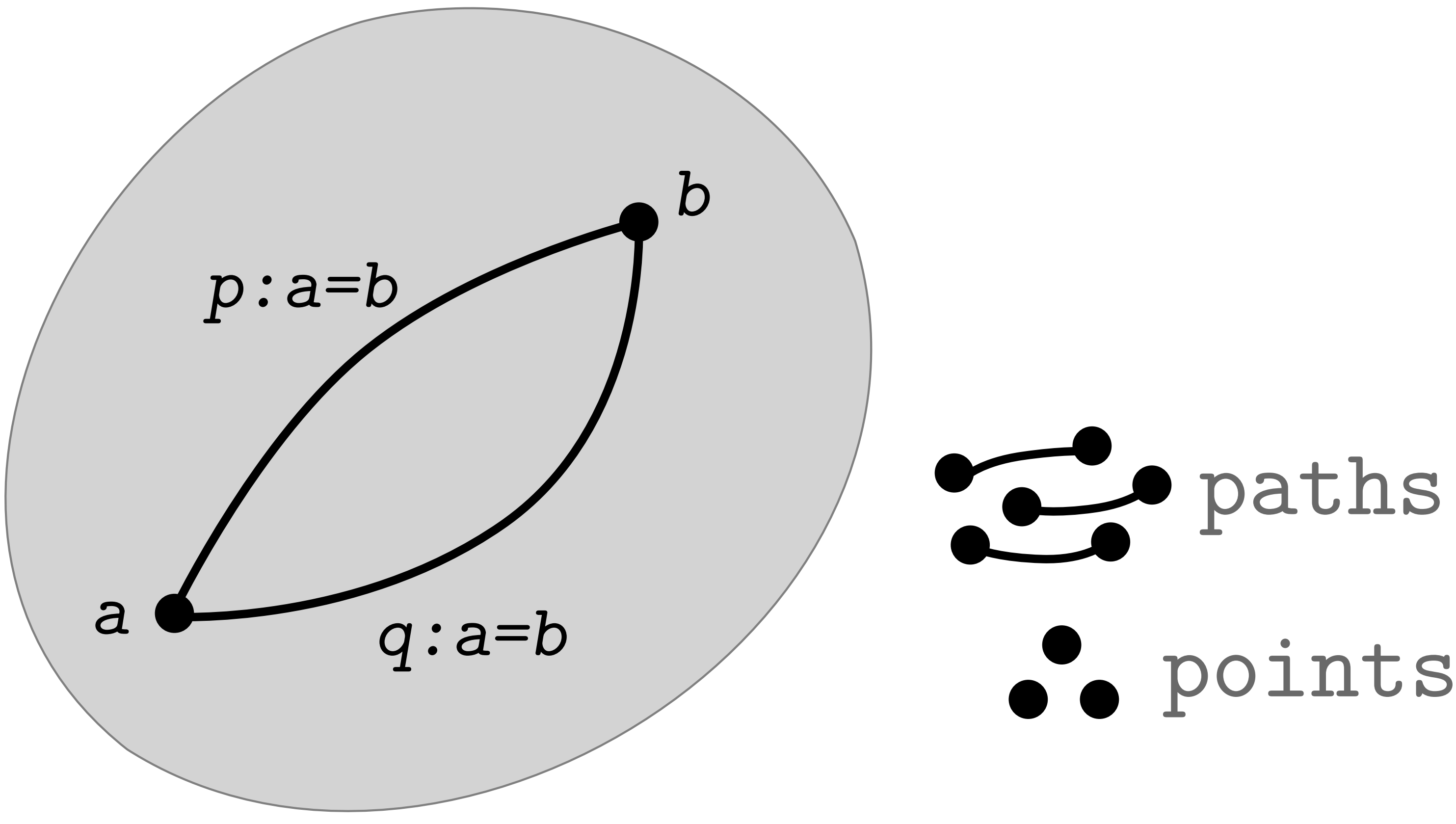
Open to new proof theories
(rules in proof assistants)
for the same theory

Canonicity always holds

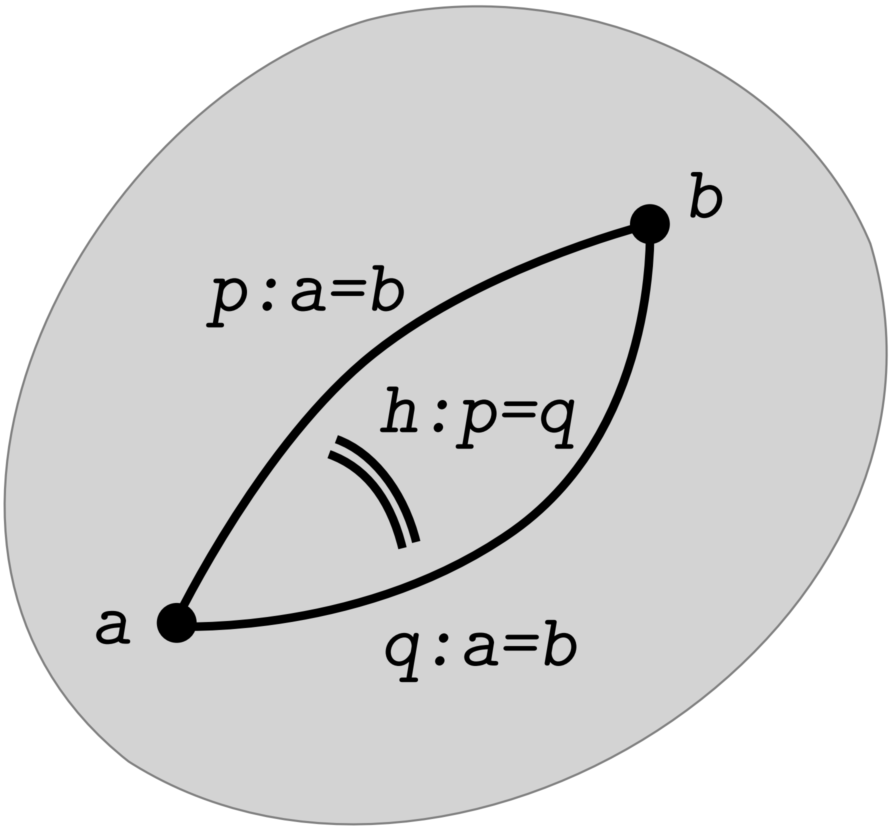
Homotopy Type Theory

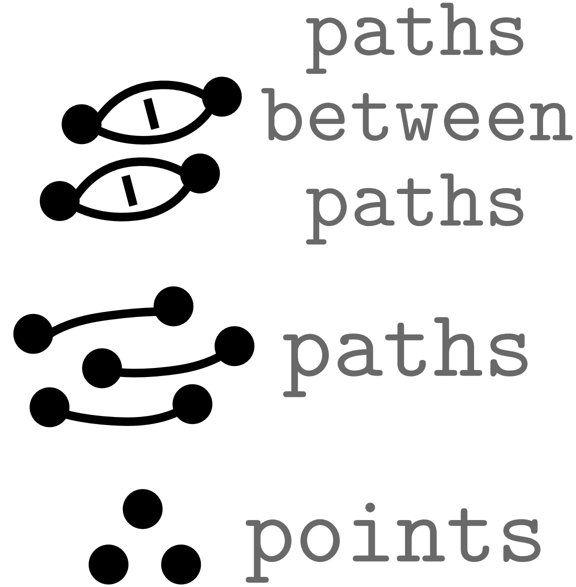


Homotopy Type Theory

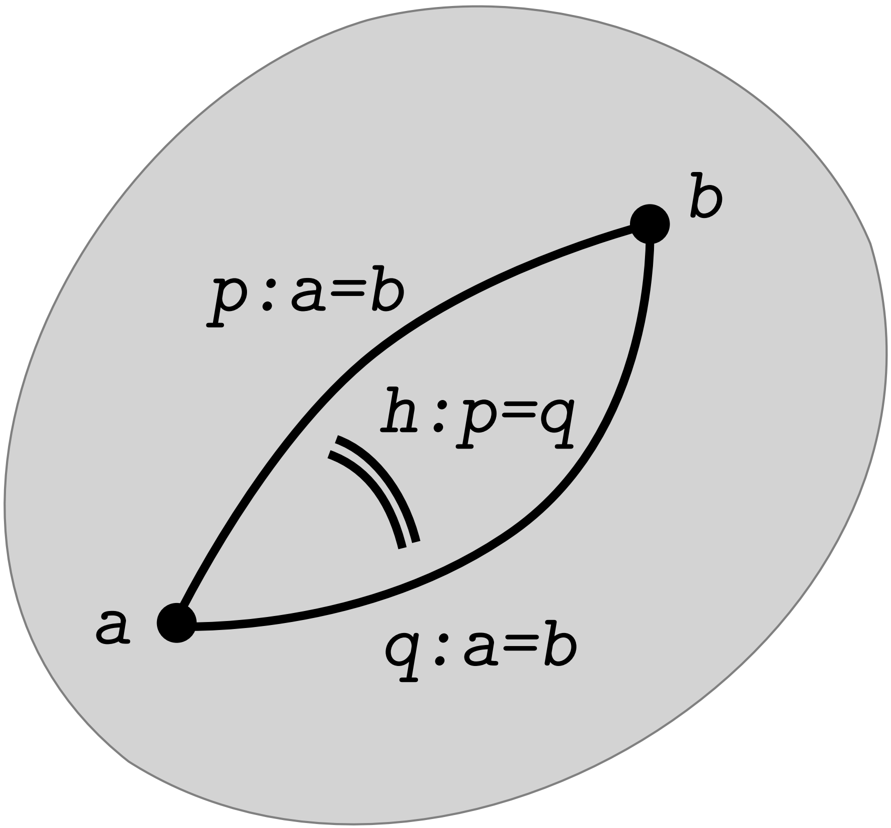


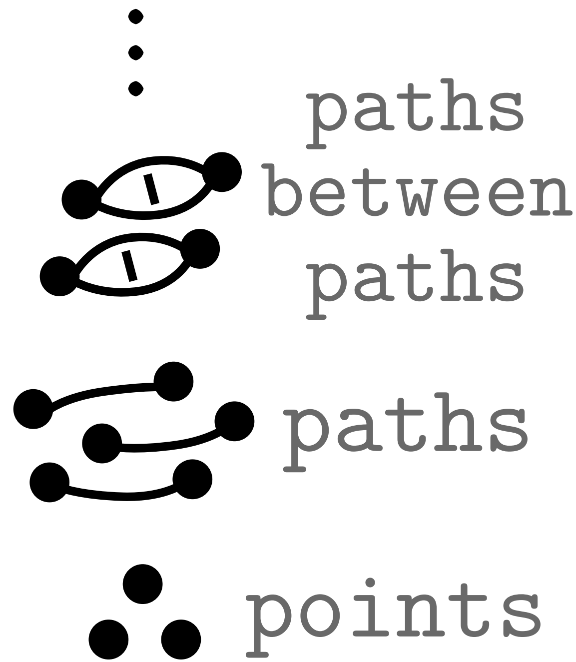
Homotopy Type Theory





Homotopy Type Theory





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

Homotopy Type Theory

Numerous results in homotopy theory mechanized through this.

In some case new proofs were discovered and inspired new results.

[Anel, Biedermann, Finster, Joyal]

Extensive works in category theory and other fields.

Key Features of HoTT

1. Identifications as paths
2. Univalence: if e is an equivalence between A and B, then $ua(e) : A = B$
3. Higher inductive types:
   generalized inductive types
   with (higher) path generators

Key Features of HoTT

1. Identifications as paths
2. Univalence: if e is an equivalence between A and B, then $ua(e) : A = B$
3. Higher inductive types:
   generalized inductive types
   with (higher) path generators

Problems: 1&2&3 give new identifications

The Poor Eliminator

elim-path[a.*C*](refl-case, path)

can only handle reflexivity

coe(*p*:*A*=*B*,a:*A*):*B*

coe(*ua*(*e*),*a*) is stuck

The Poor Eliminator

elim-path[a.*C*](refl-case, path)

can only handle reflexivity

coe(*p*:*A*=*B*,a:*A*):*B*

coe(*ua*(*e*),*a*) is stuck

Solution

each motive *C* handles paths itself

The Happy Eliminator

elim-path[a.C](refl-case, path)

each motive handles paths itself

↑

each type has cubical Kan structure

[Bezem, Coquand, Huber] [Cohen, Coquand, Huber, Mörtberg]

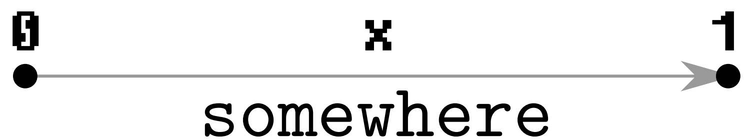
This work:

extend Nuprl by cubical Kan structures

[Angiuli, Harper, Wilson] [Angiuli, Harper] [Angiuli, Favonia, Harper] [Cavallo, Harper]

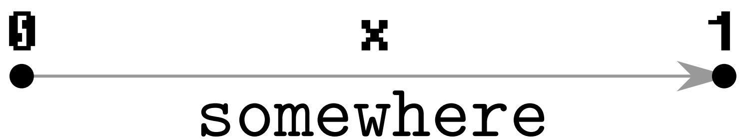
Cubical Programming

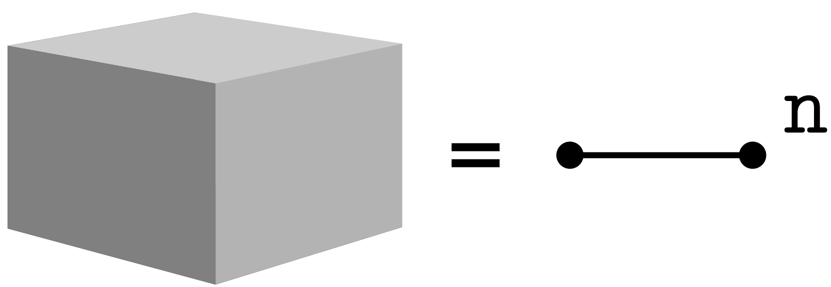
```
dim expr r := 0 | 1 | x
```



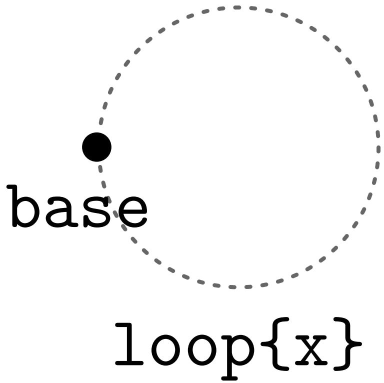
Cubical Programming

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





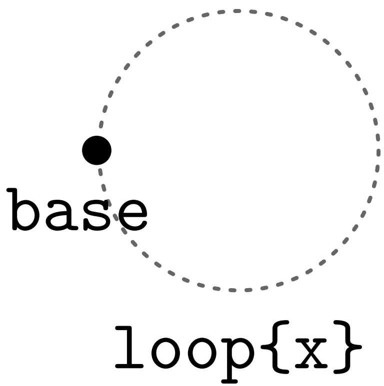
Circle



Circle

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

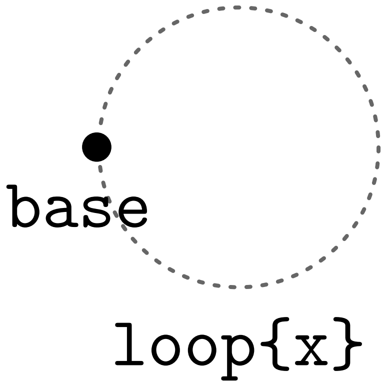
r → dim expr



Circle

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

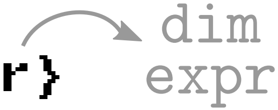
dim expr

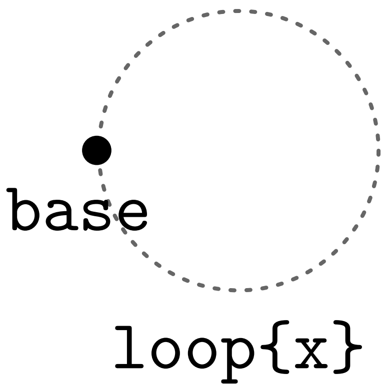


S1 val

Circle

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





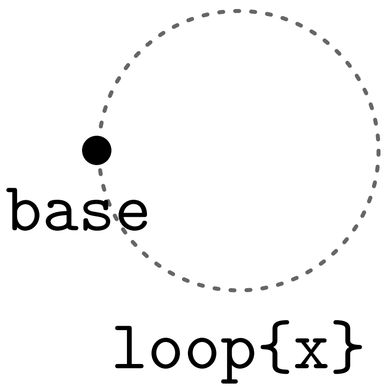
`S1 val`

`base val`

Circle

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

(r: dim expr)



`S1 val`

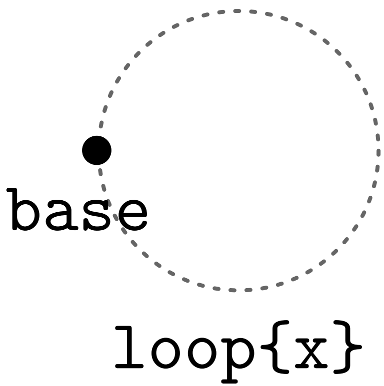
`base val`

`loop{x} val`

`loop{0} ↦ base`

`loop{1} ↦ base`

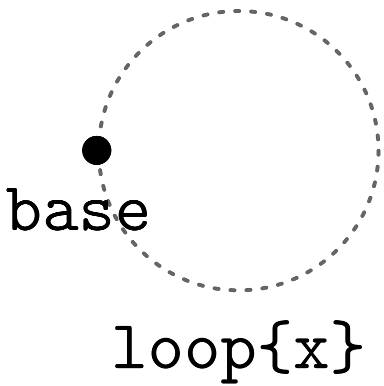
Circle




```
S1 val
```

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

Circle

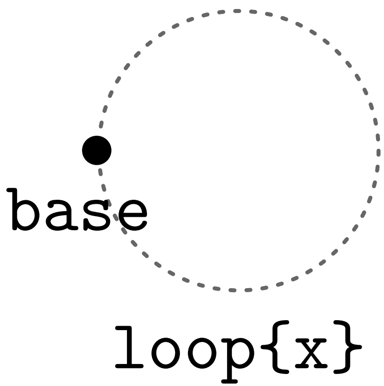



```
S1 val
```

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

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

Circle



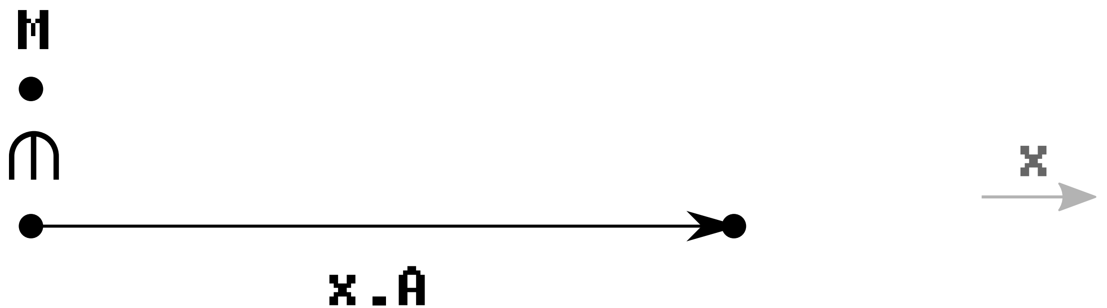

```
S1 val
```

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

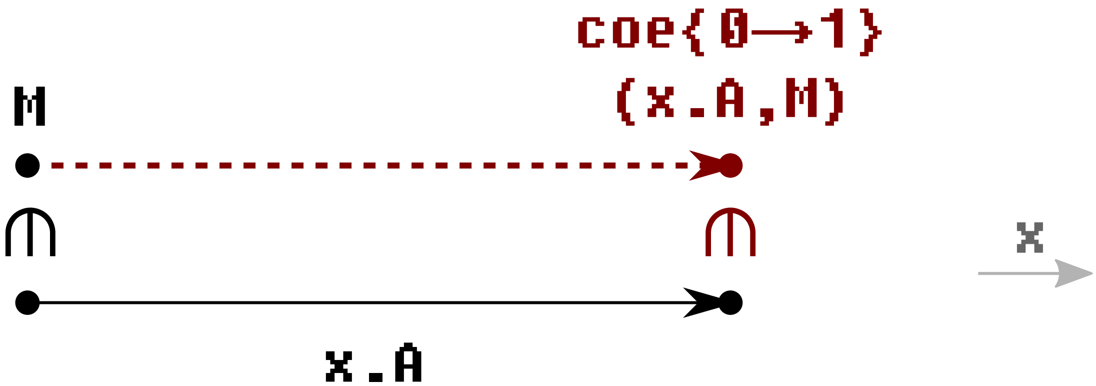
```
S1elim(a.A, base, B, x._)
  ↦ B
```

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

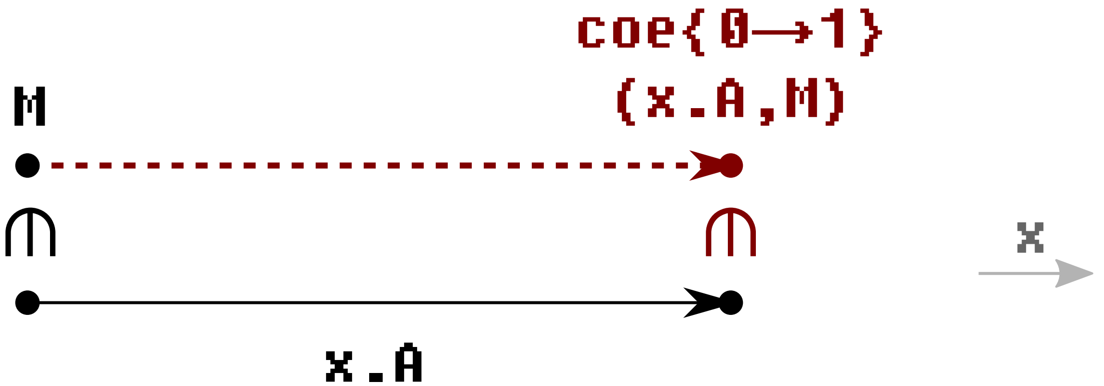
Kan 1/2: Coercion



Kan 1/2: Coercion



Kan 1/2: Coercion



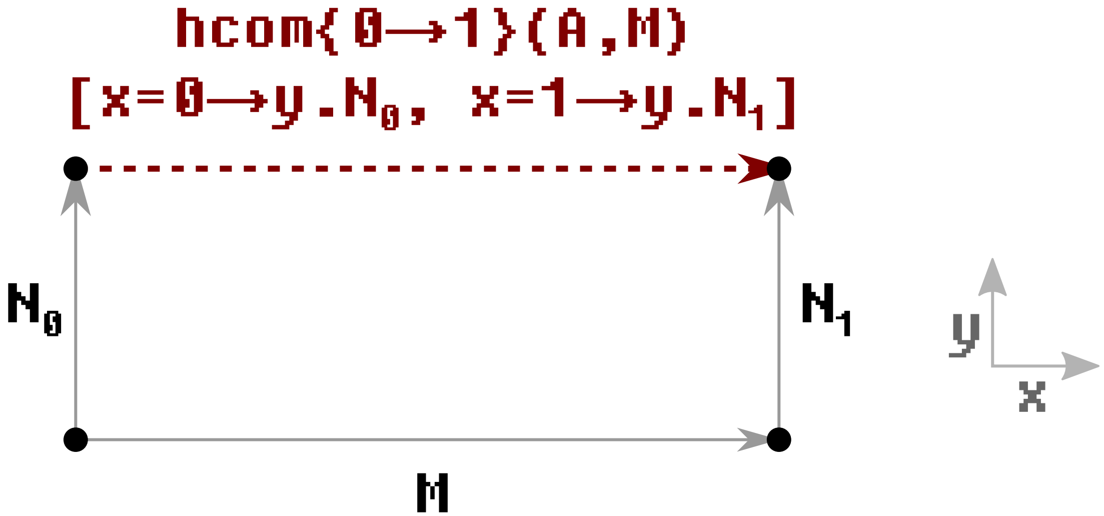
coe{r⟶r'}(x.A, M) ∈ A<r'/x>

⋔

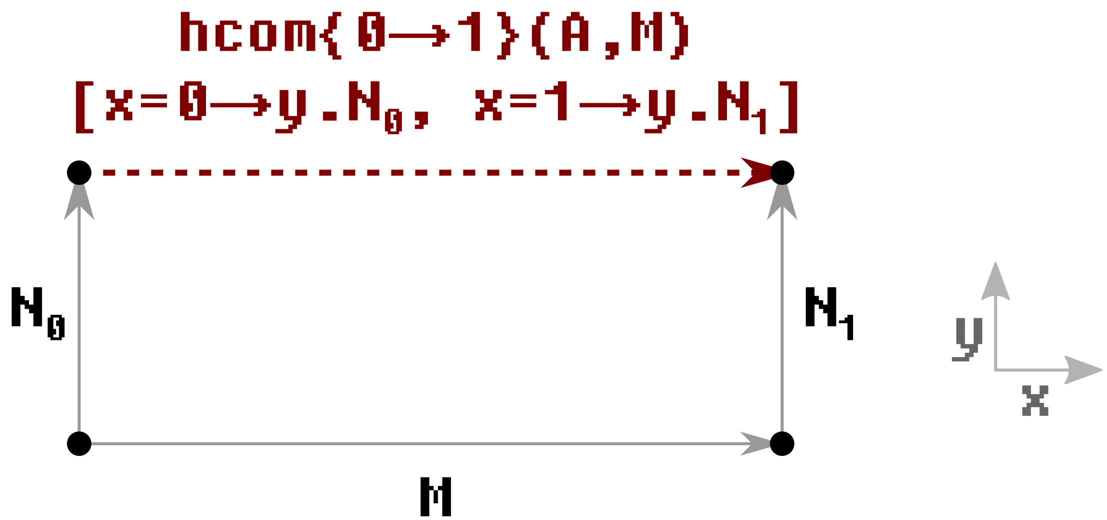
A<r/x>

Kan 2/2: Homogeneous Composition

Kan 2/2: Homogeneous Composition

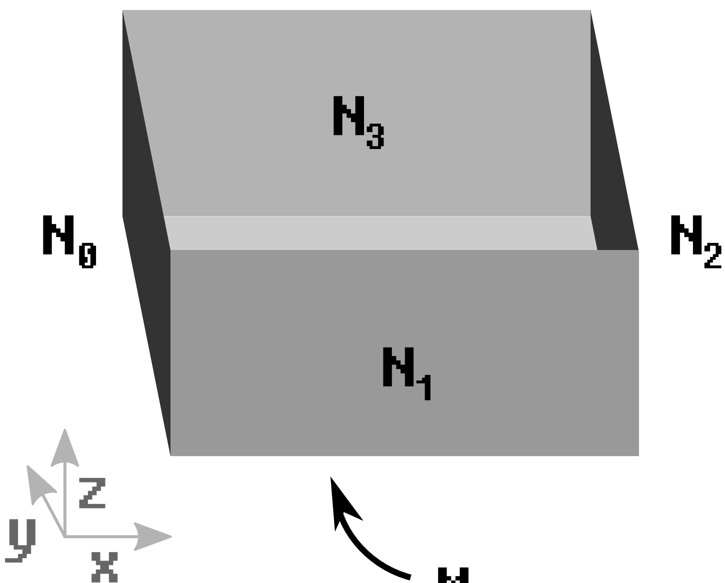


Kan 2/2: Homogeneous Composition

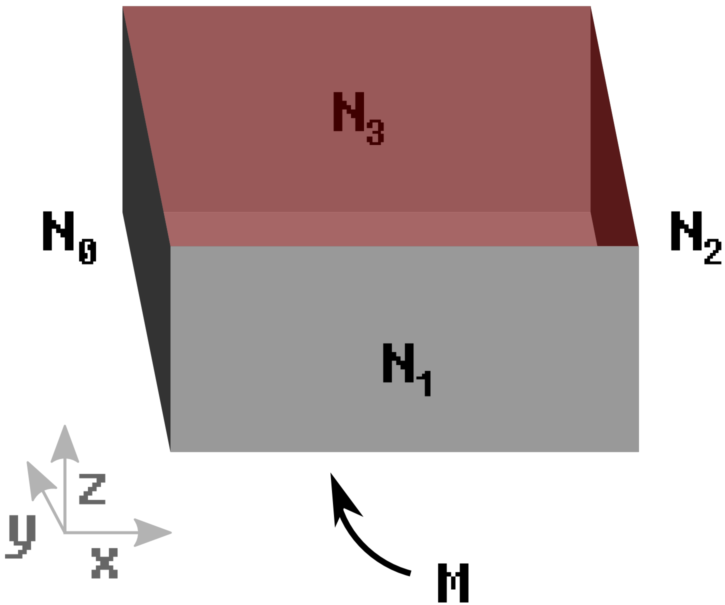



```
hcom{r→r'}(A, M)
  [..., rᵢ=r'ᵢ→y.Nᵢ, ...]
```

Kan 2/2: Homogeneous Composition



Kan 2/2: Homogeneous Composition




```
hcom{0⟶1}(A,M)
  [x=0⟶z.N₀,
   y=0⟶z.N₁,
   x=1⟶z.N₂,
   y=1⟶z.N₃]
```

Kan Circle

```
coe{r⟶r'}(_.S1, M) ↦ M
```

Kan Circle

```
coe{r⟶r'}(_.S1, M) ↦ M
```

```
hcom{r⟶r'}(S1, M)[...] ↦ fcom{r⟶r'}(M)[...]
```

formal composition

Kan Circle

```
coe{r⟶r'}(_.S1, M) ↦ M
```

```
hcom{r⟶r'}(S1, M)[...] ↦ fcom{r⟶r'}(M)[...]
```

formal composition

```
fcom{r⟶r}(M)[...] ↦ M
```

Kan Circle

```
coe{r⟶r'}(_.S1, M) ↦ M
```

```
hcom{r⟶r'}(S1, M)[...] ↦ fcom{r⟶r'}(M)[...]
```

formal composition

```
fcom{r⟶r}(M)[...] ↦ M
```

```
r≠r'   r_i=r'_i (the first i)
————————————————————————————————————————
fcom{r⟶r'}(M)[..., r_i=r'_i⟶y.N_i, ...] ↦ N_i<r'/y>
```

Kan Circle

```
coe{r⟶r'}(_.S1, M) ↦ M
```

```
hcom{r⟶r'}(S1, M)[...] ↦ fcom{r⟶r'}(M)[...]
```

formal composition

```
fcom{r⟶r}(M)[...] ↦ M
```

```
r≠r'   r_i=r'_i (the first i)
------------------------------------------------
fcom{r⟶r'}(M)[..., r_i=r'_i⟶y.N_i, ...] ↦ N_i<r'/y>
```

```
r≠r'   r_i≠r'_i for all i
----------------------------
fcom{r⟶r'}(M)[...] val
```

Kan Circle

S1elim needs to handle **fcom**

Kan Circle

S1elim needs to handle **fcom**

```
r!=r'   r_i!=r'_i
——————————————————————————————————————
S1elim(a.A, fcom{r→r'}(M)[...], B, x.L)
  ↦ com{r→r'}(y.A[fcom{r→y}(...).../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,                |-------->  L<x/y>
 loop{x}, B, y.L)                        |
                                         |
       |  <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>
        v                                v

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 and POPL papers)

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

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

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

(see our arXiv and POPL papers)

Variables

In Nuprl and friends

variables range over closed terms

In Coq, Agda, and friends

variables are indeterminate

Variables

In Nuprl and friends

variables range over closed terms

In Coq, Agda, and friends

variables are indeterminate

This work

exp variables range over closed terms
dim variables are indeterminate

Our arXiv Papers

Part1: stability and Kan

Part2: dependent types

Part3: univalence and equality

Part4: cubical inductive types

RedPRL

a proof assistant based on the new type theory

http://redprl.org

Conclusion

We extended Nuprl semantics by cubical Kan structures which justify key features of HoTT

We also built RedPRL as a prototype