Game Semantics for Dependent Types

Samson Abramsky, Radha Jagadeesan and Matthijs Vákár

Bath, 20 October, 2015
Game theoretic model of dependent type theory (DTT):

- refines model in domains and (total) continuous functions;
- call-by-name evaluation;
- faithful model of (total) DTT with Σ-, Π-, intensional Id-types and finite inductive type families;
- fully complete if Id-types limited;
Game theoretic model of dependent type theory (DTT):

- refines model in domains and (total) continuous functions;
- call-by-name evaluation;
- faithful model of (total) DTT with $\Sigma$-, $\Pi$-, intensional $\text{Id}$-types and finite inductive type families;
- fully complete if $\text{Id}$-types limited;
- $\text{Id}$-types more intensional than domain model: function extensionality fails;
- intensional in orthogonal way to HoTT (time vs space): UIP holds.
Game Semantics?

- Interpolates between operational and denotational semantics: very intensional with structural clarity of categorical model;
- Unified framework for intensional, computational semantics:
  - PCF (HO, N, AJM);

Tight correspondence with syntax (full abstraction, full faithful completeness): often unique semantics in this respect.
Game Semantics?

- Interpolates between operational and denotational semantics:
  very intensional with structural clarity of categorical model;
- Unified framework for intensional, computational semantics:
  - PCF (HO, N, AJM);
  - references, non-local control, dynamically generated local
    names, probability, non-determinism, concurrency…;
  - various evaluation strategies;
  - sums, recursive types, polymorphism;
  - propositional logic, impredicative $2^{nd}$ order quantification,
    external $1^{st}$ order quantification;
Game Semantics?

- Interpolates between operational and denotational semantics: very intensional with structural clarity of categorical model;
- Unified framework for intensional, computational semantics:
  - PCF (HO, N, AJM);
  - references, non-local control, dynamically generated local names, probability, non-determinism, concurrency...;
  - various evaluation strategies;
  - sums, recursive types, polymorphism;
  - propositional logic, impredicative 2\textsuperscript{nd} order quantification, external 1\textsuperscript{st} order quantification;
  - internal 1\textsuperscript{st} order quantification / dependent types surprisingly absent (and surprisingly hard!);
Game Semantics?

- Interpolates between operational and denotational semantics: very intensional with structural clarity of categorical model;
- Unified framework for intensional, computational semantics:
  - PCF (HO, N, AJM);
  - references, non-local control, dynamically generated local names, probability, non-determinism, concurrency...;
  - various evaluation strategies;
  - sums, recursive types, polymorphism;
  - propositional logic, impredicative 2\textsuperscript{nd} order quantification, external 1\textsuperscript{st} order quantification;
  - internal 1\textsuperscript{st} order quantification / dependent types surprisingly absent (and surprisingly hard!);
- Tight correspondence with syntax (full abstraction, full faithful completeness): often unique semantics in this respect.
<table>
<thead>
<tr>
<th>Computation / Logic</th>
<th>Games</th>
</tr>
</thead>
<tbody>
<tr>
<td>data type / proposition</td>
<td>2-player game (duality!)</td>
</tr>
<tr>
<td>computational process / argument</td>
<td>play: alternating seq. of moves</td>
</tr>
<tr>
<td>program / proof</td>
<td>Player ($P$) strategy</td>
</tr>
<tr>
<td>environment / refutation</td>
<td>Opponent ($O$) strategy</td>
</tr>
<tr>
<td>variable declarations / axiom links</td>
<td>copycat strategies</td>
</tr>
<tr>
<td>substitution / cut</td>
<td>interaction</td>
</tr>
<tr>
<td>termination / correctness</td>
<td>winning</td>
</tr>
<tr>
<td>pure sequential functional behaviour</td>
<td>history-freeness/innocence</td>
</tr>
<tr>
<td>(no state, parallelism, control ops) /..</td>
<td>+ determinism</td>
</tr>
<tr>
<td></td>
<td>+ well-bracketing of strategies</td>
</tr>
</tbody>
</table>

Samson Abramsky, Radha Jagadeesan and Matthijs Vákár

Game Semantics for Dependent Types
<table>
<thead>
<tr>
<th>Computation / Logic</th>
<th>Games</th>
</tr>
</thead>
<tbody>
<tr>
<td>data type / proposition</td>
<td>2-player game (duality!)</td>
</tr>
<tr>
<td>computational process / argument</td>
<td>play: alternating seq. of moves</td>
</tr>
<tr>
<td>program / proof</td>
<td>Player (P) strategy</td>
</tr>
<tr>
<td>environment / refutation</td>
<td>Opponent (O) strategy</td>
</tr>
<tr>
<td>variable declarations / axiom links</td>
<td>copycat strategies</td>
</tr>
<tr>
<td>substitution / cut</td>
<td>interaction</td>
</tr>
<tr>
<td>termination / correctness</td>
<td>winning</td>
</tr>
<tr>
<td>pure sequential functional behaviour</td>
<td>history-freeness/innocence</td>
</tr>
<tr>
<td>(no state, parallelism, control ops) /..</td>
<td>+ determinism</td>
</tr>
<tr>
<td></td>
<td>+ well-bracketing of strategies</td>
</tr>
</tbody>
</table>

![Chessboard image](image)
An example:

Player: \[ x : \mathbb{B}, y : (\mathbb{B} \Rightarrow \mathbb{B}) \vdash y(x) : \mathbb{B} \]

Opponent: \[ -[\text{tt}/x, (\lambda z : \mathbb{B} \rightarrow z)/y] \]

\[
\begin{array}{c|c|c|c|c}
\mathbb{B}^* & \Rightarrow & (\mathbb{B}^* & \Rightarrow & \mathbb{B}^*_*) & \Rightarrow & \mathbb{B}^*_* \\
\hline
\text{tt} & \text{tt} & \text{tt} & \text{ff} & \text{ff} & \text{ff} & \text{ff} \\
\end{array}
\]
An example:

**Player:** \( x : \mathbb{B}, y : (\mathbb{B} \Rightarrow \mathbb{B}) \vdash y(x) : \mathbb{B} \)

**Opponent:** \(- [ (\lambda z : \mathbb{B} \, \text{ff}) / y ]\)

\[
\begin{array}{cccccc}
\mathbb{B}_* & \Rightarrow & (\mathbb{B}_* & \Rightarrow & \mathbb{B}_*) & \Rightarrow & \mathbb{B}_* \\
\hline
\vdash & * & & & & & O \\
& * & & & & & P \\
& \text{ff} & & & & & O \\
& & \text{ff} & & & & P \\
\end{array}
\]
Games and winning strategies form a smcc **Game**:

- \( I \): the game with one play of length 0;
- \( A \otimes B \): playing \( A \) and \( B \) simultaneously through interleaving, where only Opponent can switch games;
- \( A \rightarrow B \): \( \text{swap}_{O,P}(A) \) and \( B \) simultaneously, Player switches.
Games and winning strategies form a smcc **Game**: 

- **I**: the game with one play of length 0;
- **$A \otimes B$**: playing $A$ and $B$ simultaneously through interleaving, where only Opponent can switch games;
- **$A \rightarrow B$**: $\text{swap}_{O,P}(A)$ and $B$ simultaneously, Player switches.

Also model simple type theory (STT): have a ccc **Game**!:

- **$A \Rightarrow B := !A \rightarrow B$**;
- **$!A$**: playing $\omega$ equivalent copies of $A$ simultaneously, where only Opponent can switch games;
- **Product $A & B$**: Opponent chooses to play $A$ or $B$ (unit: $I$).
Games and winning strategies form a smcc \textbf{Game}:

- \(I\): the game with one play of length 0;
- \(A \otimes B\): playing \(A\) and \(B\) simultaneously through interleaving, where only Opponent can switch games;
- \(A \multimap B\): \(swap_{O,P}(A)\) and \(B\) simultaneously, Player switches.

Also model simple type theory (STT): have a ccc \textbf{Game}!:

- \(A \Rightarrow B := !A \multimap B\);
- \(!A\): playing \(\omega\) equivalent copies of \(A\) simultaneously, where only Opponent can switch games;
- Product \(A \& B\): Opponent chooses to play \(A\) or \(B\) (unit: \(I\)).

Ground types (finite inductive types): for a set \(X\), game \(X_\ast\) with one Opponent move \(\ast\), followed by any of the Player moves \(x \in X\).
An example: copycats (identity morphisms)

Player: \( x : A \vdash x : A \)

Opponent:

\[
\begin{array}{c|c}
A & A \\
\hline
\Rightarrow & \\
\hline
a & O \\
\hline
a' & P \\
\hline
O & P \\
\hline
\vdash & \\
\hline
\end{array}
\]
An example: interaction (composition)

Player: \( y : \mathbb{N} \vdash 2y : \mathbb{N} \)

Opponent: \( x : \mathbb{N} \vdash x + 1 : \mathbb{N} \)

\[
\begin{array}{c|c}
\mathbb{N}_* \Rightarrow \mathbb{N}_* & \mathbb{N}_* \Rightarrow \mathbb{N}_* \\
\hline
* & * \\
* & * \\
x & x + 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
* & O \\
OP & OP \\
OP & OP \\
\end{array}
\]

parallel composition...
An example: interaction (composition)

Player: \( x : \mathbb{N} \vdash x + 1 : \mathbb{N} \vdash 2(x + 1) : \mathbb{N} \)

Opponent:

\[
\begin{array}{cccc}
\mathbb{N}_* & \Rightarrow & \mathbb{N}_* & \Rightarrow & \mathbb{N}_* \\
\Rightarrow & \Rightarrow & & & O \\
& & * & & \\
& & * & & P \\
& x & & & O \\
& x + 1 & & & P \\
2(x + 1) & & & & \\
\end{array}
\]

... plus hiding

similarly in general
Use the term simple type theory (STT) to refer to a simple λ-calculus with binary products \( \times \), function types \( \Rightarrow \) and finite inductive types \( \{ a_i \mid 1 \leq i \leq n \} \), or a total finitary PCF with binary products, with \( \beta \eta \)-rules and PCF commutative conversions for case-constructs.

Straightforward consequence of AJM:

**Theorem**

*The interpretation of STT in \( \text{Game}_! \) is fully and faithfully complete.*
Dependent type theory (DTT)?

What is it?

- Curry-Howard for predicate logic: types with free (term) variables, constructions $\Sigma, \Pi, \text{Id}$ (cf. $\exists, \forall, =)$ on types.
- Judgements like $x : A \vdash B(x)$ type and $x : A \vdash b(x) : B(x)$.
- Order in context matters!
- No clean separation syntax types and terms.
Dependent type theory (DTT)?

What is it?

- Curry-Howard for predicate logic: types with free (term) variables, constructions $\Sigma$, $\Pi$, $\text{Id}$ (cf. $\exists$, $\forall$, $=$) on types.
- Judgements like $x : A \vdash B(x)$ type and $x : A \vdash b(x) : B(x)$.
- Order in context matters!
- No clean separation syntax types and terms.

Why care?

- Move towards richer type systems: e.g. GADTs in Haskell.
- Types allowed to refer to data: e.g. $n : \mathbb{N} \vdash \text{List}(n)$ type.
- Specification by typing: certification by type checking.
- Programming logic as a proof assistant.
- Logical Frameworks.
Dependent types and their terms:
\[ n : \mathbb{N} \vdash \text{List}(n) \text{ type} \]
\[ n : \mathbb{N}, x : \text{List}(n) \vdash \text{reverse}(x) : \text{List}(n) \]

\(\Pi-\) and \(\Sigma-\)types (generalising \(\Rightarrow\) and \(\times\)):
\[ \vdash \lambda_n \lambda_x \text{reverse}(x) : \Pi_{n:\mathbb{N}} \text{List}(n) \Rightarrow \text{List}(n) \]
\[ \vdash \text{mult} : \Pi_{k,l,m:\mathbb{N}} \text{matrix}(k, l) \times \text{matrix}(l, m) \Rightarrow \text{matrix}(k, m) \]
\[ \vdash \langle 9, [\text{No, wise, fish, would, go, anywhere, without, a, porpoise}] \rangle : \Sigma_{n:\mathbb{N}} \text{List}(n) \]
Dependent types and their terms:

\[ n : \mathbb{N} \vdash \text{List}(n) \text{ type} \]
\[ n : \mathbb{N}, x : \text{List}(n) \vdash \text{reverse}(x) : \text{List}(n) \]

\(\Pi\)- and \(\Sigma\)-types (generalising \(\Rightarrow\) and \(\times\)):
\[ \vdash \lambda_n\lambda_x \text{reverse}(x) : \Pi_{n:\mathbb{N}} \text{List}(n) \Rightarrow \text{List}(n) \]
\[ \vdash \text{mult} : \Pi_{k,l,m:\mathbb{N}} \text{matrix}(k, l) \times \text{matrix}(l, m) \Rightarrow \text{matrix}(k, m) \]
\[ \vdash \left< 9, [\text{No, wise, fish, would, go, anywhere, without, a, porpoise]} \right> : \Sigma_{n:\mathbb{N}} \text{List}(n) \]

Id-type:
\[ \vdash \text{refl} : \text{Id}_{\mathbb{N}}(0, (\lambda_x x)(0)) \]
\[ x, y : B \vdash \text{case}(x, \text{case}(y, \text{refl}, \text{refl}), \text{case}(y, \text{refl}, \text{refl})) : \text{Id}_{B}(\text{lsor}(x,y), \text{rsor}(x,y)) \]

Finite inductive type families:
\[ x : \text{months} \vdash \text{days}(x) \text{ type} \]
\[ \vdash 31 : \text{days}(\text{October}). \]
Idea: DTT talks about same algorithms as STT but can assign them a more precise type/specification.

Formally: have translation of DTT into STT. Let DTT inherit the equational theory of STT to make translation faithful.
Idea: DTT talks about same algorithms as STT but can assign them a more precise type/specification.

Formally: have translation of DTT into STT. Let DTT inherit the equational theory of STT to make translation faithful.

\[ x : A \vdash (a_i \mapsto_i \{ b_{i,j} \mid j \}) \text{ type} \quad \mapsto \quad \vdash \{ b_{i,j} \mid i, j \} \text{ type} \]

\[ x : A \vdash \Sigma_{y:B} C \text{ type} \quad \mapsto \quad \vdash B^T \times C^T \text{ type} \]

\[ x : A \vdash \Pi_{y:B} C \text{ type} \quad \mapsto \quad \vdash B^T \Rightarrow C^T \text{ type} \]

\[ x : A, y : B, y' : B \vdash \text{Id}_B(y, y') \text{ type} \quad \mapsto \quad \vdash B^T \text{ type} \]

\[ x' : A' \vdash B(a) \text{ type} \quad \mapsto \quad \vdash B(x)^T \text{ type} \]

+ translation on terms.
The idea of our interpretation $\llbracket \cdot \rrbracket$ will be to construct a category $\text{CtxtGame}_!$ of dependent context games with a functor $\odot(-)$ to $\text{Game}_!$, such that

$$
\begin{array}{c}
\text{Syntax}_{DTT} \xrightarrow{\llbracket - \rrbracket} \text{CtxtGame}_! \\
\downarrow (-)^T \\
\text{Syntax}_{STT} \xrightarrow{\llbracket - \rrbracket} \text{Game}_!
\end{array}
$$

Games model of DTT will therefore automatically be faithful.
Game $B$ with dependency on $A$ consists of:

- game $\circledast(B)$ (without dependency);
- continuous function $\text{str}(A) \xrightarrow{B} \text{Sub}(\circledast(B))$ from strategies on $A$ to subgames ($\preceq$-closed subsets of plays) of $\circledast(B)$.

Alternatively:

- continuous function $\text{str}(A) \xrightarrow{\top} \text{Game}_{\preceq}$. 

---

Samson Abramsky, Radha Jagadeesan and Matthijs Vákár

Game Semantics for Dependent Types
Dependent Games and Strategies

Game $B$ with dependency on $A$ consists of:

- game $\otimes(B)$ (without dependency);
- continuous function $\text{str}(A) \xrightarrow{B} \text{Sub}(\otimes(B))$ from strategies on $A$ to subgames ($\prec$-closed subsets of plays) of $\otimes(B)$.

Alternatively:

- continuous function $\text{str}(A) \xrightarrow{B} \text{Game}_\prec$.

Can define $I$, $\&$ and $\Rightarrow$ on games with dependency and make into ccc with homset $\text{DGame}_!(A)(B, C) := \text{wstr}(\Pi_A(B \Rightarrow C))$. (Arises from model of LL.)
We define a game $\Pi_A B \subseteq A \Rightarrow 0(B)$ of dependent functions.

- Idea: the choice of a fibre $B(a)$ for the output of a dependent function $f : \Pi_A B$ is entirely the responsibility of the context that provides the argument $a$. 
We define a game $\Pi_A B \triangleleft A \Rightarrow \odot(B)$ of dependent functions.

- **Idea:** the choice of a fibre $B(a)$ for the output of a dependent function $f : \Pi_A B$ is entirely the responsibility of the context that provides the argument $a$.

- **Opponent can determine fibre $B(\tau)$ of $B$:**
  - explicitly, revealing winning strategy $\tau$ on $A$, by playing in $!A$;
  - implicitly, by playing in $\odot(B)$;

- **Player has to stay within $B(\tau)$ for all $\tau$ consistent with Opponent’s behaviour.**
We define a game $\Pi_A B \subseteq A \Rightarrow \hat{\odot}(B)$ of dependent functions.

- Idea: the choice of a fibre $B(a)$ for the output of a dependent function $f : \Pi_A B$ is entirely the responsibility of the context that provides the argument $a$.
- Opponent can determine fibre $B(\tau)$ of $B$:
  - explicitly, revealing winning strategy $\tau$ on $A$, by playing in $!A$;
  - implicitly, by playing in $\odot(B)$;
- Player has to stay within $B(\tau)$ for all $\tau$ consistent with Opponent’s behaviour.
- That is, as long as there is such a $\tau$; otherwise, anything goes.
- Indeed, Opponent is totally free and might not respect the type dependency, as $\odot(\_)$ should be faithful (to match $(-)^T$).
Non-example of dependently typed algorithm: scheduling finance meetings

Player/Academic: \( x : \text{months} \vdash 31 : \text{days}(x) \)

Opponent/Education Finance Business Manager Manager:

\[
\Pi(\text{months}_*, \text{days}_*) \quad | \\
\quad * \quad | O \\
\quad 31 \\ P
\]

Player chooses fibre: e.g. February doesn’t have a 31\(^{st}\) day. Mr Manager shouldn’t allow that...
Example of dependently typed algorithm:

\[
\begin{align*}
\text{Academic/Player: } & \quad x : \text{months} \vdash \text{case}_{\text{days}, \text{months}}(x, \{31, 1, \ldots, 1, 31\}) : \text{days}(x) \\
\text{Opponent/Education Finance Business Manager Manager: } & \quad [\text{January}/x]
\end{align*}
\]

\[
\begin{array}{c|cc}
\Pi( & \text{months*} & \text{days*} ) \\
\hline
* & O \\
* & P \\
* & O \\
* & P \\
\text{January} & O \\
31 & P \\
\end{array}
\]
Example (fibre-wise identities):

Player: $x : A, y : B(x) \vdash y : B(x)$

Opponent: choose your favourite

\[
\begin{array}{c|cc}
\Pi( [A] , [B] \Rightarrow [B] ) & \quad b & \quad O \\
& b & P \\
b' & b' & O \\
& \vdots & \\
\end{array}
\]
Theorem

*Have indexed ccc* \textbf{Game}_{\text{op}} \xrightarrow{\text{DGame}} \text{CCCat} \text{ of dependent games:}*

- \text{ob}(\text{DGame}_{\text{op}}(A)) := \{\text{continuous str}(A) \xrightarrow{B} \text{Sub}(\ominus(B))\}
- \text{hom-sets} \ DGame_{\text{op}}(A)(B, C) := \text{wstr}(\Pi_A(B \Rightarrow C))
- \text{fibrewise identities as in example;}
- \text{fibrewise comp. & change of base: usual AJM-composition.}
Theorem

Have indexed ccc Game\textsuperscript{op} \(\text{DGame} \rightarrow\) CCCat of dependent games:

- \(\text{ob}(\text{DGame}(A)) := \{\text{continuous } \text{str}(A) \xrightarrow{B} \text{Sub}(\odot(B))\}\)
- \(\text{hom-sets } \text{DGame}(A)(B, C) := \text{wstr}(\Pi_A(B \Rightarrow C))\)
- fibrewise identities as in example;
- fibrewise comp. & change of base: usual AJM-composition.

As we don’t have (additive) \(\Sigma\)-types, not a model of DTT!

"Well! I’ve often seen a cat without a grin," thought Alice; "but a grin without a cat! It's the most curious thing I ever saw in all my life."
Theorem

Have indexed ccc \( \text{Game}^{\text{op}} \xrightarrow{\text{DGame}} \text{CCCat} \) of dependent games:

- \( \text{ob}(\text{DGame}!(A)) := \{\text{continuous } \text{str}(A) \xrightarrow{B} \text{Sub}(\odot(B))\} \)

- \( \text{hom-sets } \text{DGame}!(A)(B, C) := \text{wstr}(\Pi_A(B \Rightarrow C)) \)

- fibrewise identities as in example;

- fibrewise comp. & change of base: usual AJM-composition.

As we don’t have (additive) \( \Sigma \)-types, not a model of DTT!

"Well! I’ve often seen a cat without a grin,” thought Alice; "but a grin without a cat! It's the most curious thing I ever saw in all my life.”

Theorem

Formally add them: get model \( \text{CtxtGame}^{!} \) of DTT with \( \Sigma \Pi \)-types!

Obj: lists \([X_k]_k\), where \( \text{str}(\odot(X_1) \& \cdots \& \odot(X_{k-1})) \xrightarrow{X_k} \text{Sub}(\odot(X_k)) \).

Mor: lists of winning strategies on \( \Pi \)-games of more variables.
We can also interpret

- finite inductive type families $x : A \vdash (a_i \mapsto_i \{ b_{i,j} \mid j \})$ type:

  $[a_i] \mapsto \{ b_{i,j} \mid j \}^* \sqsubseteq \{ b_{i,j} \mid i,j \}^*$
  else $\mapsto \emptyset^* \sqsubseteq \{ b_{i,j} \mid i,j \}^*$;

- identity types $x : A \vdash \text{Id}_B(s,t)$ type:

  $\sigma \mapsto (\sigma;[s]) \cap (\sigma;[t]) \sqsubseteq \odot([B])$. 
Non-Example:

Player: \( x : \mathbb{B}, y : \mathbb{B} \vdash p : \text{Id}_\mathbb{B}(x, y) \)
Opponent: \( - [tt/x, ff/y] \)

\[
\begin{array}{cccc}
\Pi( & \mathbb{B} & , & \Pi( & \mathbb{B} & , & \text{Id}_{\mathbb{B}} ) \\
* & O \\
* & P \\
tt & P \\
tt & O \\
\text{ff} & P \\
\end{array}
\]

...as \( tt \) does not lie in intersection \( tt \cap ff \).
Example:

**Player:** \( x : B \vdash \text{refl}_x : \text{Id}_B(x, x) \)

**Opponent:**

\[
\Pi(B_*, \text{Id}_{B_*}\{\text{diag}_{B_*}\})
\]

\[
\begin{array}{cccc}
\ast & & & O \\
\ast & & P & \\
x & O & P & P \\
x & & & \\
\end{array}
\]

...as \( x \) lies in \( x \cap x \).
Place in intensionality spectrum $\text{Id}$-types:

<table>
<thead>
<tr>
<th></th>
<th>Domains</th>
<th>HoTT</th>
<th>Games</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure of Equality Reflection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Streicher Intensionality Criteria ($I_1$) and ($I_2$)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Streicher Intensionality Criterion ($I_3$)</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Failure of Function Extensionality (FunExt)</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Failure of Uniqueness of Identity Proofs (UIP)</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

Discrete ground types (0-types), but function hierarchy generates (open) propositional identities: observational equivalences.
Summarising, we have

**Theorem (Soundness and Faithfulness)**

We have a faithful model of DTT with $\Sigma$-, $\Pi$-, *intensional* $\text{Id}$-types and finite inductive type families: faithful functor

$$\text{Syntax}_{DTT} \xrightarrow{\llbracket - \rrbracket} \text{Ctx} \text{tGame}!.$$
Summarising, we have

**Theorem (Soundness and Faithfulness)**

*We have a faithful model of DTT with Σ-, Π-, intensional $\text{Id}$-types and finite inductive type families: faithful functor*

$$\text{Syntax}_{DTT} \xrightarrow{[-]} \text{CtxtGame}_!.$$  

Actually, the model has strong completeness properties.

**Theorem (Full Completeness)**

*This interpretation is full when restricted to the types of the form $A$ or $\Pi_A \text{Id}_B(f,g)$ with $A$ and $B$ built without $\text{Id}$-types:*

$$\text{Syntax}_{\text{Restricted Types}}^{\text{DTT}} \xrightarrow{[-]} \text{CtxtGame}_! \quad \text{full.}$$
Future Work

Ultimate goal: intensional, computational analysis of HoTT.

- game semantics of higher inductive types / quotient types;
- examining function extensionality and univalence;
- universes and a more intensional notion of type family;
- infinite inductive type families and their definability results;
- examining completeness properties of the model for the complete type hierarchy, including Id-types;
- constructing models of DTT with effects.
In particular, for the first item:

\[ HtpyGame \xrightarrow{\infty - Gpd \times_{Set} Game} \text{Game} \]

- collapsing space-like identity, a.k.a. 0-truncation
  \[ \infty - Gpd \xrightarrow{\text{collapsing space-like identity, a.k.a. 0-truncation}} Set \]
  \[ X \xrightarrow{||X||_0} \]

- collapsing time-like identity, a.k.a. extensional collapse
  \[ A \xrightarrow{\text{wstr}(A)/\text{obs.equiv.}} \]
Bonus Example (Higher order dependent functions): approving holiday plans

Player/PhD Student: \( w : \text{years}, x : \prod(y : \text{days}(w), \prod(z : \text{holidays}(w, y), \mathbb{B})) \vdash \text{going on holidays?} : \mathbb{B} \)

Opponent/Supervisor: \( -[2015 / w, \text{acceptable holidays}/x] \)

\[
\begin{array}{c}
\prod(\text{years}*, \prod(\prod(\text{days}*, \prod(\text{holidays}*, \mathbb{B})), \mathbb{B})), \mathbb{B}*) \\\
\end{array}
\]

(ITL PD happens every year, so don’t need to ask about the year. Moreover, Player is also in charge of supplying the date, so we don’t have to determine the date before specifying the holiday.)
Bonus Example (Higher order dependent functions): approving holiday plans

Player/PhD Student: \( w : \text{years}, x : \Pi(y : \text{days}(w), \Pi(z : \text{holidays}(w, y), \mathbb{B})) \vdash \text{going on holidays?} : \mathbb{B} \)

Opponent/Supervisor: \( -[2015/w, \text{acceptable holidays}/x] \)

\[
\begin{array}{c|cccc}
\Pi(\text{years}_*, \Pi(\Pi(\text{days}_*, \Pi(\text{holidays}_*, \mathbb{B}_*)))) & \mathbb{B}_* & \ast & \ast & O \\
\ast & \ast & \ast & \ast & P \\
\ast & \ast & \ast & \ast & P \\
\ast & \ast & \ast & \ast & P \\
\text{International Talk Like a Pirate Day} & \ast & \ast & \ast & P \\
\ast & \ast & \ast & \ast & P \\
\ast & \ast & \ast & \ast & P \\
\end{array}
\]

disobedient student...
Bonus Example (Higher order dependent functions): approving holiday plans

Player/PhD Student: \[ w : \text{years}, x : \Pi(y : \text{days}(w), \Pi(z : \text{holidays}(w, y), \mathbb{B})) \vdash \text{going on holidays?} : \mathbb{B} \]

Opponent/Supervisor: \[ - [2015/w, \text{acceptable holidays}/x] \]

(Note that Holi happens every year with variable Gregorian date and that Player gets to choose the day so can even specify the holiday before the day.)