Computational Semantics for Dependent Type Theories

Samson Abramsky

Department of Computer Science, University of Oxford

Motivation

Motivation

- HoTT is hot!!

Motivation

- HoTT is hot!!
  The HoTT book, IAS year, Awodey \$7.5M grant, ...

Motivation

- HoTT is hot!!
  The HoTT book, IAS year, Awodey $7.5M grant, ...

- HoTT is essentially (intensional) Martin-Lof type theory with one new axiom. Much of the current energy and enthusiasm stems from the discovery of a new semantics in homotopy types.

Motivation

- HoTT is hot!!
  The HoTT book, IAS year, Awodey \$7.5M grant, ...

- HoTT is essentially (intensional) Martin-Lof type theory with one new axiom. Much of the current energy and enthusiasm stems from the discovery of a new semantics in homotopy types.

- Question 1:

Can we find a natural intensional computational semantics for HoTT/DTT?

Motivation

- HoTT is hot!!
  The HoTT book, IAS year, Awodey \$7.5M grant, ...

- HoTT is essentially (intensional) Martin-Lof type theory with one new axiom. Much of the current energy and enthusiasm stems from the discovery of a new semantics in homotopy types.
- Question 1:

  Can we find a natural intensional computational semantics for HoTT/DTT?

- Question 2:

  Can we combine linear and dependent types?

Sketch of a programme

Sketch of a programme

Attempt to find a game semantics for HoTT (or DTT generally).

Sketch of a programme

Attempt to find a game semantics for HoTT (or DTT generally).

Joint work with Radha Jagadeesan, Kohei Kishida, Matthijs Vákár, Norihiro Yamada.

Sketch of a programme

Attempt to find a game semantics for HoTT (or DTT generally).

Joint work with Radha Jagadeesan, Kohei Kishida, Matthijs Vákár, Norihiro Yamada.




Sketch of a programme

Attempt to find a game semantics for HoTT (or DTT generally).

Joint work with Radha Jagadeesan, Kohei Kishida, Matthijs Vákár, Norihiro Yamada.

   

Still in a preliminary stage.

Sketch of a programme

Attempt to find a game semantics for HoTT (or DTT generally).

Joint work with Radha Jagadeesan, Kohei Kishida, Matthijs Vákár, Norihiro Yamada.




Still in a preliminary stage. What we have so far:

Sketch of a programme

Attempt to find a game semantics for HoTT (or DTT generally).

Joint work with Radha Jagadeesan, Kohei Kishida, Matthijs Vákár, Norihiro Yamada.




Still in a preliminary stage. What we have so far:

- A restricted form of linear DTT (MV, `http://arxiv.org/abs/1405.0033`)

Sketch of a programme

Attempt to find a game semantics for HoTT (or DTT generally).

Joint work with Radha Jagadeesan, Kohei Kishida, Matthijs Vákár, Norihiro Yamada.

   

Still in a preliminary stage. What we have so far:

- A restricted form of linear DTT (MV, `http://arxiv.org/abs/1405.0033`)
- A coherence space semantics (SA and MV, paper soon)

Sketch of a programme

Attempt to find a game semantics for HoTT (or DTT generally).

Joint work with Radha Jagadeesan, Kohei Kishida, Matthijs Vákár, Norihiro Yamada.

   

Still in a preliminary stage. What we have so far:

- A restricted form of linear DTT (MV, `http://arxiv.org/abs/1405.0033`)
- A coherence space semantics (SA and MV, paper soon)
- Progress towards a game semantics (RJ)

The Plan

The Plan

Give a *relaxed* talk ☺

The Plan

Give a *relaxed* talk ☺

Empasise **ideas** rather than technical details.

The Plan

Give a *relaxed* talk ☺

Empasise **ideas** rather than technical details.

Firstly:

$$\boxed{\text{Why Games? Why Coherence Spaces?}}$$

The Plan

Give a *relaxed* talk ☺

Empasise **ideas** rather than technical details.

Firstly:

$\boxed{\text{Why Games? Why Coherence Spaces?}}$

- Give a "native" intensional semantics coming from the computational side

The Plan

Give a *relaxed* talk ☺

Empasise **ideas** rather than technical details.

Firstly:

$\boxed{\text{Why Games? Why Coherence Spaces?}}$

- Give a “native” intensional semantics coming from the computational side
- Naturally linear, so points to the way to linear forms of dependent type theory

The Plan

Give a *relaxed* talk ☺

Empasise **ideas** rather than technical details.

Firstly:

$$\boxed{\text{Why Games? Why Coherence Spaces?}}$$

- Give a “native” intensional semantics coming from the computational side
- Naturally linear, so points to the way to linear forms of dependent type theory
- Computational/informatic content:

  Information, time *vs.* Geometry, space

The Plan

Give a *relaxed* talk ☺

Empasise **ideas** rather than technical details.

Firstly:

$$\boxed{\text{Why Games? Why Coherence Spaces?}}$$

- Give a "native" intensional semantics coming from the computational side
- Naturally linear, so points to the way to linear forms of dependent type theory
- Computational/informatic content:

  Information, time *vs.* Geometry, space
- Can exploit techniques for building universal domains, solving "domain equations" (*i.e.* reflexive type equations).

The ideas **behind** semantics

The ideas **behind** semantics

Not just a tool for metamathematics ...

The ideas **behind** semantics

Not just a tool for metamathematics . . .

Examples:

The ideas **behind** semantics

Not just a tool for metamathematics ...

Examples:

- Homotopy semantics for DTT gve rise to HoTT

The ideas **behind** semantics

Not just a tool for metamathematics ...

Examples:

- Homotopy semantics for DTT gve rise to HoTT
- Coherence space semantics gave rise to Linear Logic

The ideas **behind** semantics

Not just a tool for metamathematics . . .

Examples:

- Homotopy semantics for DTT gve rise to HoTT
- Coherence space semantics gave rise to Linear Logic

Splitting the atom of computation, twice: a voyage through semantics

The ideas **behind** semantics

Not just a tool for metamathematics ...

Examples:

- Homotopy semantics for DTT gve rise to HoTT
- Coherence space semantics gave rise to Linear Logic

Splitting the atom of computation, twice: a voyage through semantics

A basic setting: higher-order (partial) functions

Type 0: $\mathbb{N}$
Type 1: $\mathbb{N} \rightharpoonup \mathbb{N}$
Type 2: $[\mathbb{N} \rightharpoonup \mathbb{N}] \rightharpoonup \mathbb{N}$

$\vdots$

The ideas **behind** semantics

Not just a tool for metamathematics ...

Examples:

- Homotopy semantics for DTT gve rise to HoTT
- Coherence space semantics gave rise to Linear Logic

Splitting the atom of computation, twice: a voyage through semantics

A basic setting: higher-order (partial) functions

Type 0: $\mathbb{N}$
Type 1: $\mathbb{N} \rightharpoonup \mathbb{N}$
Type 2: $[\mathbb{N} \rightharpoonup \mathbb{N}] \rightharpoonup \mathbb{N}$
$\vdots$

Functionals are not so unfamiliar: e.g. the quantifiers!

$$\forall, \exists : [\mathbb{N} \to \mathsf{B}] \to \mathsf{B}$$

Splitting the Atom I

Splitting the Atom I

Set-theoretically, higher-order functions are monsters.

Splitting the Atom I

Set-theoretically, higher-order functions are monsters.

E.g. $F : [\mathbb{N} \rightharpoonup \mathbb{N}] \rightharpoonup \mathbb{N}$ is defined by its graph:

$$\mathrm{graph}(F) \; = \; \{(f, n) \mid f \in \mathrm{dom}(F)\}$$

Splitting the Atom I

Set-theoretically, higher-order functions are monsters.

E.g. $F : [\mathbb{N} \rightharpoonup \mathbb{N}] \rightharpoonup \mathbb{N}$ is defined by its graph:

$$\mathrm{graph}(F) \;=\; \{(f, n) \mid f \in \mathrm{dom}(F)\}$$

Each ordered pair in the graph may contain an **infinite amount of information**.

Splitting the Atom I

Set-theoretically, higher-order functions are monsters.

E.g. $F : [\mathbb{N} \rightharpoonup \mathbb{N}] \rightharpoonup \mathbb{N}$ is defined by its graph:

$$\mathrm{graph}(F) \; = \; \{(f, n) \mid f \in \mathrm{dom}(F)\}$$

Each ordered pair in the graph may contain an **infinite amount of information**.

The first splitting: look at **finite pieces of information**.

Splitting the Atom I

Set-theoretically, higher-order functions are monsters.

E.g. $F : [\mathbb{N} \rightharpoonup \mathbb{N}] \rightharpoonup \mathbb{N}$ is defined by its graph:

$$\mathrm{graph}(F) \ = \ \{(f, n) \mid f \in \mathrm{dom}(F)\}$$

Each ordered pair in the graph may contain an **infinite amount of information**.

The first splitting: look at **finite pieces of information**.

Information tokens:

$$t \ := \ (\{n_1, m_1), \ldots, (n_k, m_k)\}, m)$$

$$F \models t \ \equiv \ \forall f. [\bigwedge_{i=1}^{k} f(n_i) = m_i] \rightarrow F(f) = m$$

F **should be determined** by the finite pieces of information it satisfies.

Consequences

Consequences

This idea induces a topology (the Scott topology) (Kleene, Kreisel, Nerode, Platek, . . .)

Consequences

This idea induces a topology (the Scott topology) (Kleene, Kreisel, Nerode, Platek, . . .)

This leads to **domain theory**

Consequences

This idea induces a topology (the Scott topology) (Kleene, Kreisel, Nerode, Platek, . . .)

This leads to **domain theory**

Emphasis on finite pieces of information leads to Scott domains and their representation in terms of **information systems**.

Consequences

This idea induces a topology (the Scott topology) (Kleene, Kreisel, Nerode, Platek, ...)

This leads to **domain theory**

Emphasis on finite pieces of information leads to Scott domains and their representation in terms of **information systems**.

Everything is represented in terms of tokens.

The coherence space revolution

The coherence space revolution

Gave birth to Linear Logic (Girard 1987).

The coherence space revolution

Gave birth to Linear Logic (Girard 1987).

A radically simple form of token-based finite information semantics.

The coherence space revolution

Gave birth to Linear Logic (Girard 1987).

A radically simple form of token-based finite information semantics.

So simple that:

The coherence space revolution

Gave birth to Linear Logic (Girard 1987).

A radically simple form of token-based finite information semantics.

So simple that:

- It makes visible the linear decompositions of intuitionistic types

The coherence space revolution

Gave birth to Linear Logic (Girard 1987).

A radically simple form of token-based finite information semantics.

So simple that:

- It makes visible the linear decompositions of intuitionistic types
- It allows classical dualities at the linear level

The coherence space revolution

Gave birth to Linear Logic (Girard 1987).

A radically simple form of token-based finite information semantics.

So simple that:

- It makes visible the linear decompositions of intuitionistic types
- It allows classical dualities at the linear level

On the other hand, not closed under lifting!

Review of Coherence Spaces

Review of Coherence Spaces

The underlying structure of a coherence space is just an undirected reflexive graph. The coherence space is then the set of cliques of the graph, ordered by set inclusion.

Review of Coherence Spaces

The underlying structure of a coherence space is just an undirected reflexive graph. The coherence space is then the set of cliques of the graph, ordered by set inclusion.

The novelty of coherence spaces lies in the constructions on reflexive graphs, which lead to a categorical model of full Classical Linear Logic — indeed, this model gave rise to Linear Logic.

Review of Coherence Spaces

The underlying structure of a coherence space is just an undirected reflexive graph. The coherence space is then the set of cliques of the graph, ordered by set inclusion.

The novelty of coherence spaces lies in the constructions on reflexive graphs, which lead to a categorical model of full Classical Linear Logic — indeed, this model gave rise to Linear Logic.

Notation: $(|X|, \frown_X)$. $|X|$ is the set of tokens.

Review of Coherence Spaces

The underlying structure of a coherence space is just an undirected reflexive graph. The coherence space is then the set of cliques of the graph, ordered by set inclusion.

The novelty of coherence spaces lies in the constructions on reflexive graphs, which lead to a categorical model of full Classical Linear Logic — indeed, this model gave rise to Linear Logic.

Notation: $(|X|, \supset\!\!\!\!\smile_X)$. $|X|$ is the set of tokens.

We write $\frown$ for the irreflexive part of $\supset\!\!\!\!\smile$, $\smile$ for the complement of $\supset\!\!\!\!\smile$ (which is irreflexive), and $\asymp$ for the complement of $\frown$.

Linear Type Constructions on Coherence Spaces

The linear type constructions combine set-theoretic operations on the token sets with logical operations on the coherence relations.

Linear Type Constructions on Coherence Spaces

The linear type constructions combine set-theoretic operations on the token sets with logical operations on the coherence relations.

Linear negation

Given a coherence space X, we define $X^{\perp}$ with $|X^{\perp}| = |X|$, and $\frown_{X^{\perp}} = \asymp_{X}$.
Note that $X^{\perp\perp} = X$

Linear Type Constructions on Coherence Spaces

The linear type constructions combine set-theoretic operations on the token sets with logical operations on the coherence relations.

Linear negation
Given a coherence space X, we define $X^{\perp}$ with $|X^{\perp}| = |X|$, and $\smile_{X^{\perp}} = \asymp_X$.
Note that $X^{\perp\perp} = X$

Multiplicatives
Given coherence spaces X, Y:

$$|X \otimes Y| = |X \parr Y| = |X \multimap Y| := |X| \times |Y|.$$

$$\begin{array}{lcl}
(a, b) \smile (c, d) \bmod X \otimes Y & \equiv & a \smile c \bmod X \ \wedge \ b \smile d \bmod Y \\
(a, b) \frown (c, d) \bmod X \parr Y & \equiv & a \frown c \bmod X \ \vee \ b \frown d \bmod Y \\
(a, b) \frown (c, d) \bmod X \multimap Y & \equiv & a \smile c \bmod X \ \Rightarrow \ b \frown d \bmod Y
\end{array}$$

Linear Type Constructions on Coherence Spaces

The linear type constructions combine set-theoretic operations on the token sets with logical operations on the coherence relations.

Linear negation

Given a coherence space X, we define $X^{\perp}$ with $|X^{\perp}| = |X|$, and $\smile_{X^{\perp}} = \asymp_{X}$.
Note that $X^{\perp\perp} = X$

Multiplicatives

Given coherence spaces X, Y:

$$|X \otimes Y| = |X \parr Y| = |X \multimap Y| := |X| \times |Y|.$$

$$\begin{array}{lcl}
(a, b) \smile (c, d) \bmod X \otimes Y & \equiv & a \smile c \bmod X \ \wedge \ b \smile d \bmod Y \\
(a, b) \frown (c, d) \bmod X \parr Y & \equiv & a \frown c \bmod X \ \vee \ b \frown d \bmod Y \\
(a, b) \frown (c, d) \bmod X \multimap Y & \equiv & a \smile c \bmod X \ \Rightarrow \ b \frown d \bmod Y
\end{array}$$

Note that $X \parr Y = (X^{\perp} \otimes Y^{\perp})^{\perp}$, $X \multimap Y = X^{\perp} \parr Y$.

Linear Type Constructions on Coherence Spaces

The linear type constructions combine set-theoretic operations on the token sets with logical operations on the coherence relations.

Linear negation

Given a coherence space X, we define $X^{\perp}$ with $|X^{\perp}| = |X|$, and $\smile_{X^{\perp}} = \asymp_{X}$.
Note that $X^{\perp\perp} = X$

Multiplicatives

Given coherence spaces X, Y:

$$|X \otimes Y| = |X \parr Y| = |X \multimap Y| := |X| \times |Y|.$$

$$\begin{array}{lcl}
(a, b) \smile (c, d) \bmod X \otimes Y & \equiv & a \smile c \bmod X \ \wedge \ b \smile d \bmod Y \\
(a, b) \frown (c, d) \bmod X \parr Y & \equiv & a \frown c \bmod X \ \vee \ b \frown d \bmod Y \\
(a, b) \frown (c, d) \bmod X \multimap Y & \equiv & a \smile c \bmod X \ \Rightarrow \ b \frown d \bmod Y
\end{array}$$

Note that $X \parr Y = (X^{\perp} \otimes Y^{\perp})^{\perp}$, $X \multimap Y = X^{\perp} \parr Y$.

N.B. Interdefinabilities follow from propositional calculus!

Linear Type Constructions ctd

Linear Type Constructions ctd

Additives Given coherence spaces X, Y:

$$|X \oplus Y| = |X \& Y| := |X| + |Y|.$$

We represent the disjoint union $|X| + |Y|$ concretely as $X \times \{0\} \cup Y \times \{1\}$.

$$\begin{array}{lllll} (a,i) \frown (b,j) \mod X \oplus Y & \equiv & i = j & \wedge & a \frown b \mod Z_i \\ (a,i) \frown (b,j) \mod X \& Y & \equiv & i = j & \Rightarrow & a \frown b \mod Z_i \end{array}$$

where $Z_0 = X$ and $Z_1 = Y$.

Linear Type Constructions ctd

Additives Given coherence spaces X, Y:

$$|X \oplus Y| = |X \& Y| := |X| + |Y|.$$

We represent the disjoint union $|X| + |Y|$ concretely as $X \times \{0\} \cup Y \times \{1\}$.

$$\begin{array}{lllll} (a,i) \frown (b,j) \mod X \oplus Y & \equiv & i = j & \wedge & a \frown b \mod Z_i \\ (a,i) \frown (b,j) \mod X \& Y & \equiv & i = j & \Rightarrow & a \frown b \mod Z_i \end{array}$$

where $Z_0 = X$ and $Z_1 = Y$.

Note that $X \& Y = (X^{\perp} \oplus Y^{\perp})^{\perp}$.

Linear Type Constructions ctd

Additives Given coherence spaces X, Y:

$$|X \oplus Y| = |X \& Y| := |X| + |Y|.$$

We represent the disjoint union $|X| + |Y|$ concretely as $X \times \{0\} \cup Y \times \{1\}$.

$$\begin{array}{lllll} (a,i) \smile (b,j) \mod X \oplus Y & \equiv & i = j & \wedge & a \smile b \mod Z_i \\ (a,i) \smile (b,j) \mod X \& Y & \equiv & i = j & \Rightarrow & a \smile b \mod Z_i \end{array}$$

where $Z_0 = X$ and $Z_1 = Y$.

Note that $X \& Y = (X^{\perp} \oplus Y^{\perp})^{\perp}$.

Exponentials Given a coherence space X, we define

$$|!X| = X_{\mathrm{fin}}$$

$$x \smile y \mod !X \equiv x \cup y \in X.$$

Stable functions and traces

Stable functions and traces

We recall that a function $f : X \to Y$ is **stable** if it is monotone, and preserves directed unions and pull-backs.

Stable functions and traces

We recall that a function $f : X \to Y$ is **stable** if it is monotone, and preserves directed unions and pull-backs.

Proposition

A monotone and continuous function $f : X \to Y$ is stable if and only if, whenever $b \in f(x)$ for $x \in X$, there exists a finite clique $s \subseteq x$ such that $b \in f(s)$, and moreover for all $s' \subseteq x$ such that $b \in f(s')$, $s \subseteq s'$.

Stable functions and traces

We recall that a function $f : X \to Y$ is **stable** if it is monotone, and preserves directed unions and pull-backs.

Proposition

A monotone and continuous function $f : X \to Y$ is stable if and only if, whenever $b \in f(x)$ for $x \in X$, there exists a finite clique $s \subseteq x$ such that $b \in f(s)$, and moreover for all $s' \subseteq x$ such that $b \in f(s')$, $s \subseteq s'$.

We define the **trace** of a stable function:

$$\mathrm{Tr}(f) \;:=\; \{(s, b) \in X_{\mathrm{fin}} \times |Y| \mid b \in f(s) \;\wedge\; \forall s' \subseteq s.\, b \in f(s') \;\Rightarrow\; s' = s\}.$$

Stable functions and traces

We recall that a function $f : X \to Y$ is **stable** if it is monotone, and preserves directed unions and pull-backs.

Proposition

A monotone and continuous function $f : X \to Y$ is stable if and only if, whenever $b \in f(x)$ for $x \in X$, there exists a finite clique $s \subseteq x$ such that $b \in f(s)$, and moreover for all $s' \subseteq x$ such that $b \in f(s')$, $s \subseteq s'$.

We define the **trace** of a stable function:

$$\mathrm{Tr}(f) \;:=\; \{(s, b) \in X_{\mathrm{fin}} \times |Y| \mid b \in f(s) \;\wedge\; \forall s' \subseteq s.\, b \in f(s') \;\Rightarrow\; s' = s\}.$$

Proposition

A stable function can be uniquely recovered from its trace:

$$f(x) \;=\; \{b \mid \exists (s, b) \in \mathrm{Tr}(f).\, s \subseteq x\}.$$

The trace defines a bijection between stable functions $f : X \to Y$ and the points of the coherence space $X \Rightarrow Y \;:=\; !X \multimap Y$.

Splitting the Atom II: Games

Splitting the Atom II: Games

Consider a token such as $(\{(3,2),(4,3)\},1)$.

Splitting the Atom II: Games

Consider a token such as $(\{(3,2),(4,3)\},1)$.

Many possible "schedules" for realising such a token as a process.

Splitting the Atom II: Games

Consider a token such as $(\{(3,2),(4,3)\},1)$.

Many possible "schedules" for realising such a token as a process. E.g.

$$
\begin{array}{ccccccc}
(& \mathbb{N} & \Rightarrow & \mathbb{N} &) & \Rightarrow & \mathbb{N} \\
 & & & & & & \mathrm{q} \\
 & & & \mathrm{q} & & & \\
 & \mathrm{q} & & & & & \\
 & 3 & & & & & \\
 & & & 2 & & & \\
 & & & \mathrm{q} & & & \\
 & \mathrm{q} & & & & & \\
 & 4 & & & & & \\
 & & & 3 & & & \\
 & & & & & & 1
\end{array}
$$

Coherence spaces as a collapse of game semantics

Coherence spaces as a collapse of game semantics

Forgetful map from schedules — *i.e.* plays in a game semantics — to tokens.

Coherence spaces as a collapse of game semantics

Forgetful map from schedules — *i.e.* plays in a game semantics — to tokens.

Game semantics decomposes information tokens into finer structures.

Coherence spaces as a collapse of game semantics

Forgetful map from schedules — *i.e.* plays in a game semantics — to tokens.

Game semantics decomposes information tokens into finer structures.

New subtleties arise:

Coherence spaces as a collapse of game semantics

Forgetful map from schedules — *i.e.* plays in a game semantics — to tokens.

Game semantics decomposes information tokens into finer structures.

New subtleties arise:

- Partial tokens

Coherence spaces as a collapse of game semantics

Forgetful map from schedules — *i.e.* plays in a game semantics — to tokens.

Game semantics decomposes information tokens into finer structures.

New subtleties arise:

- Partial tokens
- Sequential definability.

Coherence spaces as a collapse of game semantics

Forgetful map from schedules — *i.e.* plays in a game semantics — to tokens.

Game semantics decomposes information tokens into finer structures.

New subtleties arise:

- Partial tokens
- Sequential definability. E.g. consider composition.

Coherence spaces as a collapse of game semantics

Forgetful map from schedules — *i.e.* plays in a game semantics — to tokens.

Game semantics decomposes information tokens into finer structures.

New subtleties arise:

- Partial tokens
- Sequential definability. E.g. consider composition.

New possibilities also arise:

Coherence spaces as a collapse of game semantics

Forgetful map from schedules — *i.e.* plays in a game semantics — to tokens.

Game semantics decomposes information tokens into finer structures.

New subtleties arise:

- Partial tokens
- Sequential definability. E.g. consider composition.

New possibilities also arise:

- Full abstraction and full completeness

Coherence spaces as a collapse of game semantics

Forgetful map from schedules — *i.e.* plays in a game semantics — to tokens.

Game semantics decomposes information tokens into finer structures.

New subtleties arise:

- Partial tokens
- Sequential definability. E.g. consider composition.

New possibilities also arise:

- Full abstraction and full completeness
- Compositional model-checking

Coherence spaces as a collapse of game semantics

Forgetful map from schedules — *i.e.* plays in a game semantics — to tokens.

Game semantics decomposes information tokens into finer structures.

New subtleties arise:

- Partial tokens
- Sequential definability. E.g. consider composition.

New possibilities also arise:

- Full abstraction and full completeness
- Compositional model-checking

So we view the coherence space semantics as a stepping stone to a game semantics.

Dependent Type Theory

Dependent Type Theory

Key ideas (over and above simple propositional type theories):

Dependent Type Theory

Key ideas (over and above simple propositional type theories):

- Types depending on values of other types (syntactically, on terms).

Dependent Type Theory

Key ideas (over and above simple propositional type theories):

- Types depending on values of other types (syntactically, on terms).
  Thus no clean separation between syntax of types and terms; and order in contexts is important!

Dependent Type Theory

Key ideas (over and above simple propositional type theories):

- Types depending on values of other types (syntactically, on terms).
  Thus no clean separation between syntax of types and terms; and order in contexts is important!

- Identity types.

Dependent Type Theory

Key ideas (over and above simple propositional type theories):

- Types depending on values of other types (syntactically, on terms).
  Thus no clean separation between syntax of types and terms; and order in contexts is important!
- Identity types.
- Universes.

Dependent Type Theory

Key ideas (over and above simple propositional type theories):

- Types depending on values of other types (syntactically, on terms).
  Thus no clean separation between syntax of types and terms; and order in contexts is important!
- Identity types.
- Universes.

Judgement forms:

$\vdash \Gamma$ ctxt	Γ is a valid context
$\Gamma \vdash \sigma$ type	σ is a type in context Γ
$\Gamma \vdash M : \sigma$	M is a term of type σ in context Γ

Dependent Type Theory

Key ideas (over and above simple propositional type theories):

- Types depending on values of other types (syntactically, on terms).
  Thus no clean separation between syntax of types and terms; and order in contexts is important!
- Identity types.
- Universes.

Judgement forms:

$\vdash \Gamma$ ctxt	Γ is a valid context
$\Gamma \vdash \sigma$ type	σ is a type in context Γ
$\Gamma \vdash M : \sigma$	M is a term of type σ in context Γ

Plus equality judgements for contexts, types and terms.

Dependent Type Theory

Key ideas (over and above simple propositional type theories):

- Types depending on values of other types (syntactically, on terms).
  Thus no clean separation between syntax of types and terms; and order in contexts is important!
- Identity types.
- Universes.

Judgement forms:

$\vdash \Gamma$ ctxt	Γ is a valid context
$\Gamma \vdash \sigma$ type	σ is a type in context Γ
$\Gamma \vdash M : \sigma$	M is a term of type σ in context Γ

Plus equality judgements for contexts, types and terms.

Too many rules!

Context Formation

Rules for context formation:

$$\frac{}{\diamond \ \mathsf{ctxt}} \qquad \frac{\Gamma \vdash \sigma \ \mathsf{type}}{\Gamma, x : \sigma \ \mathsf{ctxt}}$$

Context Formation

Rules for context formation:

$$\frac{}{\diamond \ \mathsf{ctxt}} \qquad \frac{\Gamma \vdash \sigma \ \mathsf{type}}{\Gamma, x : \sigma \ \mathsf{ctxt}}$$

Note the mutual recursion between type and context formation.

Context Formation

Rules for context formation:

$$\frac{}{\diamond \text{ ctxt}} \qquad \frac{\Gamma \vdash \sigma \text{ type}}{\Gamma, x : \sigma \text{ ctxt}}$$

Note the mutual recursion between type and context formation.

A useful intuition:
In interpreting the simply-typed λ-calculus in a cartesian closed category, we interpret the comma in contexts as a product, and the turnstile in typing judgements as function space. Thus a type judgement

$$A_1, \ldots, A_n \vdash t : A$$

is interpreted as a morphism

$$f : A_1 \times \cdots \times A_n \to A.$$

Context Formation

Rules for context formation:

$$\frac{}{\Diamond \ \mathsf{ctxt}} \qquad \frac{\Gamma \vdash \sigma \ \mathsf{type}}{\Gamma, x : \sigma \ \mathsf{ctxt}}$$

Note the mutual recursion between type and context formation.

A useful intuition:
In interpreting the simply-typed λ-calculus in a cartesian closed category, we interpret the comma in contexts as a product, and the turnstile in typing judgements as function space. Thus a type judgement

$$A_1, \ldots, A_n \vdash t : A$$

is interpreted as a morphism

$$f : A_1 \times \cdots \times A_n \to A.$$

In interpreting dependent types, we should interpret the comma in contexts as a dependent sum, and the turnstile as a dependent product. This is the appropriate way to think of dependent type families.

Dependent Sums and Products

Dependent Sums and Products

Same type formation rules:

$$\frac{\Gamma \vdash \sigma \text{ type} \quad \Gamma, x : \sigma \vdash \tau \text{ type}}{\Gamma \vdash \Pi x : \sigma . \tau \text{ type}} \qquad \frac{\Gamma \vdash \sigma \text{ type} \quad \Gamma, x : \sigma \vdash \tau \text{ type}}{\Gamma \vdash \Sigma x : \sigma . \tau \text{ type}}$$

Dependent Sums and Products

Same type formation rules:

$$\frac{\Gamma \vdash \sigma \text{ type} \quad \Gamma, x : \sigma \vdash \tau \text{ type}}{\Gamma \vdash \Pi x : \sigma.\tau \text{ type}} \qquad \frac{\Gamma \vdash \sigma \text{ type} \quad \Gamma, x : \sigma \vdash \tau \text{ type}}{\Gamma \vdash \Sigma x : \sigma.\tau \text{ type}}$$

The introduction rules:

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma.M \ : \ \Pi x : \sigma.\tau} \qquad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau[M/x]}{\Gamma \vdash \mathsf{Pair}(M, N) : \Sigma x : \sigma.\tau}$$

Dependent Sums and Products

Same type formation rules:

$$\frac{\Gamma \vdash \sigma \text{ type} \quad \Gamma, x : \sigma \vdash \tau \text{ type}}{\Gamma \vdash \Pi x : \sigma.\tau \text{ type}} \qquad \frac{\Gamma \vdash \sigma \text{ type} \quad \Gamma, x : \sigma \vdash \tau \text{ type}}{\Gamma \vdash \Sigma x : \sigma.\tau \text{ type}}$$

The introduction rules:

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma.M \ : \ \Pi x : \sigma.\tau} \qquad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau[M/x]}{\Gamma \vdash \mathsf{Pair}(M, N) : \Sigma x : \sigma.\tau}$$

The “canonical elements” of these types are those given by the introduction rules.

Dependent Sums and Products

Same type formation rules:

$$\frac{\Gamma \vdash \sigma \text{ type} \quad \Gamma, x : \sigma \vdash \tau \text{ type}}{\Gamma \vdash \Pi x : \sigma.\tau \text{ type}} \qquad \frac{\Gamma \vdash \sigma \text{ type} \quad \Gamma, x : \sigma \vdash \tau \text{ type}}{\Gamma \vdash \Sigma x : \sigma.\tau \text{ type}}$$

The introduction rules:

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma.M \ : \ \Pi x : \sigma.\tau} \qquad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau[M/x]}{\Gamma \vdash \mathsf{Pair}(M, N) : \Sigma x : \sigma.\tau}$$

The “canonical elements” of these types are those given by the introduction rules.

The elimination rules give application and projections respectively.

Dependent Sums and Products

Same type formation rules:

$$\frac{\Gamma \vdash \sigma \text{ type} \quad \Gamma, x : \sigma \vdash \tau \text{ type}}{\Gamma \vdash \Pi x : \sigma.\tau \text{ type}} \qquad \frac{\Gamma \vdash \sigma \text{ type} \quad \Gamma, x : \sigma \vdash \tau \text{ type}}{\Gamma \vdash \Sigma x : \sigma.\tau \text{ type}}$$

The introduction rules:

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma.M \ : \ \Pi x : \sigma.\tau} \qquad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau[M/x]}{\Gamma \vdash \mathsf{Pair}(M, N) : \Sigma x : \sigma.\tau}$$

The "canonical elements" of these types are those given by the introduction rules.

The elimination rules give application and projections respectively.

The equality rules give the usual β-conversions.

Identity Types

Identity Types

Formation:

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathsf{Id}_\sigma(M, N) \text{ type}}$$

Identity Types

Formation:

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathsf{Id}_\sigma(M, N) \text{ type}}$$

Introduction:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \mathsf{Refl}_\sigma(M) : \mathsf{Id}_\sigma(M, M)}$$

Identity Types

Formation:

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathsf{Id}_\sigma(M, N) \ \mathsf{type}}$$

Introduction:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \mathsf{Refl}_\sigma(M) : \mathsf{Id}_\sigma(M, M)}$$

The reflexivity terms are the canonical elements.

Identity Types

Formation:

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathsf{Id}_\sigma(M, N) \ \mathsf{type}}$$

Introduction:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \mathsf{Refl}_\sigma(M) : \mathsf{Id}_\sigma(M, M)}$$

The reflexivity terms are the canonical elements.

But in the intensional theory, there can be many other elements.

Identity Types

Formation:

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathsf{Id}_\sigma(M, N) \text{ type}}$$

Introduction:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \mathsf{Refl}_\sigma(M) : \mathsf{Id}_\sigma(M, M)}$$

The reflexivity terms are the canonical elements.

But in the intensional theory, there can be many other elements.

The homotopy interpretation: paths!

Identity Types

Formation:

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathsf{Id}_\sigma(M, N) \ \mathsf{type}}$$

Introduction:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \mathsf{Refl}_\sigma(M) : \mathsf{Id}_\sigma(M, M)}$$

The reflexivity terms are the canonical elements.

But in the intensional theory, there can be many other elements.

The homotopy interpretation: paths!
i.e. **witnesses** giving ways in which one object can be transformed into another.

Identity Types

Formation:

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathsf{Id}_\sigma(M, N) \ \mathsf{type}}$$

Introduction:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \mathsf{Refl}_\sigma(M) : \mathsf{Id}_\sigma(M, M)}$$

The reflexivity terms are the canonical elements.

But in the intensional theory, there can be many other elements.

The homotopy interpretation: paths!
i.e. **witnesses** giving ways in which one object can be transformed into another.

Gives a completely new, **positive** way of thinking about intensional identity types.

Towards a coherence space semantics

Towards a coherence space semantics

It is useful to define the extended trace, which gives minimum data points for finite outputs:

$$\mathrm{Tr}^*(f) \;:=\; \{(s,t) \in X_{\mathrm{fin}} \times Y_{\mathrm{fin}} \mid t \subseteq f(s) \;\wedge\; \forall s' \subseteq s.\, t \subseteq f(s') \;\Rightarrow\; s' = s\}.$$

Towards a coherence space semantics

It is useful to define the extended trace, which gives minimum data points for finite outputs:

$$\mathrm{Tr}^*(f) \;:=\; \{(s,t) \in X_{\mathsf{fin}} \times Y_{\mathsf{fin}} \mid t \subseteq f(s) \;\wedge\; \forall s' \subseteq s.\, t \subseteq f(s') \;\Rightarrow\; s' = s\}.$$

Note that this extended trace can be derived from the basic version:

$$(s, \{b_1, \ldots, b_n\}) \in \mathrm{Tr}^*(f) \iff s = \bigcup_{i=1}^{n} s_i \;\wedge\; (s_i, b_i) \in \mathrm{Tr}(f), \quad i = 1, \ldots, n.$$

Towards a coherence space semantics

It is useful to define the extended trace, which gives minimum data points for finite outputs:

$$\mathrm{Tr}^*(f) \;:=\; \{(s,t) \in X_{\mathsf{fin}} \times Y_{\mathsf{fin}} \mid t \subseteq f(s) \;\wedge\; \forall s' \subseteq s.\, t \subseteq f(s') \;\Rightarrow\; s' = s\}.$$

Note that this extended trace can be derived from the basic version:

$$(s, \{b_1, \ldots, b_n\}) \in \mathrm{Tr}^*(f) \iff s = \bigcup_{i=1}^{n} s_i \;\wedge\; (s_i, b_i) \in \mathrm{Tr}(f), \quad i = 1, \ldots, n.$$

We shall show how to construct a **category with families**, a standard notion of semantics for DTT, based on coherence spaces.

Parameterizations

Parameterizations

We define an order relation on coherence spaces:

$$X \trianglelefteq Y \;\; \equiv \;\; |X| \subseteq |Y| \, \wedge \, \mathrel{\overset{\frown}{\smile}}_X \; = \; \mathrel{\overset{\frown}{\smile}}_Y \cap |X|^2.$$

Parameterizations

We define an order relation on coherence spaces:

$$X \trianglelefteq Y \;\; \equiv \;\; |X| \subseteq |Y| \;\wedge\; \smile_X \;=\; \smile_Y \cap |X|^2.$$

This yields a large cpo $(\mathbf{Coh}, \trianglelefteq)$ on the class of coherence spaces. This cpo is algebraic, since every graph is the directed union of its finite sub-graphs, and has pull-backs.

Parameterizations

We define an order relation on coherence spaces:

$$X \trianglelefteq Y \;\; \equiv \;\; |X| \subseteq |Y| \;\wedge\; \smile_X \;=\; \smile_Y \cap |X|^2.$$

This yields a large cpo $(\mathbf{Coh}, \trianglelefteq)$ on the class of coherence spaces. This cpo is algebraic, since every graph is the directed union of its finite sub-graphs, and has pull-backs.

A **parameterization** on a coherence space X is a stable, continuous function $F : X \to \mathbf{Coh}$. We write $\mathbf{Par}(X)$ for the set of parameterizations on X.

Dependent sums and products of coherence spaces

Dependent sums and products of coherence spaces

We now define constructions on coherence spaces corresponding to dependent sums and products.

Dependent sums and products of coherence spaces

We now define constructions on coherence spaces corresponding to dependent sums and products.

Given $F \in \mathbf{Par}(X)$, we define:

$$|\Sigma(X, F)| \; := \; \{(s, u) \mid s \in X_{\mathrm{fin}}, u \in F(s)_{\mathrm{fin}}\},$$

$$(s, u) \frown (t, v) \bmod \Sigma(X, F) \; \equiv \; s \cup t \in X \; \wedge \; u \cup v \in F(s \cup t).$$

Dependent sums and products of coherence spaces

We now define constructions on coherence spaces corresponding to dependent sums and products.

Given $F \in \mathbf{Par}(X)$, we define:

$$|\Sigma(X, F)| \;:=\; \{(s, u) \mid s \in X_{\mathsf{fin}}, u \in F(s)_{\mathsf{fin}}\},$$

$$(s, u) \frown (t, v) \bmod \Sigma(X, F) \;\equiv\; s \cup t \in X \;\wedge\; u \cup v \in F(s \cup t).$$

Given $F \in \mathbf{Par}(X)$, we define the extended trace of F:

$$\mathsf{Tr}^*(F) \;:=\; \{(s, t) \mid s \in X_{\mathsf{fin}}, b \in F(s)_{\mathsf{fin}}, \; \forall s' \subseteq s.\, t \in F(s')_{\mathsf{fin}} \;\Rightarrow\; s' = s\}.$$

Dependent sums and products of coherence spaces

We now define constructions on coherence spaces corresponding to dependent sums and products.

Given $F \in \mathbf{Par}(X)$, we define:

$$|\Sigma(X, F)| \;:=\; \{(s, u) \mid s \in X_{\mathsf{fin}}, u \in F(s)_{\mathsf{fin}}\},$$

$$(s, u) \smile (t, v) \bmod \Sigma(X, F) \;\equiv\; s \cup t \in X \;\wedge\; u \cup v \in F(s \cup t).$$

Given $F \in \mathbf{Par}(X)$, we define the extended trace of F:

$$\mathsf{Tr}^*(F) \;:=\; \{(s, t) \mid s \in X_{\mathsf{fin}}, b \in F(s)_{\mathsf{fin}},\; \forall s' \subseteq s.\, t \in F(s')_{\mathsf{fin}} \;\Rightarrow\; s' = s\}.$$

We define

$$|\Pi(X, F)| \;:=\; \mathsf{Tr}^*(F),$$

$$(s, u) \frown (t, v) \bmod \Pi(X, F) \;\equiv\; s \cup t \in X \;\Rightarrow\; u \cup v \in F(s \cup t).$$

Dependent sums and products of coherence spaces

We now define constructions on coherence spaces corresponding to dependent sums and products.

Given $F \in \mathbf{Par}(X)$, we define:

$$|\Sigma(X,F)| \;:=\; \{(s,u) \mid s \in X_{\mathrm{fin}}, u \in F(s)_{\mathrm{fin}}\},$$

$$(s,u) \smile (t,v) \bmod \Sigma(X,F) \;\equiv\; s \cup t \in X \;\wedge\; u \cup v \in F(s \cup t).$$

Given $F \in \mathbf{Par}(X)$, we define the extended trace of F:

$$\mathsf{Tr}^*(F) \;:=\; \{(s,t) \mid s \in X_{\mathrm{fin}}, b \in F(s)_{\mathrm{fin}}, \; \forall s' \subseteq s.\, t \in F(s')_{\mathrm{fin}} \;\Rightarrow\; s' = s\}.$$

We define

$$|\Pi(X,F)| \;:=\; \mathsf{Tr}^*(F),$$

$$(s,u) \frown (t,v) \bmod \Pi(X,F) \;\equiv\; s \cup t \in X \;\Rightarrow\; u \cup v \in F(s \cup t).$$

Suppose that F is constant. Define $Y := F(\varnothing)$. Then

$$\Pi(X,F) = X \Rightarrow Y, \quad \Sigma(X,F) \;=\; !X \otimes !Y \;\cong\; !(X \,\&\, Y).$$

A Category with Families based on Coherence Spaces

A Category with Families based on Coherence Spaces

- We take the category of contexts $\mathcal{C}$ to be the category of coherence spaces and stable maps.

A Category with Families based on Coherence Spaces

- We take the category of contexts $\mathcal{C}$ to be the category of coherence spaces and stable maps.
- For each coherence space X, we set $\mathrm{Ty}(X) := \mathbf{Par}(X)$. Given $F \in \mathbf{Par}(X)$, we set $\mathrm{Tm}(X, F) := \Pi(A, F)$.

A Category with Families based on Coherence Spaces

- We take the category of contexts $\mathcal{C}$ to be the category of coherence spaces and stable maps.
- For each coherence space X, we set $\mathrm{Ty}(X) := \mathbf{Par}(X)$. Given $F \in \mathbf{Par}(X)$, we set $\mathrm{Tm}(X, F) := \Pi(A, F)$.
- For the functorial action on types, given a stable map $f : X \to Y$, and $F \in \mathbf{Par}(Y)$, we define $F\{f\} \in \mathbf{Par}(X)$ by:

$$F\{f\} = F \circ f.$$

A Category with Families based on Coherence Spaces

- We take the category of contexts $\mathcal{C}$ to be the category of coherence spaces and stable maps.
- For each coherence space X, we set $\mathrm{Ty}(X) := \mathbf{Par}(X)$. Given $F \in \mathbf{Par}(X)$, we set $\mathrm{Tm}(X, F) := \Pi(A, F)$.
- For the functorial action on types, given a stable map $f : X \to Y$, and $F \in \mathbf{Par}(Y)$, we define $F\{f\} \in \mathbf{Par}(X)$ by:

$$F\{f\} \; = \; F \circ f.$$

- For the functorial action on terms, given $f : X \to Y$, and $t \in \Pi(Y, F)$, we define

$$t\{f\} \; := \; t \circ f \in \Pi(X, F\{f\}).$$

A Category with Families based on Coherence Spaces

- We take the category of contexts $\mathcal{C}$ to be the category of coherence spaces and stable maps.
- For each coherence space X, we set $\mathrm{Ty}(X) := \mathbf{Par}(X)$. Given $F \in \mathbf{Par}(X)$, we set $\mathrm{Tm}(X, F) := \Pi(A, F)$.
- For the functorial action on types, given a stable map $f : X \to Y$, and $F \in \mathbf{Par}(Y)$, we define $F\{f\} \in \mathbf{Par}(X)$ by:

$$F\{f\} \ = \ F \circ f.$$

- For the functorial action on terms, given $f : X \to Y$, and $t \in \Pi(Y, F)$, we define

$$t\{f\} \ := \ t \circ f \in \Pi(X, F\{f\}).$$

- The empty context is the terminal object $\top$ in $\mathcal{C}$.

A Category with Families based on Coherence Spaces

- We take the category of contexts $\mathcal{C}$ to be the category of coherence spaces and stable maps.
- For each coherence space X, we set $\mathrm{Ty}(X) := \mathbf{Par}(X)$. Given $F \in \mathbf{Par}(X)$, we set $\mathrm{Tm}(X, F) := \Pi(A, F)$.
- For the functorial action on types, given a stable map $f : X \to Y$, and $F \in \mathbf{Par}(Y)$, we define $F\{f\} \in \mathbf{Par}(X)$ by:

$$F\{f\} \; = \; F \circ f.$$

- For the functorial action on terms, given $f : X \to Y$, and $t \in \Pi(Y, F)$, we define

$$t\{f\} \; := \; t \circ f \in \Pi(X, F\{f\}).$$

- The empty context is the terminal object $\top$ in $\mathcal{C}$.
- Context extension is defined using the dependent sum construction. If $F \in \mathbf{Par}(X)$, then we define

$$X.F \; := \; \Sigma(X, F).$$

A Category with Families based on Coherence Spaces

- We take the category of contexts $\mathcal{C}$ to be the category of coherence spaces and stable maps.
- For each coherence space X, we set $\mathrm{Ty}(X) := \mathbf{Par}(X)$. Given $F \in \mathbf{Par}(X)$, we set $\mathrm{Tm}(X, F) := \Pi(A, F)$.
- For the functorial action on types, given a stable map $f : X \to Y$, and $F \in \mathbf{Par}(Y)$, we define $F\{f\} \in \mathbf{Par}(X)$ by:

$$F\{f\} \;=\; F \circ f.$$

- For the functorial action on terms, given $f : X \to Y$, and $t \in \Pi(Y, F)$, we define

$$t\{f\} \;:=\; t \circ f \in \Pi(X, F\{f\}).$$

- The empty context is the terminal object $\top$ in $\mathcal{C}$.
- Context extension is defined using the dependent sum construction. If $F \in \mathbf{Par}(X)$, then we define

$$X.F \;:=\; \Sigma(X, F).$$

- The projection morphism $\mathbf{p}_F : X.F \to X$ is the first projection $\Sigma(X, F) \to X$ whose trace is given by:

$$\{(\{(s, u)\}, a) \mid (s, u) \in |\Sigma(X, F)|, a \in s\}.$$

Definition of CwF ctd

Definition of CwF ctd

- The term $\mathbf{v}_F \in \Pi(\Sigma(X, F), F\{\mathbf{p}_F\})$ is the second projection, whose trace is given by:
$$\{(\{(s, u)\}, b) \mid (s, u) \in |\Sigma(X, F)|, b \in u\}.$$

Definition of CwF ctd

- The term $\mathbf{v}_F \in \Pi(\Sigma(X, F), F\{\mathbf{p}_F\})$ is the second projection, whose trace is given by:
$$\{(\{(s, u)\}, b) \mid (s, u) \in |\Sigma(X, F)|, b \in u\}.$$
- Given $f : X \to Y$, $F \in \mathbf{Par}(Y)$, and $g \in \Pi(X, F\{f\})$, we define the context morphism extension
$$\langle f, g\rangle_F : X \to \Sigma(Y, F)$$
as the pairing map, with trace:
$$\{(s \cup s', (t, u)) \in X_{\mathrm{fin}} \times |\Sigma(Y, F)| \mid (s, t) \in \mathrm{Tr}^*(f) \ \wedge \ (s', u) \in \mathrm{Tr}^*(g)\}.$$

Verification

Proposition

The above construction defines a CwF.

Proof The proof amounts to verifying a number of equations.
Firstly, for coherence spaces X, Y, Z, stable maps $f : X \to Y$, $g : Y \to Z$, $F \in \mathbf{Par}(Z)$, and $t \in \Pi(Z, F)$:

$$
\begin{array}{rcll}
F\{\mathrm{id}_X\} = F & \in & \mathrm{Ty}(X) & \text{(Ty-Id)} \\
F\{g \circ f\} = F\{g\}\{f\} & \in & \mathrm{Ty}(X) & \text{(Ty-Comp)} \\
t\{\mathrm{id}_Z\} = t & \in & \mathrm{Tm}(Z, F) & \text{(Tm-Id)} \\
t\{g \circ f\} = t\{g\}\{f\} & \in & \mathrm{Tm}(X, F\{g \circ f\})) & \text{(Tm-Comp)}
\end{array}
$$

Furthermore, for $f : X \to Y$, $g : Z \to X$, $F \in \mathbf{Par}(Y)$, and $t \in \Pi(X, F\{f\})$:

$$
\begin{array}{rcll}
\mathbf{p}_F \circ \langle f, t \rangle_F = f & : & X \to Y & \text{(Cons-L)} \\
\mathbf{v}_F\{\langle f, t \rangle_F\} = t & \in & \mathrm{Tm}(X, F\{f\}) & \text{(Cons-R)} \\
\langle f, t \rangle_F \circ g = \langle f \circ g, t\{g\} \rangle_F & : & Z \to \Sigma(Y, F) & \text{(Cons-Nat)} \\
\langle \mathbf{p}_F, \mathbf{v}_F \rangle_F = \mathrm{id}_{\Sigma(Y,F)} & : & \Sigma(Y, F) \to \Sigma(Y, F) & \text{(Cons-Id)}
\end{array}
$$

□

Identity, Totality and Intensionality

Identity, Totality and Intensionality

Identity types $\mathsf{Id}_\sigma(M, N)$ are interpreted by taking the intersections of the cliques denoted by M and N. But this gives a model of **partial type theory**, as previously studied using Scott domains by Palmgren and Stoltenberg-Hansen.

Identity, Totality and Intensionality

Identity types $\mathsf{Id}_{\sigma}(M, N)$ are interpreted by taking the intersections of the cliques denoted by M and N. But this gives a model of **partial type theory**, as previously studied using Scott domains by Palmgren and Stoltenberg-Hansen.

To get an appropriate model of standard type theory, we must extend the model with a notion of **totality**.

Identity, Totality and Intensionality

Identity types $\mathsf{Id}_\sigma(M, N)$ are interpreted by taking the intersections of the cliques denoted by M and N. But this gives a model of **partial type theory**, as previously studied using Scott domains by Palmgren and Stoltenberg-Hansen.

To get an appropriate model of standard type theory, we must extend the model with a notion of **totality**.

If this is done, we get a highly intensional (non-well-pointed) model.

Identity, Totality and Intensionality

Identity types $\mathsf{Id}_\sigma(M, N)$ are interpreted by taking the intersections of the cliques denoted by M and N. But this gives a model of **partial type theory**, as previously studied using Scott domains by Palmgren and Stoltenberg-Hansen.

To get an appropriate model of standard type theory, we must extend the model with a notion of **totality**.

If this is done, we get a highly intensional (non-well-pointed) model.

However, it is not yet clear which principles are satisfied in the internal sense of propositional equality, e.g. from UIP and Function Extensionality.

A glimpse at linear dependent types

A glimpse at linear dependent types

Syntax and Semantics of Linear Dependent Types, Matthijs Vákár, `http://arxiv.org/abs/1405.0033`.

A glimpse at linear dependent types

Syntax and Semantics of Linear Dependent Types, Matthijs Vákár, `http://arxiv.org/abs/1405.0033`.

The key issue is how to handle dependent types $\sigma(x)$ with x resource-sensitive.

A glimpse at linear dependent types

Syntax and Semantics of Linear Dependent Types, Matthijs Vákár, http://arxiv.org/abs/1405.0033.

The key issue is how to handle dependent types $\sigma(x)$ with x resource-sensitive.

If x is used in forming $\sigma(x)$, is it still available to form terms of type $\sigma(x)$?

A glimpse at linear dependent types

Syntax and Semantics of Linear Dependent Types, Matthijs Vákár, http://arxiv.org/abs/1405.0033.

The key issue is how to handle dependent types $\sigma(x)$ with x resource-sensitive.

If x is used in forming $\sigma(x)$, is it still available to form terms of type $\sigma(x)$?

This is circumvented in the current approach by considering only linear types depending on intuitionistic types, using a dual-context formulation:

$$\Gamma | \Theta \vdash M : \sigma$$

A glimpse at linear dependent types

Syntax and Semantics of Linear Dependent Types, Matthijs Vákár, `http://arxiv.org/abs/1405.0033`.

The key issue is how to handle dependent types $\sigma(x)$ with x resource-sensitive.

If x is used in forming $\sigma(x)$, is it still available to form terms of type $\sigma(x)$?

This is circumvented in the current approach by considering only linear types depending on intuitionistic types, using a dual-context formulation:

$$\Gamma | \Theta \vdash M : \sigma$$

It is not clear if this is robust with respect to universes.

A glimpse at linear dependent types

Syntax and Semantics of Linear Dependent Types, Matthijs Vákár, `http://arxiv.org/abs/1405.0033`.

The key issue is how to handle dependent types $\sigma(x)$ with x resource-sensitive.

If x is used in forming $\sigma(x)$, is it still available to form terms of type $\sigma(x)$?

This is circumvented in the current approach by considering only linear types depending on intuitionistic types, using a dual-context formulation:

$$\Gamma | \Theta \vdash M : \sigma$$

It is not clear if this is robust with respect to universes.

The underlying semantic structure is strict indexed monoidal categories with comprehension.

A glimpse at linear dependent types

Syntax and Semantics of Linear Dependent Types, Matthijs Vákár, `http://arxiv.org/abs/1405.0033`.

The key issue is how to handle dependent types $\sigma(x)$ with x resource-sensitive.

If x is used in forming $\sigma(x)$, is it still available to form terms of type $\sigma(x)$?

This is circumvented in the current approach by considering only linear types depending on intuitionistic types, using a dual-context formulation:

$$\Gamma | \Theta \vdash M : \sigma$$

It is not clear if this is robust with respect to universes.

The underlying semantic structure is strict indexed monoidal categories with comprehension.

There are a number of other delicate issues, e.g. multiplicative rather than additive forms of quantifiers.

Towards Games

Towards Games

There are some subtleties in refining the coherence space semantics to a game semantics.

Towards Games

There are some subtleties in refining the coherence space semantics to a game semantics.

In particular, consider the dependent sum.

Towards Games

There are some subtleties in refining the coherence space semantics to a game semantics.

In particular, consider the dependent sum.

In type theory, we have first and second projections, so we can access the components independently.

Towards Games

There are some subtleties in refining the coherence space semantics to a game semantics.

In particular, consider the dependent sum.

In type theory, we have first and second projections, so we can access the components independently.

However, in game terms, the dependence indicates a **causality of information flow**: to get information about the second component, we need information about the first.

Towards Games

There are some subtleties in refining the coherence space semantics to a game semantics.

In particular, consider the dependent sum.

In type theory, we have first and second projections, so we can access the components independently.

However, in game terms, the dependence indicates a **causality of information flow**: to get information about the second component, we need information about the first.

It is not clear how this can be captured in game semantics while preserving the required equations.