Polymorphic Games and Program Equivalence

Jim Laird, University of Bath
**Background**

- *Parametric Polymorphism*, or genericity, is a key feature of high-level languages, often combined with state (e.g. dynamic dispatch).

- By hiding information, it creates interesting problems for reasoning about program equivalence.

- *Games* give concrete models of higher-order objects and functions with state (and other effects), which lend themselves to verification.
Parametric Polymorphism, or genericity, is a key feature of high-level languages, often combined with state (e.g. dynamic dispatch).

By hiding information, it creates interesting problems for reasoning about program equivalence.

Games give concrete models of higher-order objects and functions with state (and other effects), which lend themselves to verification.

Object of this work: To develop games models for programming languages with effects and higher-rank polymorphism, and use it to reason about program behaviour and equivalence. Most material from “Game Semantics for a Polymorphic Programming Language”, JACM, 2013
System F Typing

Types:

\[ S, T ::= X \mid 1 \mid S \to T \mid S \times T \mid \forall X.T \]

- \[ \Theta; \Gamma, x : T \vdash x : T \quad FV(T) \subseteq \Theta \]
- \[ \Theta; \Gamma \vdash \ast : 1 \quad FV(T) \subseteq \Theta \]
- \[ \Theta; \Gamma, x : S \vdash t : T \quad \Theta; \Gamma \vdash \lambda x. s : S \to T \]
- \[ \Theta, X; \Gamma \vdash t : T \quad X \notin FV(\Gamma) \]
- \[ \Theta; \Gamma \vdash \forall X.T \quad FV(S) \subseteq \Theta \]

\[ \Theta; \Gamma \vdash \{ S \} : T[S/X] \]

\[ \Theta; \Gamma \vdash t : \{ S \} : T[S/X] \quad FV(S) \subseteq \Theta \]

We write \( I \) for \( \forall X.X \to X \)
What are the inhabitants of $S \rightarrow T$?

- **Extensionally** — functions from the set of inhabitants of $S$ to the set of inhabitants of $T$.
- **Intensionally** — a blueprint for computing an inhabitant of $T$ by interaction with an inhabitant of $S$.

What are the inhabitants of $\forall X.T$:

- **Extensionally** — a product or intersection of the sets of inhabitants of $T[S]$ over all possible $S$. (Predicativity?)
- **Intensionally** — a blueprint for computing an inhabitant of $T[S/X]$ for any $S$.

“Type variables in generic programs act as placeholders for types which may be instantiated later”.
Key Properties for our Model

- **Genericity** — capturing “uniformity” in a more intensional way than e.g. relational parametricity or dinaturality (although these play a role).

- **Full abstraction** — capturing *information hiding* in polymorphism, e.g. in equivalences such as
  \[ \lambda x.(x\{I\}) \lambda y.y \simeq \lambda x.x : \forall X.(X \to X) \to \forall X.(X \to X). \]

- **Effective presentability** — holding the prospect of verification applications. (Note that inhabitation in System F is not decidable).
A type \( T \) is generic (w.r.t. a given theory) if for all terms \( s, t : \forall X. S \), \( s\{T\} = t\{T\} \) implies \( s = t \).

**Theorem** [Longo et. al.] Extending System F with genericity at all types* yields a consistent theory.

- The syntactic (Böhm tree) model is not generic.
- Any generic *extensional* model will contain “junk” — e.g. infinitely many elements at each type.
A type $T$ is \textit{generic} (w.r.t. a given theory) if for all terms $s, t : \forall X.S$, $s\{T\} = t\{T\}$ implies $s = t$.

\textbf{Theorem} [Longo et. al.] Extending System F with genericity at all types\footnote{Note that including e.g. the unit type breaks genericity.} yields a consistent theory.

- The syntactic (Böhm tree) model is not generic.
- Any generic \textit{extensional} model will contain “junk” — e.g. infinitely many elements at each type.

\footnote{Note that including e.g. the unit type breaks genericity.}
Types with free variables are interpreted as “context arenas” with holes into which arenas may be plugged.

Second-order type structure is captured by a generalization of question/answer labelling and the bracketing condition.

To instantiate a generic strategy, the question/answer relation is used to infer a *copycat* relation between occurrences of the instantiated arena.
A context arena $A$ is a labelled bipartite *forest* $(M_A, \vdash_A, \lambda_A, \triangleright_A)$, where

- $M_A$ is a set of *moves*.
- $\vdash \subseteq M_A \times M_A$ is an *enabling relation*, making
- $\lambda_A : M_A \rightarrow \{Q, A\} \cup \mathbb{N}$ labels moves as *questions, answers* or “$i$-holes” for $i \in \mathbb{N}$.
- $\triangleright \subseteq Q_A \times M_A \times \text{Ans}_A$ is a *scoped question/answer relation* determining which (Player/Opponent) answers may be given in response to each (Opponent/Player) question.
We can interpret the syntax trees of second order types $X_1, \ldots, X_n \vdash A$ directly as context games:

- Moves are (paths to the) leaves of the tree (type variables).
- Each path enables the moves (paths) immediately to its left.
- A path which ends with a free variable $X_i$ is labelled with $i$ (an $i$-hole).

Leaves which are positively bound variables are questions.
Leaves which are negatively bound variables are answers.
Example: \( \forall X. X \rightarrow X \rightarrow X \)
The Lifted Sum

\[\forall X. (S \rightarrow X) \rightarrow (T \rightarrow X) \rightarrow X\] is the lifted sum arena \([S] \oplus [T]\).

\[\forall X. (S \rightarrow X) \rightarrow (T \rightarrow X) \rightarrow X\]
Given an alternating path through the forest, we match up answers and questions as closing and opening parentheses:

\[
\left( \left[ \left( \left[ \right] \right) \right] \right)
\]

and require that:

If answer move \( a \) closes question \( q \), there is a move \( m \) such that

\[
 m \vdash^* q, \quad m \vdash^* a \quad \text{and} \quad q \triangleright_m a.
\]
Play in $\left[ \forall X. \forall f. f \lambda x. (f \lambda y. x)\{X\} \right]$:

$$\forall X. ((X \rightarrow X) \rightarrow \forall Y. Y) \rightarrow \forall Y. Y)$$
Scoped Bracketing

Play in $[[\lambda f. f \land X. \lambda x. (f \lambda y. y)\{X\}]]$.

$[[((\forall X. (X \to X) \to \forall Y. Y) \to \forall Y. Y)]]$
Given a strategy (non-empty, even-branching, even-prefix-closed set of legal sequences) $\sigma : \forall X_{n+1}.A$, define $\sigma[B] : A[B/X_{n+1}]$ — the \textit{instantiation} of a context arena $B$ into $\sigma$ at $X_{n+1}$ by \textit{replacing each question-answer pair in} $\sigma$ \textit{with a copycat link}. 
Example

Plugging the arena $\forall Y.Y \to Y \to Y$:

$$
\begin{array}{c}
Y \\
Y \quad Y
\end{array}
$$

into the identity strategy on $\forall X.X \to X$:

$$
\begin{array}{c}
q \\
a
\end{array}
$$

gives the identity on $(\forall Y.Y \to Y \to Y) \to \forall Y.Y \to Y \to Y$:

$$
\begin{array}{c}
Y \\
Y \\
Y \\
Y
\end{array}
$$
We define an indexed category $\mathcal{G} : I^{op} \to CCC$, where:

- $I$ is the category in which objects are natural numbers and morphisms from $m$ to $n$ are $n$-tuples of second order types over $m$ free variables (composed by substitution).

- For each $n$, $\mathcal{G}(n)$ is the (cartesian closed) category in which object are second-order types over $n$ free variables and morphisms from $A$ to $B$ are strategies on $\forall X_1 \ldots \forall X_n.(A \Rightarrow B)$ (composed by parallel composition plus hiding).

- For each tuple $\langle B_1, \ldots, B_n \rangle : m \to n$, $\mathcal{G}\langle B_1, \ldots, B_n \rangle : \mathcal{G}(m) \to \mathcal{G}(n)$ is the corresponding substitution functor.

The inclusion $\{ J_{n+1} : \mathcal{G}(n) \to \mathcal{G}(n + 1) \mid n \in \mathbb{N} \}$ has an indexed left adjoint $\forall X_{n+1} : \mathcal{G}(n + 1) \to \mathcal{G}(n)$. 
**Proposition** For any type $A$, instantiation is a *retraction*


(}
Proposition For any type $A$, instantiation is a *retraction*:
$$\forall X. A[X] \subseteq A[I/X].$$

*(Corollary)* $I$ is a *generic* type for our model. (If $\llbracket M : \forall X. T \rrbracket \neq \llbracket N : \forall X. T \rrbracket$ then $\llbracket M[I] : T[I/X] \neq \llbracket N[I] : T[I/X] \rrbracket$.)
Our model is not fully complete for System F — strategies do not satisfy the \textit{visibility} condition (and so they are not \textit{innocent} or \textit{total}, either).

In particular: the instantiation strategy \(\sigma : \forall X.A \rightarrow A[B]\) does not satisfy visibility.
Example of a violation of visibility

At type $\forall X.((\forall Y.((X \to Y \to X) \to Y \to X)) \to X)$:

$\Lambda X.\lambda f^{\forall Y.((X\to Y\to X)\to Y\to X)}.(f\{X \to X\} (\lambda x^X.\lambda g^X\to X.g\ x)) \lambda y^X.y$

$[\forall X.((\forall Y.(( X \to Y \to X) \to Y \to X)) \to X)]$
We add a constant \( \text{new} : \forall X. \forall Y. (Y \to (Y \to I) \to X) \to X \)
declaring a reference cell, and denoting a strategy with the same underlying play as the preceding example.
We add a constant \( \text{new} : \forall X. \forall Y. (Y \rightarrow (Y \rightarrow I) \rightarrow X) \rightarrow X \) declaring a reference cell, and denoting a strategy with the same underlying play as the preceding example.

Using \text{new}, the section \( p : A[I/X] \rightarrow \forall X.A \) is definable as a term which stores pointers to preceding \( X \)-moves as references. We use this fact to prove finite definability/full abstraction for our model, reducing it to (essentially) the simply-typed model of Abramsky, Honda and McCusker.
Intensional semantics of polymorphic value types (“data driven”) is fundamentally different to computational types (“demand driven”).

Based on (re-engineering of) Honda and Yoshida’s semantics of call-by-value $\lambda$-calculus.
Intensional semantics of polymorphic value types (“data driven”) is fundamentally different to computational types (“demand driven”).

Based on (re-engineering of) Honda and Yoshida’s semantics of call-by-value $\lambda$-calculus.

Specifically, we need to break down moves into tuples of smaller “atoms”, including holes into which arenas may be plugged.
Intensional semantics of polymorphic value types ("data driven") is fundamentally different to computational types ("demand driven").

Based on (re-engineering of) Honda and Yoshida’s semantics of call-by-value $\lambda$-calculus.

Specifically, we need to break down moves into tuples of smaller “atoms”, including holes into which arenas may be plugged.

Quantification over types is by linking these holes with explicit pointers.
Intensional semantics of polymorphic value types ("data driven") is fundamentally different to computational types ("demand driven").

Based on (re-engineering of) Honda and Yoshida’s semantics of call-by-value $\lambda$-calculus.

Specifically, we need to break down moves into tuples of smaller "atoms", including holes into which arenas may be plugged.

Quantification over types is by linking these holes with explicit pointers.

To instantiate a generic strategy, these pointers are used to infer a *copycat link* between occurrences of the instantiated arena.

Full abstraction is established by showing that $\text{nat}$ is a generic type.
∀(X × X → X) has two initial Opponent • questions — which must be played together, and a single, contingent • answer which may point to either of them —

\[
X \times X \rightarrow X
\]

\[\langle \bullet Q, \bullet Q \rangle\]
∀(X × X → X) has two initial Opponent • questions — which must be played together, and a single, contingent • answer which may point to either of them —

\[
X \times X \rightarrow X
\]

\[
\langle \bullet Q, \bullet Q \rangle
\]
Ongoing Work

- Subtype polymorphism (System $F_{\leq}$) using dinaturality properties of instantiation to represent *bounded quantification*.
- Type-operators — extending the syntactic representation of games with $\lambda$-abstraction and application of type variables in arenas.
- Constraining the model — can we describe models with fewer/different effects, linear types, etc.
- Verification — when the references can be tamed, the model appears to be suitable for an algorithmic approach.