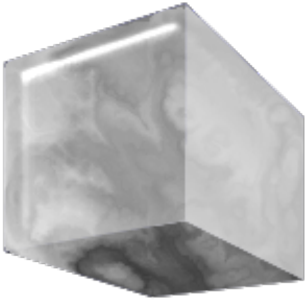
2018.02.07 Cornell

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

1. computational: canonicity by definition
2. key features from homotopy type theory (HoTT)
3. exact equality types

Cubical & Computational

Features

1. computational: canonicity by definition
2. key features from homotopy type theory (HoTT)
3. exact equality types

Advantages for computer science:

1. Equational reasoning closer to standard mathematics
2. Equivalent (good) types share the same properties
3. Quotients and other types built with relations
4. Openness to new constructs

Computational Types

$\boxed{\text{programs/ realizers}}$

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
```

$A \doteq B$ type

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

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 $A' \approx B'$

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

A Minimum Example

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

types: {bool} with syntactic equality

bool: {true, false} with syntactic equality

$$M \doteq N \in A$$

$A \doteq A$ 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 $M' \approx_{A'} N'$

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

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

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 (generally)
cannot be inductively analyzed

Variables

In Nuprl and friends

variables range over closed terms

In Coq, Agda, and friends

variables (generally)
cannot be inductively analyzed

---

closed reduction ⇔ vars over closed terms
open reduction ⇔ vars indeterminate

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

Openness

Openness

Open to new constructs

Openness

Open to new constructs

Open to new theories
for the same language

Openness

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

Openness

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

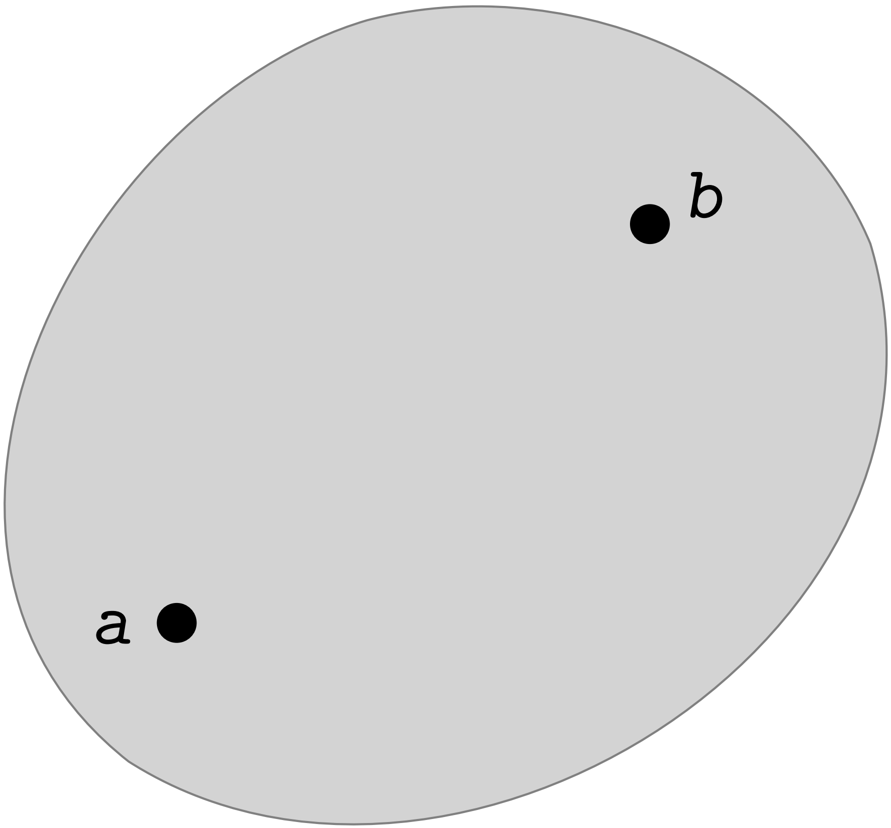
Canonicity always holds

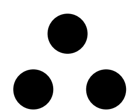
Homotopy Type Theory

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

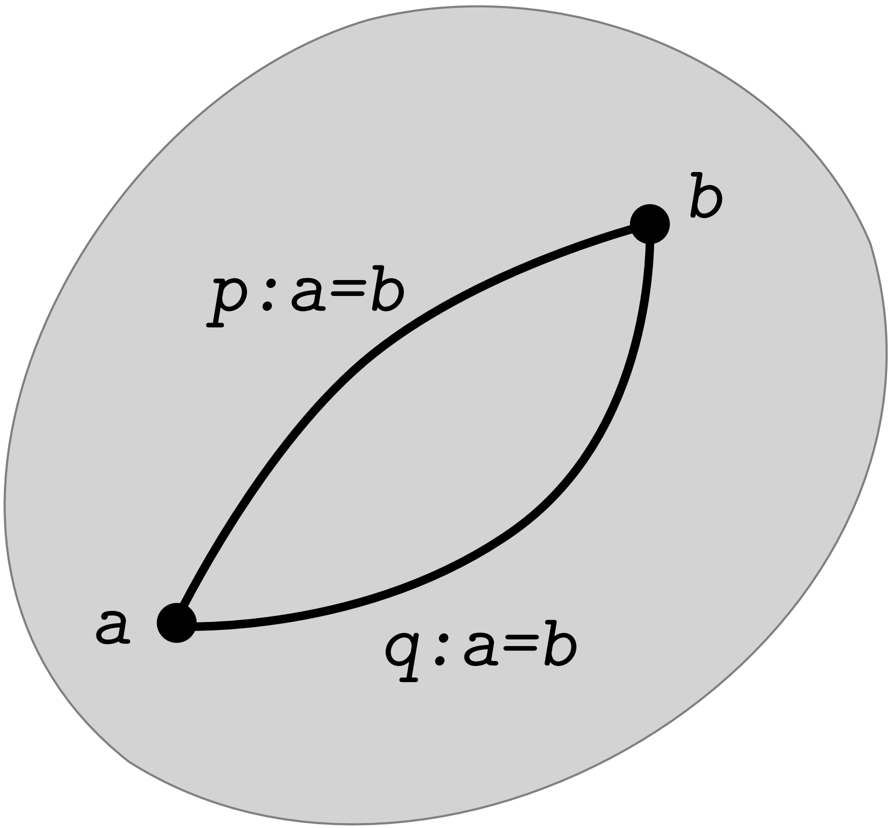
A	Type	Space
$a : A$	Element	Point
$f : A \to B$	Function	Continuous Mapping
$C : A \to Type$	Dependent Type	Fibration
$a =_A b$	Identification	Path

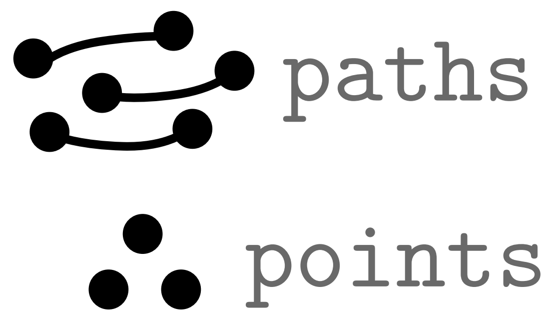
Homotopy Type Theory



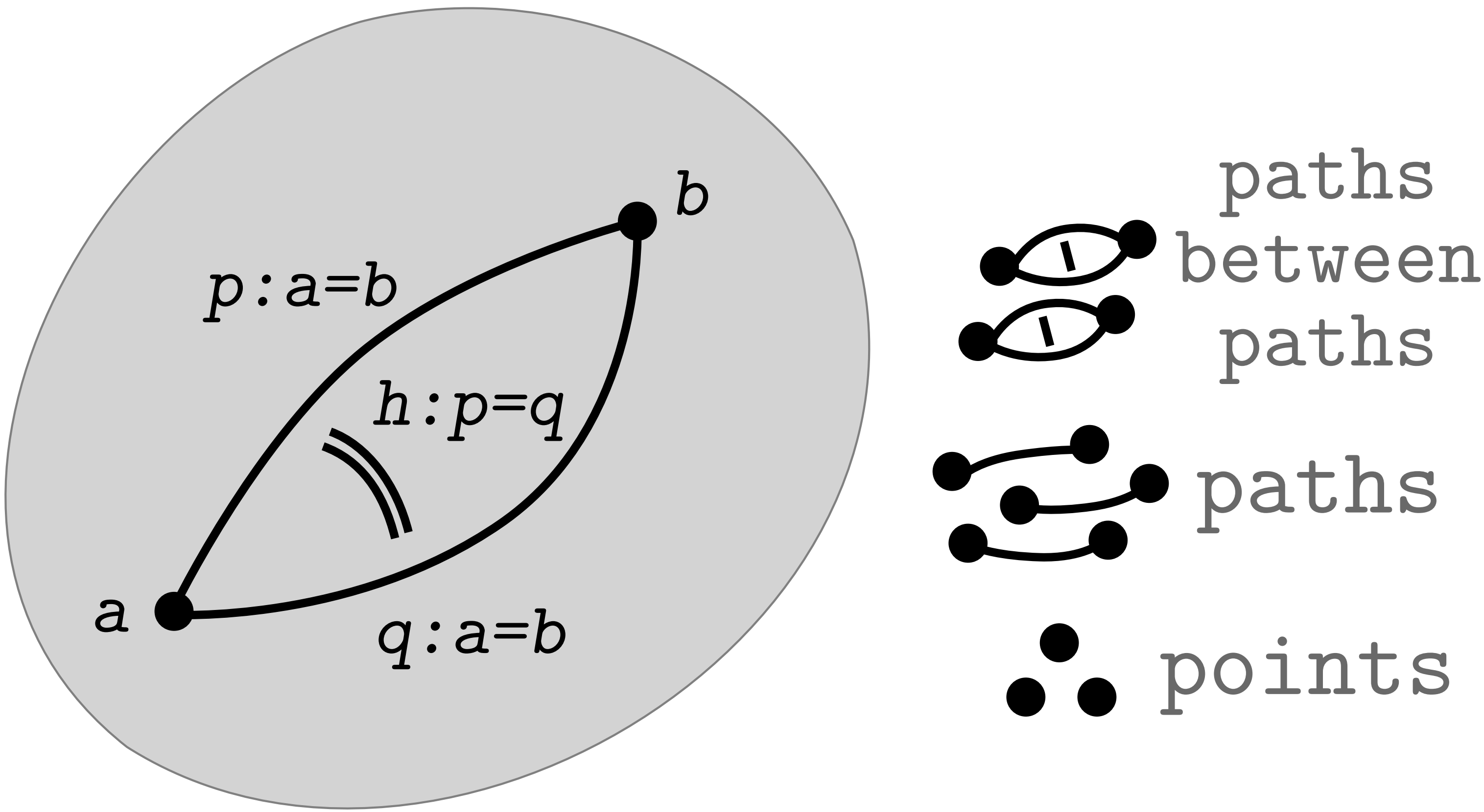


Homotopy Type Theory

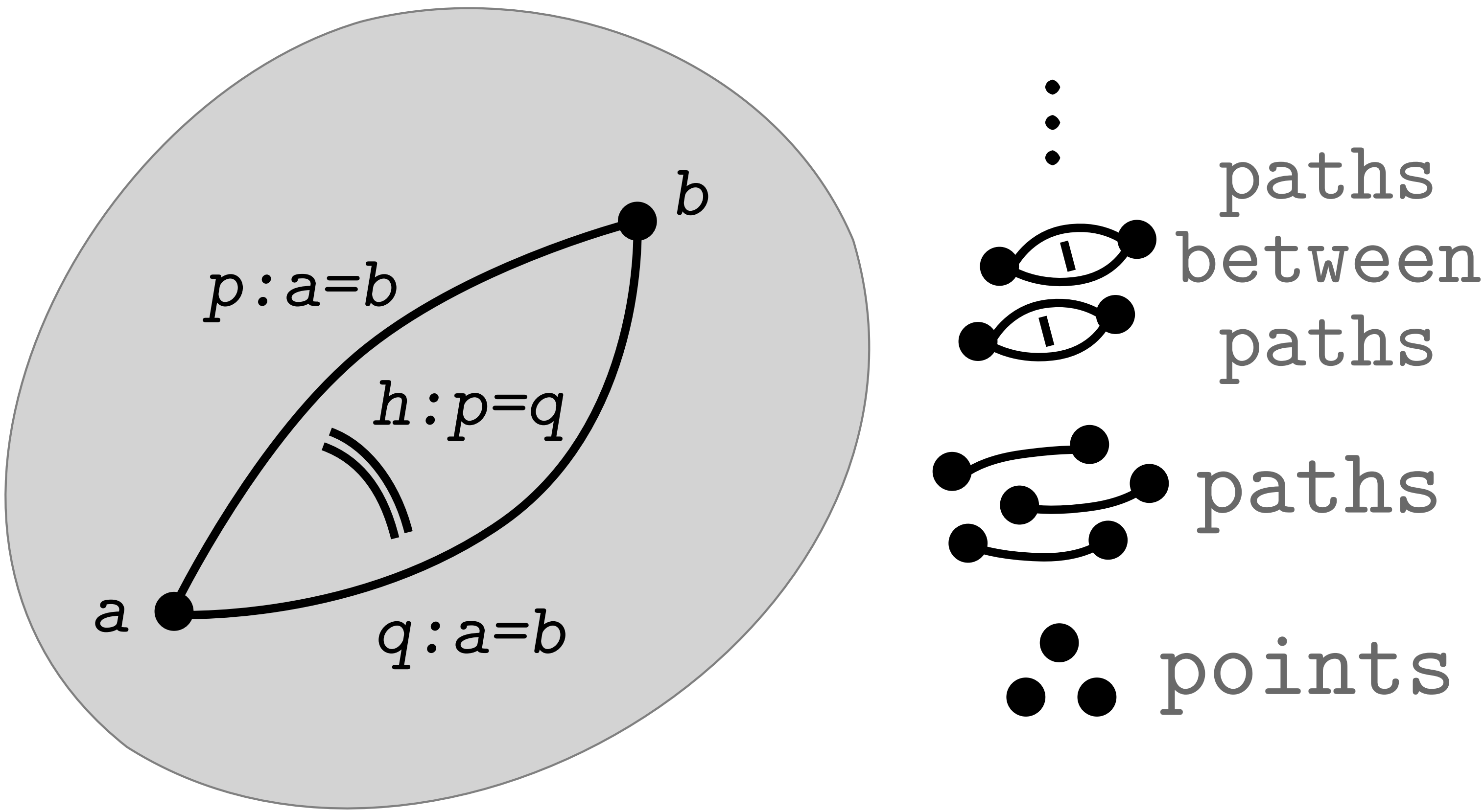




Homotopy Type Theory



Homotopy Type Theory



Homotopy Type Theory

Tons of results in homotopy theory are mechanized through this.

In some case new proofs were discovered and inspired new results.
[Anel, Biedermann, Finster, Joyal]

Also lots of works in category theory and other fields.

Key Features of HoTT

0. Based on intensional type theory
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

0. Based on intensional type theory

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: 2&3 give new identifications

The Poor J Eliminator

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

eliminator for identifications
can only handle reflexivity

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

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

The Poor J Eliminator

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

eliminator for identifications
can only handle reflexivity

$coe(p{:}A{=}B, a{:}A){:}B$

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

Solution

motive C handles paths itself

The Happy J Eliminator

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

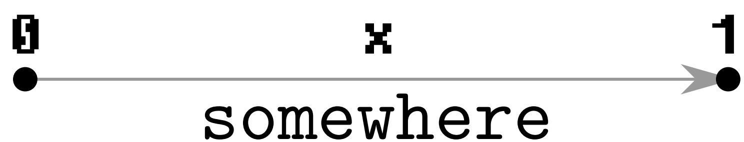
Cubical Programming

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

Cubical Programming

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

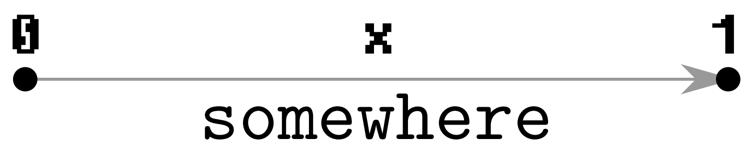
dimension variables (x)
should not be inductively analyzable



Cubical Programming

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

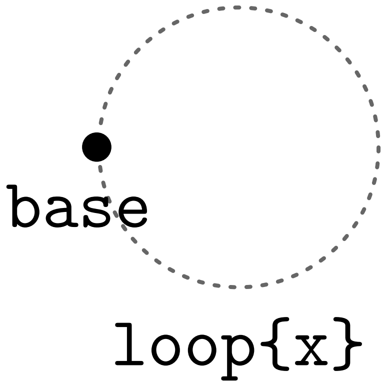
dimension variables (x)
should not be inductively analyzable



* new reduction *

closed in expression variables
open in dimension variables

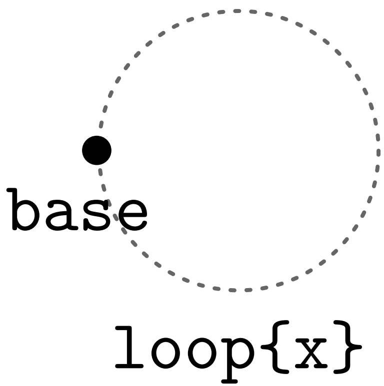
Circle



Circle

dim expr

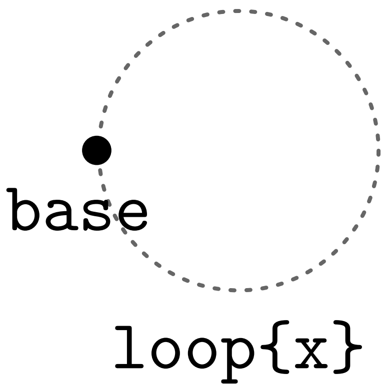
```
M := S1 | base | loop{r}
   | S1elim(a.M, M, M, x.M) | ...
```



Circle

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

dim
expr

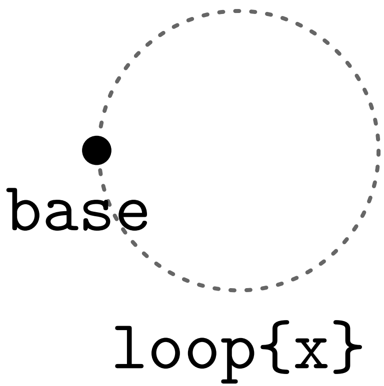


S1 val

Circle

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

dim expr

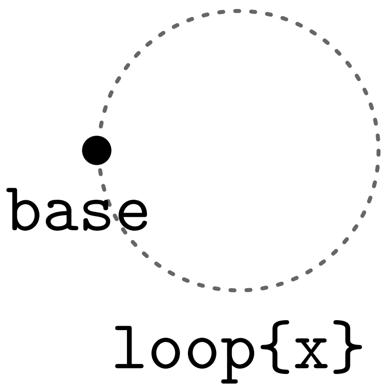


`base val`

`S1 val`

Circle

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



S1 val

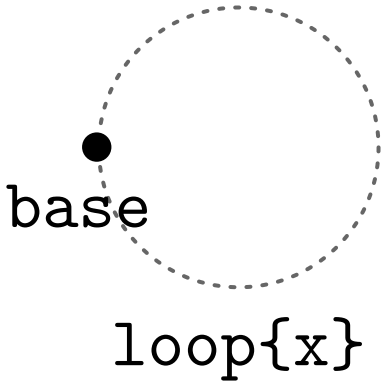
base val

loop{x} val

loop{0} ↦ base

loop{1} ↦ base

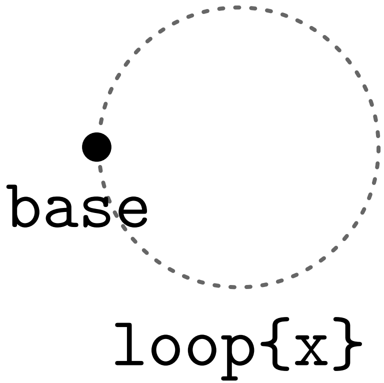
Circle




```
S1 val
```

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

Circle

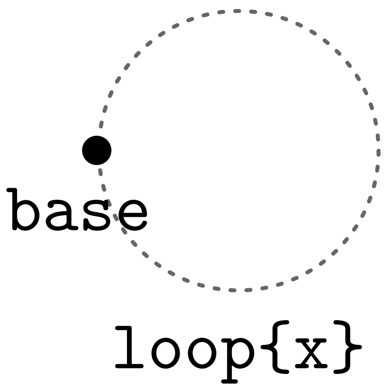


`S1 val`

```
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: Coercion

Kan: Coercion

Kan: Coercion

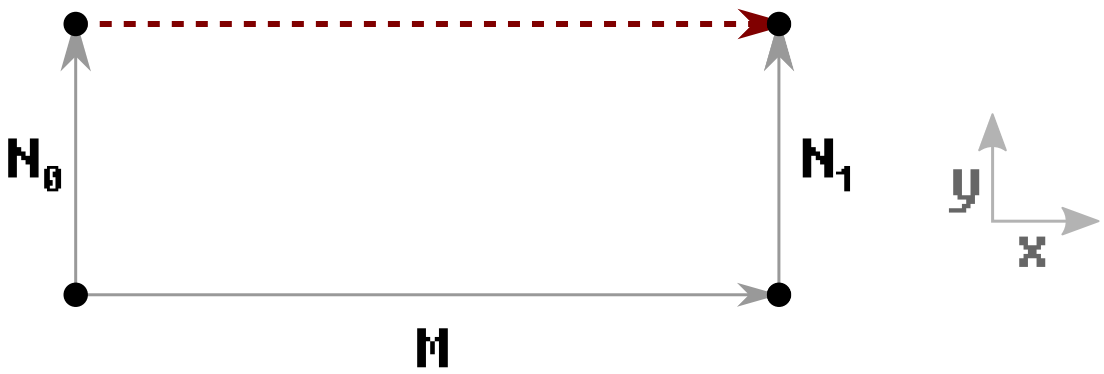
A<0/x> → A<1/x>

M → coe{0⟶1}(x.A,M)

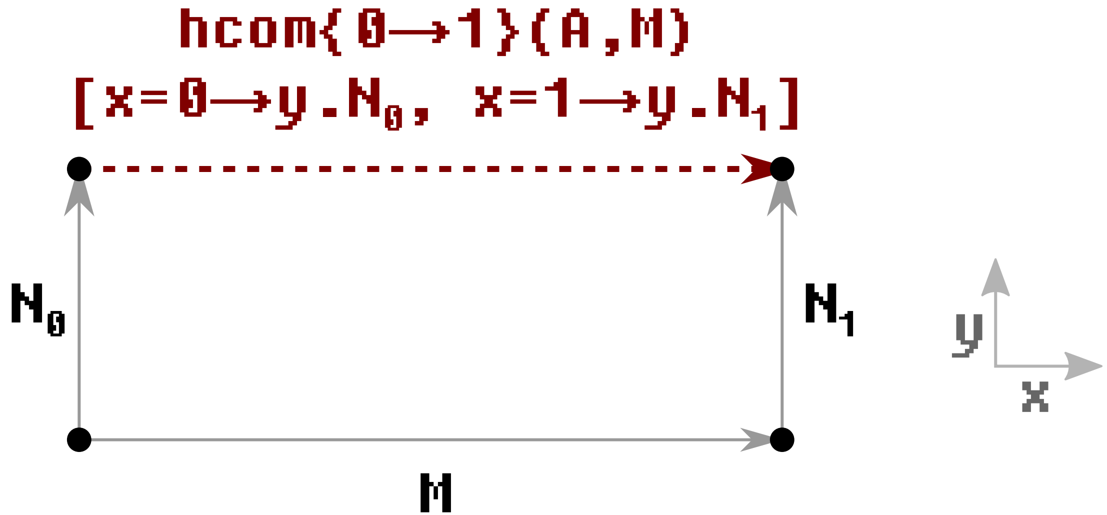
coe{r⟶r'}(x.A, M) ∈ A<r'/x>

M ∈ A<r/x>

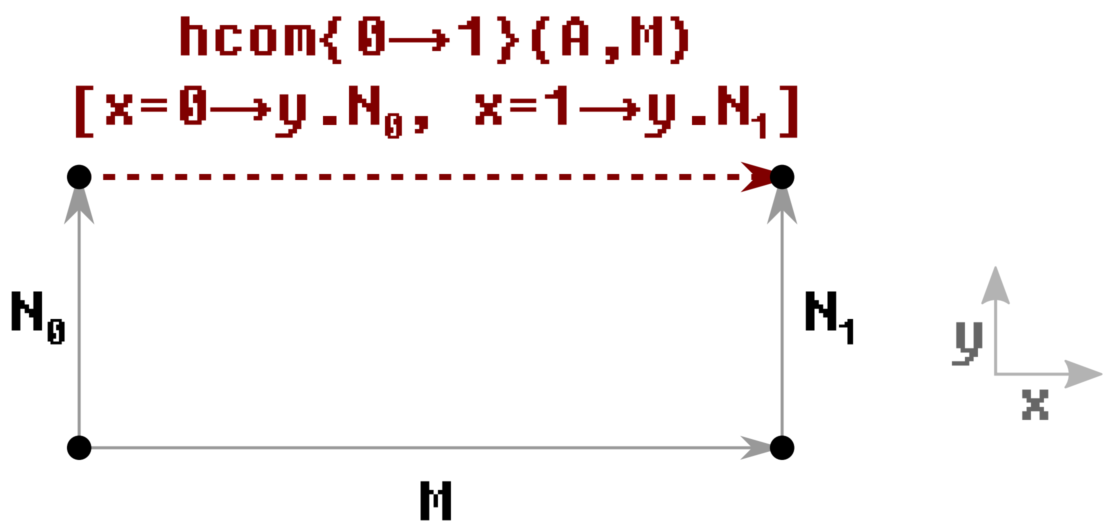
Kan: Homogeneous Comp.



Kan: Homogeneous Comp.



Kan: Homogeneous Comp.




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

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
```

$$\frac{r \neq r' \quad r_i = r'_i \text{ (the first i)}}{\texttt{fcom\{r→r'\}(M)[..., } r_i = r'_i \texttt{→y.}N_i\texttt{, ...]} \mapsto N_i\langle r'/y\rangle}$$

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!

$$
\begin{tikzcd}
\texttt{S1elim(a.A, loop\{x\}, B, y.L)} \arrow[r, mapsto, dashed] & \texttt{L<x/y>} \arrow[d, dashed, "\texttt{<0/x>}"] \\
& \texttt{L<0/y>}
\end{tikzcd}
$$

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>

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 terms

Cubical Type Theory

stability: consider every substitution

Cubical Type Theory

stability: consider every substitution

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

dim context

under any dim substitution ψ...
Aψ and Bψ stably* eval to A' and B' which 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

under any dim substitution ψ...
Aψ and Bψ stably* eval to A' and B' which stably* recognize the same stable* values and have stably* equal Kan structures

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

A ≐ A type, A⇓A',
M and N stably* eval to M' and N',
A' stably* views N' and M' as the same value

(see our arXiv and POPL papers)

Our arXiv Papers

Part1: stability

Part2: dependent types

Part3: univalence and equality

Part4: cubical inductive types

RedPRL

a proof assistant based
on the new type theory

still nascent, changing everyday

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