

# Polymorphic Game Semantics for Dynamic Binding

J. Laird  
University of Bath, UK

August 30, 2016

# Goals

Extend game semantics of generic polymorphism in three (related) directions:

- ▶ **Dynamic Binding** of terms (and types) to variables — i.e. determined at run-time, not statically.
- ▶ **Block structure** — block scoping of definitions and polymorphic copying of blocks of code.
- ▶ Prove definability and full abstraction results for different combinations of **computational effects** (control, state, ...).

# Motivations

- ▶ Discover a good semantic theory (e.g. categorical model) for dynamic binding, which is common in practice (virtual methods, overloading, dynamic dispatch).
- ▶ Understand relationship between polymorphism and dynamic scope, which seems to exist in our model at a deep level.
- ▶ Leverage the concrete/algorithmic representation of polymorphism in our semantics towards model checking for a language with polymorphism, integer state, classes,...
- ▶ Extend the “intensional hierarchy” relating **computational effects** with **constraints on strategies**.

# Simply Typed Game Semantics: Block Structure

In game semantics of PCF (Hyland & Ong '00; Abramsky, Jagadeesan & Malacaria '00) , Idealized Algol (Abramsky & McCusker '97), etc.:

- ▶ **Types** denote **games** for two players,
- ▶ who take turn to make **moves**,
- ▶ which are **questions** (procedure calls) or **answers** (returns)
- ▶ Questions and answers open and close evaluation of **blocks** of code.

E.g. — the type *com* of *commands* denotes the game with one question (*run* the command) with a single answer (*done*).

# Simply Typed Game Semantics: Block Structure

Programs denote **strategies** for Player 2 (“Proponent”) — e.g. sequential composition:

$$\begin{array}{ccccc} com & \times & com & \rightarrow & com \\ & & & & run^O \\ & & & & run^P \\ & & & & done^O \\ & & & & run^P \\ & & & & done^O \\ & & & & done^P \end{array}$$

## Second-Order Game Semantics

*Generic* Game Semantics of System F [L. 2010]

— question and answer moves denote **type variables** (occurring positively and negatively with respect to their binder)

E.g.  $\llbracket \forall X.X \rightarrow X \rrbracket$  is the arena with one question, one answer.

$\forall X.X \quad \rightarrow \quad X$

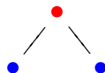
$Q^o$

$A^p$

Questions and answers represent **holes** into which we can plug any other game tree — answering a question corresponds to **playing copycat** between these plugged-in games.

## Example

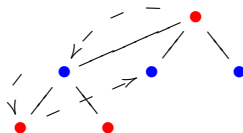
Plugging the arena  $\forall Y. Y \rightarrow Y \rightarrow Y$ :



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



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



# Subsuming Simple Games in Polymorphic Games

- ▶ Second order games **subsume** simply-typed ones via well-known System F encodings — e.g.  $\forall X.X \rightarrow X$  denotes the same game as *com*,
- ▶ *But* we lose block structuring, for which we need basic types as atoms.
- ▶ Instantiation breaks static (and block) scoping in this model — to prove **definability** (fullness of the interpretation functor) and **full abstraction** (completeness w.r.t. observational equivalence), we need to add **general references**.

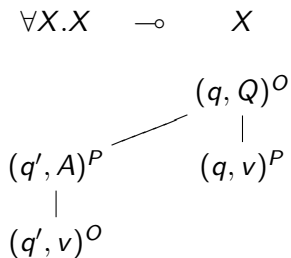


# Polymorphic Game Semantics with Block Structure

Two levels of question-answer (bracketing):

- ▶ A move is either a **simple question**  $q$  or **simple answer**  $(q, v)$  (a simple question and a value).
- ▶ A simple question may be further labelled as a **second-order question**  $(q, Q)$  or **second-order answer**  $(q, A)$ .
- ▶ Simple questions are endowed with **enabling** and **conflict** relations, which we use to build a model of (intuitionistic) linear type theory.

# Example: New Interpretation for $\forall X.(X \multimap X)$



## Constraints: New and Old

Legal positions are determined by the unanswered (simple) questions, corresponding to the **stack** of open procedure calls — each move must be:

- ▶ **enabled** by a question on the stack if it is a simple question — only a function may call its arguments
- ▶ **conflict-free** with all moves on the stack if it is a simple question — only one call to a function at each nesting depth.
- ▶ a response to the question at the top of the stack if it is a second-order answer (**well-bracketing**)
- ▶ A simple answer **copying** the preceding value if and only if that is a simple answer which closes a second-order answer — polymorphic copycat condition.

## Example

The only strategies on  $\forall X.(X \multimap X)$  are  $\{\varepsilon\}$  and the “generic identity” copycat  $(\sum_{v \in \mathcal{V}} (q, Q)^O (q', A)^P (q', v)^O (q, v)^P)^*$ .

$$\begin{array}{ccc} \forall X.X & \multimap & X \\ & & (q, Q)^O \\ & & (q', A)^P \\ & & (q', v)^O \\ & & (q, v)^P \\ & & (q, Q)^O \\ & & (q', A)^P \\ & & (q', v')^O \\ & & (q, v')^P \\ & & \vdots \end{array}$$

# Dynamic Binding: Typing

Extend System F with a set of ground types  $B$  and a **non-binding** *let* operation.

$$\frac{\Theta \vdash \Gamma, T}{\Gamma, x: T, \Gamma' \vdash_{\Theta} x: T}$$

$$\frac{\Gamma, x: S, \Gamma' \vdash_{\Theta} N: T \quad \Gamma, x: S, \Gamma' \vdash_{\Theta} M: S}{\Gamma, x: S, \Gamma' \vdash_{\Theta} \text{let } x = M \text{ in } N: T}$$

$$\frac{\Gamma, x: S \vdash_{\Theta} M: T}{\Gamma \vdash_{\Theta} \lambda x^S. M: S \rightarrow T}$$

$$\frac{\Gamma \vdash_{\Theta} M: S \rightarrow T \quad \Gamma \vdash_{\Theta} N: S}{\Gamma \vdash_{\Theta} M N: T}$$

$$\frac{\Gamma \vdash_{\Theta, X} M: T \quad \Theta \vdash \Gamma}{\Gamma \vdash_{\Theta} \Lambda X. M: \forall X. T}$$

$$\frac{\Gamma \vdash_{\Theta} M: \forall X. T \quad \Theta \vdash S}{\Gamma \vdash_{\Theta} M\{S\}: T[S/X]}$$

# Dynamic Binding: Operational Semantics

Programs are evaluated with respect to a **stack** of bindings — e.g.

$$\frac{M; \mathcal{E} \Downarrow 0; \mathcal{E}'}{\text{If } 0 \ M; \mathcal{E} \Downarrow \lambda xy. x; \mathcal{E}}$$

$$\frac{M; \mathcal{E} \Downarrow \lambda x. M'; \mathcal{E}'_a \quad M'[a/x]; \mathcal{E}'_a, (a, N) \Downarrow C; \mathcal{E}''}{M \ N; \mathcal{E} \Downarrow C; \mathcal{E}''}$$

$$\frac{M; \mathcal{E}, (a, M), \mathcal{E}_a \Downarrow C; \mathcal{E}''}{a; \mathcal{E}, (a, M), \mathcal{E}_a \Downarrow C; \mathcal{E}''}$$

$$\frac{M; \mathcal{E} \Downarrow n+1; \mathcal{E}'}{\text{If } 0 \ M; \mathcal{E} \Downarrow \lambda xy. y; \mathcal{E}}$$

$$\frac{M; \mathcal{E} \Downarrow \Lambda X. M'; \mathcal{E}' \quad M'[T/X]; \mathcal{E}' \Downarrow C; \mathcal{E}''}{M \{ T \}; \mathcal{E} \Downarrow C; \mathcal{E}''}$$

$$\frac{N; \mathcal{E}, (a, M) \Downarrow C; \mathcal{E}'}{\text{let } a = M \ \text{in } N; \mathcal{E} \Downarrow C; \mathcal{E}'}$$

# Examples

- ▶ **Overriding of definitions:** e.g.  
 $let\ f = (let\ x = 1\ in\ M)\ in\ let\ x = 0\ in\ N,$
- ▶ **Escape from static scope:**  $g : (R \rightarrow S \rightarrow S) \rightarrow R \rightarrow T \vdash$   
 $\lambda x^R.(g(\lambda y^T.\lambda z^S.let\ x = y\ in\ z))\ x : R \rightarrow T$  — the second  
argument to  $g$  may be captured from inside the scope of the  
first.
- ▶ **No escape from block scope** — e.g.  $\lambda x^{nat}.if\ 0\ x\ then\ x\ else\ x$   
and  $\lambda x^{nat}.x$  are observationally equivalent (without integer  
state).

# Dynamic Binding: Denotational Semantics

Like game semantics of references, exceptions, channels, locks, ... dynamically bound variables of type  $A$  may be considered as **objects** with two **methods**:

- ▶ **definition** (*let*) of “type”  $!A \multimap \forall X.(X \multimap X)$
- ▶ **invocation** of type  $A$

which we represent as projections from

$$\S A = !(!A \multimap \forall X.(X \multimap X)) \otimes !A.$$

So we interpret  $x_1 : S_1, \dots, x_n : S_n \vdash M : T$  as a morphism from  $\S[S_1] \otimes \dots \otimes \S[S_n]$  to  $\llbracket T \rrbracket$ .



## Dynamic variable declaration strategy

To **declare** a new variable, compose with a strategy  $dec_A : !A \rightarrow \S A$  which dynamically connects definitions to invocations:

$$\begin{array}{ccc}
 !A & \rightarrow & !( !A \multimap \forall X. X \multimap X ) \otimes !A \\
 & & \begin{array}{c} (q, Q)^O \\ (q, A)^P \end{array} \\
 & & q^O \\
 & & \begin{array}{c} q^P \\ (q, v)^O \end{array} \\
 & & (q, v)^P \\
 & & \begin{array}{c} (q, v')^O \\ (q, v')^P \end{array} \\
 & & q^O \\
 q^P & & \\
 \vdots & & 
 \end{array}$$

# Dynamic Binding: Categorical Semantics

To prove soundness of our semantics, we capture its **categorical structure**.

- ▶  $!A$  ( $\omega$ -indexed copies of  $A$ , opened in order) is a **minimal invariant** (least fixed point) for  $A \otimes \_$ , where
- ▶  $\otimes$  is a **monoidal action** of our category of games on a subcategory  $\mathcal{L}$  of strict maps, which is **dual** to  $\multimap$  — i.e.

$$\frac{\mathcal{L}(A \otimes B, C)}{\mathcal{L}(A, B \multimap C)} \qquad \frac{\mathcal{L}(B \multimap A, C)}{\mathcal{L}(A, C \otimes B)}$$

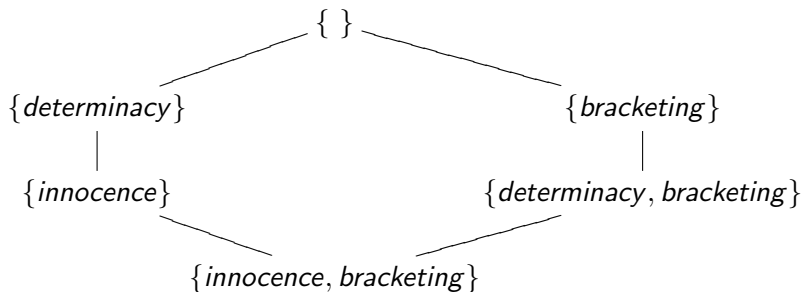
- ▶ We use these categorical properties to (corecursively) define declaration of dynamically bound variables, and prove its soundness.

# Computational Effects

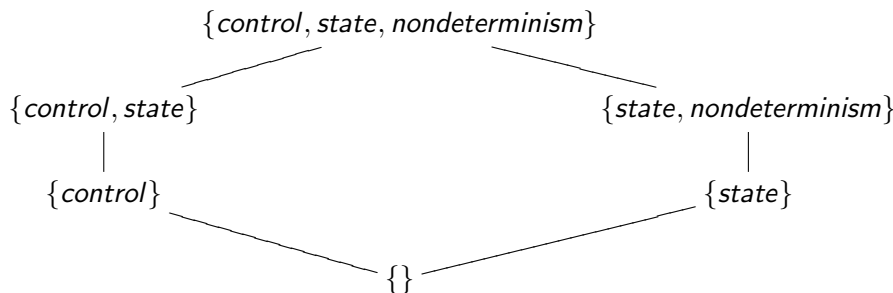
The “intensional hierarchy” on simple games relates:

- ▶ **Constraints** on strategies — used to identify **PCF-definable** strategies
- ▶ with computational **effects** — e.g. relaxing **innocence** gives a model of integer state (Idealized Algol).

# Hierarchy of Constraints

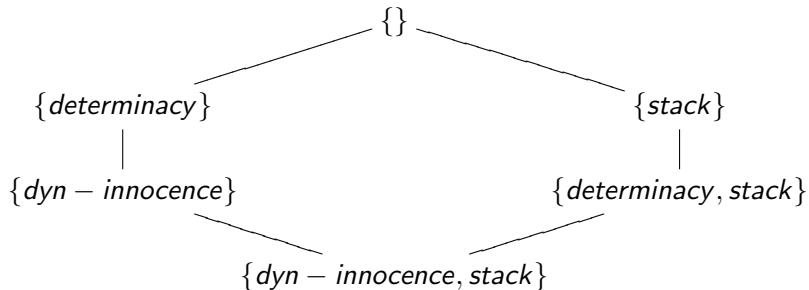


# Hierarchy of Effects



# An Intensional Hierarchy for Dynamic Binding

We define constraints on our model giving an intensional hierarchy of models with dynamic binding:



# Definability and Full Abstraction

We prove

- ▶  $PCF_d^2$ -definability for simply-typed strategies (by PCF-style [decomposition](#)).
- ▶  $PCF_d^2$ -definability for strategies at all types, using a [definable retraction](#)  $\forall X. T \sqsubseteq T[(\forall Y. Y \rightarrow Y)/X]$
- ▶  $IA_d^2$ -definability and full abstraction for strategies which break dynamic innocence by an Idealized-Algol style [factorization](#).

# Questions?