Computational Higher-dimensional Type Theory & RedPRL

2017.07.04 @ INI

Carlo Angiuli
Evan Cavallo
Favonia
Bob Harper
Jon Sterling
Todd Wilson

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

Computational Types

We use an un(i)typed lambda calculus as in Nuprl.

Computational Types

We use an un(i)typed lambda calculus as in Nuprl.

```
A ≐ B type => A evals to A', B evals to B',
              A' and B' recognize the same values
```

Computational Types

We use an un(i)typed lambda calculus as in Nuprl.

$A \doteq B$ type => A evals to A', B evals to B',
A' and B' recognize the same values

$M \doteq N \in A$ => $A \doteq A$ type, A evals to A', M to M',
N to N', A' views N' and M' as
the same value

Computational Types

We use an un(i)typed lambda calculus as in Nuprl.

```
A ≐ B type => A evals to A', B evals to B',
              A' and B' recognize the same values

M ≐ N ∈ A => A ≐ A type, A evals to A', M to M',
             N to N', A' views N' and M' as
             the same value
```

Ex: "bool type" because "bool" evals to "bool"
& exactly "true" and "false" are in "bool"

Computational Types

We use an un(i)typed lambda calculus as in Nuprl.

A ≐ B type => A evals to A', B evals to B',
A' and B' recognize the same values

M ≐ N ∈ A => A ≐ A type, A evals to A', M to M',
N to N', A' views N' and M' as
the same value

Ex: "bool type" because "bool" evals to "bool"
& exactly "true" and "false" are in "bool"

Ex: "if(true;false,42) ∈ bool" because
"if(true;false,42)" evals to "false"
and "false" is in "bool".

Go Higher-Dimensional

The Book HoTT and the CCHM system have higher-dim. interpretations

Can we do the same?

Go Higher-Dimensional

The Book HoTT and the CCHM system
have higher-dim. interpretations

Can we do the same?

realizers/programs
higher-dim. (dims & Kan)
interesting spaces
universes + univalence

Potential Benefits

Potential Benefits

1. $\doteq$ is closer to equational reasoning in standard math

   functional extensionality
   full universal properties
   (ex: eta for natural numbers)
   ...

Potential Benefits

1. $\doteq$ is closer to equational reasoning in standard math

   functional extensionality
   full universal properties
   (ex: eta for natural numbers)
   ...

2. adding more theorems cannot break computation

   realizers & proof theory separated
   perfect for programming

Clarification #0

computational type theory

IS NOT

"extensional type theory"

Clarification #0

computational type theory
IS NOT
"extensional type theory"

Realizers can be
a model of "ETT", **a model of** "ITT",
or potentially **a model of** HoTT.

You can collect your favorite **theorems**
as rules in your favorite theory

Clarification #1

computational type theory

IS NOT

operational sem. + canonicity

Clarification #1

computational type theory

IS NOT

operational sem. + canonicity

most formal type theories (defined by rules)
want decidable type-checking

Clarification #1

computational type theory

IS NOT

operational sem. + canonicity

most formal type theories (defined by rules)
want **decidable type-checking**

$$f \doteq g \in \texttt{nat -> nat}$$

not decidable in general
but can be proved by **induction** in CTT
(as proving a **theorem** about realizers)

Clarification #2

most formal type theories
want decidable type-checking

=> undecidable rules were ruled out

computational type theory
is fine with "undecidable rules"

=> ask users for guidance in practice
(can be interactive & tactic-based)

Cubical Realizers

higher-dimensional programming

[Angiuli, Harper & Wilson]

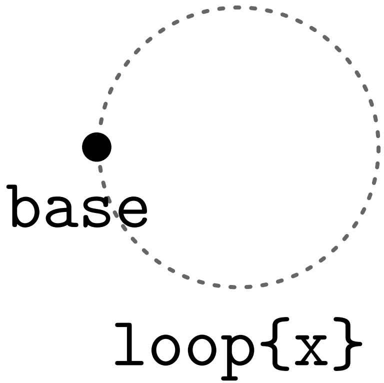
Cubical Realizers

higher-dimensional programming

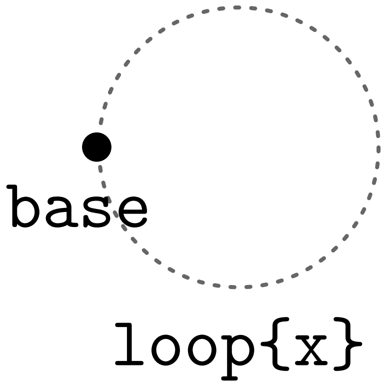
[Angiuli, Harper & Wilson]

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

Circle

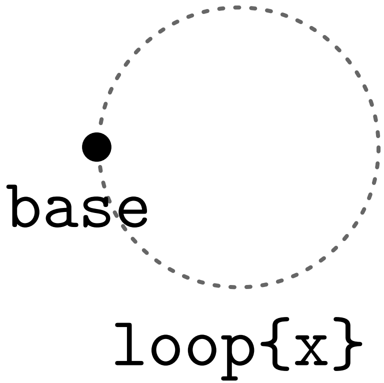


Circle



S1 value

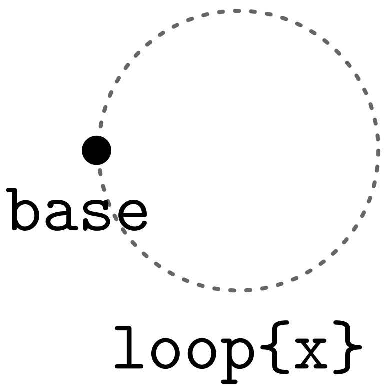
Circle



base value

S1 value

Circle



S1 value

base value

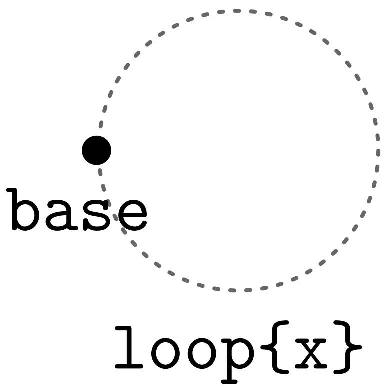
loop{r} → dim expr

loop{x} value

loop{0} ↦ base

loop{1} ↦ base

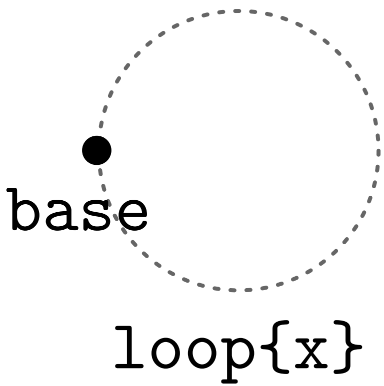
Circle



S1 value

```
M ↦ M'
---------------------
S1elim(a.A, M, B, u.L)
 ↦ S1elim(a.A, M', B, w.L)
```

Circle

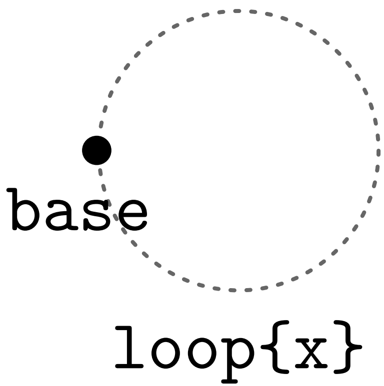


S1 value

```
M ↦ M'
----------------------
S1elim(a.A, M, B, u.L)
  ↦ S1elim(a.A, M', B, w.L)
```

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

Circle



S1 value

```
M ↦ M'
---------------------
S1elim(a.A, M, B, u.L)
 ↦ S1elim(a.A, M', B, w.L)
```

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

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

Kan: Coercions

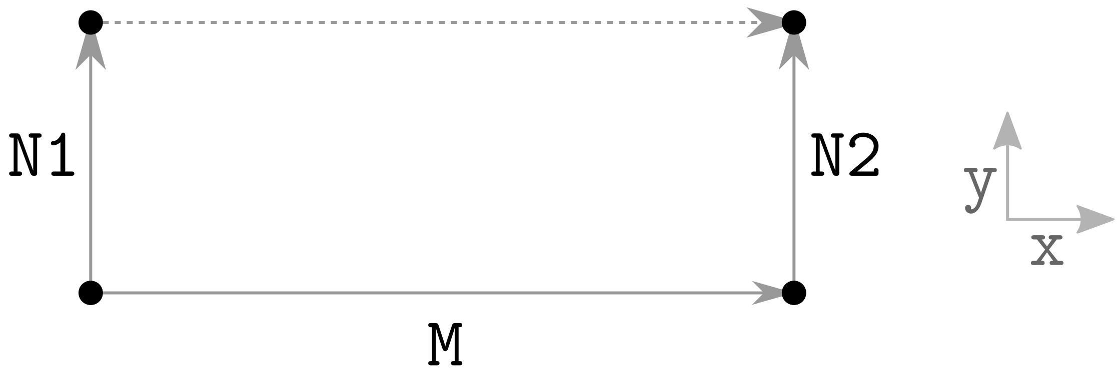
Kan: Coercions

coe{r->r'}(x.A, M) ∈ A<r'/x>

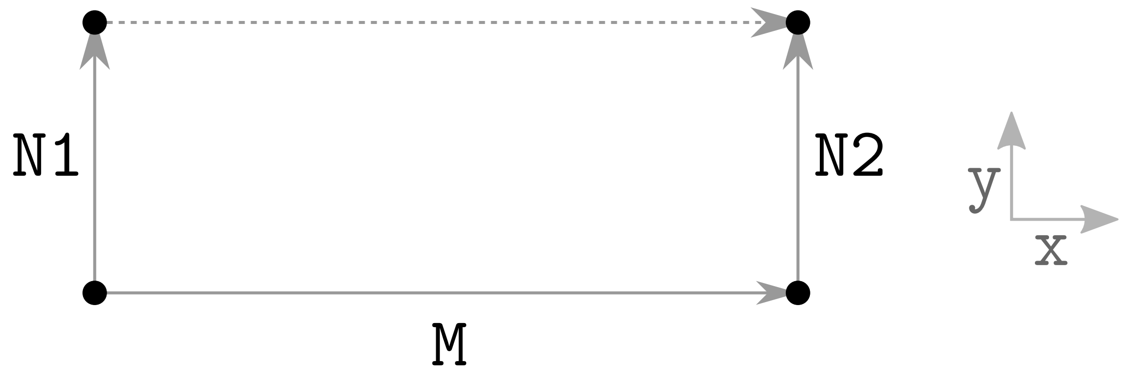
⋔

A<r/x>

Kan: Homogeneous Comp.



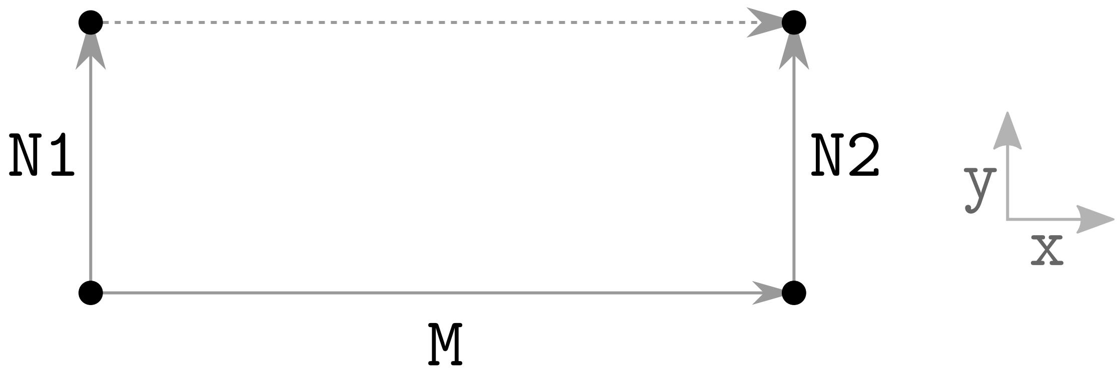
Kan: Homogeneous Comp.




```
hcom{r->r'}(A, M)
  [x -> (y.N1, y.N2)]
```

adjacency: needs exact equality (≐)

Kan: Homogeneous Comp.




```
hcom{r->r'}(A, M)
   [x -> (y.N1, y.N2)]
```

adjacency: needs exact equality (≐)

note: we forbid empty systems

Kan Circle

```
coe{r->r'}(_.S1, M) ↦ M
```

Kan Circle

```
coe{r->r'}(_.S1, M) ↦ M
```

```
hcom{r->r}(S1, M)... ↦ M
```

Kan Circle

```
coe{r->r'}(_.S1, M) ↦ M
```

```
hcom{r->r}(S1, M)... ↦ M
```

```
r != r'  ri is 0 or 1 (the first)
-----------------------------------
hcom{r->r'}(S1, M)...[ri -> (y.N_0, y.N_1)]...
 ↦ N_ri<r'/y>
```

Kan Circle

```
coe{r->r'}(_.S1, M) ↦ M
```

```
hcom{r->r}(S1, M)... ↦ M
```

```
r != r'  ri is 0 or 1 (the first)
---------------------------------
hcom{r->r'}(S1, M)...[ri -> (y.N_0, y.N_1)]...
 ↦ N_ri<r'/y>
```

```
r != r'  ri != 0  ri != 1
--------------------------
hcom{r->r'}(S1, M)... value
```

formal composition

Kan Circle

S1elim needs to handle new values

Kan Circle

S1elim needs to handle new values

```
r != r'  ri != 0  ri != 1
-----------------------------------
S1elim(a.A, hcom{r->r'}(S1, M)..., B, w.L)
 ↦ com{r->r'}(z.A[hcom{r->z}(...).../a],
      S1elim(M, B, w.L))...
```

(using the definable heterogeneous comp.)

Cubical Stability

Dimension substs. do not commute with evaluation!

Cubical Stability

Dimension substs. do not commute with evaluation!

```
S1elim(a.A,
 loop{x}, B, w.L)  |----------> L<x/w>
                                  |
                                  |  <0/x>
                                  v

                                L<0/w>
```

Cubical Stability

Dimension substs. do not commute with evaluation!

```
S1elim(a.A,
 loop{x}, B, w.L)   |----------> L<x/w>
                                    ¦
        ¦ <0/x>                     ¦ <0/x>
        v                           v

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

Cubical Stability

Dimension substs. do not commute with evaluation!

```
S1elim(a.A,
 loop{x}, B, w.L)  |----------> L<x/w>
                                  ¦
      ¦ <0/x>                     ¦ <0/x>
      v                           v
S1elim(a.A,
 base, B, w.L)  |--> B  <=??=> L<0/w>
```

Judgments for cubically stable types, memberships, values, etc

Cubical Type Theory

stability: consider every substitution

Cubical Type Theory

stability: consider every substitution

A ≐ B type =>
under any further substitution ψ...
Aψ and Bψ stably* eval to A' and B',
stably* recognizing the same stable* values
and having stably* equal Kan structures

(*see our arXiv & POPL papers for details)

Cubical Type Theory

stability: consider every substitution

A ≐ B type =>
under any further substitution ψ...
Aψ and Bψ stably* eval to A' and B',
stably* recognizing the same stable* values
and having stably* equal Kan structures

M ≐ N ∈ A =>
A ≐ A type and A evals to A',
M and N stably* eval to M' and N',
A' stably* views N' and M' as the same value

(*see our arXiv & POPL papers for details)

Comparison with CCHM

based on realizability

(closer to standard math equality)

no connections

no empty systems

(handled by coe)

Current Progress

Done:

dependent functions
dependent pairs
strict bools
"weak" bools
circle
...
univalence for
strict isomorphisms

Current Progress

Done:

dependent functions
dependent pairs
strict bools
"weak" bools
circle
...
univalence for
strict isomorphisms

Still working:

Univalence & Kan universes

RedPRL

a proof assistant based on cubical realizability

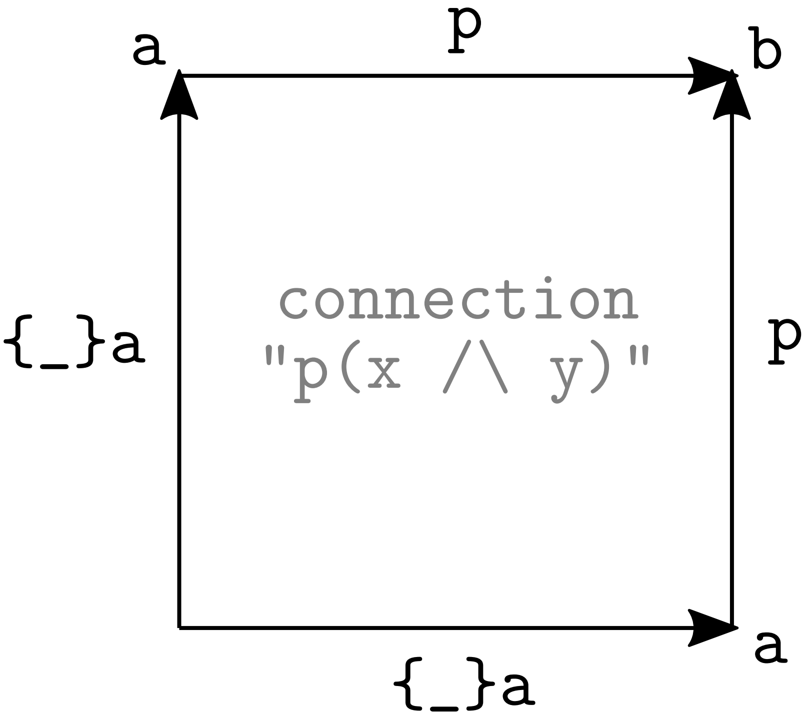
incorporates recent advances such as dependent subgoals

(inspired by Spiwack's work)

still nascent, changing everyday

https://github.com/RedPRL/sml-redprl

RedPRL Example



Conclusion

Cubical extension of Nuprl realizers

Canonicity on points (diff. from CCHM)

Still working on Kan universes

RedPRL (to be) an implementation

Conclusion

Cubical extension of Nuprl realizers

Canonicity on points (diff. from CCHM)

Still working on Kan universes

RedPRL (to be) an implementation

Acknowledgements

Our work is strongly influenced by the BCH and CCHM papers, work by Licata and Brunerie, and many inspiring discussions within the HoTT community.