Game Semantics for Interface Middleweight Java

Andrzej S. Murawski
DIMAP and Department of Computer Science
University of Warwick

Nikos Tzevelekos
School of Electronic Engineering and Computer Science
Queen Mary, University of London

Abstract

We consider an object calculus in which open terms interact with the environment through interfaces. The calculus is intended to capture the essence of contextual interactions of Middleweight Java code. Using game semantics, we provide fully abstract models for the induced notions of contextual approximation and equivalence. These are the first denotational models of this kind.

Categories and Subject Descriptors D.3.1 [Formal Definitions and Theory]: Semantics; F.3.2 [Semantics of Programming Languages]: Denotational semantics

Keywords Full Abstraction, Game Semantics, Contextual Equivalence, Java

1. Introduction

Denotational semantics is charged with the construction of mathematical universes (denotations) that capture program behaviour. It concentrates on compositional, syntax-independent modelling with the aim of illuminating the structure of computation and facilitating reasoning about programs. Many developments in denotational semantics have been driven by the quest for full abstraction [21]: a model is fully abstract if the interpretations of two programs are the same precisely when the programs behave in the same way (i.e. are contextually equivalent). A faithful correspondence like this opens the path to a broad range of applications, such as compiler optimisation and program transformation, in which the preservation of semantics is of paramount importance.

Recent years have seen game semantics emerge as a robust denotational paradigm [4, 6, 12]. It has been used to construct the first fully abstract models for a wide spectrum of programming languages, previously out of reach of denotational semantics. Game semantics models computation as an exchange of moves between two players, representing respectively the program and its computational environment. Accordingly, a program is interpreted as a strategy in a game corresponding to its type. Intuitively, the plays that game semantics generates constitute the observable patterns that a program produces when interacting with its environment, and this is what underlies the full abstraction results. Game semantics is compositional: the strategy corresponding to a compound program phrase is obtained by canonical combinations of those corresponding to its sub-phrases. An important advance in game semantics was the development of nominal games [3, 17, 26], which underpinned full abstraction results for languages with dynamic generative behaviours, such as the $\nu$-calculus [3], higher-order concurrency [18] and ML references [24]. A distinctive feature of nominal game models is the presence of names (e.g. memory locations, references names) in game moves, often along with some abstraction of the store.

The aim of the present paper is to extend the range of the game approach towards real-life programming languages, by focussing on Java-style objects. To that end, we define an imperative object calculus, called Interface Middleweight Java (IMJ), intended to capture contextual interactions of code written in Middleweight Java (MJ) [9], as specified by interfaces with inheritance. We present both equational (contextual equivalence) and inequational (contextual approximation) full abstraction results for the language. To the best of our knowledge, these are the first denotational models of this kind.

Related Work While the operational semantics of Java has been researched extensively [7], there have been relatively few results regarding its denotational semantics. More generally, most existing models of object-oriented languages, such as [8, 15], have been based on global state and consequently could not be fully abstract.

On the other hand, contextual equivalence in Java-like languages has been studied successfully using operational approaches such as trace semantics [2, 13, 14] and environmental bisimulation [16]. The trace-based approaches are closest to ours and the three papers listed also provide characterizations of contextual equivalence. The main difference is that traces are derived operationally through a carefully designed labelled transition system and, thus, do not admit an immediate compositional description in the style of denotational semantics.

However, similarities between traces and plays in game semantics indicate a deeper correspondence between the two areas, which also manifested itself in other cases, e.g. [20] vs [19]. At the time of writing, there is no general methodology for moving smoothly between the two approaches, but we believe that there is scope for unifying the two fields in the not so distant future.

In comparison to other game models, ours has quite lightweight structure. For the most part, playing consists of calling the opponent’s methods and returning results to calls made by the opponent. In particular, there are no justification pointers between moves. This can be attributed to the fact that Java does not feature first-class higher-order functions and that methods in Java objects cannot be updated. On the other hand, the absence of pointers makes definitions of simple notions, such as well-bracketing, less direct, since the dependencies between moves are not given explicitly any
more and need to be inferred from plays. The latter renders strategy composition non-standard. Because it is impossible to determine statically to which arena a move belongs, the switching conditions (cf. [6]) governing interactions become crucial for determining the strategy responsible for each move. Finally, it is worth noting that traditional copiycat links are by definition excluded from our setting: a call/return move for a given object cannot be copiycat by the other player, as a move has a fixed polarity, determined by the ownership of the object. In fact, identity strategies contain plays of length at most two!

**Further Directions** In future work, we would like to look for automata-theoretic representations of fragments of our model in order to use them as a foundation for a program verification tool for Java programs. Our aim is to take advantage of the latest developments in automata theory over infinite alphabets [10], and fresh-register automata in particular [23, 27], to account for the nominal features of the model.

## 2. The language IMJ

We introduce an imperative object calculus, called Interface Mid-
lweight Java (IMJ), in which objects are typed using interfaces. The calculus is a stripped down version of Middletweight Java (MJ), expressive enough to expose the interactions of MJ-style objects with the environment.

**Definition 1.** Let Ints, Flds and Meths be sets of interface, field and method identifiers. We range over them respectively by $I, f, m$ and variants. The types $\Theta$ of IMJ are given below, where $\theta$ stands for a sequence $\theta_1, \ldots, \theta_n$ of types (for any $n$). An interface definition $\Theta$ is a finite set of typed fields and methods. An **interface table** $\Delta$ is a finite assignment of interface definitions to interface identifiers.

$$
\Theta \ni \theta := \text{void} \ | \ \text{int} \ | \ I
$$

$IDfns \ni \Theta := \emptyset \ | \ (f : \Theta), \Theta \ | \ (m : \eta \to \theta), \Theta$

$ITbls \ni \Delta := \emptyset \ | \ (I : \Theta), \Delta \ni (I(I') : \Theta), \Delta$

We write $I(I') : \Theta$ for interface extension: interface $I$ extends $I'$ with fields and methods from $\Theta$. We stipulate that the extension relation must not lead to circular dependencies. Moreover, each identifier $f, m$ can appear at most once in each $\Theta$, and each $I$ can be defined at most once in $\Delta$ (i.e., there is at most one element of $\Delta$ of the form $I : \Theta$ or $I(I') : \Theta$). Thus, each $\Theta$ can be seen as a finite partial function $\Theta : (Ints \cup Meths) \to Types$. We write $\Theta(f)$ for $\Theta(f)$ and $\Theta.m$ for $\Theta(m)$. Similarly, $\Delta$ defines a partial function $\Delta : Ints \to IDfns$ given by

$$
\Delta(I) = \begin{cases}
\Theta & (I : \Theta) \ni \Delta
\emptyset & (I(I') : \Theta) \ni \Delta
\text{undefined} & \text{otherwise}
\end{cases}
$$

An interface table $\Delta$ is **well-formed** if, for all interface types $I, I'$:

- if $I'$ appears in $\Delta(I)$ then $I' \in \text{dom}(\Delta)$,
- if $(I(I') : \Theta) \in \Delta$ then $\text{dom}(\Delta(I')) \cap \text{dom}(\Theta) = \emptyset$.

Henceforth we assume that interface tables are well-formed. Interface extensions yield a subtyping relation. Given a table $\Delta$, we define $\Delta \vdash \theta_1 \leq \theta_2$ by the following rules.

$$
(\Delta(I', \Theta), \Delta) \vdash \Delta \leq \theta
$$

$$
\Delta \vdash \theta_1 \leq \theta_2 \leq \theta_3 \vdash \Delta \vdash \theta_1 \leq \theta_3 \leq \theta_2
$$

We might omit $\Delta$ from subtyping judgements for economy.

**Definition 2.** Let $A$ be a countably infinite set of **object names**, which we range over by $a$ and variants. IMJ **terms** are listed below, where we let $x$ range over a set of variables $Vars$, and $i$ over $\mathbb{Z}$. Moreover, $\otimes$ is selected from some set of binary numeric operands. $\mathcal{M}$ is a **method-set implementation**. Again, we stipulate that each $m$ appear in each $\mathcal{M}$ at most once.

$$
\begin{align*}
M := & \ x \ | \ a \ | \ \text{skip} \ | \ \text{null} \ | \ i \ | \ M \oplus M \ | \ \text{let} \ x = M \ni M
M = & \ M \ | \ M \ni \text{if} M \text{then } M \text{else } M
M.f := & \ M \ni \text{new}(x : I; M)
M := [m, f : M, m_0]
\text{MImps} := & \emptyset \ | \ (\ell : \otimes \text{M})
\end{align*}
$$

The terms are typed in contexts comprising an interface table $\Delta$ and a variable context $\Gamma = \{x_1 : \theta_1, \ldots, x_n : \theta_n\} \ni \{a_1 : I_1, \ldots, a_m : I_m\}$ such that any interface in $\Gamma$ occurs in $\text{dom}(\Delta)$. The typing rules are given in Figure 1.

For the operational semantics, we define the sets of term values, heap configurations and states by:

$$
TVals := \{\text{skip}, \text{null} \ | \ a\} \ni V
HConf := \{\emptyset \ | \ (f : v), V\}
\text{States} := S \ni \text{Ints} \times (HConf \times \text{MImps})
$$

If $S(a) = (I, (V, M))$ then we write $S(a) : I$, while $S(a, f)$ and $S(a, M, m)$ stand for $V.f$ and $M_m$ respectively, for each $f$ and $m$.

Given an interface table $\Delta$ such that $I \in \text{dom}(\Delta)$, we let the default heap configuration of type $I$ be

$$
V_I = \{f : v_f \ | \ (\Delta(I), f) = \emptyset\}
$$

where $v_{\text{void}} = \text{skip}$, $v_{\text{null}} = 0$ and $v_{\text{null}} = \text{null}$. The operational semantics of IMJ is given by means of a small-step transition relation

---

**Figure 1. Typing rules for IMJ terms and method-set implementations**
\[(S, i \oplus i') \rightarrow (S, j), \text{ if } j = i \oplus i'\]  
\[(S, \text{if } 0 \text{ then } M \text{ else } M') \rightarrow (S, M')\]  
\[(S, I(a)) \rightarrow (S, a), \text{ if } S(a) : I' \land I' \subseteq I\]  
\[(S, \text{new}(x : I; M)) \rightarrow (S \uplus \{(a, I, V_x, M[a/x])\}, a)\]  
\[(S, \text{a.m}(i)) \rightarrow (S, M[i/x])\], if \(S(a).m = \lambda x.M\)  
\[(S, \text{a.f}(= v)) \rightarrow (S[a \mapsto (V, \forall f \mapsto v), M]), \text{skip}\], if \(S(a) = (V, M)\)  
\[(S, E[M]) \rightarrow (S', E[M'])\], if \((S, M) \rightarrow (S', M')\)
need the full power of the theory but mainly the basic notion of name-permutation. For an element \( x \) belonging to a (nominal) set \( X \) we write \( \nu(x) \) for its name-support, which is the set of names occurring in \( x \). Moreover, for any \( x, y \in X \), we write \( x \sim y \) if there is a permutation \( \pi \) such that \( x = \pi(y) \).

We proceed to define a category of games. The objects of our category will be arenas, which are nominal sets carrying specific type information.

**Definition 4.** An arena is a pair \( (M, \Xi_A) \) where:
- \( M_A \) is a nominal set of moves;
- \( \Xi_A : M_A \rightarrow (\Delta \rightarrow Ints) \) is a nominal typing function;

such that, for all \( m \in M_A \), \( \text{dom}(\Xi_A(m)) = \nu(m) \).

We start by defining the following basic arenas,
- \( 1 = \{*, (v, \emptyset)\} \quad \mathbb{Z} = (\mathbb{Z}, \{(i, \emptyset)\}) \)
- \( I = (\emptyset \cup \{n!\}, \{(n!), \emptyset\} \cup \{(a, a, I)\}) \)

for all interfaces \( I \). Given arenas \( A \) and \( B \), we can form the arena \( A \times B \) by:

\[
M_{A \times B} = \{(m, n) \in M_A \times M_B \mid a \in \nu(m) \cap \nu(n) \}
\]

for each type \( \theta \), we set \( \text{Val}_{\theta} \) to be the set of semantic values of type \( \theta \), given by:

\[
\text{Val}_{\nu = \emptyset} = \text{M}_1, \quad \text{Val}_{\nu = \emptyset} = \Xi_A = \emptyset, \quad \text{Val}_I = \Xi_I.
\]

For each type sequence \( \theta = \theta_1, \ldots, \theta_n \), we set \( \text{Val}_{\theta} = \text{Val}_{\theta_1} \times \cdots \times \text{Val}_{\theta_n} \).

We let a store \( \Sigma \) be a type-preserving finite partial function from names to object types and field assignments, that is, \( \Sigma : \emptyset \rightarrow Ints \times (\text{Files} \rightarrow \text{Val}) \) such that \( |\Sigma| \) is finite and

\[
\Sigma(a) : I \land \Delta(i), f = \theta \implies \Sigma(a), f = v \land \Delta(i) \vdash v \leq \theta,
\]

where the new notation is explained below. First, assuming \( \Sigma(a) = (I', \phi) \), the judgement \( \Sigma(a) : I \) holds iff \( I = I' \) and \( \Sigma(a), f \) stands for \( \phi(f) \). Next we define typing rules for values in store contexts:

\[
v \in \text{Val}_{\nu = \emptyset} \quad \text{Val}_I = \Xi_I \vdash v : \text{null} \quad \Delta(i) \vdash v \leq \theta
\]

and write \( \Delta \vdash v : \theta \) for \( \Delta(i) \vdash v : \theta \) for each legal sequence \( I \subseteq \Delta \).

We let \( \text{Sto} \) be the set of all stores. We write \( \text{dom}(\Sigma(a)) \) for the set of all \( f \) such that \( \Sigma(a), f \) is defined. We let \( \text{Sto}_0 \) contain all stores \( \Sigma \) such that:

\[
\forall a \in \text{dom}(\Sigma), f \in \text{dom}(\Sigma(a)) \quad \Sigma(a), f \in \{*, 0, \text{null}\}
\]

and we call such a \( \Sigma \) a default store.

Given arenas \( A \) and \( B \), plays in \( AB \) will consist of sequences of moves (with store) which will be either moves from \( M_A \cup M_B \), or moves representing method calls and returns. Formally, we define:

\[
M_{A \times B} = M_A \cup M_B \cup \text{Calls} \cup \text{Retns}
\]

where we set \( \text{Calls} = \{ \text{call } a.m(v) \mid a \in A \land v \in \text{Val} \} \) and \( \text{Retns} = \{ \text{ret } a.m(v) \mid a \in A \land v \in \text{Val} \} \).

**Definition 5.** A legal sequence in \( AB \) is a sequence of moves from \( M_{A \times B} \) that adheres to the following grammar (Well-Bracketing), where \( m_A \) and \( m_B \) range over \( M_A \) and \( M_B \) respectively.

\[
L_{A \times B} ::= \epsilon \mid m_A X \mid m_A Y m_B X
\]

\[
X ::= Y \mid \text{call } a.m(v) X
\]

\[
Y ::= \epsilon \mid Y Y \mid \text{call } a.m(v) Y \text{ ret } a.m(v)
\]

We write \( L_{A \times B} \) for the set of legal sequences in \( AB \). In the last clause above, we say that \( \text{call } a.m(v) \) justifies \( \text{ret } a.m(v) \).

To each \( s \in L_{A \times B} \) we assign a polarity function \( p \) from move occurrences in \( s \) to the set \( P_\text{call} = \{O, P\} \). Polarities represent the two players in our game reading of programs: \( O \) is the Opponent and \( P \) is the Proponent in the game. The latter corresponds to the modelled program, while the former models the possible computational environments surrounding the program. Polarities are complemented via \( O = \{P\} \) and \( P = \{O\} \). In addition, the polarity function must satisfy the condition:

- For all \( m_X \in M_X \) (\( X = A, B \)) occurring in \( s \) we have \( p(m_X) = O \) and \( p(m_B) = P \) (O-starting).
- If \( mn \) are consecutive moves in \( s \) then \( p(n) \in \{P\} \). (Alternation)

It follows that there is a unique \( p \) for each legal sequence, namely the one which assigns \( O \) precisely to those moves appearing in odd positions in \( s \).

A move-with-store in \( AB \) is a pair \( m^\Sigma \) with \( \Sigma \in \text{Sto} \) and \( m \in M_{A \times B} \). For each sequence \( s \) of move-with-store we define the set of available names of \( s \) by:

\[
\text{Av}(\epsilon) = \emptyset, \quad \text{Av}(s.m^\Sigma) = \Sigma^*(\text{Av}(s) \cup \nu(m))
\]

where, for each \( X \subseteq A \), we let \( \text{Av}^*(X) = \bigcup \Sigma^*(X) \), with

\[
\Sigma^0(X) = X, \quad \Sigma^{n+1}(X) = \nu(\Sigma(X)).
\]

That is, a name is available in \( s \) just if it appears inside a move in \( s \), or it can be reached from an available name through some store in \( s \). We write \( \Sigma \) for the underlying sequence of moves of \( s \) (i.e. \( \pi_1(s) \)), and let \( \Sigma^s \) denote the prefix relation between sequences. If \( s^m, s^m' \subseteq s \) and \( a \in \nu(m^\Sigma) \cup \nu(s') \) then we say \( a \) is introduced by \( m^\Sigma \) in \( s \).

In such a case, we define the owner of the name \( a \), written \( o(a) \), to be \( p(m) \) (where \( p \) is the polarity associated with \( s \)). For each polarity \( X \in \{O, P\} \) we let

\[
\text{X}(s) = \{a \in \nu(s) \mid o(a) = X\}
\]

be the set of names in \( s \) owned by \( X \).

**Definition 6.** A play in \( AB \) is a sequence of moves-with-store \( s \) such that \( s \) is a legal sequence and, moreover, for all \( s^{m^\Sigma}, s^{m^\Sigma'} \subseteq s \):

- It holds that \( \text{dom}(\Sigma) = \text{Av}(s^{m^\Sigma}) \). (Frugality)
- If \( a \in \text{dom}(\Sigma) \) with \( \Sigma(a) : T \) then:
  - if \( m \in M_X \), for \( X \subseteq \{A, B\} \), then \( I \leq \xi_X(m, a) \);
  - for all \( n^T \in s \), if \( a \in \text{dom}(T) \) then \( T(a) = I \);
  - if \( \Delta(T), m = \theta \rightarrow \theta \) then:
    - if \( m = \text{call } a.m(v) \) then \( \Delta(v) : \theta' \) for some \( \theta' \leq \theta' \);
    - if \( m = \text{ret } a.m(v) \) then \( \Delta(v) : \theta' \) for some \( \theta' \leq \theta' \).
- (Well-classing)
- If \( m = \text{call } a.m(v) \) then \( o(a) \in \text{p}(m) \). (Well-calling)

We write \( P_{A \times B} \) for the set of plays in \( AB \).

---

1 By abuse of notation, we frequently write instead “\( a \) is introduced by \( m \) in \( s \)”.

Recall also that \( \nu(s) \) collects all names appearing in \( s \); in particular, \( \nu(m_a^1 \cdots m_a^n) = \nu(m_1) \cup \nu(m_2) \cup \cdots \cup \nu(m_n) \cup \nu(\Sigma_i) \).
Note above that, because of well-bracketing and alternation, if \( m = \text{ret}\ a.m(v) \) then well-calling implies \( o(a) = p(m) \). Thus, the frugality condition stipulates that names cannot appear in a play in unreachable parts of a store (cf. [17]). Moreover, well-calling ensures that the typing information in stores is consistent and adheres to the constraints imposed by \( A \) and the underlying arenas. Finally, well-calling implements the specification that each player need only call the other player’s methods. This is because calls to each player’s own methods cannot in general be observed and so should not be accounted for in plays.

Given arenas \( A, B, C \), next we define interaction sequences, which show how plays from \( AB \) and \( BC \) can interact to produce a play in \( AC \). The sequences will rely on moves with stores, where the moves come from the set:

\[
M_{ABC} = M_A \cup M_B \cup M_C \cup \text{Calls} \cup \text{Retns}.
\]

The moves will be assigned polarities from the set:

\[
Pol2 = \{ O_L, P_L, P_L, P_L, P_L, O_R, P_R \}.
\]

The index \( L \) stands for “left”, while \( R \) means “right”. The indices indicate which part of the interaction (\( A, B \) or \( C \)) a move comes from, and what polarity it has therein. We also consider an auxiliary notion of pseudo-polarities:

\[
\begin{align*}
\text{OO} &= \{ O_L, O_R \}, & \text{PO} &= \{ P_L, P_L, O_R \}, & \text{OP} &= \{ P_R, O_L, P_R \}. \\
\text{Each polarity has an opposite pseudo-polarity determined by:} & \\
\bar{\text{OO}} &= \{ O_L, O_R \}, & \bar{\text{PO}} &= \{ P_L, P_L, O_R \}, & \bar{\text{OP}} &= \{ P_R, O_L, P_R \}. \\
\text{Finally, each } X \in \{ AB, BC, AC \} \text{ has a designated set of polarities given by:} & \\
p(AB) &= \{ O_L, P_L, P_L, P_L, O_R, P_R \}, & p(BC) &= \{ O_R, P_R, P_R, P_R, O_L, P_L \}, & p(AC) &= \{ O_L, P_L, O_R, P_R \}.
\end{align*}
\]

Note the slight abuse of notation with \( p \), as it is also used for move polarities.

Suppose \( X \in \{ AB, BC, AC \} \). Consider a sequence \( s \) of moves-with-store from \( ABC \) (i.e. a sequence with elements \( m^X \) with \( m \in M_{ABC} \)) along with an assignment \( p \) of polarities from \( Pol2 \) to moves of \( s \). Let \( \sigma[s] \) be the subsequence of \( s \) containing those moves-with-store \( m^X \) of \( s \) for which \( p(m) \subseteq \sigma(X) \). Additionally, we define \( s \upharpoonright X \) to be \( \gamma(s \upharpoonright X) \), where the function \( \gamma \) acts on moves-with-store by restricting the domains of stores to available names:

\[
\gamma(e) = e, \quad \gamma(sm^X) = \gamma(s)(m^{\Sigma(Aw(sm^X))}).
\]

**Definition 7.** An interaction sequence in \( ABC \) is a sequence \( s \) of moves-with-store in \( ABC \) satisfying the following conditions:

* For each \( s \) such that \( s \subseteq s \), \( \text{dom} (\Sigma) = \text{Aw}(s^{X}) \). (Frugality)
* If \( s \subseteq s \) and \( a \in \text{dom}(\Sigma) \) with \( \Sigma(a) = T \) then:
  - if \( m \in M_X \), for \( X \in \{ A, B, C \} \), then \( T \subseteq \chi(m, a) \);
  - for all \( n_T \) in \( s \), if \( a \in \text{dom}(T) \) then \( T(a) = T \);
  - if \( T(a) \neq \emptyset \) then:
    - \( m = \text{call}\ a.m(v) \) then \( T \upharpoonright v : \theta \downharpoonright \delta \) for some \( \delta < \theta \);
    - \( m = \text{ret}\ a.m(v) \) then \( T \upharpoonright v : \theta \downharpoonright \delta \) for some \( \delta < \theta \).
  (Well-calling)
* There is a polarity function \( p \) from move occurrences in \( s \) to \( Pol2 \) such that:
  - For all \( m \in M_X \) (\( X = A, B, C \)) occurring in \( s \) we have \( p(m_A) = O_L, p(m_B) = P_L, p(m_C) = P_R \);
  - If \( mn \) are consecutive moves in \( s \), then \( p(n) \subseteq p(m) \).
(Alternation)

**Lemma 1.** Each \( s \in \text{Int}(ABC) \) has a unique polarity function \( p \).

**Proof.** Suppose \( s \in \text{Int}(ABC) \). We claim that the alternation, well-calling, projecting and well-returning conditions uniquely specify \( p \). Consider the interaction diagram of Figure 3, which we read as an automaton accepting \( s \), for each \( s \in \text{Int}(ABC) \). The edges represent moves by their polarities, while the labels of vertices specify the polarity of the next (outgoing) move. For example, from \( \text{OO} \) we can only have a move \( m \) with \( p(m) \subseteq \{ O_L, O_R \} \), for any \( p \).

We write \( \text{Int}(ABC) \) for the set of interaction sequences in \( ABC \).

Note that, by projecting and well-returning, each return move in \( s \) has a unique justifier. Next we show that the polarities of moves inside an interaction sequence are uniquely determined by the interaction diagram of Figure 3. The diagram can be seen as an automaton accepting \( s \), for each \( s \in \text{Int}(ABC) \). The edges represent moves by their polarities, while the labels of vertices specify the polarity of the next (outgoing) move. By projecting we obtain that the first element of \( s \) is some \( m_A \) and, by alternation, its polarity is \( O_L \). Thus, \( \text{OO} \) is the initial state.

We now use induction on \( |s| \) to show that \( A \) has a unique run on \( s \). The base case is trivial, so suppose \( s = s'm \). By induction hypothesis, \( A \) has a unique run on \( s' \), which reaches some state \( X \). We do a case analysis on \( m \). If \( m \in M_A \cup M_B \cup M_C \) then there is a unique edge accepting \( m \) and, by alternation, this edge must depart from \( X \). If, on the other hand, \( m = \text{call}\ a.m(v) \) then the fact that \( o(a) = p(m) \) gives two possible edges for accepting \( m \). But observe that no combination of such edges can depart from \( X \). Finally, let \( m = \text{ret}\ a.m(v) \) be justified by some \( n \) in \( s' \). Then, by well-bracketing, \( n \) is the justifier of \( m \) in all projections, and hence the edge accepting \( m \) must be the opposite of the one accepting \( n \) (e.g. if \( m \) is accepted by \( O_L \) then \( n \) is accepted by \( P_L \)).

Next we show that interaction sequences project to plays. The projection of interaction sequences in \( ABC \) on \( AB, BC \) and \( AC \)
leads to the following definition of projections of polarities,
\[ \pi_{AB}(X) = X \quad \pi_{AB}(Y) = \text{undef.} \]
\[ \pi_{BC}(X) = \text{undef.} \quad \pi_{BC}(Y) = Y \quad \pi_{BC}(R) = Y \]
\[ \pi_{AC}(X) = X \quad \pi_{AC}(Y) = \text{undef.} \quad \pi_{AC}(R) = Y \]
where \( X, Y \in \{ O, P \} \). We can now show the following.

**Lemma 2.** Let \( s \in \text{Int}(ABC) \). Then, for each \( X \in \{ AB, BC, AC \} \)
and each \( m^s \in s \), if \( p(m) \in p(X) \) then \( \pi_X(p(m)) = p_X(m) \),
where \( p_X \) is the polarity function of \( X \) \( S \).

**Proof.** We show this for \( X = AB \), the other cases are proven similarly, by induction on \( |s| \geq 0 \); the base case is trivial. For the inductive case, if \( m \) is the first move in \( s \) with polarity in \( p(AB) \) then, by projecting, \( m \in M_A \) and therefore \( p(m) = O_L \) and \( p_{AB}(m) = O \), as required. Otherwise, let \( n \) be the last move in \( s \) with polarity in \( p(AB) \) before \( m \). By IH, \( p_{AB}(n) = \pi_{AB}(p(n)) \). Now, by projecting, \( p_{AB}(m) = p_{AB}(n) \) and observe that, for all \( X \in p(n) \), \( \pi_{AB}(X) = \pi_{AB}(p(n)) \), in particular \( \pi_{AB}(p(m)) = \pi_{AB}(p(n)) = p_{AB}(n) = p_{AB}(m) \).

The following lemma formulates a taxonomy on names appearing in interaction sequences.

**Lemma 3.** Let \( s \in \text{Int}(ABC) \). Then,
1. \( \nu(s) = O(s \upharpoonright \gamma, AC) \uplus P(s \upharpoonright \gamma, AB) \uplus P(s \upharpoonright \gamma, BC) \);
2. if \( s = tm^s \) and:
   - \( p(m) \in O_O \) and \( s \upharpoonright \gamma AC = tm^s \upharpoonright m^s \),
   - or \( p(m) \in O_P \) and \( s \upharpoonright \gamma AB = t \upharpoonright m^s \),
   - or \( p(m) \in O_P \) and \( s \upharpoonright \gamma BC = t \upharpoonright m^s \),
then \( \nu(t) \cap \nu(m^s) \subseteq \nu(t) \) and, in particular, if \( m \) introduces name \( a \) in \( t \upharpoonright m^s \) then \( m \) introduces \( a \) in \( s \).

**Proof.** For 1, by definition of interactions we have that these sets are disjoint. It therefore suffices to show the left-to-right inclusion. Suppose that \( a \in \nu(s) \) is introduced in some \( m^s \in s \) with \( p(m) \in PO \) and let \( s \upharpoonright \gamma AB = \cdots \cdots \). If \( a \in \nu(m^s) \) then \( a \in P(s \upharpoonright \gamma AB) \), as required. Otherwise, by Laird’s last set of conditions, \( a \) is copied from the store of the move preceding \( m^s \) in \( s \), a contradiction to its being introduced in \( m^s \). Similarly if \( p(m) \in PO \). Finally, if \( p(m) \in OO \) then we work similarly, considering \( O(s \upharpoonright \gamma AC) \).

For 2, we show the first case, and the other cases are similar. It suffices to show that \( \nu(m^s) \setminus \nu(t) \cap \nu(t) = \emptyset \). Suppose \( a \in \nu(m^s) \setminus \nu(t) \), therefore \( a \in O(s \upharpoonright \gamma AC) \). But then we cannot have \( a \in \nu(t) \) as the latter, by item 1, would imply \( a \in P(s \upharpoonright \gamma AB) \cup P(s \upharpoonright \gamma BC) \).

**Proposition 4.** For all \( s \in \text{Int}(ABC) \), the projections \( s \upharpoonright \gamma AB, s \upharpoonright \gamma BC \), and \( s \upharpoonright \gamma AC \) are plays in \( AB, BC \), and \( AC \) respectively.

**Proof.** By frugality of \( s \) and application of \( \gamma \), all projections satisfy frugality. Moreover, well-classing is preserved by projections. For well-calling, let \( m = \text{call}\_a\_m^s \) be a move in \( s \) and not \( m^s \) be the move introducing \( a \) in \( s \). Suppose \( p(m) \in p(AB) \) and let \( a \in O(s \upharpoonright \gamma AC) \). By well-calling of \( s \), we have that \( a \in O(s \upharpoonright \gamma AC) \). Thus, \( p(n) \in PO \) and, by Lemma 3, \( a \) introduces \( a \) in \( s \upharpoonright \gamma AB \) and therefore \( o_{AB}(m) = P \). Thus, \( p(n) \in PO \) and, by Lemma 3, \( a \) introduces \( a \) in \( s \upharpoonright \gamma AC \) and therefore \( o_{AC}(m) = P \). Thus, by the same lemma, \( a \notin P(s \upharpoonright \gamma AB) \) and hence \( o_{AB}(a) = O \). The cases for the other projections are shown similarly.

In our setting programs will be represented by strategies between arenas. We shall introduce them next after a few auxiliary definitions. Intuitively, strategies capture the observable computational patterns produced by a program.

Let us define the following notion of subtyping between stores.

For \( \Sigma, \Sigma' \subseteq S_{\text{str}} \), \( \Sigma \subseteq \Sigma' \) holds if, for all names \( a \),
\[ \Sigma'(a) : \Sigma' \rightarrow \Sigma' \leq T \wedge \forall f \in \text{dom}(\Sigma(a)).(\Sigma(a).f = \Sigma'(a).f). \]

In particular, if \( a \) is in the domain of \( \Sigma' \), \( \Sigma \) may contain more information about \( a \) because of assigning to \( a \) a larger interface. Accordingly, for plays \( s', s'' \in PA_B \), we say that \( s \) is an O-extension of \( s' \) if \( s \) and \( s'' \) agree on their underlying sequences, while their stores may differ due to subtyping related to O-names. Where such subtyping leads to \( s \) having stores with more fields than those in \( s'' \), \( P \) is assumed to copy the values of those fields. Formally, \( s \leq_O s'' \) is defined by the rules:
\[ s \leq_O s' \quad \Sigma \leq \Sigma' \quad \Sigma \setminus P(sm^s) \subseteq \Sigma' \]
\[ \nu(s) \cup \Sigma \subseteq s' \nu(s) \cup \Sigma' \]
\[ \nu(s) = s' \nu(s) \quad \Sigma = s' \Sigma \]
\[ sm^s \leq s' m^s \]
where \( \Sigma \) extends \( \Sigma' \) by \( T \) if:
1. for all \( a \in \text{dom}(\Sigma) \setminus \text{dom}(\Sigma') \), \( \Sigma(a) = T(a) \);
2. for all \( a \in \text{dom}(\Sigma) \) and \( f \in \text{dom}(\Sigma(a)) \setminus \text{dom}(\Sigma'(a)) \), \( \Sigma(a).f = T(a).f \).

The utility of O-extension is to express semi-technically the fact that the environment of a program may use up-casting to inject in its objects additional fields (and methods) not accessible to the program.

**Definition 8.** A strategy \( \sigma \in AB \) is a non-empty set of even-length plays from \( PA_B \) satisfying the conditions:
1. if \( sm^s \in \sigma \) then \( s \in \sigma \). (Even-prefix closure)
2. if \( sm^s \in \sigma \) and \( sm^t \sim sm^s \). (Determinacy)
3. if \( s \in \sigma \) and \( s \sim t \) then \( t \in \sigma \). (Equivariance)
4. if \( s \in \sigma \) and \( t \leq_O s \) then \( t \in \sigma \). (O-extension)

We write \( \sigma : A \rightarrow B \) when \( \sigma \) is a strategy in \( AB \). If \( \sigma : A \rightarrow B \) and \( \tau : B \rightarrow C \), we define their composition \( \sigma; \tau \) by:
\[ \sigma; \tau = \{ s \mid AC \mid s \in \sigma \upharpoonright \tau \} \]
where \( \sigma \upharpoonright \tau = \{ s \in \text{Int}(ABC) \mid s \upharpoonright AB \in \sigma \wedge s \upharpoonright BC \in \tau \} \).

In definitions of strategies we may often leave the presence of the empty sequence implicit, as the latter is a member of every strategy. For example, for each arena \( A \), we define the strategy:
\[ \text{id}_A : A \rightarrow A = \{ m^s_A m^s_A \in PA_A \} \]

The next series of lemmata allow us to show that strategy composition is well defined.

**Lemma 5.** If \( sm^s \sim sm^t \in \sigma \upharpoonright \tau \), with \( p(m) \notin O_O \) then \( sm^s \sim sm^t \).

**Proof.** For the latter part, if \( s_1 = \pi \cdot s_2 \), then, since \( \pi \cdot (s_2m^t) \in \sigma \upharpoonright \tau \), by former part of the claim we have \( s_1m^s \sim \pi \cdot (s_2m^t) \) so \( s_1m^s \sim s_2m^t \).

Recall that, for any nominal set \( X \) and \( x, y \in X \), we write \( x \sim y \) just if there is a permutation \( \pi \) such that \( x = \pi \cdot y \).
with $s' m^\Sigma = s_m^\Sigma \restriction \gamma AB$ and $s' n^T = s_n^T \restriction \gamma AB$. We therefore have $(s', m^\Sigma) \sim (s', n^T)$ and, trivially, $(s, s') \sim (s, s')$. Moreover, by Lemma 3, $\nu(m^\Sigma) \cap \nu(s) \subseteq \nu(s')$ and $\nu(n^T) \cap \nu(s) \subseteq \nu(s')$ hence, by Strong Support Lemma [26], $s^{m^\Sigma} \sim s^{n^T}$. By Laird’s last set of conditions, the remaining values of $\Sigma, T$ are determined by the last store in $s$, hence $s^{m^\Sigma} \sim s^{n^T}$. □

**Lemma 6.** If $s_1, s_2 \in \sigma \tau$ end in moves with polarities in $p(AC)$ and $s_1 \restriction AC = s_2 \restriction AC$ then $s_1 \sim s_2$.

Proof. By induction on $|s_1 \restriction AC| > 0$. The base case is encompassed in $s_1 = s'_1 m^\Sigma$ with $p(m) \in OO, i = 1, 2$, where note that by IH $m$ will have the same polarity in $s_1, s_2$. Then, by IH we get $s_1 \approx \pi \cdot s_2$, for some $\pi$. Let $s'_1 m^\Sigma = s_1 \restriction AC$, for $i = 1, 2$, so in particular $s'_1 = \pi \cdot s'_2$ and therefore $(s'_1, s'_2) \sim (s'_1, s'_2)$. Moreover, by hypothesis, we trivially have $(m^\Sigma, s'_1) \sim (m^\Sigma, s'_2)$ and hence, by Lemma 3 and Strong Support Lemma [26], we obtain $s'_1 m^\Sigma \sim s'_2 m^\Sigma$ which implies $s_1 \sim s_2$ by Laird’s conditions.

Suppose now $s_1 = s'_1 m^\Sigma, i = 1, 2$, with $(p(m) \in P(AC) \setminus OO$ and the last move in $s'_1$ being the last move in $s'_1$ having polarity in $p(AC)$. By IH, $s'_1 \sim s'_2$. Then, by consecutive applications of Lemma 5, we obtain $s_1 \sim s_2$. □

**Proposition 7.** If $\sigma : A \to B$ and $\tau : B \to C$ then $\sigma; \tau : A \to C$.

Proof. We show that $\sigma; \tau$ is a strategy. Even-prefix closure and equivariance are clear. Moreover, since each $s \in \sigma; \tau$ has even-length projections in $AB$ and $BC$, we can show that its projection in $AC$ is even-length too. For O-extension, if $s \in \sigma; \tau$ and $t \leq s$ with $s = u \restriction \gamma AC$ and $u \in \sigma; \tau$, we can construct $v \in Int\{ABC\}$ such that $t = v \restriction \gamma AC$ and $v \leq u$, where $u \leq 0$ is defined for interaction sequences in an analogous way as for plays (with condition $p(m) = O$ replaced by $p(m) \in OO$, and $p(m) = P$ by $p(m) \in PO \cup OP$). Moreover, $v \restriction \gamma AB \leq u \restriction \gamma AB$ and $v \restriction \gamma BC \leq u \restriction \gamma BC$, so $t \in \sigma; \tau$. Finally, for determinacy, let $s^{m^\Sigma}, s^{n^T} \in \sigma; \tau$ be due to $s_1 s_2 m^\Sigma, s_2 s_2 n^T \in \sigma; \tau$ respectively, where $s_1, s_2$ both end in the last move of $s$. By Lemma 6, we have $s_1 \sim s_2$ and thus, by consecutive applications of Lemma 5, we get $s_1 s_2 m^\Sigma \sim s_2 s_1 n^T$, so $s^{m^\Sigma} \sim s^{n^T}$. □

The above result shows that strategies are closed under composition. We can prove that composition is associative and, consequently, obtain a category of games.  

**Proposition 8.** For all $\rho : A \to B$, $\sigma : B \to C$ and $\tau : C \to D$, $(\rho; \sigma; \tau) = \rho (\sigma; \tau)$.

**Definition 9.** Given a class table $\Delta$, we define the category $\mathcal{G}_\Delta$ having arenas as objects and strategies as morphisms. Identity morphisms are given by $1_{\Delta}$, for each arena $\Delta$.

Note that neutrality of identity strategies easily follows from the definitions and, hence, $\mathcal{G}_\Delta$ is well defined. In the sequel, when $\Delta$ can be inferred from the context, we shall write $\mathcal{G}_\Delta$ simply as $\mathcal{G}$. As a final note, for class tables $\Delta \subseteq \Delta'$, we can define a functor $\Delta / \Delta' : \mathcal{G}_\Delta \to \mathcal{G}_{\Delta'}$ which acts as the identity map on arenas, and sends each $\sigma : A \to B$ of $\mathcal{G}_\Delta$ to:

$$(\Delta / \Delta')(\sigma) = \{ s \in P^\Delta A_B \mid \exists t \in \sigma, s \leq_\Delta t \}$$

where $P^\Delta A_B$ refers to plays in $\mathcal{G}_\Delta$. In the other direction, we can define a strategy transformation:

$$(\Delta' / \Delta)(\sigma) = \sigma \cap P^\Delta A_B$$

which satisfies $\Delta' / \Delta(\Delta / \Delta')(\sigma) = \sigma$.

**4. Soundness**

Here we introduce constructions that will allow us to build a model of IMJ. We begin by defining a special class of strategies. A strategy $\sigma : A \to B$ is called evaluated if there is a function $f_\sigma : M_A \to M_B$ such that:

$$\sigma = \{ m^\Sigma_A m^B_B \in P_A B \mid m^B_B = f_\sigma m_A \} .$$

Note that equivariance of $\sigma$ implies that, for all $m_A \in M_A$ and permutations $\pi$, it holds that $\pi \cdot f_\sigma m_A = f_\sigma (\pi \cdot m_A)$. Thus, in particular, $\nu(f_\sigma m_A) \subseteq \nu(m_A)$.

Recall that, for arenas $A$ and $B$, we can construct a product arena $A \times B$. We can also define projection strategies:

$$\pi_1 : A \times B \to A = \{ (m_A, m_B) \in P(A \times B) A \}$$

and, analogously, $\pi_2 : A \times B \to B$. Note that the projections are evaluated. Moreover, for each object $A$, $![A] = \{ m^\Sigma_A m^B_B \in P_A A \}$ is the unique evaluated strategy of type $A \to 1$.

Given strategies $\sigma : A \to B$ and $\tau : A \to C$, with $\tau$ evaluated, we define:

$$\langle \sigma, \tau \rangle : A \times B \times C = \{ m^\Sigma_A \sigma(m_B, f_\tau (m_A)) m^\Sigma_B \mid m^\Sigma_A, m^\Sigma_B \}$$

where we write $s[m^\Sigma_B]$ for the sequence obtained from $s$ by replacing any occurrences of $m_B$ in it by $m^\Sigma_B$ (note that there can be at most one occurrence of $m_B$ in $s$).

The above structure yields products for evaluated strategies.

**Lemma 9.** Evaluated strategies form a wide subcategory of $\mathcal{G}$ which has finite products, given by the above constructions. Moreover, for all $\sigma : A \to B$ and $\tau : A \to C$ with $\tau$ evaluated, $\langle \sigma, \tau \rangle = \sigma ; \tau$.

Using the above result, we can extend pairings to general $\sigma : A \to B$ and $\tau : A \to C$ by:

$$\langle \sigma, \tau \rangle = A^{(\sigma; \tau)} B \times A B^{(\tau; \sigma)} C$$

where $\cong$ is the isomorphism $\langle \sigma, \tau \rangle$. The above represents a notion of left-pairing of $\sigma$ and $\tau$, where the effects of $\sigma$ precede those of $\tau$. We can also define a left-tensor between strategies:

$$\sigma \times \tau = A \times B \times C^{(\sigma; \tau)} B^{\sigma; \tau} C^{\tau; \sigma}$$

for any $\sigma : A \to A'$ and $\tau : B \to B'$.

**Lemma 10.** Let $\tau' : A' \to A$, $\sigma : A \to B_1 \times B_2$, $\sigma_1 : B_1 \to C_1$ and $\sigma_2 : B_2 \to C_2$, with $\tau$ and $\tau'$ evaluated. Then $\tau'(\sigma, \tau) ; (\sigma_1, \pi_2, \sigma_2) = (\tau'\sigma, \sigma, \tau_1, \sigma_2)$.

Proof. The result follows from the simpler statements:

$$\tau ; (\sigma, 1 \Delta) = (\tau; \sigma, \tau), \quad (\sigma, 1 \Delta ; (\sigma', \pi_2, \sigma_2) = (\sigma; \sigma, \tau, \tau_1, \sigma_2), \quad$$

for all appropriately typed $\sigma, \sigma'$, $\tau$, with $\tau$ evaluated, and Lemma 9. □

An immediate consequence of the above is:

$$A = A^{(\sigma; \tau)} B \times (\sigma; \tau) B^{\sigma; \tau} C \times C^{\tau; \sigma}$$

More generally, Lemma 10 provides us with naturality conditions similar to those present in Freyd categories [25] or, equivalently, categories with monadic products [22].

We also introduce the following weak notion of coproduct. Given strategies $\sigma, \tau : A \to B$, we define:

$$(\sigma, \tau) : Z \times A \to B = \{ (1, m^\Sigma_A m^B_B s) \in \} \cup \{ (0, m^\Sigma_A m^B_B s) \in \}$$
Setting $i : 1 \to Z = \{ * i \}$, for each $i \in Z$, we can follow the following.

**Lemma 11.** For all strategies $\sigma' : A' \to A$ and $\sigma, \tau : A \to B$,
- $\langle ;, \emptyset, \emptyset; [\sigma, \emptyset]; [\tau, \emptyset] \rangle = \sigma$;
- if $\sigma'$ is evaluated then $(1, \emptyset, \emptyset; [\sigma', \emptyset]; [\tau, \emptyset]) = \sigma'$.

Method definitions in IMJ induce a form of evaluation:

$$\bigwedge_{i=1}^{n}(\Delta \Gamma \uplus \{ \delta_i \} : M_i : \theta_i) \quad \Theta = \{ m_0, \delta_i \to \theta_i | 1 \leq i \leq n \} \land M = \{ \lambda \Sigma \delta_i, M_i | 1 \leq i \leq n \}$$

the modelling of which requires some extra semantic machinery. Traditionally, in call-by-value game models, evaluation leads to ‘effectless’ strategies, corresponding to higher-order value terms. In our case, higher-order values are methods, manifesting themselves via the objects they may inhabit. Hence, evaluation necessarily passes through generation of fresh object names containing these values. These considerations give rise to two classes of strategies introduced below.

We say that an even-length play $s \in P_{AB}$ is **total** if it is either empty or $s = m_0^2 \Sigma m_1^2 \Sigma^s$ and:
- $\Gamma \vdash \Sigma \vdash \tau \rightarrow \forall \nu (m_0 m_1 \Sigma)$.
- if $s' = m_0^2 \Sigma m_1^2 \Sigma^t$ and $a \in \text{dom}(\Sigma) \land \nu (\gamma (m_0^2 \Sigma m_1^2 \Sigma^t \Sigma^s))$, then $\Sigma_0 \vdash \Sigma_{\gamma (m_0^2 \Sigma m_1^2 \Sigma^t \Sigma^s)} \in P_{AB}$, then $a \notin \nu (m_1)$ and $\Gamma \vdash \Sigma$.

We write $P_{AB}^{\uparrow}$ for the set of total plays in $P_{AB}$. Thus, in total plays, the initial move $m_0$ is immediately followed by a move $m_1$, and the initial store $\Sigma$ is invisible to $P$ because $P$ cannot use its names nor their values. A strategy $\phi : A \to B$ is called **thread-independent** if it consists of total plays and satisfies the conditions: $^3$

- for all $m_0 \Sigma m_1^2 \Sigma^s \in P_{AB}$ there is $m_0 \Sigma m_1^2 \Sigma^s \in \phi$;
- if $m_0 \Sigma m_1^2 \Sigma^t \Sigma^s \in \phi$ then $m_0 \Sigma m_1^2 \Sigma^t \Sigma^s \in \phi$, for $\Sigma_0 \vdash \Sigma$.
- if $m_0 \Sigma m_1^2 \Sigma^t \Sigma^s \in \phi$, then there is $\Sigma$.

Thus, single-threaded strategies reply to every initial move $m_0 \Sigma$ with a move $m_1$, which depends only on $m_0$ (i.e., $P$ does not read before playing). Moreover, $m_0 \Sigma$ does not change the values of $\Sigma$ ($P$ does not write) and may introduce some fresh objects, albeit with default values. Finally, plays of single-threaded strategies consist of just one thread, where a thread is a total play in which there can be at most one call to names introduced by its second move. Conversely, given a total play starting with $m_0 \Sigma m_1^2 \Sigma^s$, we can extract its threads by tracing back for each move in $s$ the method call of the object $\nu (\Gamma)$, if $\nu (\Gamma)$ is related to $\Gamma$. Formally, for each total play $s = m_0 \Sigma m_1^2 \Sigma^s$, with $\gamma (s') > 0$, the **threader** move of $s$, written thr$(s)$ is given by induction:

- if $m_0 \Sigma m_1^2 \Sigma^t \Sigma^s$, then $\text{thr}(s')$,
- if $m_0 \Sigma m_1^2 \Sigma^t \Sigma^s$, then $\text{thr}(s')$.

where the restriction retains only those moves $n^T$ of $s'$ such that thr$(n^T) = m_0^2 \Sigma$. We extend this to the case of $|s| = 2$ by setting $|s| = 2$. Finally, we call a total play $s \in P_{AB}$ **thread-independent** if for all $s' m_0^2 \Sigma \subseteq \text{even}s$ with $\gamma (s') > 2$:
- if $\gamma (m_0^2 \Sigma) \subseteq \gamma (s')$ and $\nu (\gamma (m_0^2 \Sigma))$ then $\Sigma (a) = \Gamma (a)$.
- if $s$' ends in some $n^T$ and $a \in \text{dom}(\Sigma) \land \nu (\gamma (m_0^2 \Sigma))$ then $\Sigma (a) = \Gamma (a)$.

We write $P_{AB}^{\uparrow \downarrow}$ for the set of thread-independent plays in $AB$.

**Lemma 12.** $\phi^\downarrow$ is a strategy, for each single-threaded $\phi$.

**Proof.** Equivariance, Even-prefix closure and O-extension follow from the corresponding conditions on $\phi$. For determinacy, if $m_0^2 \Sigma m_1^2 \Sigma^s \in \phi \land |s| > 0$ then, using determinacy of $\phi$ and the fact that $P$-moves do not change the current thread, nor do they modify or use names from other threads, we can show that $m_0^2 \Sigma m_1^2 \Sigma^s \in \phi$. We say that a strategy $\sigma$ is **thread-independent** if $\sigma \equiv \uparrow$ for some single-threaded strategy $\sigma$. Thus, thread-independent strategies do not depend on initial stores and behave in each of their threads in an independent manner. Note in particular that evaluated strategies are thread-independent (and single-threaded).

**Lemma 13.** Let $\sigma : A \to B$ and $\tau : A \to C$ be strategies with $\tau$ thread-independent. Then, $\langle \sigma, \tau; \pi_1 \tau \pi_2 \sigma \rangle = \Gamma : A \times B \times C$.

**Proof.** The former claim is straightforward. For the latter, we observe that the initial effects of $\sigma$ and $\tau$ commute: on initial move $m_0 \Sigma^s \tau \rightarrow m_0 \Sigma^s \tau$, does not read the store updates that $\sigma$ includes in its response $m_0^2 \Sigma^s \tau$, while $\sigma$ cannot access the names created by $\tau$ in its second move $m_0^2 \Sigma^s \tau$.

It is worth noting that the above lemma does not suffice for obtaining categorical products. Allowing thread-independent strategies to create fresh names in their second move breaks universality of pairings. Considering, for example, the strategy:

$$\sigma : 1 \to \Sigma \times I = \{ * (a, a) \Sigma \in P_{1 \times 2} | \Sigma \in \text{Sto}_0 \}$$

we can see that $\sigma \neq \langle \sigma; \pi_1 \tau \pi_2 \sigma \rangle$, as the right-hand-side contains plays of the form $(a, b)^T$ with $a \neq b$.

We can now define an appropriate notion of exponential for our games. Let us assume a translation assigning an arena $\overrightarrow{\Theta}$ to each type scheme $\overrightarrow{\theta}$. Moreover, let $\overrightarrow{I}$ be an interface such that $\Delta (\overrightarrow{I}) \land \overrightarrow{M eths} = \{ m_0 : \overrightarrow{\theta}_0 \to \theta_1, \ldots, m_n : \overrightarrow{\theta}_n \to \theta_n \}$

where $\overrightarrow{\theta}_k : \theta_{k1}, \ldots, \theta_{km_k}$, for each $i$. For any arena $A$, given single-threaded strategies $\phi_1, \ldots, \phi_n : A \to I$ such that, for each $i$, if $m_0^2 \Sigma^s \Sigma^t \Sigma^s \in \phi_i$ then $\Gamma \notin \nu (\Sigma) \land T (\sigma) : A \times \{ \text{call} a. m (\overrightarrow{\iota}) \in s \Rightarrow m = m_0 \}$, we can group them into one single-threaded strategy:

$$(\langle \phi_1, \ldots, \phi_n \rangle) : A \to I = \bigcup_{* i=1}^{n} \phi_i$$

Note that the $a$ above is fresh for each $m_0^2 \Sigma^s$ (i.e. $a \notin \nu (m_0^2 \Sigma^s)$).
Let now \(\sigma_1, \ldots, \sigma_n\) be strategies with \(\sigma_i : A \times [\vec{b}] \rightarrow [\theta_i]\). For each \(i\), we define the single-threaded strategy \(\Lambda(\sigma_i) : A \rightarrow I\):
\[
\Lambda(\sigma_i) = \{m_A \cdot \gamma(\alpha, \vec{b})_{\sigma_i} \in P_{\text{Ax}} | \gamma((m_A, \vec{b})_{\sigma_i}) \in \sigma_i \}
\]
where \(a \notin \nu(\Sigma, \vec{v}, \vec{s}, \Sigma')\) and \(T(a) : I\). By definition, \(\Lambda(\sigma_i)\) is single-threaded. Therefore, setting
\[
\Lambda(\sigma_1, \ldots, \sigma_n) = \langle\langle \Lambda(\sigma_1), \ldots, \Lambda(\sigma_n) \rangle\rangle^1 : A \rightarrow I,
\]
we obtain a thread-independent strategy implementing a simultaneous currying of \(\sigma_1, \ldots, \sigma_n\). In particular, given translations \([M_i]\) for each method in a method-set implementation \(M\), we can construct:
\[
[M] : [\Gamma] \rightarrow I = \Lambda([M_1], \ldots, [M_m]).
\]

Finally, we define evaluation strategies \(ev_{m_i} : I \times [\vec{b}] \rightarrow [\theta_i]\) by (taking even-length prefixes of):
\[
ev_{m_i} = \{ (\alpha, \vec{b})_{\sigma_i} \in \text{call} \cdot a.m(\vec{v})_{\sigma_i} \text{ret a.m}(\vec{v})_{\sigma_i} P_{\text{Ax}} | \gamma((\alpha, \vec{b})_{\sigma_i}) \in \sigma_i \}
\]
where \(A = (I \times [\vec{b}])[\theta_i]\). We can now show the following natural mapping from groups of strategies in \(A \times [\vec{b}] \rightarrow [\theta_i]\) to thread-independent ones in \(A \rightarrow I\).

**Lemma 14.** Let \(\sigma_1, \ldots, \sigma_n\) be as above, and let \(\tau : A' \rightarrow A\) be evaluated. Then:
- \(\Lambda(\sigma_1, \ldots, \sigma_n) \circ \tau\), \(\tau \circ \Lambda(\sigma_1, \ldots, \sigma_n)\), and \(\sigma_i \circ \tau\) are the same plays. We can now define the semantic translation of \(\tau\) as:
\[
\tau \in \text{domain}(\text{thread-independent}) \rightarrow I
\]

Apart from dealing with exponentials, in order to complete our translation we need also to address the appearance of \(x : I\) in the rule:
\[
\Gamma, x : I \vdash M : \Theta \quad \rightarrow \quad \Gamma, x \vdash \text{new}(x : I ; M) \quad \Delta_{\text{Methods}} = \Theta
\]
where \(\Delta_{\text{Methods}} = \Theta\)

Recall that
\[
[M] : [\Gamma] \times I \rightarrow I
\]

for each field \(x\). Thus, object creation involves creating a pair of names \((a', a)\) with \(a : I\) and \(a' : I\), where \(a\) is the name of the object we want to return. The name \(a'\) serves as a store where the handle of the method implementations, that is, the name created by the second move of \([M]\), will be passed. The strategy \(\nu_{\text{new}}\), upon receiving a request \(\text{call} \cdot a.m(\vec{v})_{\sigma_i}\), simply forwards it to the respective method of \(a', a'\) and, once it receives a return value, copies it back as the return value of the original call.

**Definition 10.** The semantic translation is given as follows:
- Contexts \(\Gamma \in \{ x_1 : \theta_1, \ldots, x_n : \theta_n \} \cup \{ a_1 : I, \ldots, a_m : I \}\) are translated into arenas by
\[
[\Gamma] = [\theta_1] \times \cdots \times [\theta_n] \times (#(I_1, \ldots, I_m), [I_1]_{[M_1]}, \ldots, [I_m]_{[M_m]}).
\]

In order to prove that the semantics is sound, we will also need to interpret terms inside state contexts. Let \(\Gamma \vdash M : \Theta\), with \(\Gamma = \tau_1 \cup \tau_2\), where \(\tau_1\) contains only variables and \(\text{dom}(\tau_2) = \text{dom}(S)\). A term-in-state-context \((S, M)\) is translated into the strategy:
\[
[\Gamma \vdash (S, M)] = [\Gamma_1]_{[S_1]} \times [\Gamma_1] \times [\Gamma_1]_{[M_1]} \times [\Gamma_1]_{[M_2]} \times [\Gamma_1]_{[M_3]} \times [\Gamma_1]_{[M_4]} \times [\Gamma_1]_{[M_5}, \ldots, [\Gamma_1]_{[M_n]}.
\]
where \([\text{void}] = 1\) and \([\text{Int}] = [\text{Z}]\) and \([\text{I}] \approx [\text{Z}]\).

Terms are translated as in Figure 4 (top part).

In order to prove that the semantics is sound, we will also need to interpret terms inside state contexts. Let \(\Gamma \vdash M : \Theta\), with \(\Gamma = \tau_1 \cup \tau_2\), where \(\tau_1\) contains only variables and \(\text{dom}(\tau_2) = \text{dom}(S)\). A term-in-state-context \((S, M)\) is translated into the strategy:
\[
[\Gamma \vdash (S, M)] = [\Gamma_1]_{[S_1]} \times [\Gamma_1] \times [\Gamma_1]_{[M_1]} \times [\Gamma_1]_{[M_2]} \times [\Gamma_1] \times [\Gamma_1]_{[M_3]} \times [\Gamma_1]_{[M_4]} \times [\Gamma_1]_{[M_5}, \ldots, [\Gamma_1]_{[M_n]}.
\]
where \([\text{void}] = 1\) and \([\text{Int}] = [\text{Z}]\) and \([\text{I}] \approx [\text{Z}]\).

The first stage, \([S_1]\), creates the objects in \(\text{dom}(S)\) and implements their methods. The second stage of the translation, \([S_2]\), initialises the fields of the newly created objects.

In the rest of this section we show soundness of the semantics. Let us call \(\text{NEW}, \text{FIELDUP}, \text{FIELDAC}\) and \(\text{METHODCL}\) respectively the transition rules in Figure 2 which involve state. Given a rule \(\tau\), we write \((S, M) \rightarrow (S', M')\) if the transition \((S, M) \rightarrow (S', M')\) involves applying \(\tau\) and context rules.

**Proposition 15 (Correctness).** Let \((S, M)\) be a term-in-state-context and suppose \((S, M) \rightarrow (S', M')\).
1. If the transition \(\tau\) is not stateful then \([M] = [M']\).
2. If \(\tau\) is one of \(\text{FIELDAC}\) or \(\text{FIELDUP}\) then \([S_2] \cup (\text{Id} \times \#(\vec{I})]) = [S'_2] \cup (\text{Id} \times \#(\vec{I})]) = [M'_2].\)
3. If \(\tau\) is one of \(\text{METHODCL}\) or \(\text{NEW}\) then \(\langle[S, M]\rangle = \langle[S', M']\rangle\).

**Proof.** Claim 1 is proved by using the naturality results of this section. For the let construct, we show by induction on \(M\) that \([M_{\text{if}}/x] = (\text{Id} \times [v])\) and \([M]\). For we use the following properties of field assignment and access:
\[
\begin{align*}
\text{assign}_1 \cup \text{assign}_2 & = (\text{assign}_1 \cup \text{assign}_2) : I \times [\theta] \rightarrow [\theta]\n\text{assign}_1 \cup \text{assign}_2 & = (\text{Id} \times \text{assign}_2) : I \times [\theta] \times [\theta] \rightarrow [\theta]\n\text{assign}_1 \cup \text{assign}_2 & = (\text{Id} \times \text{assign}_1) : I \times [\theta] \times [\theta] \rightarrow [\theta]\n\end{align*}
\]
which are easily verifiable (the former one states that assigning a field value and accessing it returns the same value; the latter that two assignments in a row have the same effect as just the last one). The final claim follows by showing that the diagrams below
\[
\begin{align*}
\end{align*}
\]

\[\text{Note that } x \text{ may appear free in } M; \text{ it stands for the keyword this of Java.}\]
\[\text{Here we omit wrapping } [M] \text{ inside } \Delta', \text{as well as wrapping the whole } [\text{new}(x : I,A)] \text{ in } \Delta'/\Delta, \text{ for conciseness.}\]
• \[ \Gamma \vdash x_i : \theta_i \] 
• \[ \Gamma \vdash \text{skip} : \text{void} \] 
• \[ \Gamma \vdash \text{let } x = M'/\theta : \Gamma ] = \[ \Gamma ] \mid \Pi_M \mid \Gamma \vdash \text{null} : \text{I} \] 
• \[ \Gamma \vdash \text{i : int} \] 
• \[ \Gamma \vdash \text{let } x = M' \in M : \theta : \[ \Gamma ] \mid \Pi_M \mid \Gamma \vdash \text{null} \] 
• \[ \Gamma \vdash (I) M : \Gamma ] = \[ \Gamma ] \mid \Pi_M \mid \Gamma \vdash \text{null} \] 
• \[ \Gamma \vdash \text{M + M' : int} \] 
• \[ \Gamma \vdash \text{M = M' : int} \] 
• \[ \Gamma \vdash \text{if } \theta : \[ \Gamma ] \mid \Pi_M \mid \Gamma \vdash \text{null} \] 
• \[ \Gamma \vdash \text{M.f := M' : void} \] 
• \[ \Gamma \vdash \text{M.f : } \theta : \[ \Gamma ] \mid \Pi_M \mid \Gamma \vdash \text{null} \] 
• \[ \Gamma \vdash \text{M.m(M' : M} : \theta : \[ \Gamma ] \mid \Pi_M \mid \Gamma \vdash \text{null} \] 

![Figure 4. The semantic translation of IMJ.](image)

commute (we write \( A \) for \( \Gamma ] \mid \Pi_M \mid \Gamma \vdash \text{null} \),

\[
\begin{align*}
\vec{M} \text{ to } \vec{M} & \text{ to } \vec{M} \\
\Gamma \vdash A \text{ to } \vec{M} & \text{ to } \vec{M} \\
\Gamma \vdash \vec{M} \text{ to } A & \text{ to } \vec{M} \\
\vec{M} \text{ to } A & \text{ to } \vec{M} \\
\end{align*}
\]

with \( \delta = (\text{id}, \text{id}) \). The former diagram says that, assigning method implementations \( \vec{M} \) to object stores \( \vec{a} \) and calling \( M_i \) on some method \( m \) is the same as assigning \( \vec{M} \) to \( \vec{a} \) and evaluating instead a new copy of \( M_i \) on \( m \). The reason the diagram commutes is that the copy of \( M_i \) differs from the original just in the handle name (the one returned in the codomain of \( [M_i] \)), but the latter is hidden via composition with \( \text{env}_m \). The latter diagram stipulates that if we create \( \vec{a} \) with methods \( \vec{M} \), then calling \( a_i \) on \( m \) is the same as calling \( M_i \) on \( m \). The latter holds because of the way that \( \text{env}_m \) manipulates calls inside the interaction, by delegating calls to methods of \( a_i \) to \( M_i \).

**Proposition 16 (Computational Soundness).** For all \( \vdash M : \text{void} \), if \( M \not\downarrow \) then \( [M] = \{*\} \) (i.e., \( [M] = \{\text{skip}\})).

Proof. This directly follows from Correctness.

**Proposition 17 (Computational Adequacy).** For all \( \vdash M : \text{void} \), if \( [M] = \{*\} \) then \( M \not\downarrow \).

Proof. Suppose, for the sake of contradiction, that \( [M] = \{*\} \) and \( M \not\downarrow \). We notice that, by definition of the translation for blocking constructs (castings and conditionals may block) and due to Correctness, if \( M \not\downarrow \) were due to some reduction step being blocked then the semantics would also block. Thus, \( M \not\downarrow \) must be due to divergence. Now, the reduction relation restricted to all rules but MethodCL is strongly normalising, as each transition decreases the size of the term. Hence, if \( M \) diverges then it must involve infinitely many MethodCL reductions and our argument below shows that the latter would imply \( [M] = \{\varepsilon\} \).

For any term \( \Gamma \vdash N : \theta \) and \( a \in A \setminus \text{dom}(\Gamma) \), construct \( \Gamma_a \vdash N_a \) where \( \Gamma_a = \Gamma \cup \{a : \text{Var}_1\} \) by recursively replacing each subterm of \( N \) of the shape \( N'.m(N) \) with \( a.f := (a.f + 1); N'.m(N) \).
Var₁ is an interface with a sole field f : int. Observe that each s ∈ [Γ₁ + N₀] induces some s' ∈ [Γₐ₁ + Nₐ₁] such that a appears in s' only in stores (and in a single place in the initial move) and O never changes the value of a.f, while P never decreases the value of a.f. We write [Γ₁ + N₀] for the subset of [Γ₁ + N₀] containing precisely these plays. Then, take M₀ to be the term let z = new(x : Var₁) in (M₀[x/a]; x.f), where x a fresh variable. Because ss ∈ [M₀], we get s_j ∈ [M₀] for some j ∈ Z.

Consider now the infinite reduction sequence of (θ, M). It must have infinitely many MethodCL steps, so suppose (θ, M) ---→ (S, M') contains j + 1 such steps. Then, we obtain (θ, M₀) ---→ (Sₐ, Mₐ') by sₐ(a.f) = j + 1. By Correctness, we have that s_j ∈ [Sₐ, Mₐ']; a.f] = [Sₐ]; (i.d × #); [(Mₐ'); a.f].

Since in (Mₐ')ₐ the value of a cannot decrease, and its initial value is j + 1 (as stipulated by Sₐ), we reach a contradiction.

5. Full Abstraction

Recall that, given plays s, s’, we call s an O-extension of s’ (written s ≤ₕ O s’) if s, s’ are identical except the type information regarding O-names present in stores: the types of O-names in s may be subtypes of those in s’. We shall write s ≤ₕ s’ for the dual notion involving P-names, i.e., s ≤ₕ s’ if s, s’ are the same, but the types of P-names in s’ may be subtypes of those in s. Then, given X ∈ {O, P} and fixed A, B, let us define clx(s) = {s’ ∈ PₐB | s ≤ₕ s’} and clx(σ) = Uₜ∈clx(s). We write PₐB[clx] for P₁[clx]. A play will be called complete if it is of the form m₁Y₁m₂Y₂.

Next we establish a definability result stating that any complete play (together with other plays implied by O-closure) originates from a term.

Lemma 18 (Definability). Let s ∈ Pₐ₁[clx] be a complete play.

There exists ∆' ⊇ ∆ and ∆'[Γ] = M : 0 such that [∆'[Γ] ⊢ M : 0] clₐ(s).

Proof. The argument proceeds by induction on |s|. For s = ε, any divergent term suffices. For example, one can take ∆’ = ∆ ⊕ [Div → (m : void → void)], and pre-compose any term of type (m : Σ₁ → Σ₂) with new(x : Div; m : Σ₁(m)(m)).

Suppose s = q[s]s₁. Then the second move can be a question or an answer. We first show how to reduce the former case to the latter, so that only the latter needs to be attacked directly.

Suppose s’ = q’[s]s₁s₉, where o : Σ₁ and ∆’(I) : T → I₀. Consider ∆’ = ∆ ⊕ Σ₉[I → I₀] and the following play from Pₐ₁[clx]:

s’ = q’[s]s₁s₉s₂s₃s₄s₅s₆s₇s₈s₉

where p ∉ o(s), Σ₈ = Σ₁ ⊕ Σ₂, Σ₃ = {p ⊢ (Tᵢₙ, m : Iₖ → Iᵢₖ)} and s₈ is the same as s₉ except that each store is extended by Σ₈. If ∆’[Γ] = M : T satisfies the Lemma for s’ then, s, one can take

let x₀ = M’ in x₀.m(y.m(x₀.f)), where y refers to o, i.e., y is of the shape x.f, where x ∈ dom Γ and f is a sequence of fields that points at o in Σ₈.

Thanks to the reduction given above we can now assume that s ∈ Pₐ₁[clx] is non-empty and

s = q[s]s₁s₂s₃s₄s₅s₆s₇s₈s₉,

where s₉ is an answer. We are going to enrich s in two ways so that it is easier to decompose. Ultimately, the decomposition of s will be based on the observation that the mₚ₁, ..., mₚ₂k segment can be viewed as an interleaving of threads, each of which is started by a move of the form call p for some P-name p. A thread consists of the starting move and is generated according to the following two rules:

- mₚₙ begins to the thread of mₚₙ₋₁ and every answer-move belongs to the same thread as the corresponding question-move.
- The first transformation of s brings forward the point of P-name creation to the second move. In this way, threads will never create objects and, consequently, it will be possible to compose without facing the problem of object fusion.

Suppose Pₙ = pₙ and p₁ : Pₙ₁. Let ∆ₙ = ∆ ⊕ (Iₙ → Iₙ). Consider s’ = {sₙ.q}Σₙpₙ₀mₙ₁,...mₙ₂k, where

Σₙ = Σₚ ⊕ {n → (Iₙ, null)} and Σₚ = Σₚ ⊕ {n → (Iₙ, pₙ)} ⊕ (p₁ → (Iₙ₁, null) | Σₚ(p₁) undefined, p₁ ∈ P(s)).

Let Γ’ = (xₙ : Iₙ) ⊔ Γ. Observe that s’ ∈ Pₐ₁[clx].

The second transformation consists in storing the unfolding play in a global variable. It should be clear that the recursive structure of types along with the ability to store names is sufficient to store plays in objects. Let Iₚ play be a signature that makes this possible. This will be used to enforce the intended interleaving of threads after their composition (in the style of Innocent Factorization [5]). Let ∆” = ∆ ⊕ Σ₁[History → play : Iₚ play] and ∆”’ = {xₙ : History} ⊔ Γ. Consider

s”’ = (h, n, q)q”’m”’₁m”’₂...m”’₂tk,

where

Σ”’ = Σ” ⊔ {h → (History, play → null)},

Σ”’₂tk = Σ”’₂tk ⊔ {h → (History, play → s”’ₙ ≤ₕ m”’₂tk)},

Σ”’₂tk+₁ = Σ”’₂tk+₁ ⊔ {h → (History, play → s”’ₙ ≤ₕ m”’₂tk+₁)}.

Observe that s”’ ∈ Pₐ₁[clx].

Now we shall decompose mₚ₁,...,mₚ₂k into threads. Recall that each of them is a subsequence of s” of the form

call p.m(u)Σₑ t ret p.m.v(Σₑ),

where the segment t contains moves of the form call o or ret o for some o ∈ O(s). We would now like to invoke the IH for each thread but, since a thread is not a play, we do so for the closely related play (h, n, q, u)Σₑ t” v”’r. Let us call the resultant term Mₚₙₐ,u,Σₑ. Next we combine terms related to the same p : Iₚ into an object definition by

Mₚₙₐ,u,Σₑ = new(x : Iₚ, m : λₚᵥ.case(u, Σₑ[Mₚₙₐ,u,Σₑ]).

The case statement, which can be implemented in IMJ using nested ifs, is needed to recognize instances of u and Σₑ that really occur in threads related to p. In such cases the corresponding term Mₚₙₐ,u,Σₑ will be run. Otherwise, the statement leads to divergence.

The term M for s can now be obtained by taking

let xₙ = new(x : Iₚ) ;

let xₙ = new(x : History) ;

let xₙ = Mₚₙᵢ, in

assert(q”’); xₙ.f₁ = xₙ.p₁; make(Σ”’ᵢ); play(m₁);

where xₙ = Mₚₙᵢ represents a series of bindings (one for each P-name pᵢ ∈ P(s)), assert(h, n, q)q”’ is a conditional that converges if and only if the initial values of free Σ identifiers as well as values accessible through them are consistent with q and Σₑ respectively, make(Σ”’ᵢ) is a sequence of assignments that set values to those specified in Σ”’ᵢ (up-casts need to be performed to ensure typability) and play(m₁) is skip, i, null or, if m₁ is a name,
Proof. The preceding result implies that $M_1 \simeq M_2$ if and only if $\text{cl}_P([M_1]_{\text{comp}}) = \text{cl}_P([M_2]_{\text{comp}})$. We show that this implies $[M_1]_{\text{comp}} = [M_2]_{\text{comp}}$. Let $s \in [M_1]_{\text{comp}}$. By $\text{cl}_P([M_1]_{\text{comp}}) = \text{cl}_P([M_2]_{\text{comp}})$, it must be the case that $s \in \text{cl}_P([M_2]_{\text{comp}})$, i.e., there exists $s' \in [M_2]_{\text{comp}}$ such that $s \in \text{cl}_P(s')$. Again, by $\text{cl}_P([M_1]_{\text{comp}}) = \text{cl}_P([M_2]_{\text{comp}})$, it follows that $s' \in [M_1]_{\text{comp}}$, i.e., there exists $s'' \in [M_1]_{\text{comp}}$ such that $s'' \in \text{cl}_P(s'')$. So, we have $s \in \text{cl}_P(s')$ and $s' \in \text{cl}_P(s'')$, which implies $s \in \text{cl}_P(s'')$. However, $s, s'' \in [M_1]_{\text{comp}}$, so $s \in \text{cl}_P(s'')$ entails $s = s''$. Hence, $s \in \text{cl}_P(s')$ and $s' \in \text{cl}_P(s)$, and $s = s'$ follows. Because $s' \in [M_2]_{\text{comp}}$, we showed $s \in [M_2]_{\text{comp}}$. The other inclusion is derived analogously.

References


