Game Semantics for Interface Middleweight Java

ANDRZEJ S. MURAWSKI, University of Oxford
NIKOS TZEVELEKOS, Queen Mary University of London

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.

CCS Concepts: • **Theory of computation** → **Denotational semantics**;

Additional Key Words and Phrases: Full Abstraction, Game Semantics, Contextual Equivalence, Java

ACM Reference Format:
Andrzej S. Murawski and Nikos Tzevelekos. 2020. Game Semantics for Interface Middleweight Java. 1, 1 (October 2020), 52 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The last two and half decades have seen *game semantics* emerge as a robust denotational paradigm in the theory of programming languages [7, 37]. It has been used to construct the first *fully abstract* models for a wide spectrum of languages [4, 6, 12, 15, 16, 24, 28], previously out of reach of denotational semantics. 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.

The pioneering full abstraction results for the purely functional language PCF [5, 17, 39], obtained in the 1990s, have recently been acknowledged by the Alonzo Church Award. The aim of the present paper is to demonstrate how far the field has developed in the meantime and how the range of the game approach can now be extended to capture real-life programming features, such as 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) [10], as specified by interfaces with inheritance. We present both equational (contextual equivalence) and inequational (contextual approximation) full abstraction results for the language.

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

Authors' addresses: Andrzej S. Murawski, University of Oxford; Nikos Tzevelekos, Queen Mary University of London.

Manuscript submitted to ACM

those corresponding to its sub-phrases. An important advance in game semantics was the development of nominal games [3, 25, 38, 45], which underpinned full abstraction results for languages with dynamic generative behaviours, such as the ν-calculus [3], higher-order concurrency [26] and ML references [33]. 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. In the setting of an object-oriented language like IMJ, reference names are used to model object identifiers. The game semantics of a term then consists of sequences of call/return moves referring to methods of objects created by one of the players and revealed to the other.

Example 1.1. Consider interfaces $\mathcal{I}$ and $\mathsf{HashFun}_{\mathcal{I}}$, where $\mathsf{HashFun}_{\mathcal{I}}$ contains a method $\mathsf{hash} : \mathcal{I} \to \mathsf{int}$, which can be thought of as an integer-valued hashing function for objects of type $\mathcal{I}$. Let o be an object of type $\mathsf{HashFun}_{\mathcal{I}}$. Its semantics will contain plays of the form

$$o.\mathsf{call}\,\mathsf{hash}(a_1) \quad o.\mathsf{ret}\,\mathsf{hash}(i_1) \quad o.\mathsf{call}\,\mathsf{hash}(a_2) \quad o.\mathsf{ret}\,\mathsf{hash}(i_2) \cdots$$

where $a_1, a_2, \ldots$ are object names of type $\mathcal{I}$ and $i_1, i_2, \ldots$ are the corresponding hash values. The moves labelled as calls belong to the *Opponent* player (representing the environment), while those that are returns belong to the *Proponent* player (corresponding to the modelled object).

Now, suppose o has an additional method $\mathsf{reset} : \mathsf{HashFun}_{\mathcal{I}} \to \mathsf{void}$ that receives an object of type $\mathsf{HashFun}_{\mathcal{I}}$ as input and, from then on, delegates hashing to that object's hash method. In this case, the plays become more involved, as each player can issue calls to hash methods of objects created by the other player. For instance, we could have the following plays, in which we have tagged the moves according to the player issuing them.

$$o.\mathsf{call}\,\mathsf{hash}(a_1)_O \quad o.\mathsf{ret}\,\mathsf{hash}(i_1)_P \quad o.\mathsf{call}\,\mathsf{hash}(a_2)_O \quad o.\mathsf{ret}\,\mathsf{hash}(i_2)_P \cdots \quad o.\mathsf{call}\,\mathsf{reset}(o')_O \quad o.\mathsf{ret}\,\mathsf{reset}()_P$$
$$o.\mathsf{call}\,\mathsf{hash}(a_1')_O \quad o'.\mathsf{call}\,\mathsf{hash}(a_1')_P \quad o'.\mathsf{ret}\,\mathsf{hash}(i_1')_O \quad o.\mathsf{ret}\,\mathsf{hash}(i_1')_P \cdots$$

In particular, note that, since reset was used (first line), the subsequent move $o.\mathsf{call}\,\mathsf{hash}(a_1')_O$ is followed by $o'.\mathsf{call}\,\mathsf{hash}(a_1')_P$, which corresponds to requesting the hash code of a_1' according to o'. Only after the hash code is provided (as i_1' in $o'.\mathsf{ret}\,\mathsf{hash}(i_1')_O$), can the call $o.\mathsf{call}\,\mathsf{hash}(a_1')_O$ be answered.

The interactions are by no means restricted to the previous format. Opponent has no obligation to return with the hash of a_1' immediately and could instead make another call:

$$o.\mathsf{call}\,\mathsf{hash}(a_1)_O \quad o.\mathsf{ret}\,\mathsf{hash}(i_1)_P \quad o.\mathsf{call}\,\mathsf{hash}(a_2)_O \quad o.\mathsf{ret}\,\mathsf{hash}(i_2)_P \cdots \quad o.\mathsf{call}\,\mathsf{reset}(o')_O \quad o.\mathsf{ret}\,\mathsf{reset}()_P$$
$$o.\mathsf{call}\,\mathsf{hash}(a_1')_O \quad o'.\mathsf{call}\,\mathsf{hash}(a_1')_P \quad o.\mathsf{call}\,\mathsf{hash}(a_2')_O \quad o'.\mathsf{call}\,\mathsf{hash}(a_2')_P \quad o.\mathsf{call}\,\mathsf{reset}(o'')_O \quad o.\mathsf{ret}\,\mathsf{reset}()_P \cdots$$

More generally, the setting makes it possible to express all the call/return behaviours anticipated in our scenario and, as we demonstrate in this paper, any interaction produced by IMJ objects.

The full abstraction results for IMJ were first presented in [35]. In this paper we provide an extended account of the work, including proofs, examples and additional explanations.

Related Work. While the operational semantics of Java has been researched extensively [8], there have been relatively few results regarding its denotational semantics. More generally, most existing models of object-oriented languages, such as [9, 22], 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, 20, 21] and environmental bisimulations [23]. 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. [29] vs [27]. This correlation has been formally explored in [18].

Next we compare our model to existing game models for other languages.

- Broadly speaking, our model follows the methodology of nominal game semantics: names will be weaved into play, moves will be accompanied by a component representing the (visible) store and all the main concepts underpinning the model will be name-invariant. Previous work in that strand has led to models for the ν-calculus [3], first-order references [34], storage of names [25], higher-order references [33], higher-order concurrency [26] and nominal exceptions [36]. In this paper we show how to apply the methodology to an object-oriented framework, which has not been attempted before. In particular, the modelling approach covers dynamic object creation, interface and object subtyping, run-time type cast and self-reference.
- At the technical level, our main contribution lies in identifying new notions of play and strategy, along with the corresponding concepts of strategy composition and a pre-order on strategies, which taken together can be shown to characterise contextual interactions of objects through a full abstraction result.
  In comparison to other game models, the game model of IMJ has a relatively lightweight feel. Arenas have flat structure and, for the most part, playing consists of calling the other player's methods or returning results for calls made by the other player, subject to a well-bracketing condition. This is governed by two new conditions, called *well-calling* and *well-classing*. The former requires that each player can only call the other player's methods. Intuitively, this is because calls to one's own methods cannot be observed and so should not be visible in a fully abstract model. Well-classing ensures that playing is compatible with the subtyping relation.
- In contrast to the nominal game models mentioned above, we do not use justification pointers between moves, which have been a common feature in models of higher-order computation [17]. Conceptually, this simplification can be attributed to the fact that, unlike for higher-order references [33], methods in Java objects cannot be updated and, consequently, each function can be referred using an object name.
- On the other hand, the absence of justification pointers makes definitions of some simple notions, such as polarity, less direct, since the dependencies between moves are not given explicitly any more and need to be inferred from the history of play. In particular, following the principle that each player calls only methods of objects created by the other player or returns results of their own methods, we can (recursively) determine the polarity of a method call based on the polarity of the move introducing its object in the play. This will render strategy composition non-standard. Because it is impossible to determine statically to which arena a move belongs, the switching conditions (cf. [7]) governing interactions will be crucial for determining the strategy responsible for each move.
- Finally, it is worth noting that identity strategies (typically consisting of lengthy "tit-for-tat" interactions) will be particularly simple in our setting: they will contain plays of length at most two. This is a consequence of well-calling: if O plays an object and P copies it, O will not be able to make any further moves, because he is not allowed to call methods of his object.

Further Directions. The model presented here was used as a theoretical foundation for classifying decidable fragments of IMJ with respect to contextual equivalence [31], and for implementing the equivalence verification tool Coneqct [30]:

https://bitbucket.org/sjr/coneqct

This was made possible by Theorem 6.5 (full abstraction), which provides an explicit characterisation of contextual equivalence, along with the use of automata over infinite alphabets [11] to account for the nominal features of the model. In particular, the decidable fragments turned out to be faithfully representable using a combination of fresh-register and pushdown register automata [32, 46].

2 THE LANGUAGE IMJ

We introduce an imperative object calculus, called Interface Middleweight Java (IMJ), in which objects are typed using interfaces. The calculus is a stripped down version of Middleweight Java (MJ) [10], expressive enough to expose the interactions of MJ-style objects with the environment.

Definition 2.1. Let *Ints*, *Flds* and *Meths* be disjoint sets of ***interface***, ***field*** and ***method*** identifiers, ranged over respectively by $\mathcal{I}$, f, m and variants. The ***types*** θ of IMJ include void, int and all interface identifiers. An ***interface definition*** Θ is a finite set of typed fields and methods. An ***interface table*** Δ is a finite assignment of interface definitions to interface identifiers. These are given below, where $\vec{\theta}$ stands for a sequence $\theta_1, ..., \theta_n$ of types (for any n).

$$\begin{array}{lll} \mathit{Ints} \ni \mathcal{I} & \mathit{Types} \ni \theta & ::= \ \mathsf{void} \mid \mathsf{int} \mid \mathcal{I} \\ \mathit{Flds} \ni \mathsf{f} & \mathit{IDfns} \ni \Theta & ::= \ \emptyset \mid (\mathsf{f} : \theta), \Theta \mid (\mathsf{m} : \vec{\theta} \to \theta), \Theta \\ \mathit{Meths} \ni \mathsf{m} & \mathit{ITbls} \ni \Delta & ::= \ \emptyset \mid (\mathcal{I} : \Theta), \Delta \mid (\mathcal{I}\langle\mathcal{I}'\rangle : \Theta), \Delta \end{array}$$

We write $\mathcal{I}\langle\mathcal{I}'\rangle : \Theta$ for *interface extension*: interface $\mathcal{I}$ extends $\mathcal{I}'$ with fields and methods from Θ.[1] We stipulate that the extension relation must not lead to circular dependencies. Moreover, each identifier f/m may appear in each Θ at most once, and each $\mathcal{I}$ can be defined at most once in Δ (i.e. there is at most one element of Δ of the form $\mathcal{I} : \Theta$ or $\mathcal{I}\langle\mathcal{I}'\rangle : \Theta$). Thus, each Θ can be seen as a finite partial function $\Theta : (\mathit{Flds} \cup \mathit{Meths}) \rightharpoonup \mathit{Types}^*$. We write $\Theta.\mathsf{f}$ for $\Theta(\mathsf{f})$ and $\Theta.\mathsf{m}$ for $\Theta(\mathsf{m})$. Similarly, Δ can be used to define a partial function $\Delta : \mathit{Ints} \rightharpoonup \mathit{IDfns}$ as the least (wrt domain size) partial function such that

- if $(\mathcal{I} : \Theta) \in \Delta$ then $\Delta(\mathcal{I}) = \Theta$;
- if $(\mathcal{I}\langle\mathcal{I}'\rangle : \Theta) \in \Delta$, $\mathcal{I}' \in \mathsf{dom}(\Delta)$ and $\mathsf{dom}(\Delta(\mathcal{I}')) \cap \mathsf{dom}(\Theta) = \emptyset$ then $\Delta(\mathcal{I}) = \Delta(\mathcal{I}') \cup \Theta$.

An interface table Δ is ***well-formed*** if $\Delta(\mathcal{I})$ is defined for any interface type $\mathcal{I}$ occurring in Δ. Formally, well-formedness can be defined using the rules in Figure 1, which rely on judgments of the form $\mathcal{D} \mid \mathcal{U} \vdash \Delta$, where $\mathcal{D}$ and $\mathcal{U}$ track defined and undefined interfaces respectively. An interface table Δ is then well-formed, if there exists $\mathcal{D}$ such that $\mathcal{D} \mid \emptyset \vdash \Delta$. Henceforth we assume that interface tables are well-formed.

Interface extensions yield a subtyping relation. Given a table Δ, we define $\Delta \vdash \theta_1 \leq \theta_2$ by the following rules.

$$\frac{}{(\mathcal{I}\langle\mathcal{I}'\rangle : \Theta), \Delta \vdash \mathcal{I} \leq \mathcal{I}'} \qquad \frac{}{\Delta \vdash \theta \leq \theta} \qquad \frac{\Delta \vdash \theta_1 \leq \theta_2 \quad \Delta \vdash \theta_2 \leq \theta_3}{\Delta \vdash \theta_1 \leq \theta_3}$$

We might omit Δ from subtyping judgements for economy. For illustration, we give several example interface tables next.

[1] The notation here could be misleading for the Java enthusiasts: angle brackets do *not* stand for polymorphism as in Java generics.

$$\frac{}{\emptyset \mid \emptyset \vdash \emptyset} \qquad \frac{\mathcal{D} \mid \mathcal{U} \vdash \Delta \qquad \mathcal{I} \notin \mathcal{D}}{\mathcal{D}' \mid \mathcal{U}' \vdash (\mathcal{I} : \Theta), \Delta} \qquad \frac{\mathcal{D} \mid \mathcal{U} \vdash (\mathcal{I}' : \Theta'), \Delta \qquad \mathcal{I} \notin \mathcal{D} \qquad \mathrm{dom}(\Theta) \cap \mathrm{dom}(\Theta') = \emptyset}{\mathcal{D}' \mid \mathcal{U}' \vdash (\mathcal{I}\langle \mathcal{I}' \rangle : \Theta), (\mathcal{I}' : \Theta'), \Delta}$$

Ints(Θ) stands for the set of interface names occurring in Θ. We let $\mathcal{D}' = \mathcal{D} \cup \{\mathcal{I}\}$ and $\mathcal{U}' = (\mathcal{U} \cup \mathit{Ints}(\Theta)) \setminus \mathcal{D}'$.

Fig. 1. Well-formedness rules for interface tables

Example 2.2. The simplest interface is the empty one, called Empty, which contains no fields or methods. Any other interface can be set to extend Empty. For example, the following interface tables are valid:

$$\Delta_1 = \{\mathsf{Empty} : \emptyset,\ \mathsf{Point} : (\mathsf{x} : \mathsf{int}, \mathsf{y} : \mathsf{int})\},$$
$$\Delta_2 = \{\mathsf{Empty} : \emptyset,\ \mathsf{Point}\langle\mathsf{Empty}\rangle : (\mathsf{x} : \mathsf{int}, \mathsf{y} : \mathsf{int})\}.$$

Note also that interfaces can be defined recursively. For instance, the interface table

$$\Delta = \{\, \mathcal{I} : (\mathsf{m}_1 : \mathcal{I} \to \mathsf{void},\ \mathsf{m}_2 : \mathsf{void} \to \mathcal{I})\,\}$$

is well-formed.

We shall observe a notational convention when writing down interface tables: while both interface tables and interface definitions are sets, curly brackets will be used for the former and plain brackets for the latter.

Remark 2.3. Our notion of an interface conveys information about an object's type signature, which coincides with the information that the environment needs to interact with an IMJ object. The interfaces specify simply what fields and methods are available. This is more permissive than what is allowed in Java interfaces, in which fields are restricted to constants. Some of this expressivity can be regained in Java with abstract classes, though. Other object-oriented languages also allow for more expressive interfaces (e.g. traits in Scala).

Next we turn to terms of our language.

Definition 2.4. Let Names 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, $\oplus$ is selected from some set of binary numeric operations. $\mathcal{M}$ stands for ***method-set implementations***. Again, we stipulate that each m appear in each $\mathcal{M}$ at most once.

$$M ::= x \mid \mathsf{null} \mid a \mid \mathsf{skip} \mid i \mid M \oplus M \mid \mathsf{let}\, x = M \,\mathsf{in}\, M \mid M = M \mid \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M$$
$$\mid (\mathcal{I})M \mid \mathsf{new}(x : \mathcal{I}; \mathcal{M}) \mid M.\mathsf{f} \mid M.\mathsf{f} := M \mid M.\mathsf{m}(\overrightarrow{M})$$

$$\mathit{MImps} \ni \mathcal{M} ::= \emptyset \mid (\mathsf{m} : \lambda\vec{x}.M), \mathcal{M}$$

The terms are typed in contexts comprising an interface table Δ, a variable context $\Gamma = \{x_1 : \theta_1, \cdots, x_n : \theta_n\}$, a name context $u = \{a_1 : \mathcal{I}_1, \cdots, a_m : \mathcal{I}_m\}$ such that any interface in Γ and u occurs in $\mathrm{dom}(\Delta)$. The typing rules are given in Figure 2.

As usual, we write $M; N$ for $\mathsf{let}\, x = M \,\mathsf{in}\, N$, where x is not free in N. We will also be writing $\mathsf{new}(\_ : \mathcal{I}; \mathcal{M})$ for $\mathsf{new}(x : \mathcal{I}; \mathcal{M})$ where x does not occur freely in $\mathcal{M}$. In typing judgements, when the name context u is empty we may omit them altogether and write e.g. $\Delta|\Gamma \vdash M : \theta$.

$$\frac{}{\Delta|\Gamma; u \vdash \mathsf{skip} : \mathsf{void}} \qquad \frac{}{\Delta|\Gamma; u \vdash \mathsf{null} : \mathcal{I}}\ {\scriptstyle \mathcal{I} \in \mathrm{dom}(\Delta)} \qquad \frac{}{\Delta|\Gamma; u \vdash i : \mathsf{int}}$$

$$\frac{}{\Delta|\Gamma; u \vdash x : \theta}\ {\scriptstyle (x:\theta)\in\Gamma} \qquad \frac{}{\Delta|\Gamma; u \vdash a : \mathcal{I}}\ {\scriptstyle (a:\mathcal{I})\in u} \qquad \frac{\Delta|\Gamma; u \vdash M : \mathsf{int} \quad \Delta|\Gamma; u \vdash M' : \mathsf{int}}{\Delta|\Gamma; u \vdash M \oplus M' : \mathsf{int}}$$

$$\frac{\Delta|\Gamma; u, x : \theta' \vdash M : \theta \quad \Delta|\Gamma; u \vdash M' : \theta'}{\Delta|\Gamma; u \vdash \mathsf{let}\, x = M' \,\mathsf{in}\, M : \theta} \qquad \frac{\Delta|\Gamma; u \vdash M : \mathcal{I} \quad \Delta|\Gamma; u \vdash M' : \mathcal{I}}{\Delta|\Gamma; u \vdash M = M' : \mathsf{int}} \qquad \frac{\Delta|\Gamma; u \vdash M : \mathcal{I}'}{\Delta|\Gamma; u \vdash (\mathcal{I})M : \mathcal{I}}\ {\scriptstyle \Delta \vdash \mathcal{I} \le \mathcal{I}' \text{ or } \Delta \vdash \mathcal{I}' \le \mathcal{I}}$$

$$\frac{\Delta|\Gamma; u \vdash M : \mathsf{int} \quad \Delta|\Gamma; u \vdash M', M'' : \theta}{\Delta|\Gamma; u \vdash \mathsf{if}\ M\ \mathsf{then}\ M'\ \mathsf{else}\ M'' : \theta} \qquad \frac{\Delta|\Gamma, x : \mathcal{I}; u \vdash \mathcal{M} : \Theta}{\Delta|\Gamma; u \vdash \mathsf{new}(x : \mathcal{I}; \mathcal{M}) : \mathcal{I}}\ {\scriptstyle \Delta(\mathcal{I}) \upharpoonright Meths = \Theta}$$

$$\frac{\Delta|\Gamma; u \vdash M : \mathcal{I} \quad \Delta|\Gamma; u \vdash M' : \theta}{\Delta|\Gamma; u \vdash M.\mathsf{f} := M' : \mathsf{void}}\ {\scriptstyle \Delta(\mathcal{I}).\mathsf{f}=\theta} \qquad \frac{\Delta|\Gamma; u \vdash M : \mathcal{I}}{\Delta|\Gamma; u \vdash M.\mathsf{f} : \theta}\ {\scriptstyle \Delta(\mathcal{I}).\mathsf{f}=\theta}$$

$$\frac{\Delta|\Gamma; u \vdash M : \mathcal{I} \quad \bigwedge_{i=1}^{n}(\Delta|\Gamma; u \vdash M_i : \theta_i)}{\Delta|\Gamma; u \vdash M.\mathsf{m}(M_1, \cdots, M_n) : \theta}\ {\scriptstyle \Delta(\mathcal{I}).\mathsf{m}=\vec{\theta}\to\theta} \qquad \frac{\bigwedge_{i=1}^{n}(\Delta|\Gamma \uplus \{\vec{x}_i : \vec{\theta}_i\}; u \vdash M_i : \theta_i)}{\Delta|\Gamma; u \vdash \mathcal{M} : \Theta}\ {\scriptstyle \Theta=\{\mathsf{m}_i:\vec{\theta}_i\to\theta_i \mid 1\le i\le n\} \atop \mathcal{M}=\{\mathsf{m}_i:\lambda\vec{x}_i.M_i \mid 1\le i\le n\}}$$

Fig. 2. Typing rules for IMJ terms and method-set implementations

Remark 2.5. Note that, in the typing rule for $\mathsf{new}(x : \mathcal{I}; \mathcal{M})$, the type of the variable x matches that of the whole term $\mathsf{new}(x : \mathcal{I}; \mathcal{M})$. This is because x represents the identity of the object, like the keyword `this` in Java. Accordingly, occurrences of x in $\mathcal{M}$ represent self-reference, i.e. they make it possible to refer to the fields and methods of the same object in other methods.

For the operational semantics, we define the sets of ***term values***, ***field assignments*** and ***states*** by:

$$\begin{aligned} \mathit{TVals} \ni V &::= \mathsf{skip} \mid i \mid \mathsf{null} \mid a \\ \mathit{FAsgs} \ni F &::= \emptyset \mid (\mathsf{f} : v), F \\ \mathit{States} \ni S &: \mathsf{Names} \rightharpoonup \mathit{Ints} \times (\mathit{FAsgs} \times \mathit{MImps}) \end{aligned}$$

If $S(a) = (\mathcal{I}, (F, \mathcal{M}))$ then we write $S(a) : \mathcal{I}$, while $S(a).\mathsf{f}$ and $S(a).\mathsf{m}$ stand for $F.\mathsf{f}$ and $\mathcal{M}.\mathsf{m}$ respectively, for each f and m.

Given an interface table Δ such that $\mathcal{I} \in \mathrm{dom}(\Delta)$, we let the default field assignment of type $\mathcal{I}$ be

$$F_{\mathcal{I}} = \{\mathsf{f} : V_\theta \mid \Delta(\mathcal{I}).\mathsf{f} = \theta\},$$

where $V_{\mathsf{void}} = \mathsf{skip}$, $V_{\mathsf{int}} = 0$ and, for each interface $\mathcal{I}'$, $V_{\mathcal{I}'} = \mathsf{null}$. The operational semantics of IMJ is given by means of a small-step transition relation between terms-in-state, presented in Figure 3.

The transition relation uses *evaluation contexts* E that are defined as follows.

$$\begin{aligned} E ::= {} & \bullet \mid \mathsf{let}\, x = E \,\mathsf{in}\, M \mid E \oplus M \mid i \oplus E \mid E = M \mid V = E \mid \mathsf{if}\ E\ \mathsf{then}\ M\ \mathsf{else}\ M' \mid (\mathcal{I})E \\ & \mid E.\mathsf{f} \mid E.\mathsf{f} := M \mid a.\mathsf{f} := E \mid E.\mathsf{m}(\vec{M}) \mid a.\mathsf{m}(v_1, \cdots, v_i, E, M_{i+2}, \cdots, M_n) \end{aligned}$$

Given $\Delta|\emptyset \vdash M : \mathsf{void}$, we write $M \Downarrow$ if there exists S such that $(\emptyset, M) \longrightarrow^* (S, \mathsf{skip})$.

$$(S, i \oplus i') \longrightarrow (S, j), \text{ if } j = i \oplus i' \qquad (S, \text{let } x = V \text{ in } M) \longrightarrow (S, M[V/x])$$

$$(S, \text{if } 0 \text{ then } M \text{ else } M') \longrightarrow (S, M') \qquad (S, \text{if } i \text{ then } M \text{ else } M') \longrightarrow (S, M), \text{ if } i \neq 0$$

$$(S, V = V) \longrightarrow (S, 1) \qquad (S, V = V') \longrightarrow (S, 0), \text{ if } V \neq V'$$

$$(S, (\mathcal{I})\text{null}) \longrightarrow (S, \text{null}) \qquad (S, (\mathcal{I})a) \longrightarrow (S, a), \text{ if } S(a) : \mathcal{I}' \wedge \mathcal{I}' \leq \mathcal{I}$$

$$(S, \text{new}(x : \mathcal{I}; \mathcal{M})) \longrightarrow (S \uplus \{(a, \mathcal{I}, (F_{\mathcal{I}}, \mathcal{M}[a/x]))\}, a) \qquad (S, a.\text{f}) \longrightarrow (S, S(a).\text{f})$$

$$(S, a.\text{m}(\vec{V})) \longrightarrow (S, M[\vec{V}/\vec{x}]), \text{ if } S(a).\text{m} = \lambda\vec{x}.M \qquad (S, a.\text{f} := V) \longrightarrow (S[a \mapsto (\mathcal{I}, (F[\text{f} \mapsto V], \mathcal{M}))], \text{skip}),$$
$$\text{if } S(a) = (\mathcal{I}, (F, \mathcal{M}))$$

$$(S, E[M]) \longrightarrow (S', E[M']), \text{ if } (S, M) \longrightarrow (S', M')$$

Fig. 3. Operational semantics of IMJ.

Remark 2.6. For technical convenience, IMJ features the let construct, even though it is definable: given $\Delta|\Gamma, x : \theta'; u \vdash M : \theta$ and $\Delta|\Gamma; u \vdash M' : \theta'$, consider $\text{new}(\, y : \mathcal{I}; \text{m} : \lambda x.M\,).\text{m}(M')$, where $\mathcal{I}$ is a fresh interface with a single method $\text{m} : \theta \to \theta'$.

Remark 2.7. Although IMJ does not have explicit local variables, they could easily be introduced by taking

$$\text{let } (x = \text{new}(\, \_ : \text{Var}_\theta;)) \text{ in } \cdots,$$

where Var_θ is an interface with a single field of type θ, e.g. $\text{Var}_\theta : (\text{val} : \theta)$. This is reminiscent of how locally-scoped references are introduced in ML-style languages. For examples of interesting equivalences involving local state, we direct the reader to [43].

Such locally declared objects can also be used to simulate fields and methods that are *private* to objects and invisible to the environment via interfaces. For example, in order to create an object of type $\mathcal{I}$ with private fields $f_1 : \theta_1, \cdots, f_k : \theta_k$, we can use an open IMJ term $\text{new}(x : \mathcal{I}; \mathcal{M})$ with free variables $f_1 : \text{Var}_{\theta_1}, \cdots, f_k : \text{Var}_{\theta_k}$. The variables may then occur freely in methods and field access/update can be simulated by dereferencing/updating the public field $f_i.\text{val} : \theta_i$ from Var_{θ_i}. Privacy (state encapsulation) can then be achieved by binding each $f_i : \text{Var}_{\theta_i}$ to a fresh local object as follows:

$$\begin{array}{l} \text{let } f_1 = \text{new}(\, \_ : \text{Var}_{\theta_1};\,) \text{ in} \\ \qquad \vdots \\ \text{let } f_k = \text{new}(\, \_ : \text{Var}_{\theta_k};\,) \text{ in } \text{new}(x : \mathcal{I}; \mathcal{M}). \end{array}$$

This ability to simulate local state in IMJ influenced our decision not to include explicit private fields in IMJ, for the sake of minimal design. The encoding of private fields outlined above will often be used in our examples.

Example 2.8. The next term creates a hash-function object over the interface $\mathcal{I}$. Its hashing method delegates the job to another object (*priv*.val) of the same type. The latter is stored internally, its initial value will be trivial, but it can be

modified later via reset. The objects have type $\mathsf{HashFun}_{\mathcal{I}} : (\mathsf{hash} : \mathcal{I} \to \mathsf{int},\ \mathsf{reset} : \mathsf{HashFun}_{\mathcal{I}} \to \mathsf{void})$.

$$\begin{array}{l}\mathsf{let}\ \mathit{priv} = \mathsf{new}(\_ : \mathsf{Var}_{\mathsf{HashFun}_{\mathcal{I}}};)\ \mathsf{in}\\ \quad (\mathit{priv}.\mathsf{val} := \mathsf{new}(\_ : \mathsf{HashFun}_{\mathcal{I}}; \mathcal{M}_0));\ \mathsf{new}(\_ : \mathsf{HashFun}_{\mathcal{I}}; \mathcal{M}_1)\end{array}$$

$$\mathcal{M}_0 = (\mathsf{hash} :\ \lambda\_.\, 0,\ \mathsf{reset} :\ \lambda\_.\, \mathsf{skip}),$$
$$\mathcal{M}_1 = (\mathsf{hash} :\ \lambda z.\ \mathit{priv}.\mathsf{val}.\mathsf{hash}(z),\ \mathsf{reset} :\ \lambda h.\ \mathit{priv}.\mathsf{val} := h).$$

Finally, we define a notion of equivalence of terms, which will be the main target of our denotational model: the model will equate two terms just if they are equivalent in this sense.

Definition 2.9. Given $\Delta|\Gamma \vdash M_i : \theta$ $(i = 1, 2)$, we shall say that $\Delta|\Gamma \vdash M_1 : \theta$ ***contextually approximates*** $\Delta|\Gamma \vdash M_2 : \theta$ if, for all $\Delta' \supseteq \Delta$ and all contexts C such that $\Delta'|\emptyset \vdash C[M_i] : \mathsf{void}$, if $C[M_1] \Downarrow$ then $C[M_2] \Downarrow$. We then write $\Delta|\Gamma \vdash M_1 \sqsubseteq\!\!\!\sim M_2 : \theta$. Two terms are ***contextually equivalent*** (written $\Delta|\Gamma \vdash M_1 \cong M_2 : \theta$) if they approximate each other.

To illustrate equivalences and refinements that may arise in IMJ (or failures thereof), we consider several examples next. The first one is based on comparisons between reference names, the second one uses local variables to hide implementation details and the third one discusses type casts and subtyping.

Example 2.10 (extended from [43]). The terms below manipulate reference names, and use methods from names to names. Let

$$\Delta = \{\mathsf{Empty} : \emptyset,\ \mathsf{EtoE} : (\mathsf{m} : \mathsf{Empty} \to \mathsf{Empty})\}$$

and consider the terms $\Delta|\emptyset \vdash M_i : \mathsf{EtoE}$ $(i = 1, 2, 3)$ defined by

$$\begin{array}{lll} M_1 \equiv & \mathsf{let}\ x = \mathsf{new}(\_ : \mathsf{Empty};)\ \mathsf{in}\ \ \mathsf{new}(\_ : \mathsf{EtoE}; \mathcal{M}_1) \\ M_2 \equiv & \mathsf{let}\ x = \mathsf{new}(\_ : \mathsf{Empty};)\ \mathsf{in}\ \mathsf{let}\ y = \mathsf{new}(\_ : \mathsf{Empty};)\ \mathsf{in}\ \ \mathsf{new}(\_ : \mathsf{EtoE}; \mathcal{M}_2) \\ M_3 \equiv & \mathsf{let}\ x = \mathsf{new}(\_ : \mathsf{Empty};)\ \mathsf{in}\ \mathsf{let}\ y = \mathsf{new}(\_ : \mathsf{Empty};)\ \mathsf{in}\ \ \mathsf{new}(\_ : \mathsf{EtoE}; \mathcal{M}_3) \end{array}$$

with

$$\begin{array}{ll} \mathcal{M}_1 = & (\mathsf{m} : \lambda z.\, x) \\ \mathcal{M}_2 = & (\mathsf{m} : \lambda z.\, \mathsf{if}\ (z = x)\ \mathsf{then}\ x\ \mathsf{else}\ y), \\ \mathcal{M}_3 = & (\mathsf{m} : \lambda z.\, \mathsf{if}\ (z = x)\ \mathsf{then}\ y\ \mathsf{else}\ x). \end{array}$$

The term M_1 will simply keep on returning the same name all the time. M_2 seemingly has two options (x or y) but, in order to respond with x the context would have to guess it. As x is local and not divulged to the environment otherwise, this will never be the case, so M_1 and M_2 are contextually equivalent. However, M_3 is different: it reveals x when called for the first time, which can be exploited to trigger the second response y in a subsequent call. Thus, we have $\Delta|\emptyset \vdash M_1 \cong M_2 \ncong M_3 : \mathsf{EtoE}$. Note that x (resp. x, y) have been declared *locally* in M_1 (resp. in M_2, M_3). Next we explain how the equivalence and inequivalence will be captured by our game model. Indeed, the game semantics of the first two terms will turn out to consist of the same plays, of the shape

$$*^{\emptyset}\ a^{\Sigma_0}\ \mathsf{call}\ a.\mathsf{m}(a_1)^{\Sigma_1'}\ \mathsf{ret}\ a.\mathsf{m}(a_0)^{\Sigma_1}\ \mathsf{call}\ a.\mathsf{m}(a_2)^{\Sigma_2}\ \mathsf{ret}\ a.\mathsf{m}(a_0)^{\Sigma_2} \cdots \mathsf{call}\ a.\mathsf{m}(a_k)^{\Sigma_k}\ \mathsf{ret}\ a.\mathsf{m}(a_0)^{\Sigma_k}$$

where $a_1 \neq a_0$ and:

$$\Sigma_0 = \{a \mapsto (\mathsf{EtoE}, \emptyset)\}$$
$$\Sigma_1' = \Sigma_0 \cup \{a_1 \mapsto (\mathsf{Empty}, \emptyset)\}$$
$$\Sigma_1 = \Sigma_1' \cup \{a_0 \mapsto (\mathsf{Empty}, \emptyset)\}$$
$$\Sigma_{i+1} = \Sigma_i \cup \{a_{i+1} \mapsto (\mathsf{Empty}, \emptyset)\} \quad (i > 0).$$

The first move can be viewed as the environment starting an interaction by providing values for the free variables. Since the term does not have any, it is simply $*$. Each move will be accompanied by a store that collects all the names have been used in play. As no names have been used so far, $*$ appears together with the empty store $\emptyset$. The second move represents the program revealing the name of the object that has been created by M_i. Only its name is revealed and the associated store becomes Σ_0. Note that the store contains the type EtoE of the name. If the corresponding interface featured any public fields then their initial values would have been mentioned, but EtoE does not have any, hence a is mapped to $(\mathsf{EtoE}, \emptyset)$. Note that method bodies are never mentioned, these will always be hidden in our interpretation.

The subsequent sequences of call- and return-moves correspond to the environment calling the m method of the object a (on an object name) and the program returning the same name a_0 in response. As soon as a name is played, it becomes part of every subsequent store, which reflects the fact that the object has become accessible to both players and thus observable. The names $a_1, \cdots, a_k$ are the arguments used by the environment. They need not be different. However, $a_1 \neq a_0$, as a_0 represents a local name that the environment cannot predict.

Note that inside stores, the names $a_1, \cdots, a_k$ are associated with $(\mathsf{Empty}, \emptyset)$, because Empty is both the argument and result type of m, as specified by the interface EtoE. A notable feature of our way of modelling (and game semantics, in general) is the fact that local storage (and related state changes) are invisible in plays unless the relevant location (reference name) has been revealed by the term to the environment (leaked). This can be seen in the second move a^{Σ_0}, which does not contain any mention of x (or x, y for M_2). For M_1, the name bound to x is revealed only in the fourth move, once it is actually returned as a result of the first call to m. Similarly, for M_2, the fourth move reveals the name bound to y, but the play will never use the name associated with x.

Our model makes it possible to distinguish M_1, M_2 from M_3, because M_3 will also generate the following play:

$$*^{\emptyset}\ a^{\Sigma_0}\ \mathsf{call}\ a.\mathsf{m}(a_1)^{\Sigma_1'}\ \mathsf{ret}\ a.\mathsf{m}(a_0)^{\Sigma_1}\ \mathsf{call}\ a.\mathsf{m}(a_0)^{\Sigma_1}\ \mathsf{ret}\ a.\mathsf{m}(a_0')^{\Sigma_1 \cup \{a_0' \mapsto (\mathsf{Empty}, \emptyset)\}}$$

where $a_0 \neq a_0'$. It does not follow the previous pattern, because the second result is different from the first one. This time both the names corresponding to x and y (in M_3) get revealed. The name a_0 corresponds to the object bound to x and, after the environment feeds a_0 to m in the fifth move, the sixth move contains the name a_0', which corresponds to the name bounde to y in M_3.

Example 2.11 (extended from [23]). The terms below simulate local storage of a single name (Var_E) with access (get) and update (set) methods. Integer storage (Var_I) will be used to vary the behaviour of the implementations. M_1 will use a single private variable to store the name. Two such variables are used in M_2 and accessed alternately but, the variables always hold the same value, so it does not matter which of the variables is used to return the value. On the other hand, in M_3 the two variables are used to hold the last two values stored, and the switch b is used to record which of them contains the most recent one.

Let

$$\begin{array}{rcl}\Delta & = \{ & \mathsf{Empty} : \emptyset, \\ & & \mathsf{Cell} : (\mathsf{get} : \mathsf{void} \to \mathsf{Empty}, \mathsf{set} : \mathsf{Empty} \to \mathsf{void}), \\ & & \mathsf{Var}_E : (\mathsf{val} : \mathsf{Empty}), \\ & & \mathsf{Var}_I : (\mathsf{val} : \mathsf{int}) \quad \}\end{array}$$

and consider the terms $\Delta|\emptyset \vdash M_i : \mathsf{Cell}$ $(i = 1, 2, 3)$ defined by

$$M_1 \equiv \quad \mathsf{let}\ z = \mathsf{new}(\_ : \mathsf{Var}_E;)\ \mathsf{in}\ \ \mathsf{new}(\_ : \mathsf{Cell}; \mathcal{M}_1)$$

$$M_2 \equiv \quad \mathsf{let}\ b = \mathsf{new}(\_ : \mathsf{Var}_I;)\ \mathsf{in}\ \mathsf{let}\ z_1 = \mathsf{new}(\_ : \mathsf{Var}_E;)\ \mathsf{in}\ \mathsf{let}\ z_2 = \mathsf{new}(\_ : \mathsf{Var}_E;)\ \mathsf{in}\ \ \mathsf{new}(\_ : \mathsf{Cell}; \mathcal{M}_2)$$

$$M_3 \equiv \quad \mathsf{let}\ b = \mathsf{new}(\_ : \mathsf{Var}_I;)\ \mathsf{in}\ \mathsf{let}\ z_1 = \mathsf{new}(\_ : \mathsf{Var}_E;)\ \mathsf{in}\ \mathsf{let}\ z_2 = \mathsf{new}(\_ : \mathsf{Var}_E;)\ \mathsf{in}\ \ \mathsf{new}(\_ : \mathsf{Cell}; \mathcal{M}_3)$$

with

$$\begin{array}{rl}\mathcal{M}_1 = & (\mathsf{get} : \lambda().(z.\mathsf{val}), \\ & \mathsf{set} : \lambda y.\ (z.\mathsf{val} := y)) \\ \mathcal{M}_2 = & (\mathsf{get} : \lambda().\mathsf{if}\ (b.\mathsf{val})\ \mathsf{then}\ (b.\mathsf{val} := 0; z_1.\mathsf{val})\ \mathsf{else}\ (b.\mathsf{val} := 1; z_2.\mathsf{val}), \\ & \mathsf{set} : \lambda y.\ (z_1.\mathsf{val} := y;\ z_2.\mathsf{val} := y)) \\ \mathcal{M}_3 = & (\mathsf{get} : \lambda().\mathsf{if}\ (b.\mathsf{val})\ \mathsf{then}\ z_1.\mathsf{val}\ \mathsf{else}\ z_2.\mathsf{val}, \\ & \mathsf{set} : \lambda y.\mathsf{if}\ (b.\mathsf{val})\ \mathsf{then}\ (b.\mathsf{val} := 0; z_2.\mathsf{val} := y)\ \mathsf{else}\ (b.\mathsf{val} := 1; z_1.\mathsf{val} := y)).\end{array}$$

We have $\Delta|\emptyset \vdash M_1 \cong M_2 \cong M_3 : \mathsf{Cell}$. Note that, as in the previous example, we are relying on local variables b, z_1, z_2. They are local with respect to the object and play the role of private fields.

The game semantics of the three terms will turn out to consist of plays of the shape

$$*^{\emptyset}\ a^{\Sigma_0}\ \mathit{Get}_0^*\ \mathit{Set}_1\ \mathit{Get}_1^*\ \cdots\ \mathit{Set}_k\ \mathit{Get}_k^*,$$

where

$$\begin{array}{rcl}\mathit{Get}_i & = & \begin{cases}\mathsf{call}\ a.\mathsf{get}(*)^{\Sigma_0}\ \ \mathsf{ret}\ a.\mathsf{get}(\mathtt{nul})^{\Sigma_0} & i = 0 \\ \mathsf{call}\ a.\mathsf{get}(*)^{\Sigma_i}\ \ \mathsf{ret}\ a.\mathsf{get}(v_i)^{\Sigma_i} & i > 0\end{cases} \\ \mathit{Set}_i & = & \mathsf{call}\ a.\mathsf{set}(v_i)^{\Sigma_i}\ \ \mathsf{ret}\ a.\mathsf{set}(*)^{\Sigma_i} \\ \Sigma_i & = & \{a \mapsto (\mathsf{Cell}, \emptyset)\} \cup \{v_j \mapsto (\mathsf{Empty}, \emptyset) \mid 0 < j \leq i,\ v_j \neq \mathtt{nul}\}.\end{array}$$

Here a ranges over Names and each v_i ranges over $(\mathsf{Names} \setminus \{a\}) \cup \{\mathtt{nul}\}$. Intuitively, the plays describe all possible interactions of a Cell object. The first two moves $*^{\emptyset}\ a^{\Sigma_0}$ correspond to object creation. Only the object creation corresponding to the outcome of evaluating M_i is represented - the local objects are not represented because they will never get revealed by the term.

After the first two moves, the Get_i segments represent the environment reading the current content via get (initially having null value), while the Set_i segments correspond to updating the content with the value provided by the environment via set. The stores Σ_i attached to moves consist of all names that have been introduced during the interaction so far. They include a and the names provided by the environment as arguments to set, but never the object names corresponding to the local object names b, z_1, z_2.

It is worth noting that, because IMJ has explicit casting, a context can always guess the actual interface of an object and extract any information we may want to hide through casting.

Example 2.12. Let $\Delta = \{\mathsf{Empty} : \emptyset, \mathsf{Point}\langle\mathsf{Empty}\rangle : (\mathsf{x} : \mathsf{int}, \mathsf{y} : \mathsf{int})\}$ and consider the terms $\Delta|\emptyset \vdash M_i : \mathsf{Empty}$ $(i = 1, 2)$ defined by:

$$M_1 \equiv \quad \mathsf{new}(x : \mathsf{Empty};),$$

$$M_2 \equiv \quad \mathsf{let}\, p = \mathsf{new}(x : \mathsf{Point};)\, \mathsf{in}\ \ p.\mathsf{x} := 0; p.\mathsf{y} := 1; (\mathsf{Empty})p.$$

In our model they will be interpreted by the following strategies respectively: $\sigma_1 = \{\epsilon, *^{\emptyset}\, a^{\{a \mapsto (\mathsf{Empty}, \emptyset)\}}\}$ and $\sigma_2 = \{\epsilon, *^{\emptyset}\, a^{\{a \mapsto (\mathsf{Point}, \{\mathsf{x} \mapsto 0, \mathsf{y} \mapsto 1\})\}}\}$. This time, the strategies are very concise: there are no call- or return-moves, because the interfaces involved do not contain any methods. The essence of the object is entirely captured by the information about their name and type and, for M_2, the initial values assigned to fields.

The reader may wonder how subsequent field updates are handled, as they are not represented explicitly in σ_2. Because such updates are triggered by the interaction of the object with other objects, they will be integrated in the process of strategy composition, which will keep a record of how the fields evolve. Such updates need not become part of the semantics of the object, because their flavour is highly uniform (just propagate the most recent value) and not specific to the object. However, if the object had methods, the strategy would contain call- and return-moves with stores indicating how the field values are affected by the calls and returns respectively. Using, for example, the casting context $C \equiv (\mathsf{Point})\bullet; \mathsf{skip}$, we can see that $\Delta|\emptyset \vdash M_2 \not\sqsubseteq M_1 : \mathsf{Empty}$. On the other hand, Theorem 6.3 will imply $\Delta|\emptyset \vdash M_1 \sqsubseteq M_2 : \mathsf{Empty}$.

Remark 2.13. When designing IMJ, we aimed to arrive at a minimalistic calculus that is expressive enough to capture interactions of MJ-like objects with the environment. Our guiding principle was that these interactions are carried out via interfaces, which specify the publicly accessible fields and methods. Accordingly, we suppressed the introduction of explicit class hierarchy, as it would remain invisible to the environment and any class-based internal computations can be represented using standard object encodings [1]. In the same spirit, we did not add explicit private fields/methods to IMJ, because:

(1) their effect (i.e. local state, encapsulation, hiding) can already be achieved through other IMJ constructs (Remark 2.7),
(2) private fields/methods are not explicitly involved in contextual interactions anyway.

Despite differing from Java in several details mentioned in earlier remarks, IMJ amounts to a compact calculus that strips down Middleweight Java to the bare essentials needed for interface-based interaction. In particular, it accounts for such features of Java-like objects as

- field access and assignment;
- object creation, identity and self-reference;
- base types, reference types and null pointers;
- interfaces with subtyping and casting.

At the moment the calculus allows for single inheritance for interfaces only, but extending it to multiple inheritance is not problematic. The following semantic developments only rely on the assumption that $\leq$ must not give rise to circularities.

3 THE GAME MODEL

In this section we show how to interpret IMJ terms in a nominal game model. We will present the translation in steps, starting from the translation of IMJ types into *arenas*, and building up an arsenal of notions that will lead to a translation of typed IMJ terms into *strategies*. In our discussion below, we assume a fixed interface table Δ along with the induced subtyping relation $\leq$.

We start off by translating values of IMJ into *semantic values*. For each type θ, we let Val_θ be the set of *semantic values* of type θ, given by:

$$Val_{\mathsf{void}} = \{*\}, \quad Val_{\mathsf{int}} = \mathbb{Z}, \quad Val_{\mathcal{I}} = \mathsf{Names} \cup \{\mathtt{nul}\},$$

for each interface type $\mathcal{I}$. Thus, the void type has a single value, represented by the asterisk symbol. On the other hand, values of interface type, in addition to names, feature a distinguished value nul, representing the null value. We write Val for $Val_{\mathsf{void}} \cup Val_{\mathsf{int}} \cup Val_{\mathcal{I}}$. For each type sequence $\vec{\theta} = \theta_1, \cdots, \theta_n$, we set $Val_{\vec{\theta}} = Val_{\theta_1} \times \cdots \times Val_{\theta_n}$.

3.1 Nominal sets

The game model will be constructed using mathematical objects (moves, plays, strategies) that feature *names* drawn from a designated set Names. The set Names will in particular consist of object names. Although names underpin various elements of our model, we do not want to delve into the precise nature of the sets containing them. Hence, all of our definitions preserve name-invariance, i.e. our objects are (strong) *nominal sets* [13, 45]. Note that we do not need the full power of the theory but mainly the basic notion of name-permutation. We will construe a *nominal set* to be a set whose elements contain elements from Names. More specifically, given nominal sets X, Y:

- For an element x belonging to X we write $\nu(x)$ for its name-support, which is the set of names occurring in x. By assumption, every $\nu(x)$ is going to be finite.
- Moreover, for any $x \in X$ and permutation π of Names, we write $\pi \cdot x$ for the result of applying π elementwise to all names in x. We stipulate that $x \in X$ implies $\pi \cdot x \in X$.
- For any $x, y \in X$, we write $x \sim y$ if there is a permutation π such that $x = \pi \cdot y$.
- A relation $R \subseteq X \times Y$ is called *nominal* if it is closed under permutations: if xRy then $(\pi \cdot x)R(\pi \cdot y)$, for any permutation π. Accordingly, $f : X \to Y$ is a nominal function just if $\pi \cdot (f(x)) = f(\pi \cdot x)$ for all x and π.

The objects of our category of games will be nominal sets carrying specific type information.

3.2 Arenas, moves-with-store and plays

Our semantic translation will be into a category of *games*, which will feature *arenas* as objects and *strategies* as morphisms. In particular, each typed term $\Delta|\Gamma; u \vdash M : \theta$ will be translated into a strategy $[\![M]\!] : [\![\Gamma; u]\!] \to [\![\theta]\!]$. Thus, arenas will serve as type representations in our model. They will provide the *defining moves* from which all other moves will be derived. Given arenas A and B, a *play* in $A \to B$ will be a sequence of moves from A and B adhering to certain well-formedness conditions. In addition, each move will carry its own representation of the current (visible) state. A strategy will then be a set of such plays describing semantically the visible behaviour of M.

We start off by defining arenas.

Definition 3.1. An ***arena*** is a pair $A = (M_A, \xi_A)$ where:

- M_A is a nominal set of ***defining moves***,
- $\xi_A : M_A \to (\mathsf{Names} \rightharpoonup \mathit{Ints})$ is a nominal *typing function*,

such that, for all $m \in M_A$, $\mathsf{dom}(\xi_A(m)) = \nu(m)$.

Thus, the typing function ξ_A assigns interface types to all object names appearing in the defining moves of A. It can be seen as the semantic counterpart of syntactic typing.

Since arenas function as representations of IMJ types, we define the following basic arenas representing respectively the types void, int and $\mathcal{I}$:

$$1 = (\{*\}, \{(*, \emptyset)\}),$$
$$\mathbb{Z} = (\mathbb{Z}, \{(i, \emptyset)\}),$$
$$\mathcal{I} = (\mathsf{Names} \cup \{\mathtt{nul}\}, \{(\mathtt{nul}, \emptyset)\} \cup \{(a, (a \mapsto \mathcal{I}))\}),$$

for all interfaces $\mathcal{I}$.[2] We notice that $M_1 = Val_{\mathsf{void}}$, $M_{\mathbb{Z}} = Val_{\mathsf{int}}$ etc, i.e. the moves in the arena corresponding to each of the basic types coincide with the semantic values of that type. Moreover, the typing function is trivial for moves that contain no names (e.g. the move $*$), but is meaningful for moves containing names (e.g. it assigns the type $\mathcal{I}$ to each name a in a move a of the arena $\mathcal{I}$).

Another important arena is the *fresh combination arena* $\#(\mathcal{I}_1, \cdots, \mathcal{I}_n) = (M_{\#(\vec{\mathcal{I}})}, \xi_{\#(\vec{\mathcal{I}})})$, with:

$$M_{\#(\vec{\mathcal{I}})} = \{(a_1, \cdots, a_n) \in \mathsf{Names}^n \mid a_i\text{'s distinct}\}$$
$$\xi_{\#(\vec{\mathcal{I}})}((a_1, \cdots, a_n), a_i) = \mathcal{I}_i$$

for all $n \in \mathbb{N}$ (where we write $\xi(x)(y)$ as $\xi(x, y)$ to save on brackets). In particular, $\#() = 1$. The arena can be used to represent tuples of different objects, such as those featuring in the state component of our operational semantics. In what follows, we shall rely on the special arenas specified above as well as their combinations obtained through the product construction, defined next.

Definition 3.2 (Product arena). Given arenas A and B, we can form the arena $A \times B$ by:

$$M_{A\times B} = \{(m, n) \in M_A \times M_B \mid a \in \nu(m) \cap \nu(n) \implies \xi_A(m, a) \leq \xi_B(n, a) \vee \xi_B(n, a) \leq \xi_A(m, a)\}$$
$$\xi_{A\times B}((m, n), a) = \begin{cases} \xi_A(m, a) & a \notin \nu(n) \vee \xi_A(m, a) \leq \xi_B(n, a) \\ \xi_B(n, a) & \text{otherwise} \end{cases}$$

Remark 3.3. As one may expect, $M_{A\times B}$ consists of pairs of moves, taken from A and B respectively. The moves need not share names but, if they do, the types of the names (as prescribed by ξ_A and ξ_B) must be compatible, i.e. comparable via $\leq$. Then $\xi_{A\times B}$ is taken to be the lower type.

With basic arenas and products we can now translate arbitrary sequences of types into arenas. In particular, anticipating the full definition of the semantic translation (Definition 5.1), we can translate contexts $\Gamma = \{x_1 : \theta_1, \cdots, x_n : \theta_n\}$, $u = \{a_1 : \mathcal{I}_1, \cdots, a_m : \mathcal{I}_m\}$ into arenas by

$$[\![\Gamma; u]\!] = [\![\theta_1]\!] \times \cdots \times [\![\theta_n]\!] \times \#(\mathcal{I}_1, \cdots, \mathcal{I}_m) \tag{1}$$

where $[\![\mathsf{void}]\!] = 1$, $[\![\mathsf{int}]\!] = \mathbb{Z}$ and $[\![\mathcal{I}]\!] = \mathcal{I}$.

States are going to be represented in the model via *stores*. Recall that a state is a finite map from names to interfaces paired with heap configurations and method implementations. Since method implementations are not visible to the

[2] There is an obvious abuse of notation here: $\mathcal{I}$ is used to refer to both the interface type $\mathcal{I}$ and the arena representing it. This is done for notational brevity.

environment of an object (i.e. the environment can interact with an object's methods but not look at their code), stores will only register the interface information and the field assignment of each object.

Definition 3.4. We let a ***store*** Σ be a type-preserving finite partial function from names to interfaces and field assignments,

$$\Sigma : \mathsf{Names} \rightharpoonup \mathit{Ints} \times (\mathit{Flds} \rightharpoonup \mathit{Val})$$

satisfying several healthiness conditions listed below. To specify the first component of $\Sigma(a)$, we shall write $\Sigma(a) : \mathcal{I}$ if $\Sigma(a) = (\mathcal{I}, \phi)$ for some ϕ. Similarly, $\Sigma(a).\mathsf{f}$ will stand for $\phi(\mathsf{f})$.
We stipulate that $|\Sigma|$ be finite and the following well-formedness condition be satisfied

$$\forall a, \mathcal{I}, \mathsf{f}, \theta.\ \Sigma(a) : \mathcal{I} \wedge \Delta(\mathcal{I}).\mathsf{f} = \theta \implies \exists v, \mathcal{I}'.\ \Sigma(a).\mathsf{f} = v \wedge \Sigma \vdash v : \mathcal{I}' \wedge \mathcal{I}' \leq \theta$$

where the typing rules for values in store contexts are given below.

$$\frac{v \in \mathit{Val}_{\mathsf{void}}}{\Sigma \vdash v : \mathsf{void}} \qquad \frac{v \in \mathit{Val}_{\mathsf{int}}}{\Sigma \vdash v : \mathsf{int}} \qquad \frac{\Sigma(v) : \mathcal{I} \vee v = \mathtt{nul}}{\Sigma \vdash v : \mathcal{I}}$$

We let Sto be the set of all stores and write $\mathsf{dom}(\Sigma(a))$ for the set of all f such that $\Sigma(a).\mathsf{f}$ is defined.
We let Sto_0 contain all stores Σ such that:

$$\forall a \in \mathsf{dom}(\Sigma).\forall \mathsf{f} \in \mathsf{dom}(\Sigma(a)).\ \Sigma(a).\mathsf{f} \in \{*, 0, \mathtt{nul}\}$$

and we call such a Σ a *default store*.
Finally, given stores Σ and Σ', and $X \subseteq \mathsf{Names}$, we define:

- the *restricted* store $\Sigma \restriction X = \{(a, \Sigma(a)) \mid a \in \mathsf{dom}(\Sigma) \cap X\}$
- the *updated* store $\Sigma[\Sigma'] = \Sigma' \cup (\Sigma \restriction (\mathsf{dom}(\Sigma) \setminus \mathsf{dom}(\Sigma')))$
- the *defaulted* store $\mathsf{Dft}(\Sigma)$ as the unique $\Sigma_0 \in \mathit{Sto}_0$ such that $\forall a, \mathcal{I}.\ \Sigma(a) : \mathcal{I} \iff \Sigma_0(a) : \mathcal{I}$.

Example 3.5. Our previous definitions have laid a correspondence between syntactic and semantic values, and between states and stores. For example, below we list a couple of states along with the corresponding stores (where $\mathcal{M}_i$ and the interface types used are defined as in Example 2.11).

$$\{a \mapsto (\mathsf{Var}_I, (\{\mathsf{val} \mapsto 0\}, \emptyset)), b \mapsto (\mathsf{Var}_E, (\{\mathsf{val} \mapsto \mathtt{nul}\}, \emptyset))\} \longmapsto \{a \mapsto (\mathsf{Var}_I, \{\mathsf{val} \mapsto 0\}), b \mapsto (\mathsf{Var}_E, \{\mathsf{val} \mapsto \mathtt{nul}\})\}$$

$$\{a \mapsto (\mathsf{Cell}, (\emptyset, \mathcal{M}_i)), b \mapsto (\mathsf{Empty}, (\emptyset, \emptyset))\} \longmapsto \{a \mapsto (\mathsf{Cell}, \emptyset), b \mapsto (\mathsf{Empty}, \emptyset)\}$$

Remark 3.6. As mentioned at the start of this section, a typed term $\Delta|\Gamma; u \vdash M : \theta$ will be mapped into a strategy $[\![M]\!] : [\![\Gamma; u]\!] \to [\![\theta]\!]$. A strategy $\sigma : A \to B$ is going to be a set of sequences of moves-with-store from the arenas A and B adhering to several well-formedness conditions. In what follows, we will specify these conditions and motivate them with examples. We will often label the associated notions (such as move and play) with AB, which will stand as abbreviation for the pair (A, B).

Given arenas A and B, plays in AB will consist of sequences of moves annotated with stores, where the moves will be either coming from $M_A \cup M_B$ or representing method calls and returns. Formally, we define:

$$M_{AB} = M_A \cup M_B \cup \mathit{Calls} \cup \mathit{Retns}$$

where we set

$$\mathit{Calls} \;=\; \{\mathsf{call}\ a.\mathsf{m}(\vec{v}) \mid a \in \mathsf{Names} \wedge \mathsf{m} \in \mathit{Meths} \wedge \vec{v} \in \mathit{Val}^*\},$$
$$\mathit{Retns} \;=\; \{\mathsf{ret}\ a.\mathsf{m}(v) \mid a \in \mathsf{Names} \wedge \mathsf{m} \in \mathit{Meths} \wedge v \in \mathit{Val}\}.$$

Our first step towards defining plays in AB is to specify well-formedness conditions for the underlying sequences of moves. Recall that A and B stand for arbitrary arenas. In practice, in the majority of cases, we shall rely on the special arenas $1, \mathbb{Z}, \mathcal{I}$ introduced earlier and their products.

Definition 3.7. A ***legal sequence*** in AB is a sequence of moves from M_{AB} that adheres to the following grammar (*Well-Bracketing*), where m_A and m_B range over M_A and M_B respectively.

$$L_{AB} \;::=\; \epsilon \mid m_A\mathcal{X} \mid m_A\mathcal{Y}m_B\mathcal{X}$$
$$\mathcal{X} \;::=\; \mathcal{Y} \mid \mathcal{Y}\,(\mathsf{call}\ a.\mathsf{m}(\vec{v}))\,\mathcal{X}$$
$$\mathcal{Y} \;::=\; \epsilon \mid (\mathsf{call}\ a.\mathsf{m}(\vec{v}))\,\mathcal{Y}\,(\mathsf{ret}\ a.\mathsf{m}(v))\,\mathcal{Y}$$

We write L_{AB} for the set of legal sequences in AB. In the last clause above, we say that $\mathsf{call}\,a.\mathsf{m}(\vec{v})$ ***justifies*** $\mathsf{ret}\ a.\mathsf{m}(v)$.

To each $s \in L_{AB}$ we relate a ***polarity*** function p from move occurrences in s to the set $\mathit{Pol}_1 = \{O, P\}$ by setting:

- for all $m_A \in M_A$ occurring in s we have $p(m_A) = O$; (*Well-starting*)
- if mn are consecutive moves in s then $p(n) \neq p(m)$. (*Alternation*)

Polarities are complemented via $\overline{O} = \{P\}$ and $\overline{P} = \{O\}$.

Remark 3.8. Note that a non-empty legal sequence always begins with a move from M_A but no other moves from M_A are allowed afterwards. A legal sequence may also contain at most one element of M_B. All other moves are from $\mathit{Calls} \cup \mathit{Retns}$. Below we relate the shape of legal sequence to our modelling needs.

Recall that our typing judgments have the form $\Delta|\Gamma; u \vdash M : \theta$. When modelling terms, the initial move m_A will represent a tuple of values corresponding to a value assignment for the variables in Γ and an enlisting of all names in u. m_B in turn represents the moment when a value of type θ is generated by the interaction between the term and its environment. If M is already a value then m_B will immediately follow m_A. In general this need not be the case and m_A may be preceded by a well-balanced segment $\mathcal{Y}$ of calls and returns corresponding to the history of interaction before successful evaluation.

Once m_B is played, the following moves $\mathcal{X}$ are sequences of calls and returns that may contain "open" calls but returns follow the call/return stack discipline. $\mathcal{X}$ is used to represent the history of computation after evaluation.

Remark 3.9. 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. By definition, there exists a unique polarity function p for each legal sequence s, namely the one which assigns O precisely to those moves appearing in odd positions in s (i.e. the first move, the third one, etc.). Moreover, it follows that for all $m_B \in M_B$ occurring in s we have $p(m_B) = P$. Finally, we warn the reader that, although the complement of a polarity is currently determined uniquely, later on it will become a non-singleton set of polarities.

Example 3.10. Consider the following interface with a single callable method:

$$\mathsf{Callable} = (\mathsf{foo} : \mathsf{void} \to \mathsf{void})$$

and let us examine legal sequences in Callable1. In anticipation of Definition 3.14, we will be looking at legal sequences that can be extended to plays.

- The arena Callable comprises moves from the set $\{\mathtt{nul}\} \cup \mathsf{Names}$. Thus, a legal sequence in Callable1 must start with a move m_0 that is either $\mathtt{nul}$ or some name a. Moreover, m_0 would have polarity O.
- Next, following Definition 3.7, we can either have call move, or the unique move in 1. In both cases, the new move would have polarity P. In the latter case, we obtain the legal sequence: m_0*.
  In the former case, Definition 3.7 allows us to call any method on any object and with any argument values. Later on, in Definition 3.14, we will constrain such method calls so that they make sense: the method called should be part of the object it is called upon, and the arguments it is called with be of compatible types (*well-classing* conditions). In our case here, a sensible call would be to the method foo with argument $*$. If m_0 was a non-null object a, this would yield the legal sequence: $a\ \mathsf{call}\ a.\mathsf{foo}(*)$.
- The legal sequence m_0* corresponds to $m_A m_B$ in the notation of Definition 3.7. Hence, we could only extend it by picking another call, the polarity of which would be O. As we shall see, though, Definition 3.14 would exclude this possibility: either because $m_0 = \mathsf{nul}$ or, for $m_0 = a$, because O cannot call a method on a, since a was created by O (a player can only call methods on objects created by the other player, *well-calling* condition).
  Instead, let us focus on how to extend $a\ \mathsf{call}\ a.\mathsf{foo}(*)$. Definition 3.7 allows us to add either a return of foo on a, or a new call. We saw above that another call would be contrary to the conditions of Definition 3.14, so a sensible way to extend our legal sequence would be: $a\ \mathsf{call}\ a.\mathsf{foo}(*)\ \mathsf{ret}\ a.\mathsf{foo}(*)$.

Re-iterating the last two steps above, the general pattern for legal sequences that can be extended to plays is:

$$a\ (\mathsf{call}\ a.\mathsf{foo}(*)\ \mathsf{ret}\ a.\mathsf{foo}(*))^* *$$

and prefixes thereof, for any name a.

Example 3.11. In Example 2.11 we examined terms $\Delta|\emptyset \vdash M_i : \mathsf{Cell}$ $(i = 1, 2, 3)$ with:

$$\mathsf{Cell} : (\mathsf{get} : \mathsf{void} \to \mathsf{Empty}, \mathsf{set} : \mathsf{Empty} \to \mathsf{void})$$

The terms are translated into strategies $[\![M_i]\!] : [\![\emptyset]\!] \to [\![\mathsf{Cell}]\!]$, so the arenas of interest are:

$$[\![\emptyset]\!] = 1 = (\{*\}, \{(*, \emptyset)\}) \quad \text{and} \quad [\![\mathsf{Cell}]\!] = \mathsf{Cell} = (\mathsf{Names} \cup \{\mathtt{nul}\}, \{(\mathtt{nul}, \emptyset)\} \cup \{(a, (a \mapsto \mathsf{Cell}))\}).$$

From these, we build the relevant set of moves:

$$M_{1\mathsf{Cell}} = \{*, \mathtt{nul}\} \cup \mathsf{Names} \cup \{\mathsf{call}\ a.\mathsf{m}(\vec{v}) \mid a \in \mathsf{Names} \wedge \vec{v} \in Val^*\} \cup \{\mathsf{ret}\ a.\mathsf{m}(v) \mid a \in \mathsf{Names} \wedge v \in Val\}.$$

Relating these moves to the previous definition, m_A is simply $*$, while m_B can be $\mathtt{nul}$ or any name $a \in \mathsf{Names}$. Thus, non-empty legal sequences from 1Cell can be in one of the following forms:

- $*s$: where s is a sequence of calls and returns that is well-bracketed, i.e. in the language of $\mathcal{X}$.
- $*sms'$: where s is a well-balanced sequence (i.e. in $\mathcal{Y}$), m is either $\mathtt{nul}$ or a name, and s' is a well-bracketed sequence (i.e. in $\mathcal{X}$).

The meaning of these moves is the following. The move $*$ simply represents the context of the term M_i, which is empty. The sequence s represents any computations performed by M_i before evaluating. In practice, since the context is empty, these computations will be internal to M_i and therefore s must also be empty. If M_i evaluates to null then m will be $\mathtt{nul}$,

and therefore there is no more interaction to expect, i.e. s' is empty. On the other hand, if M_i evaluates to some object a then we will have $m = a$ and s' will contain a sequence of calls and returns of the get and set methods on a.

Legal sequences cannot fully represent the interaction between a term and its environment, as they lack information on field updates. This is where the role of stores becomes important: the plays of our games will be legal sequences whose moves have been annotated with stores.

Remark 3.12. The early game models of the 1990s, e.g. [4–6, 12, 15–17, 24], did not rely on moves annotated with a store. In contrast, one of the characteristic features of nominal game semantics, is the reliance on such moves to keep track of the names that become visible to the environment and to express the evolution of visible store, e.g. [3, 25, 33, 38, 45].

Definition 3.13. A ***move-with-store*** in AB is a pair m^Σ with $\Sigma \in Sto$ and $m \in M_{AB}$.

In order to be able to specify the valid uses of object names inside a play, we need to introduce the notions of *name availability* and *name ownership*. These will allow us to impose, for example, that a call of a method on an object a cannot be made by the player who owns this object (i.e. introduced it first in the play). Note that from here onwards we will reserve s for sequences of moves-with-store. Given such a sequence s, we use the notation $\underline{s}$ to refer to its underlying sequence of moves (i.e. the one obtained by erasing all stores from s).

For each sequence s of moves-with-store we define the set of ***available names*** of s by:

$$\mathsf{Av}(\epsilon) = \emptyset, \quad \mathsf{Av}(sm^\Sigma) = \Sigma^*(\mathsf{Av}(s) \cup \nu(m))$$

where, for each $X \subseteq \mathsf{Names}$, we let $\Sigma^*(X) = \bigcup_i \Sigma^i(X)$, with

$$\Sigma^0(X) = X, \quad \Sigma^{i+1}(X) = \nu(\Sigma(\Sigma^i(X))).$$

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 use $\sqsubseteq$ to denote the prefix relation between sequences. Let s be a sequence of moves-with-store, and let p be the polarity function of its underlying $\underline{s}$. If $s'm^\Sigma \sqsubseteq s$ and $a \in \nu(m^\Sigma) \setminus \nu(s')$ then we say a ***is introduced*** by m^Σ in s,[3] and we define the ***owner*** of the name a in s, written $o(a)$, to be $p(m)$. We moreover define ownership sets:

$$O(s) = \{a \in \nu(s) \mid o(a) = O\} \quad \text{and} \quad P(s) = \{a \in \nu(s) \mid o(a) = P\}$$

for O and P respectively.

Plays will consist of sequences of moves-with-store adhering to a set of well-formedness conditions. Before presenting their formal definition, let us first discuss these conditions informally. Suppose we have a valid play s' and a new candidate move-with-store m^Σ is given. In order for $s'm^\Sigma$ to be valid, we want to check the following.

(1) The store Σ must give a complete account of the field values of all names that have been revealed since, in each move, the player playing the move may have changed the field's value. Therefore, the domain of Σ should include the set $\mathsf{Av}(s'm^\Sigma)$. Moreover, it should not contain any additional names: any such names would be unreachable to the other player, and revealing their field values in the play would be revealing information that is hidden to the other player.

[3] 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, $\nu(m_1^{\Sigma_1} \cdots m_i^{\Sigma_i}) = \nu(m_1) \cup \nu(\Sigma_1) \cup \cdots \cup \nu(m_i) \cup \nu(\Sigma_i)$.

(2) Any name appearing in the domain of Σ should be typed consistently with s' and the underlying arenas. For example, if a name a appears in s' and has type $\mathcal{I}$ (i.e. it appears in some store T in s' and $T(a) : \mathcal{I}$), then $\Sigma(a) : \mathcal{I}$. Or, if m is a move from an arena M_A and $\xi_A(m, a) = \mathcal{I}$, then Σ must assign to a a subtype of $\mathcal{I}$.
(3) If m is a call to a method of an object a, then the owner of a must be the opposite of the owner of m. Put otherwise, a player cannot call their own 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. Additionally, the name a must have been revealed in s', i.e. be part of its available names.

Definition 3.14. A ***play*** in AB is a sequence of moves-with-store s such that $\underline{s}$ is a legal sequence with polarity function p and, for all $s'm^\Sigma \sqsubseteq s$:

- It holds that $\mathrm{dom}(\Sigma) = \mathsf{Av}(s'm^\Sigma)$. (*Frugality*)
- If $a \in \mathrm{dom}(\Sigma)$ with $\Sigma(a) : \mathcal{I}$ then:
  - if $m \in M_X$, for $X \in \{A, B\}$, and $a \in v(m)$ then $\mathcal{I} \leq \xi_X(m, a)$;
  - for all n^T in s', if $a \in \mathrm{dom}(T)$ then $T(a) : \mathcal{I}$;
  - if m is a call or return of some method m on a, then $\Delta(\mathcal{I}).\mathsf{m} = \vec{\theta} \to \theta$ such that:
    - if $m = \mathsf{call}\, a.\mathsf{m}(\vec{v})$ then $\Sigma \vdash \vec{v} : \vec{\theta'}$ for some $\vec{\theta'} \leq \vec{\theta}$,
    - if $m = \mathsf{ret}\, a.\mathsf{m}(v)$ then $\Sigma \vdash v : \theta'$ for some $\theta' \leq \theta$.

  (*Well-classing*)
- If $m = \mathsf{call}\, a.\mathsf{m}(\vec{v})$ then $o(a) \in \overline{p(m)}$. (*Well-calling*)

We write P_{AB} for the set of plays in AB.

Remark 3.15. It is worth noting the following:

- Well-calling implements the specification that each player need only call the other player's methods. Moreover, it stipulates that a name already be available in order for its methods to be called: in the well-calling condition, in order for the owner of a to be different from the player playing m, a must have been introduced in s'.
- Because of well-bracketing, alternation and well-calling, if $m = \mathsf{ret}\, a.\mathsf{m}(v)$ then $o(a) = p(m)$. That is, while method calls on each name a are issued by the player that does not own a, they are answered by the owner of a.
- The frugality condition stipulates that names cannot appear in a play in unreachable parts of a store (cf. [25]).
- Well-classing ensures that the typing information in stores is consistent and adheres to the constraints imposed by Δ and the underlying arenas.

Example 3.16. Let us look again at the plays produced in Example 2.11. The terms examined were typed as $\Delta|\emptyset \vdash M_i : \mathsf{Cell}$ and, as we already saw in Example 3.11, the plays comprising $[\![M_i]\!]$ are plays in 1 Cell. As we mentioned in Example 2.11, these are of the form:

$$s = *^{\emptyset}\, a^{\Sigma_0}\, (\mathsf{call}\, a.\mathsf{get}(*)^{\Sigma_0}\, \mathsf{ret}\, a.\mathsf{get}(\mathsf{nul})^{\Sigma_0})^*\, \mathsf{call}\, a.\mathsf{set}(v_1)^{\Sigma_1}\, \mathsf{ret}\, a.\mathsf{set}(*)^{\Sigma_1}\, (\mathsf{call}\, a.\mathsf{get}(*)^{\Sigma_1}\, \mathsf{ret}\, a.\mathsf{get}(v_1)^{\Sigma_1})^* \cdots$$

where $a \in \mathsf{Names}$, $\Sigma_0 = \{a \mapsto (\mathsf{Cell}, \emptyset)\}$ and $\Sigma_i = \Sigma_0 \cup \{v_j \mapsto (\mathsf{Empty}, \emptyset) \mid 0 < j \leq i,\ v_j \neq \mathsf{nul}\}$ for $i > 0$. Let us analyse such a play s. The underlying sequence of moves is:

$$\underline{s} = *\, a\, (\mathsf{call}\, a.\mathsf{get}(*)\, \mathsf{ret}\, a.\mathsf{get}(\mathsf{nul}))^*\, \mathsf{call}\, a.\mathsf{set}(v_1)\, \mathsf{ret}\, a.\mathsf{set}(*)\, (\mathsf{call}\, a.\mathsf{get}(*)\, \mathsf{ret}\, a.\mathsf{get}(v_1))^* \cdots$$

It is straightforward to see that this is well bracketed: each call is immediately followed by its return. Note also the polarities here: each call has polarity O, while each return has P. Moreover, the move a has polarity P. These polarities convey the fact that the object a is created by P (i.e. the term M_i), so O can call its methods and P will return from them.

Since we established that $\underline{s}$ is legal, let us look at the other play conditions for s:

- Frugality. The available names at each point in the play, after the first move, are the name a and any of the v_i's that have been played and are different to `nul`. The domain of each Σ_i contains precisely those names.
- Well-classing. This is clearly adhered to as all method calls and returns type-check, a is well-typed in each Σ_i and each Σ_i is also well-typed.
- Well-calling. As we noted above, a is owned by P and its methods are correctly called by O.

Hence, s is a valid play.

3.3 Interaction

We next look at how plays compose. In this section, we fix arenas A, B and C, and examine how plays from AB and BC can interact to produce a play in AC. More precisely, we will define play interactions, called *interaction sequences in ABC*, which will be moves-with-store representing the synchronisation of plays from AB and BC into a common sequence of moves. By projecting these interaction sequences onto AC, we will obtain the desired play compositions.

In general in game semantics, play interactions are performed by "parallel composition plus hiding", whereby moves in the common component B are matched between the plays in AB and BC, and then hidden in order to produce plays in AC. In our case, though, plays do not only contain moves from A, B and C: the most interesting moves are calls and returns, which do not form part of the arenas. But while these moves cannot be attributed to an arena, they can be attributed to a component in the interaction (either AB, or BC, or both) and a player – e.g. a move call $a.\mathsf{m}(5)$ will be played by the opposite player to the one that introduced the name a. We will formalise this tracking of moves using an extended set of polarities, to account for the fact that in an interaction over ABC there are two components, AB and BC, and the same move may correspond to different players in different components. The same polarities will ensure that our interactions are alternating, well-calling and well-returning. They will also be used in order to define projections of interaction sequences in ABC onto plays in AB, BC and AC.

Interaction sequences will rely on moves with stores where the moves come from the set:

$$M_{ABC} = M_A \cup M_B \cup M_C \cup \mathit{Calls} \cup \mathit{Retns}\,.$$

The moves will be assigned polarities from a set of six polarities:

$$\mathit{Pol}_2 = \{\ O_L, P_L, O_L P_R, P_L O_R, O_R, P_R\ \}.$$

The index L stands for "left" and refers to the AB constituent of the interaction, while R means "right" and refers to BC. Polarities indicate which component of the interaction (AB or BC) a move comes from and what polarity it has in it. For instance, a move labelled O_L is a move played by O in AB alone (i.e. and not played in BC), while a move labelled P_R is played by P in BC alone (and not played in AB). A move labelled $O_L P_R$, on the other hand, is played in both components and has polarity O in AB and P in BC. We can group the polarities designated to each of the two components as:

$$p(AB) = \{O_L, P_L, O_L P_R, P_L O_R\},$$
$$p(BC) = \{O_R, P_R, O_L P_R, P_L O_R\}.$$

The above polarities also allow us to determine whether a move forms part of the component AC, i.e. the component obtained after composing in AB and BC. A move will be part of AC just if it is not a move played in both AB and BC, i.e. it is not a move that needs to be "hidden" after synchronisation between these two components. We can therefore designate these polarities to AC:

$$p(AC) = \{O_L, P_L, O_R, P_R\}.$$

Note the slight abuse of notation with p, as it is also used for denoting a move polarity function.

An interaction over ABC is essentially an interaction between three "coarse" players: the player P in AB, the player P in BC and the player O in AC. Each of these players has two related polarities: e.g. P in AB relates to P_L and $P_L O_R$, as P . It is useful to group the polarities of each of the three players in what we call *pseudo-polarities*, which are sets of polarities defined by:

$$P_1 = \{P_L, P_L O_R\}, \quad P_2 = \{P_R, O_L P_R\}, \quad O_3 = \{O_L, O_R\},$$

where the indices 1, 2 and 3 stand for the components AB, BC and AC respectively.

Pseudo-polarities will be useful for determining the complement of a move's polarity, which in turn will be used for determining the next player after a given move, or e.g. the player who is allowed to call a method on an object introduced by a given player. For instance, a move with polarity O_L (i.e. a move played by O in AB alone), may only be followed by a move played by P in AB, i.e. a move with polarity P_L or $P_L O_R$, i.e. a move with pseudo-polarity P_1. Applying this reasoning to each polarity, we can define:

$$\overline{O_L} = \overline{O_L P_R} = P_1, \quad \overline{O_R} = \overline{P_L O_R} = P_2, \quad \overline{P_L} = \overline{P_R} = O_3,$$

as the polarity complementation function. In particular, the complement of a polarity is a pseudo-polarity.

Projecting interaction sequences in ABC onto AB, BC and AC will involve using a polarity function to specify what moves to retain in the projection, and a filtering function γ to remove superfluous names from the stores of the projection. Consider a sequence s of moves-with-store from ABC (i.e. a sequence with elements m^Σ with $m \in M_{ABC}$) along with a map p assigning to moves of s polarities from Pol_2. For each $X \in \{AB, BC, AC\}$, we define:

$$s \upharpoonright X = \begin{cases} (s' \upharpoonright X)\, m^\Sigma & \text{if } s = s' m^\Sigma \text{ and } p(m) \in p(X) \\ s' \upharpoonright X & \text{if } s = s' m^\Sigma \text{ and } p(m) \notin p(X) \\ \epsilon & \text{if } s = \epsilon \end{cases}$$

$$s \upharpoonright_\gamma X = \gamma(s \upharpoonright X) \quad \text{where} \quad \gamma(\epsilon) = \epsilon, \quad \gamma(s m^\Sigma) = \gamma(s)\, m^{\Sigma \upharpoonright \mathsf{Av}(s m^\Sigma)}.$$

i.e. $s \upharpoonright X$ is the subsequence of s containing those moves-with-store m^Σ of s for which $p(m) \in p(X)$; and $s \upharpoonright_\gamma X$ is the sequence we obtain from the latter by retaining the same moves but restricting the domains of their stores to available names. The role of γ is to *frugalise* the filtered interaction so that it becomes a play.

We can now formally define interaction sequences. These will be sequences of moves-with-store from ABC adhering to conditions that generalise play conditions. Below, for each name a in an interaction sequence s with polarity function p, and each polarity $\Pi \in Pol_2$, we let $o(a) = \Pi$ just if a is introduced in s by a move-with-store m^Σ with $p(m) = \Pi$.

Definition 3.17. An ***interaction sequence*** in ABC is a sequence s of moves-with-store in ABC, satisfying the following.

- For each $s' m^\Sigma \sqsubseteq s$, $\mathrm{dom}(\Sigma) = \mathsf{Av}(s' m^\Sigma)$. (*Frugality*)
- If $s' m^\Sigma \sqsubseteq s$ and $a \in \mathrm{dom}(\Sigma)$ with $\Sigma(a) : \mathcal{I}$ then:
  - if $m \in M_X$, for $X \in \{A, B, C\}$, and $a \in \nu(m)$ then $\mathcal{I} \leq \xi_X(m, a)$;

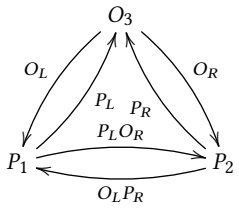


Fig. 4. Polarity diagram for interaction sequences in ABC. Transitions are labelled by move polarities, while the initial state is O_3.

  - for all n^T in s', if $a \in \mathrm{dom}(T)$ then $T(a) : \mathcal{I}$;
  - if m is a call or return of some method m on a, then $\Delta(\mathcal{I}).\mathsf{m} = \vec{\theta} \to \theta$ and:
    * if $m = \mathrm{call}\, a.\mathsf{m}(\vec{v})$ then $\Sigma \vdash \vec{v} : \vec{\theta'}$ for some $\vec{\theta'} \leq \vec{\theta}$,
    * if $m = \mathrm{ret}\, a.\mathsf{m}(v)$ then $\Sigma \vdash v : \theta'$ for some $\theta' \leq \theta$.

  (*Well-classing*)
- There is a polarity function p from move occurrences in $\underline{s}$ to Pol_2 such that:
  - For all $m_X \in M_X$ ($X = A, B, C$) occurring in $\underline{s}$ we have $p(m_A) = O_L$, $p(m_B) = P_L O_R$ and $p(m_C) = P_R$. (*Well-starting*)
  - If mn are consecutive moves in $\underline{s}$ then $p(n) \in \overline{p(m)}$. (*Alternation*)
  - If $s' m^\Sigma \sqsubseteq s$ then $m = \mathrm{call}\, a.\mathsf{m}(\overline{v})$ implies $o(a) \in \overline{p(m)}$. (*Well-calling*)
  - If $s' m^\Sigma \sqsubseteq s$ and $m = \mathrm{ret}\, a.\mathsf{m}(v)$ then there is a move n^T in s' such that, for all X such that $p(m) \in p(X)$, n is the justifier of m in $\underline{s \upharpoonright X}$. (*Well-returning*)
  - For each $X \in \{AB, BC, AC\}$, $\underline{s \upharpoonright X} \in L_X$. (*Projecting*)
  - The following *Laird conditions* hold:
    * $P(s \upharpoonright_\gamma AB)$, $P(s \upharpoonright_\gamma BC)$ and $O(s \upharpoonright_\gamma AC)$ are pairwise disjoint;
    * For each $s' \sqsubseteq s$ ending in $m^\Sigma n^T$ and each $a \in \mathrm{dom}(T)$, we must have $\Sigma(a) = T(a)$ whenever:
      · $p(m) \in P_1$ and $a \notin \nu(s' \upharpoonright_\gamma AB)$,
      · or $p(m) \in P_2$ and $a \notin \nu(s' \upharpoonright_\gamma BC)$,
      · or $p(m) \in O_3$ and $a \notin \nu(s' \upharpoonright_\gamma AC)$.

We write $Int(ABC)$ for the set of interaction sequences in ABC.

Remark 3.18. Up to well-returning, the conditions listed above are extensions of play conditions to interaction sequences. Well-returning in particular is the dual of well-calling, and in this case needs to be stated explicitly as it does not follow from the other conditions (as was the case for plays). Well-returning implies that each return move $\mathrm{ret}\, a.\mathsf{m}(v)^\Sigma$ in an interaction sequence has a unique justifier of the shape $\mathrm{call}\, a.\mathsf{m}(\vec{v})^T$.

The remaining conditions in the above definition come from standard game semantics (projecting) and nominal game semantics (Laird), see [25] for both. Projecting imposes that the interaction follows the legal move patterns of AB and BC. The Laird conditions govern the privacy of names between the two components, AB and BC. The first Laird condition stipulates that there are three players who uniquely own all names in an interaction: P in AB, P in BC, and O in AC. The second condition ensures that if a name has not been revealed in one of the components then its field values cannot be modified in that component.

Recall that, when we looked at legal sequences in Definition 3.7, the polarity function of each legal sequence was uniquely defined by well-starting (O starts) and alternation (consecutive moves have opposite polarities). In interaction sequences, these two conditions only partially specify the potential polarity functions. In Figure 4 we have drawn a diagram capturing these specifications. It can be viewed as an automaton accepting the pair $(\underline{s}, p)$, for each $s \in Int(ABC)$ with polarity function p, starting from state O_3 (all states are accepting). The edges in the diagram represent moves by their polarities, while the labels of vertices specify the pseudo-polarity of the next (outgoing) move. For example, from O_3 we can only have a move m with $p(m) \in O_3 = \{O_L, O_R\}$.

In Lemma 3.20 we shall see that each interaction sequence has a unique polarity function, i.e. if s is an interaction sequence by using either p_1 or p_2 as polarity functions then $p_1 = p_2$. Consequently, we will usually refer to *the polarity of a move* in an interaction sequence, without mentioning a polarity function.

Before proceeding to the Lemma, we explore interaction sequences and their conditions with an example.

Example 3.19. Consider the following interfaces:

$$\mathsf{CellVarInt} = (\mathsf{set} : \mathsf{Var}_I \to \mathsf{void},\ \mathsf{get} : \mathsf{void} \to \mathsf{Var}_I), \quad \mathsf{Var}_I = (\mathsf{val} : \mathsf{int}).$$

We look at interaction sequences in $\mathsf{CellVarInt}_1\ \mathsf{CellVarInt}_2\ 1$, where we tag the two occurrences of the CellVarInt arena for clarity. The interface CellVarInt specifies objects with a getter and a setter method that consume and return respectively an object of type Var_I (i.e. an integer variable).

In the next diagram we depict a possible interaction sequence in $\mathsf{CellVarInt}_1\ \mathsf{CellVarInt}_2\ 1$. We write move sequences vertically, instead of horizontally, in order to specify the component they belong to (e.g. a move written under 1 comes from the component $\mathsf{CellVarInt}_2 1$ and is not present in $\mathsf{CellVarInt}_1 \mathsf{CellVarInt}_2$). We choose a scenario where each setter call provides a fresh name of type Var_I, and the same happens with every getter return. Moreover, *all available* objects of type Var_I have their values changed in each move, with said values oscillating between 42 and 24. For brevity, we write $\Sigma[\hat{a}]$ for $\Sigma[a.\mathsf{val} \mapsto 66 - a.\mathsf{val}]$.

$\mathsf{CellVarInt}_1$	$\mathsf{CellVarInt}_2$	1	*polarity*	*stores*
$c_1^{\Sigma_0}$			O_L	$\Sigma_0 = \{c_1 \mapsto (\mathsf{CellVarInt}, \emptyset)\}$
$\mathsf{call}\, c_1.\mathsf{get}(*)^{\Sigma_0}$			P_L	
$\mathsf{ret}\, c_1.\mathsf{get}(a_1)^{\Sigma_1}$			O_L	$\Sigma_1 = \Sigma_0 \cup \{a_1 \mapsto (\mathsf{Var}_I, \{\mathsf{val} \mapsto 42\})\}$
$\mathsf{call}\, c_1.\mathsf{set}(a_2)^{\Sigma_2}$			P_L	$\Sigma_2 = \Sigma_1[\hat{a_1}] \cup \{a_2 \mapsto (\mathsf{Var}_I, \{\mathsf{val} \mapsto 42\})\}$
$\mathsf{ret}\, c_1.\mathsf{set}(*)^{\Sigma_3}$			O_L	$\Sigma_3 = \Sigma_2[\hat{a_1}, \hat{a_2}]$
	$c_2^{\Sigma_4}$		$P_L O_R$	$\Sigma_4 = \Sigma_3[\hat{a_1}, \hat{a_2}] \cup \{c_2 \mapsto (\mathsf{CellVarInt}, \emptyset)\}$
	$\mathsf{call}\, c_2.\mathsf{get}(*)^{\Sigma_4}$		$O_L P_R$	
	$\mathsf{ret}\, c_2.\mathsf{get}(a_3)^{\Sigma_5}$		$P_L O_R$	$\Sigma_5 = \Sigma_4[\hat{a_1}, \hat{a_2}] \cup \{a_3 \mapsto (\mathsf{Var}_I, \{\mathsf{val} \mapsto 42\})\}$
	$\mathsf{call}\, c_2.\mathsf{set}(a_4)^{\Sigma_6}$		$O_L P_R$	$\Sigma_6 = \Sigma_5[\hat{a_3}] \cup \{a_4 \mapsto (\mathsf{Var}_I, \{\mathsf{val} \mapsto 42\})\}$
$\mathsf{call}\, c_1.\mathsf{set}(a_5)^{\Sigma_7}$			P_L	$\Sigma_7 = \Sigma_6[\hat{a_1}, \hat{a_2}, \hat{a_3}, \hat{a_4}] \cup \{a_5 \mapsto (\mathsf{Var}_I, \{\mathsf{val} \mapsto 42\})\}$
$\mathsf{ret}\, c_1.\mathsf{set}(*)^{\Sigma_8}$			O_L	$\Sigma_8 = \Sigma_7[\hat{a_1}, \hat{a_2}, \hat{a_5}]$
	$\mathsf{ret}\, c_2.\mathsf{set}(*)^{\Sigma_9}$		$P_L O_R$	$\Sigma_9 = \Sigma_8[\hat{a_1}, \hat{a_2}, \hat{a_3}, \hat{a_4}, \hat{a_5}]$
		$*^{\Sigma_{10}}$	P_R	$\Sigma_{10} = \Sigma_9[\hat{a_3}, \hat{a_4}]$

The projections of the interaction on $\mathsf{CellVarInt}_1 \mathsf{CellVarInt}_2$, $\mathsf{CellVarInt}_2 1$ and $\mathsf{CellVarInt}_1 1$ are obtained by suppressing the moves in the third, first and second column respectively, and then applying γ. We can thus see that the projection condition is satisfied.

Let us call the interaction sequence above s and look at how the Laird conditions apply to it. The projections of s on each of the three components are as below.

$s_1 = s \upharpoonright_\gamma \mathsf{CellVarInt}_1\mathsf{CellVarInt}_2$		$s_2 = s \upharpoonright_\gamma \mathsf{CellVarInt}_2 1$		$s_3 = s \upharpoonright_\gamma \mathsf{CellVarInt}_1 1$		
$\mathsf{CellVarInt}_1$	$\mathsf{CellVarInt}_2$	$\mathsf{CellVarInt}_2$	1	$\mathsf{CellVarInt}_1$	1	*frugalised stores*
$c_1^{\Sigma_0}$				$c_1^{\Sigma_0}$		
$\mathsf{call}\, c_1.\mathsf{get}(*)^{\Sigma_0}$				$\mathsf{call}\, c_1.\mathsf{get}(*)^{\Sigma_0}$		
$\mathsf{ret}\, c_1.\mathsf{get}(a_1)^{\Sigma_1}$				$\mathsf{ret}\, c_1.\mathsf{get}(a_1)^{\Sigma_1}$		
$\mathsf{call}\, c_1.\mathsf{set}(a_2)^{\Sigma_2}$				$\mathsf{call}\, c_1.\mathsf{set}(a_2)^{\Sigma_2}$		
$\mathsf{ret}\, c_1.\mathsf{set}(*)^{\Sigma_3}$				$\mathsf{ret}\, c_1.\mathsf{set}(*)^{\Sigma_3}$		
	$c_2^{\Sigma_4}$	$c_2^{\Sigma'_4}$				
	$\mathsf{call}\, c_2.\mathsf{get}(*)^{\Sigma_4}$	$\mathsf{call}\, c_2.\mathsf{get}(*)^{\Sigma'_4}$				$\Sigma'_4 = \Sigma_4 \upharpoonright \{c_2\}$
	$\mathsf{ret}\, c_2.\mathsf{get}(a_3)^{\Sigma_5}$	$\mathsf{ret}\, c_2.\mathsf{get}(a_3)^{\Sigma'_5}$				$\Sigma'_5 = \Sigma_5 \upharpoonright \{c_2, a_3\}$
	$\mathsf{call}\, c_2.\mathsf{set}(a_4)^{\Sigma_6}$	$\mathsf{call}\, c_2.\mathsf{set}(a_4)^{\Sigma'_6}$				$\Sigma'_6 = \Sigma_6 \upharpoonright \{c_2, a_3, a_4\}$
$\mathsf{call}\, c_1.\mathsf{set}(a_5)^{\Sigma_7}$				$\mathsf{call}\, c_1.\mathsf{set}(a_5)^{\Sigma''_7}$		$\Sigma''_7 = \Sigma_7 \upharpoonright \{c_1, a_1, a_2, a_5\}$
$\mathsf{ret}\, c_1.\mathsf{set}(*)^{\Sigma_8}$				$\mathsf{ret}\, c_1.\mathsf{set}(*)^{\Sigma''_8}$		$\Sigma''_8 = \Sigma_8 \upharpoonright \{c_1, a_1, a_2, a_5\}$
	$\mathsf{ret}\, c_2.\mathsf{set}(*)^{\Sigma_9}$	$\mathsf{ret}\, c_2.\mathsf{set}(*)^{\Sigma'_9}$				$\Sigma'_9 = \Sigma_9 \upharpoonright \{c_2, a_3, a_4\}$
			$*^{\Sigma'_{10}}$		$*^{\Sigma''_{10}}$	$\Sigma'_{10} = \Sigma_{10} \upharpoonright \{c_2, a_3, a_4\}$
						$\Sigma''_{10} = \Sigma_{10} \upharpoonright \{c_1, a_1, a_2, a_5\}$

All names (c_1, c_2, a_1-a_5) remain available in $\mathsf{CellVarInt}_1\mathsf{CellVarInt}_2$ after projecting, as they are part of moves played in $\mathsf{CellVarInt}_1\mathsf{CellVarInt}_2$. On the other hand, only names c_2, a_3, a_4 are created in $\mathsf{CellVarInt}_2 1$ and are therefore available in it. Similarly for $\mathsf{CellVarInt}_1 1$ and the names c_1, a_1, a_2, a_5. Checking in each projection the polarity of the move that first introduces each name, we have:

$$P(s_1) = \{c_2, a_2, a_3, a_5\}, \quad P(s_2) = \{a_4\}, \quad O(s_3) = \{c_1, a_1\}.$$

Thus, the first Laird condition is satisfied. For the second one, we see that, for example, the move $\mathsf{call}\, c_2.\mathsf{get}(*)$ has polarity $O_L P_R \in P_2$ and a_1, a_2 are not available in the projection on $\mathsf{CellVarInt}_2\, 1$. Therefore, no field value can change in that move and, thus, the store remains Σ_4. Looking at the name-privacy situation at that move, the restriction makes sense: $\mathsf{call}\, c_2.\mathsf{get}(*)$ is played by P in $\mathsf{CellVarInt}_2\, 1$ and, in that component, none of a_1, a_2 has been revealed. On the other hand, the following move $\mathsf{ret}\, c_2.\mathsf{get}(a_3)$ is played by P in $\mathsf{CellVarInt}_1\mathsf{CellVarInt}_2$ and in there both of a_1, a_2 are available and can have their values changed.

Lemma 3.20. *Each $s \in Int(ABC)$ has a unique polarity function p.*

Proof. Suppose $s \in Int(ABC)$. We claim that the well-starting, alternation, well-calling, projecting and well-returning conditions uniquely specify p. Consider the polarity diagram of Figure 4. We shall read it as an automaton $\mathcal{A}$ which accepts $\underline{s}$ and at the same time constructs a polarity function p for $\underline{s}$ by means of the corresponding (accepting) path; and show that $\underline{s}$ has a unique path in $\mathcal{A}$.

First, by projecting we obtain that the first element of $\underline{s}$ is some m_A and, by well-starting, its polarity is O_L. Thus, we can pick O_3 as the initial state of $\mathcal{A}$.

We now use induction on $|s|$ to show that $\underline{s}$ has a unique path in $\mathcal{A}$. The base case is trivial, so suppose $\underline{s} = s'm$. By induction hypothesis, $\mathcal{A}$ has a unique path for 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 indeed departs from X. If, on the other hand, $m = \mathsf{call}\, a.\mathsf{m}(\vec{v})$ then the fact that $o(a) \in \overline{p(m)}$ gives two possible edges for accepting m. But observe

that no combination of such edges can depart from X. Finally, let $m = \mathrm{ret}\, a.\mathrm{m}(v)$ 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 componentwise opposite of the one accepting n (e.g. if m is accepted by O_L then n be accepted by P_L, etc.). □

We now proceed to show that interaction sequences project to plays. This is done in Proposition 3.23, using the following two lemmata. The first lemma states that projections preserve polarities. The projection of interaction sequences in ABC on AB, BC and AC leads to the following definition of projections of polarities,

$$\begin{array}{lll} \pi_{AB}(X_L) = X & \pi_{AB}(X_L Y_R) = X & \pi_{AB}(Y_R) = \text{undef.} \\ \pi_{BC}(X_L) = \text{undef.} & \pi_{BC}(X_L Y_R) = Y & \pi_{BC}(Y_R) = Y \\ \pi_{AC}(X_L) = X & \pi_{AC}(X_L Y_R) = \text{undef.} & \pi_{AC}(Y_R) = Y \end{array}$$

where $X, Y \in \{O, P\}$. We establish the following.

Lemma 3.21. *Let $s \in Int(ABC)$. Then, for each $X \in \{AB, BC, AC\}$ and each m^{Σ} in s, if $p(m) \in p(X)$ then $\pi_X(p(m)) = p_X(m)$, where p_X is the polarity function of $s \upharpoonright X$.*

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, $p_{AB}(m) = \overline{p_{AB}(n)}$ and observe that, for all $X \in \overline{p(n)}$, $\pi_{AB}(X) = \overline{\pi_{AB}(p(n))}$, so in particular $\pi_{AB}(p(m)) = \overline{\pi_{AB}(p(n))} = \overline{p_{AB}(n)} = p_{AB}(m)$. □

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

Lemma 3.22. *Let $s \in Int(ABC)$. Then,*

(1) $\nu(s) = O(s \upharpoonright_\gamma AC) \uplus P(s \upharpoonright_\gamma AB) \uplus P(s \upharpoonright_\gamma BC)$;
(2) *if $s = tm^{\Sigma}$ and:*
   - *$p(m) \in O_3$ and $s \upharpoonright_\gamma AC = t'm^{\Sigma'}$,*
   - *or $p(m) \in P_1$ and $s \upharpoonright_\gamma AB = t'm^{\Sigma'}$,*
   - *or $p(m) \in P_2$ and $s \upharpoonright_\gamma BC = t'm^{\Sigma'}$,*

   then $\nu(t) \cap \nu(m^{\Sigma'}) \subseteq \nu(t')$ and, in particular, if m introduces name a in $t'm^{\Sigma'}$ 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^{Σ} in s, with $p(m) \in PO$, and let $s \upharpoonright_\gamma AB = \cdots m^{\Sigma'} \cdots$. If $a \in \nu(m^{\Sigma'})$ 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^{Σ} in s, a contradiction to its being introduced at m^{Σ}. Similarly if $p(m) \in P_2$. Finally, if $p(m) \in O_3$ 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 $(\nu(m^{\Sigma'}) \setminus \nu(t')) \cap \nu(t) = \emptyset$. So suppose $a \in \nu(m^{\Sigma'}) \setminus \nu(t')$, 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 3.23. *For all $s \in Int(ABC)$, 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 γ, all projections satisfy frugality. Moreover, well-classing is preserved by projections. For well-calling, let $m = \mathsf{call}\ a.\mathsf{m}(\vec{v})$ be a move in $\underline{s}$ and let n^T be the move introducing a in s. Suppose $p(m) \in p(AB)$ and let us assume $p_{AB}(m) = O$. We need to show that $o_{AB}(m) = P$. By $p_{AB}(m) = O$ we obtain that $p(m) \in \{O_L, O_L P_R\}$ and, by well-calling of s, we have that $o(a) \in P_1$. Thus, $p(n) \in P_1$ and, by Lemma 3.22, n introduces a in $s \upharpoonright_\gamma AB$ and therefore $o_{AB}(n) = P$, as required. If, on the other hand, $p_{AB}(m) = P$ then we obtain $p(n) \in O_3 \cup P_2$ and therefore, by Lemma 3.22, $a \in P(s \upharpoonright_\gamma BC) \cup O(s \upharpoonright_\gamma AC)$. 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. □

Remark 3.24. A consequence of Proposition 3.23 is that, to compose plays $s_1 \in P_{AB}$ and $s_2 \in P_{BC}$ into plays in P_{AC}, it suffices to pick an interaction sequence $s \in Int(ABC)$ such that $s \upharpoonright_\gamma AB = s_1$ and $s \upharpoonright_\gamma BC = s_2$, and project s on AC by taking $s \upharpoonright_\gamma AC$. Note that the selection of an s which projects as s_1 and s_2 need not be unique. We look at an example of this next.

Example 3.25. We consider the interfaces:

$$\mathsf{CellCallable} = (\mathsf{set} : \mathsf{Callable} \to \mathsf{void},\ \mathsf{get} : \mathsf{void} \to \mathsf{Callable}), \quad \mathsf{Callable} = (\mathsf{foo} : \mathsf{void} \to \mathsf{void}).$$

along with the following interaction sequences s^1 and s^2 in CellCallable 1 Callable, written vertically for clarity:

CellCallable	1	Callable	*pol.*
c^{Σ_0}			O_L
$\mathsf{call}\ c.\mathsf{set}(a)^{\Sigma_1}$			P_L
$\mathsf{ret}\ c.\mathsf{set}(*)^{\Sigma_1}$			O_L
	$*^{\Sigma_1}$		$P_L O_R$
		b^{Σ_2}	P_R
		$\mathsf{call}\ b.\mathsf{foo}(*)^{\Sigma_2}$	O_R
		$\mathsf{ret}\ b.\mathsf{foo}(*)^{\Sigma_2}$	P_R
$\mathsf{call}\ a.\mathsf{foo}(*)^{\Sigma_2}$			O_L
$\mathsf{ret}\ a.\mathsf{foo}(*)^{\Sigma_2}$			P_L

CellCallable	1	Callable	*pol.*
c^{Σ_0}			O_L
$\mathsf{call}\ c.\mathsf{set}(a)^{\Sigma_1}$			P_L
$\mathsf{ret}\ c.\mathsf{set}(*)^{\Sigma_1}$			O_L
	$*^{\Sigma_1}$		$P_L O_R$
		b^{Σ_2}	P_R
$\mathsf{call}\ a.\mathsf{foo}(*)^{\Sigma_2}$			O_L
$\mathsf{ret}\ a.\mathsf{foo}(*)^{\Sigma_2}$			P_L
		$\mathsf{call}\ b.\mathsf{foo}(*)^{\Sigma_2}$	O_R
		$\mathsf{ret}\ b.\mathsf{foo}(*)^{\Sigma_2}$	P_R

with $\Sigma_0 = \{c \mapsto (\mathsf{CellCallable}, \emptyset)\}$, $\Sigma_1 = \Sigma_0[a \mapsto (\mathsf{Callable}, \emptyset)]$ and $\Sigma_2 = \Sigma_1[b \mapsto (\mathsf{Callable}, \emptyset)]$. We notice that the projections on CellCallable1 and 1Callable are the same for the two sequences:

$$s_1 = s^i \upharpoonright_\gamma \mathsf{CellCallable1} = c^{\Sigma_0}\ \mathsf{call}\ c.\mathsf{set}(a)^{\Sigma_1}\ \mathsf{ret}\ c.\mathsf{set}(*)^{\Sigma_1}\ \mathsf{call}\ a.\mathsf{foo}(*)^{\Sigma_2 \upharpoonright \{c,a\}}\ \mathsf{ret}\ a.\mathsf{foo}(*)^{\Sigma_2 \upharpoonright \{c,a\}}$$
$$s_2 = s^i \upharpoonright_\gamma \mathsf{1Callable} = *^{\emptyset}\ b^{\Sigma_2 \upharpoonright \{b\}}\ \mathsf{call}\ b.\mathsf{foo}(*)^{\Sigma_2 \upharpoonright \{b\}}\ \mathsf{ret}\ b.\mathsf{foo}(*)^{\Sigma_2 \upharpoonright \{b\}}$$

On the other hand, s^1 and s^2 are distinct, and their projections on CellCallable Callable:

$$s_3^1 = c^{\Sigma_0}\ \mathsf{call}\ c.\mathsf{set}(a)^{\Sigma_1}\ \mathsf{ret}\ c.\mathsf{set}(*)^{\Sigma_1}\ b^{\Sigma_2}\ \mathsf{call}\ b.\mathsf{foo}(*)^{\Sigma_2}\ \mathsf{ret}\ b.\mathsf{foo}(*)^{\Sigma_2}\ \mathsf{call}\ a.\mathsf{foo}(*)^{\Sigma_2}\ \mathsf{ret}\ a.\mathsf{foo}(*)^{\Sigma_2}$$
$$s_3^2 = c^{\Sigma_0}\ \mathsf{call}\ c.\mathsf{set}(a)^{\Sigma_1}\ \mathsf{ret}\ c.\mathsf{set}(*)^{\Sigma_1}\ b^{\Sigma_2}\ \mathsf{call}\ a.\mathsf{foo}(*)^{\Sigma_2}\ \mathsf{ret}\ a.\mathsf{foo}(*)^{\Sigma_2}\ \mathsf{call}\ b.\mathsf{foo}(*)^{\Sigma_2}\ \mathsf{ret}\ b.\mathsf{foo}(*)^{\Sigma_2}$$

are also distinct. Put otherwise, composing s_1 and s_2 we can obtain both of s_3^1 and s_3^2.

In practice, composing plays by picking interaction sequences which project to them is impractical. In the remainder of this section we will introduce an equivalent operational way to compose plays.

Definition 3.26. Let $s_1 \in P_{AB}$ and $s_2 \in P_{BC}$. Using the rules of Figure 5, we define a transition system whose states are tuples of the form (X, s_1', s_2', s_3, p) where:

$$(O_3, m_A^{\Sigma} s_1, s_2, \epsilon, \emptyset) \to (P_1, s_1, s_2, m_A^{\Sigma}, \{m_A \mapsto O_L\})$$

$$(O_3, m^{\Sigma} s_1, s_2, s_3, p) \to (P_1, s_1, s_2, s_3 m^{\Sigma_3'[\Sigma]}, p[m \mapsto O_L]) \qquad (O_3, s_1, m^{\Sigma} s_2, s_3, p) \to (P_2, s_1, s_2, s_3 m^{\Sigma_3'[\Sigma]}, p[m \mapsto O_R])$$

$$(P_1, m^{\Sigma} s_1, s_2, s_3, p) \to (O_3, s_1, s_2, s_3 m^{\Sigma_3[\Sigma]}, p[m \mapsto P_L]) \quad (P_1, m^{\Sigma_1} s_1, m^{\Sigma_2} s_2, s_3, p) \to (P_2, s_1, s_2, s_3 m^{\Sigma_3[\Sigma_1]\cup\Sigma_2}, p[m \mapsto P_L O_R])$$

$$(P_2, s_1, m^{\Sigma} s_2, s_3, p) \to (O_3, s_1, s_2, s_3 m^{\Sigma_3[\Sigma]}, p[m \mapsto P_R]) \quad (P_2, m^{\Sigma_1} s_1, m^{\Sigma_2} s_2, s_3, p) \to (P_1, s_1, s_2, s_3 m^{\Sigma_3[\Sigma_2]\cup\Sigma_1}, p[m \mapsto O_L P_R])$$

Fig. 5. Transition system for play composition (cf. Definition 3.26). All rules are subject to the side-condition that the resulting s_3 be an interaction sequence with polarity function the resulting p. Σ_3 stands for the last store in s_3, and Σ_3' ranges over all stores.

- $X \in \{P_1, P_2, O_3\}$,
- s_1', s_2' are suffixes of s_1 and s_2 respectively,
- $s_3 \in Int(ABC)$ with polarity function p.

We then set: $s_1 \| s_2 = \{\, s_3 \mid (O_3, s_1, s_2, \epsilon, \emptyset) \to^* (X, \epsilon, \epsilon, s_3, p) \,\}$.

Lemma 3.27. *For s_1, s_2 as above and $s_3 \in Int(ABC)$, the following are equivalent.*

(1) $s_1 = s_3 \upharpoonright_\gamma AB$ *and* $s_2 = s_3 \upharpoonright_\gamma BC$,
(2) $s_3 \in s_1 \| s_2$.

Therefore: $s_1 \| s_2 = \{s_3 \in Int(ABC) \mid s_3 \upharpoonright_\gamma AB = s_1 \wedge s_3 \upharpoonright_\gamma BC = s_2\}$.

Proof. $2 \Rightarrow 1$. By inspection of the transition rules, we have $\underline{s_1} = \underline{s_3 \upharpoonright AB}$ and $\underline{s_2} = \underline{s_3 \upharpoonright BC}$. We also notice that the stores of s_1 and s_2 are included in s_3 (they are merely extended with extra names that are not available when projecting), so applying the availability function we indeed retrieve s_1 and s_2.

$1 \Rightarrow 2$. We show that for every prefix s_3' of s_3 there is a transition sequence $(O_3, s_1, s_2, \epsilon, \emptyset) \to^* (X, s_1', s_2', s_3', p)$ such that $s_1 = s_1^0 s_1'$, $s_2 = s_2^0 s_2'$, $s_1^0 = s_3' \upharpoonright_\gamma AB$ and $s_2^0 = s_3' \upharpoonright_\gamma BC$. We do induction on s_3'. The cases where s_3' has length at most 1 are straightforward. Suppose $s_3' = s_3'' m^{\Sigma}$. By IH, there is a transition sequence $(O_3, s_1, s_2, \epsilon, \emptyset) \to^* (X, s_1', s_2', s_3'', p')$. We do a case analysis on the polarity of m (taken from s_3).

Suppose the polarity is $P_L O_R$ ($O_L P_R, P_L, P_R$ are treated similarly). Since the transition sequence follows the alternation diagram of Figure 4, X will be equal to P_1. Since $s_1^0 = s_3' \upharpoonright_\gamma AB$ and $s_1 = s_3 \upharpoonright_\gamma AB$, and $s_2^0 = s_3' \upharpoonright_\gamma BC$ and $s_2 = s_3 \upharpoonright_\gamma BC$, it must be the case that $s_1' = m^{\Sigma_1} s_1''$ for some Σ_1 and s_1'', and $s_2' = m^{\Sigma_2} s_2''$ for some Σ_2 and s_2''. We apply the reduction $(P_1, m^{\Sigma_1} s_1'', m^{\Sigma_2} s_2'', s_3', p) \to (P_2, s_1'', s_2'', s_3' m^{\Sigma_3[\Sigma_1]\cup\Sigma_2}, p[m \mapsto P_L O_R])$. It now suffices to show that $\Sigma_3[\Sigma_1] \cup \Sigma_2 = \Sigma$. Note that Σ_1 and Σ_2 are both restrictions of Σ (by hypothesis), and Σ_3 is the store preceding Σ in s_3. Thus, $\mathrm{dom}(\Sigma_i) \subseteq \mathrm{dom}(\Sigma)$ for $i = 1, 2, 3$. we have $\mathrm{dom}(\Sigma_3[\Sigma_1 \cup \Sigma_2]) \subseteq \mathrm{dom}(\Sigma)$. If $a \in \mathrm{dom}(\Sigma) \setminus \mathrm{dom}(\Sigma_3)$ then, by Lemma 3.22, $a \in \mathrm{dom}(\Sigma_1)$, so $\mathrm{dom}(\Sigma_3[\Sigma_1]) = \mathrm{dom}(\Sigma)$. Moreover, if $a \in \mathrm{dom}(\Sigma_3) \setminus \mathrm{dom}(\Sigma_1)$ then $\Sigma_3(a) = \Sigma_1(a)$ (by Laird conditions), and thus $\Sigma_3[\Sigma_1] = \Sigma$. Since Σ_2 is a restriction of Σ, $\Sigma_3[\Sigma_1] \cup \Sigma_2$ is a valid store and equal to Σ.

Suppose the polarity is O_L (O_R is treated similarly). Similarly to above, X will be equal to O_3 and in this case we will have $s_1 = s_1^0 s_1'$, $s_1^0 = s_3' \upharpoonright_\gamma AB$ and $s_1' = m^{\Sigma_1} s_1''$ for some Σ_1 and s_1''. We apply the reduction $(O_3, m^{\Sigma_1} s_1'', s_2', s_3', p) \to (P_1, s_1'', s_2', s_3' m^{\Sigma[\Sigma_1]}, p[m \mapsto O_L])$. Since $\Sigma[\Sigma_1] = \Sigma$, we are done.

Thus, we have that $(O_3, s_1, s_2, \epsilon, \emptyset) \to^* (X, s_1', s_2', s_3, p)$ such that $s_1 = s_1^0 s_1'$, $s_2 = s_2^0 s_2'$, $s_1^0 = s_3 \upharpoonright_\gamma AB$ and $s_2^0 = s_3 \upharpoonright_\gamma BC$. By the hypothesis, $s_1' = s_2' = \epsilon$, which yields $s_3 = s_1 \| s_2$. □

3.4 Strategies and the category of games

Programs will be represented as *strategies* between arenas. We shall introduce them next after some auxiliary definitions. Intuitively, strategies capture the observable computational patterns produced by a program.

Let us first define the following notion of subtyping between stores. For $\Sigma, \Sigma' \in \mathit{Sto}$, $\Sigma \leq \Sigma'$ holds if, for all names a,

$$\Sigma'(a) : \mathcal{I}' \implies \Sigma(a) \leq \mathcal{I}' \wedge \forall \mathsf{f} \in \mathsf{dom}(\Sigma'(a)).\Sigma(a).\mathsf{f} = \Sigma'(a).\mathsf{f}$$

Put otherwise, $\Sigma \leq \Sigma'$ means that Σ *extends* Σ' in this way: if Σ' assigns to some name a an interface and field values, then Σ assigns to a the same field values, and the same interface type or a subtype thereof. In particular, Σ may contain more information about a because of assigning to a a larger interface. Accordingly, for plays $s, s' \in P_{AB}$, 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:

$$\frac{}{\epsilon \leq_O \epsilon} \qquad \frac{s \leq_O s' \quad \Sigma \leq \Sigma' \quad \Sigma \upharpoonright P(sm^{\Sigma}) \subseteq \Sigma'}{sm^{\Sigma} \leq_O s'm^{\Sigma'}}\, p(m){=}O \qquad \frac{sn^{T} \leq_O s' \quad \Sigma \leq \Sigma' \quad \Sigma \text{ extends } \Sigma' \text{ by } T}{sn^{T}m^{\Sigma} \leq_O s'm^{\Sigma'}}\, p(m){=}P$$

where Σ extends Σ' by T if:

- for all $a \in \mathsf{dom}(\Sigma) \setminus \mathsf{dom}(\Sigma')$, $\Sigma(a) = T(a)$;
- for all a and $\mathsf{f} \in \mathsf{dom}(\Sigma(a)) \setminus \mathsf{dom}(\Sigma'(a))$, $\Sigma(a).\mathsf{f} = T(a).\mathsf{f}$.

The utility of O-extension is to express semantically 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.

Strategies shall be sets of plays, representing how Proponent should behave in the given game. The sets will satisfy a number of conditions stipulating that:

- each play in a strategy is even-length, thus representing a configuration in the game where P has just played (because of alternation and the fact that O always plays first);
- if a play can be reached by a strategy, then all of its even-length prefixes can be reached as well;
- the strategy is deterministic, i.e. for each O move extending a play from a strategy there is at most one move that P can play after it;
- the names that appear in a strategy should be interchangeable – the strategy manipulates names in the same way up to permutation;
- the behaviour of P should be invariant under extensions via subclassing of objects belonging to O.

We formalise strategies and these conditions next.

Definition 3.28. A ***strategy*** σ in AB is a non-empty set of even-length plays from P_{AB} satisfying the conditions:

- If $sm^{\Sigma}n^{T} \in \sigma$ then $s \in \sigma$. (*Even-prefix closure*)
- If $sm^{\Sigma}, sn^{T} \in \sigma$ then $sm^{\Sigma} \sim sn^{T}$. (*Determinacy*)
- 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 $\sigma : A \to B$ when σ is a strategy in AB.

[4]Recall that, for any nominal set X and $x, y \in X$, we write $x \sim y$ just if there is a permutation π such that $x = \pi \cdot y$.

We can check that the plays listed in Examples 2.10, 2.11 and 2.12 form strategies. 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:

$$\mathtt{id}_A : A \to A = \{m_A^\Sigma m_A^\Sigma \in P_{AA}\}$$

Below is a more interesting example, involving O-extension.

Example 3.29. Consider $\Delta = \{\mathsf{Empty} : \emptyset, \mathsf{Point}\langle\mathsf{Empty}\rangle : (\mathsf{x} : \mathsf{int}, \mathsf{y} : \mathsf{int})\}$ from Example 2.12. By *O-extension*, any strategy $\sigma : \mathsf{Empty} \to \mathsf{Empty}$ containing $s_a = a^{\{a \mapsto (\mathsf{Empty}, \emptyset)\}}\, a^{\{a \mapsto (\mathsf{Empty}, \emptyset)\}}$ must also contain

$$s_a^{i,j} = a^{\{a \mapsto (\mathsf{Point}, \{\mathsf{x} \mapsto i, \mathsf{y} \mapsto j\})}\, a^{\{a \mapsto (\mathsf{Point}, \{\mathsf{x} \mapsto i, \mathsf{y} \mapsto j\})}$$

for any $i, j \in \mathbb{Z}$. Consequently, the set $\sigma^- = \{s_a \mid a \in \mathsf{Names}\}$ cannot be a strategy. However, σ^- can be extended to a strategy σ as follows:

$$\sigma = \sigma^- \cup \{s_a^{i,j} \mid a \in \mathsf{Names},\ i, j \in \mathbb{Z}\}$$

Note also that $\{s_a^{i,j} \mid a \in \mathsf{Names},\ i, j \in \mathbb{Z}\}$ is also a strategy in $\mathsf{Empty}\,\mathsf{Empty}$. While the former corresponds to $\Delta|x : \mathsf{Empty} \vdash x : \mathsf{Empty}$, the latter will turn out to model $\Delta|x : \mathsf{Empty} \vdash (\mathsf{Empty})(\mathsf{Point})x : \mathsf{Empty}$.

We saw in the previous section how plays can be composed via interaction sequences. Strategies can also be composed, simply by composing their plays.

Definition 3.30. Given $\sigma : A \to B$ and $\tau : B \to C$, we define their ***composition*** $\sigma; \tau$ by:

$$\sigma; \tau = \{s \upharpoonright_\gamma AC \mid s \in \sigma \| \tau\}$$

where $\sigma \| \tau = \{s \in \mathit{Int}(ABC) \mid s \upharpoonright_\gamma AB \in \sigma \wedge s \upharpoonright_\gamma BC \in \tau\}$.

A simple class of examples of strategy composition involves composing with the identity strategy. We first look at an example of this, before proving that identity strategies are identities (i.e. neutral) under composition.

Example 3.31. We recall the interface $\mathsf{Callable} = (\mathsf{foo} : \mathsf{void} \to \mathsf{void})$ of Example 3.10, and consider the strategies:

$$\mathtt{id}_{\mathsf{Callable}} : \mathsf{Callable} \to \mathsf{Callable} = \{a^\Sigma a^\Sigma \mid a \in \mathsf{Names} \wedge \Sigma = \{a \mapsto (\emptyset, \mathsf{Callable})\}\}$$

$$\sigma : \mathsf{Callable} \to 1 = \{a^\Sigma \mathsf{call}\, a.\mathsf{foo}(*)^\Sigma,\ a^\Sigma \mathsf{call}\, a.\mathsf{foo}(*)^\Sigma \mathsf{ret}\, a.\mathsf{foo}(*)^\Sigma *^\Sigma \mid a \in \mathsf{Names} \wedge \Sigma = \{a \mapsto (\emptyset, \mathsf{Callable})\}\}$$

with the aim to compose them to $\mathtt{id}_{\mathsf{Callable}}; \sigma$. To obtain this, we look at all interaction sequences in $\mathsf{Callable}_1\mathsf{Callable}_2 1$ which project respectively in $\mathtt{id}_{\mathsf{Callable}}$ and σ. Since these strategies have a unique maximal play up to permutation, the common interaction sequences are all subsequences of the following $s_3 \in \mathit{Int}(\mathsf{Callable}_1\mathsf{Callable}_2 1)$.

$\mathsf{Callable}_1$	$\mathsf{Callable}_2$	1	*pol.*
a^Σ			O_L
	a^Σ		$P_L O_R$
		$\mathsf{call}\, a.\mathsf{foo}(*)^\Sigma$	P_R
		$\mathsf{ret}\, a.\mathsf{foo}(*)^\Sigma$	O_R
		$*^\Sigma$	P_R

We can now compute $s_3 \upharpoonright_\gamma \mathsf{Callable}_1 1 = a^\Sigma \mathsf{call}\, a.\mathsf{foo}(*)^\Sigma \mathsf{ret}\, a.\mathsf{foo}(*)^\Sigma *^\Sigma$. Considering all even-length subsequences of the latter, we conclude that $\mathtt{id}_{\mathsf{Callable}}; \sigma = \sigma$.

Proposition 3.32. *For all $\sigma : A \to B$, we have $\mathtt{id}_A; \sigma = \sigma; \mathtt{id}_B = \sigma$.*

Proof. We show that $\mathtt{id}_A;\sigma = \sigma$, and the other equality is proven similarly. Let us tag the two copies of arena A by setting $\mathtt{id}_A : A_1 \longrightarrow A_2$ and $\sigma : A_2 \longrightarrow B$. By definition,

$$\mathtt{id}_A;\sigma = \{s \restriction_\gamma A_1B \mid s \in Int(A_1A_2B) \wedge s \restriction_\gamma A_1A_2 \in \mathtt{id}_A \wedge s \restriction_\gamma A_2B \in \sigma\}.$$

So, let $s \in Int(A_1A_2B)$ be such that $s \restriction_\gamma A_1A_2 = m^\Sigma_{A_1} m^\Sigma_{A_2}$ (for some m^Σ_A) and $s_2 = s \restriction_\gamma A_2B \in \sigma$. By Definition 3.26, it must be the case that $s = m^\Sigma_{A_1} m^\Sigma_{A_2} s'$ and, since $s \restriction_\gamma A_1A_2$ contains only two moves, all moves in s' have polarities from $p(m) \in \{P_R, O_R\}$. Therefore, $s \restriction_\gamma A_1B = m^\Sigma_A(s' \restriction_\gamma A_1B) = m^\Sigma_A(s' \restriction_\gamma A_2B) = s \restriction_\gamma A_2B$ and, hence, $\mathtt{id}_A;\sigma \subseteq \sigma$. Conversely, if $m^\Sigma_A s' \in \sigma$ then this yields a sequence $s = m^\Sigma_{A_1} m^\Sigma_{A_2} s' \in Int(A_1A_2B)$. Reasoning as above, s has the same projection on A_1B and A_2B, and we conclude that $\sigma \subseteq \mathtt{id}_A;\sigma$. □

Example 3.33. Recall the two interacting plays from Example 3.19.

$$\begin{aligned} s_1 \;=\;& c_1^{\Sigma_0}\ \mathsf{call}\, c_1.\mathsf{get}(*)^{\Sigma_0}\ \mathsf{ret}\, c_1.\mathsf{get}(a_1)^{\Sigma_1}\ \mathsf{call}\, c_1.\mathsf{set}(a_2)^{\Sigma_2}\ \mathsf{ret}\, c_1.\mathsf{set}(*)^{\Sigma_3}\ c_2^{\Sigma_4}\ \mathsf{call}\, c_2.\mathsf{get}(*)^{\Sigma_4}\ \mathsf{ret}\, c_2.\mathsf{get}(a_3)^{\Sigma_5} \\ & \mathsf{call}\, c_2.\mathsf{set}(a_4)^{\Sigma_6}\ \mathsf{call}\, c_1.\mathsf{set}(a_5)^{\Sigma_7}\ \mathsf{ret}\, c_1.\mathsf{set}(*)^{\Sigma_8}\ \mathsf{ret}\, c_2.\mathsf{set}(*)^{\Sigma_9} \\ s_2 \;=\;& c_2^{\Sigma_4 \restriction \{c_2\}}\ \mathsf{call}\, c_2.\mathsf{get}(*)^{\Sigma_4 \restriction \{c_2\}}\ \mathsf{ret}\, c_2.\mathsf{get}(a_3)^{\Sigma_5 \restriction \{c_2,a_3\}}\ \mathsf{call}\, c_2.\mathsf{set}(a_4)^{\Sigma_6 \restriction \{c_2,a_3,a_4\}} \end{aligned}$$

Let $\sigma : \mathsf{CellVarInt}_1 \longrightarrow \mathsf{CellVarInt}_2$ and $\tau : \mathsf{CellVarInt}_2 \longrightarrow 1$ be the smallest strategies containing s_1 and s_2 respectively, which can be obtained in this case by adding new plays to satisfy Even-prefix closure and Equivariance. Then $\sigma;\tau : \mathsf{CellVarInt}_1 \longrightarrow 1$ is the smallest strategy containing the play:

$$\begin{aligned} s_3 \;=\;& c_1^{\Sigma_0}\ \mathsf{call}\, c_1.\mathsf{get}(*)^{\Sigma_0}\ \mathsf{ret}\, c_1.\mathsf{get}(a_1)^{\Sigma_1}\ \mathsf{call}\, c_1.\mathsf{set}(a_2)^{\Sigma_2}\ \mathsf{ret}\, c_1.\mathsf{set}(*)^{\Sigma_3}\ \mathsf{call}\, c_1.\mathsf{set}(a_5)^{\Sigma_7 \restriction \{c_1,a_1,a_2,a_5\}} \\ & \mathsf{ret}\, c_1.\mathsf{set}(*)^{\Sigma_8 \restriction \{c_1,a_1,a_2,a_5\}}\ *^{\Sigma_{10} \restriction \{c_1,a_1,a_2,a_5\}} \end{aligned}$$

obtained by projecting the interaction sequence s on $\mathsf{CellVarInt}_1$ 1.

The next two lemmata will allow us to show that strategy composition is well defined (Proposition 3.36). Moreover, strategy composition is associative (Proposition 3.37) and thus arenas and strategies form a category (Definition 3.38).

Lemma 3.34. *If $sm^\Sigma, sn^T \in \sigma\|\tau$ with $p(m) \notin O_3$ then $sm^\Sigma \sim sn^T$. Hence, if $s_1m^\Sigma, s_2n^T \in \sigma\|\tau$ with $p(m) \notin O_3$ and $s_1 \sim s_2$ then $s_1m^\Sigma \sim s_2n^T$.*

Proof. For the first part, suppose WLOG that $p(m) \in PO$. Then, by the diagram in Figure 4, we also have $p(n) \in PO$. As $sm^\Sigma, sn^T \restriction_\gamma AB \in \sigma$, by determinacy of σ we get $s'm^{\Sigma'} \sim s'n^{T'}$ with $s'm^{\Sigma'} = sm^\Sigma \restriction_\gamma AB$ and $s'n^{T'} = sn^T \restriction_\gamma AB$. We therefore have $(s', m^{\Sigma'}) \sim (s', n^T)$ and, trivially, $(s, s') \sim (s, s')$. Moreover, by Lemma 3.22, $\nu(m^{\Sigma'}) \cap \nu(s) \subseteq \nu(s')$ and $\nu(n^{T'}) \cap \nu(s) \subseteq \nu(s')$ hence, by Strong Support Lemma [45], $sm^{\Sigma'} \sim sn^{T'}$. By Laird's last set of conditions, the remaining values of Σ, T are determined by the last store in s, hence $sm^\Sigma \sim sn^T$.

For the second part, suppose $s_1 = \pi \cdot s_2$. Then, since $\pi \cdot (s_2n^T) \in \sigma\|\tau$, by the first part we have $s_1m^\Sigma \sim \pi \cdot (s_2n^T)$, so $s_1m^\Sigma \sim s_2n^T$. □

Lemma 3.35. *If $s_1, s_2 \in \sigma\|\tau$ end in moves with polarities in $p(AC)$ and $s_1 \restriction_\gamma AC = s_2 \restriction_\gamma AC$ then $s_1 \sim s_2$.*

Proof. By induction on $|s_1 \restriction_\gamma AC| > 0$. The base case is encompassed in $s_i = s_i'm^{\Sigma_i}$ with $p(m) \in O_3$, $i = 1, 2$, where note that by IH m will have the same polarity in s_1, s_2. Then, by IH we get $s_1' = \pi \cdot s_2'$, for some π. Let $s_i''m^{\Sigma'} = s_i \restriction_\gamma AC$, for $i = 1, 2$, so in particular $s_1'' = \pi \cdot s_2''$ and therefore $(s_1', s_1'') \sim (s_2', s_2'')$. Moreover, by hypothesis, we trivially have $(m^{\Sigma'}, s_1'') \sim (m^{\Sigma'}, s_2'')$ and hence, by Lemma 3.22 and Strong Support Lemma [45], we obtain $s_1'm^{\Sigma'} \sim s_2'm^{\Sigma'}$ which implies $s_1 \sim s_2$ by Laird's conditions. Suppose now $s_i = s_i's_i''m^{\Sigma_i}$, $i = 1, 2$, with $p(m) \in P(AC) \setminus O_3$ and the last move in

s_i' being the last move in $s_i's_i''$ having polarity in $p(AC)$. By IH, $s_1' \sim s_2'$. Then, by consecutive applications of Lemma 3.34, we obtain $s_1 \sim s_2$. □

Proposition 3.36. *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 $t \leq_O s$ 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 $v \leq_O u$, where $\leq_O$ is defined for interaction sequences in an analogous way as for plays (with condition $p(m) = O$ replaced by $p(m) \in O_3$, and $p(m) = P$ by $p(m) \in PO \cup P_2$). Moreover, $v \restriction_\gamma AB \leq_O u \restriction_\gamma AB$ and $v \restriction_\gamma BC \leq_O u \restriction_\gamma BC$, so $t \in \sigma;\tau$. Finally, for determinacy, let $sm^\Sigma, sn^T \in \sigma;\tau$ be due to $s_1s_1'm^{\Sigma'}, s_2s_2'n^{T'} \in \sigma\|\tau$ respectively, where s_1, s_2 both end in the last move of s. By Lemma 3.35, we have $s_1 \sim s_2$ and thus, by consecutive applications of Lemma 3.34, we get $s_1s_1'm^{\Sigma'} \sim s_2s_2'n^{T'}$, so $sm^\Sigma \sim sn^T$. □

Proposition 3.37. *For all $\rho : A \to B$, $\sigma : B \to C$ and $\tau : C \to D$, $(\rho;\sigma);\tau = \rho;(\sigma;\tau)$.*

Proof. We delegate the technical argument to Appendix A. □

Definition 3.38. Given an interface table Δ, we define the category $\mathcal{G}_\Delta$ having arenas as objects and strategies as morphisms. Identity morphisms are given by $\mathtt{id}_A$, for each arena A.

The dependence of the category $\mathcal{G}_\Delta$ on the interface table Δ is due to the fact that the stores appearing in plays (inside strategies in $\mathcal{G}_\Delta$) must obey to well-formedness conditions imposed by Δ (cf. Definition 3.4). In the sequel, when Δ 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:

$$(\Delta/\Delta')(\sigma) = \{s \in P_{AB}^{\Delta'} \mid \exists t \in \sigma.\, s \leq_O t\}$$

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

$$(\Delta'/\Delta)(\sigma) = \sigma \cap P_{AB}^{\Delta}$$

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

4 MODEL STRUCTURE

The aim of this section is to bring out the categorical structure that will allow us to build a model of IMJ. To that end, we shall employ several classes of strategies to demonstrate that our setting is compatible with categorical requirements for modelling call-by-value evaluation.

The first class of strategies will be called *evaluated*. These strategies, say in $A \to B$, after the initial move m_A^Σ, respond immediately with m_B^Σ without modifying the content of the store nor depending on it. Intuitively, this corresponds to terms that have already been evaluated, i.e. skip, null, i, a. These strategies will turn out to support finite products and a notion of left pairing (needed to simulate the order of evaluation). At the abstract level, this will lead us to a setting with the same properties as Freyd categories [40], known to be the categorical counterparts of call-by-value program calculi. We will also define a weak notion of coproduct, useful for modelling the conditionals in the language.

Other prominent classes of strategies will be the *single-threaded* ones. They are similar to evaluated strategies in that m_B is played immediately after m_A, and that the initial store content cannot be modified. However, in contrast to evaluated strategies, they allow for the introduction of new names in the store component (object creation). Further, some interaction is allowed with the new object, but only through a single call, i.e. only a single thread of play is allowed. *Well-threaded* strategies generalise single-threaded strategies by allowing multiple calls. However, the resultant threads need to be uniform so that each well-threaded strategy can be viewed as being generated by a single-threaded strategy. This gives rise to an adjunction that leads to a notion of exponentiation that can be used to model λ-abstraction and, consequently, method definitions (i.e. $\mathsf{m} : \lambda\vec{x}.M$). Single-threaded strategies will correspond to single method invocations, whereas thread-independent ones to method-set implementations (i.e. sets of method definitions).

Finally, to model self-referencing in objects, i.e. x in $\mathsf{new}(x : \mathcal{I}; \mathcal{M})$, we use a dedicated 'copycat' construction between interfaces.

4.1 Evaluated strategies

A strategy $\sigma : A \to B$ is called ***evaluated*** if there is a function $f_\sigma : M_A \to M_B$ such that:

$$\sigma = \{m_A^\Sigma m_B^\Sigma \in P_{AB} \mid m_B = f_\sigma(m_A)\}\,.$$

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

For example, identity strategies are evaluated. More importantly, since evaluated strategies are free from state dependence, restricting strategies to evaluated ones yields a category with products.

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

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

and, analogously, $\pi_2 : A \times B \to B$. Clearly, these strategies are evaluated. Moreover, for each object A,

$$!_A = \{m_A^\Sigma *^\Sigma \mid m_A^\Sigma \in P_{A1}\}$$

is the unique evaluated strategy of type $A \to 1$.

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

$$\langle \sigma, \tau \rangle : A \to B \times C = \{m_A^\Sigma s[(m_B, f_\tau(m_A))/m_B] \mid m_A^\Sigma s \in \sigma\}$$

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

From the previous definitions we obtain the following properties.

Lemma 4.1. *Evaluated strategies form a wide subcategory $\mathcal{V}$ of $\mathcal{G}$ with finite products given by the above constructions. Moreover, for all $\sigma : A \to B$ and $\tau : A \to C$ with τ evaluated, $\langle \sigma, \tau \rangle ; \pi_1 = \sigma$ and $\langle \sigma, \tau \rangle = \langle \sigma, \mathsf{id}_A \rangle ; \langle \pi_1, \pi_2 ; \tau \rangle$.*

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

$$\langle \sigma, \tau \rangle \;=\; A \xrightarrow{\langle \sigma, \mathsf{id}_A \rangle} B \times A \xrightarrow{\langle \pi_2 ; \tau, \pi_1 \rangle} C \times B \xrightarrow{\cong} B \times C$$

where $\cong$ is the isomorphism $\langle \pi_2, \pi_1 \rangle$. The above represents a notion of *left-pairing* of σ and τ, where the effects of σ precede those of τ. We can also define a *left-tensor* between strategies:

$$\sigma \times \tau \;=\; A \times B \xrightarrow{\langle \pi_1; \sigma, \pi_2 \rangle} A' \times B \xrightarrow{\langle \pi_1, \pi_2; \tau \rangle} A' \times B'$$

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

Lemma 4.2. *Let $\tau' : A' \to A$, $\sigma : A \to B_1$, $\tau : A \to B_2$, $\sigma_1 : B_1 \times B_2 \to C_1$ and $\sigma_2 : B_2 \to C_2$, with τ and τ' evaluated. Then $\tau'; \langle \sigma, \tau \rangle; \langle \sigma_1, \pi_2; \sigma_2 \rangle = \langle \tau'; \langle \sigma, \tau \rangle; \sigma_1, \tau'; \tau; \sigma_2 \rangle$.*

Proof. The result follows from the simpler statements:

$$\tau; \langle \sigma, \mathtt{id} \rangle = \langle \tau; \sigma, \tau \rangle, \quad \langle \sigma, \mathtt{id} \rangle; \langle \sigma', \pi_2 \rangle = \langle \langle \sigma; \mathtt{id} \rangle; \sigma', \mathtt{id} \rangle,$$

for all appropriately typed σ, σ', τ, with τ evaluated, and Lemma 4.1. □

An immediate consequence of the above is:

$$A \xrightarrow{\langle \sigma; \tau \rangle} B_1 \times B_2 \xrightarrow{\sigma_1 \times \sigma_2} C_1 \times C_2 \;=\; A \xrightarrow{\langle \sigma; \sigma_1, \tau; \sigma_2 \rangle} C_1 \times C_2$$

More generally, we can use Lemma 4.2 to show that $\mathcal{V}$ and $\mathcal{G}$, along with the inclusion functor $I : \mathcal{V} \to \mathcal{G}$, yield a Freyd category [40, 41].

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

$$[\sigma, \tau] : \mathbb{Z} \times A \to B = \{(i, m_A)^{\Sigma} s \mid i \neq 0 \wedge m_A^{\Sigma} s \in \sigma\} \cup \{(0, m_A)^{\Sigma} s \mid m_A^{\Sigma} s \in \tau\}$$

Setting $\hat{i} : 1 \to \mathbb{Z} = \{* \, i\}$, for each $i \in \mathbb{Z}$, we can show the following.

Lemma 4.3. *For all strategies $\sigma' : A' \to A$ and $\sigma, \tau : A \to B$,*

- *$\langle !; \hat{0}, \mathtt{id} \rangle; [\sigma, \tau] = \tau$ and $\langle !; \hat{i}, \mathtt{id} \rangle; [\sigma, \tau] = \sigma$ if $i \neq 0$;*
- *if σ' is evaluated then $(\mathtt{id}_{\mathbb{Z}} \times \sigma'); [\sigma, \tau] = [\sigma'; \sigma, \sigma'; \tau]$.*

4.2 Single-threaded and thread-independent strategies

Method definitions in IMJ amount to a form of exponentiation:

$$\frac{\bigwedge_{i=1}^{n} (\Delta | \Gamma \uplus \{\vec{x}_i : \vec{\theta}_i\}; u \vdash M_i : \theta_i)}{\Delta | \Gamma; u \vdash \mathcal{M} : \Theta} \quad \Theta = \{\mathsf{m}_i : \vec{\theta}_i \to \theta_i \mid 1 \leq i \leq n\} \wedge \mathcal{M} = \{\mathsf{m}_i : \lambda \vec{x}_i . M_i \mid 1 \leq i \leq n\}$$

the modelling of which requires some extra semantic machinery. Traditionally, in call-by-value game models, exponentiation 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, exponentiation 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 $s = m_A^{\Sigma} m_B^{\Sigma \uplus T} s'$ and:

- $T \in Sto_0$ and $\nu(m_B) \cap \nu(\Sigma) \subseteq \nu(m_A)$,
- if $s'' m^{\Sigma'} n^{T'} \sqsubseteq s'$ is even-length and $a \in \mathsf{dom}(\Sigma) \setminus \nu(\gamma(m_A^{\Sigma_0} m_B^{\Sigma_0 \uplus T} s'' m^{\Sigma'}))$, for $\Sigma_0 = \mathrm{Dft}(\Sigma) \restriction \nu(m_A)$, then $T'(a) = \Sigma'(a)$ and $a \notin \nu(\gamma(m_A^{\Sigma_0} m_B^{\Sigma_0 \uplus T} s'' m^{\Sigma'} n^{T'}))$.

We write P^{t}_{AB} for the set of total plays in AB. Thus, in total plays, the initial move m^{Σ}_{A} is immediately followed by a move $m^{T'}_{B}$, and the initial store Σ is *invisible* to P in the sense that P cannot use its names or their values (can neither read from nor write to Σ). Moreover, $m^{T'}_{B}$ may introduce some fresh objects, albeit with default values.

A consequence of totality is that the first store played can be replaced by a default one.

Lemma 4.4. *If $m^{\Sigma}_{A} m^{\Sigma \uplus T}_{B} s'$ is a total play then so is $\gamma(m^{\Sigma_0}_{A} m^{\Sigma_0 \uplus T}_{A} s')$, where $\Sigma_0 = \mathsf{Dft}(\Sigma) \upharpoonright \nu(m_A)$.*

Hence, total plays impose that the only initial effect available to P is the creation of fresh objects, which appear in the second move of the play. Next, we look at strategies that impose that, to those objects, at most one call can be made in the remainder of the play.

Definition 4.5. A strategy $\phi : A \to B$ is called ***single-threaded*** if it consists of total plays and satisfies the conditions:[5]

- for all $m^{\Sigma}_{A} \in P_{AB}$ there is $m^{\Sigma}_{A} m^{T}_{B} \in \phi$;
- for all $m^{\Sigma}_{A}\, m^{\Sigma \uplus T}_{B} s \in P^{\mathsf{t}}_{AB}$ and $\Sigma_0 = \mathsf{Dft}(\Sigma) \upharpoonright \nu(m_A)$, $m^{\Sigma}_{A}\, m^{\Sigma \uplus T}_{B} s \in \phi$ iff $\gamma(m^{\Sigma_0}_{A} m^{\Sigma_0 \uplus T}_{B} s) \in \phi$;
- if $m^{\Sigma}_{A} m^{\Sigma \uplus T}_{B} s\, \mathsf{call}\, a.\mathsf{m}(\vec{v})^{\Sigma'} s' \in \phi$ and $a \in \nu(T)$ then $s = \epsilon$.

The first two conditions above are strengthening totality by imposing that the initial move be always replied to (i.e. there is no initial divergence) and that changing the initial store to a default one leave the strategy behaviour unaffected (i.e. the initial store is not read). 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 $m^{\Sigma}_{A} m^{\Sigma \uplus T}_{B}$, we can extract its threads by tracing back for each move in s the method call of the object $a \in \nu(T)$ it is related to. Formally, for each total play $s = m^{\Sigma}_{A} m^{\Sigma \uplus T}_{B} s'$ with $|s'| > 0$, the *threader* move of s, written $\mathsf{thrr}(s)$, is given by induction:

- $\mathsf{thrr}(s' m^{\Sigma'}) = \mathsf{thrr}(s')$, if $p(m) = P$;
- $\mathsf{thrr}(s' \mathsf{call}\, a.\mathsf{m}(\vec{v})^{\Sigma'}) = \mathsf{call}\, a.\mathsf{m}(\vec{v})^{\Sigma'}$, if $a \in \nu(T)$;
- $\mathsf{thrr}(s' n^{T'} s'' \mathsf{call}\, a.\mathsf{m}(\vec{v})^{\Sigma'}) = \mathsf{thrr}(s' n^{T'})$, if $a \in P(s) \setminus \nu(T)$ and n introduces a;
- $\mathsf{thrr}(s' n^{T'} s'' m^{\Sigma'}) = \mathsf{thrr}(s' n^{T'})$, if $p(m) = O$ and n justifies m.

If $s = s' n^{T'} s''$ with $|s'| \geq 2$, we set $\mathsf{thrr}(n^{T'}) = \mathsf{thrr}(s' n^{T'})$. Then, the ***current thread*** of s is the subsequence of s containing only moves with the same threader move as s, that is, if $\mathsf{thrr}(s) = m^{\Sigma'}$ and $s = m^{\Sigma}_{A} m^{\Sigma \uplus T}_{B} s'$ then

$$\lceil s \rceil = m^{\Sigma}_{A} m^{\Sigma \uplus T}_{B}\, (s' \upharpoonright m^{\Sigma'})$$

where the restriction retains only those moves $n^{T'}$ of s' such that $\mathsf{thrr}(n^{T'}) = m^{\Sigma'}$. We extend this to the case of $|s| \leq 2$ by setting $\lceil s \rceil = s$. Finally, we call a total play $s \in P_{AB}$ ***thread-independent*** if for all $s' m^{\Sigma'} \sqsubseteq^{even} s$ with $|s'| > 2$:

- if $\gamma(\lceil s' m^{\Sigma'} \rceil) = s'' m^{\Sigma''}$ then $\nu(\Sigma'') \cap \nu(s') \subseteq \nu(s'')$;
- if s' ends in some $n^{T'}$ and $a \in \mathsf{dom}(\Sigma') \setminus \nu(\gamma(\lceil s' m^{\Sigma'} \rceil))$ then $\Sigma'(a) = T'(a)$.

We write P^{ti}_{AB} for the set of thread-independent plays in AB.

We can now define strategies which occur as interleavings of single-threaded ones.

Definition 4.6. Given single-threaded $\phi : A \to B$, we define: $\phi^{\dagger} = \{s \in P^{\mathsf{ti}}_{AB} \mid \forall s' \sqsubseteq^{even} s.\ \gamma(\lceil s' \rceil) \in \phi\}$. We call a strategy σ ***thread-independent*** if $\sigma = \tau^{\dagger}$ for some single-threaded strategy τ.

[5] Note that the use of the term "thread" here is internal to game semantics parlance and in particular should not be confused with Java threads.

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 4.7. *$\phi^\dagger$ is a strategy, for each single-threaded ϕ.*

Proof. Equivariance, Even-prefix closure and O-extension follow from the corresponding conditions on ϕ. For determinacy, if $sm^\Sigma, sn^T \in \phi^\dagger$ with $|s| > 0$ then, using determinacy of ϕ 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 $sm^\Sigma \sim sn^T$. □

Thread-independent strategies have good naturality properties with respect to the product construction on arenas and pairing/projection on strategies. This is shown in the next lemma. Note though that this 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:

$$\sigma : 1 \to \mathcal{I} \times \mathcal{I} = \{*\,(a,a)^\Sigma \in P_{1(\mathcal{I}\times\mathcal{I})} \mid \Sigma \in Sto_0\}$$

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

Lemma 4.8. *Let $\sigma : A \to B$ and $\tau : A \to C$ be strategies with τ thread-independent. Then, $\langle \sigma, \tau\rangle; \pi_1 = \sigma$ and:*

$$\langle \sigma, \tau\rangle \;=\; A \xrightarrow{\langle \tau, \sigma\rangle} C \times B \xrightarrow{\cong} B \times C\,.$$

Proof. The former claim is straightforward. For the latter, we observe that the initial effects of σ and τ commute: on initial move m_A^Σ, τ does not read the store updates that σ includes in its response $m_B^{\Sigma'}$, while σ cannot access the names created by τ in its second move $m_C^{\Sigma' \uplus T}$. □

We can now define an appropriate notion of exponential for our games.Let us assume a translation assigning an arena $[\![\vec{\theta}]\!]$ to each type sequence $\vec{\theta}$. Moreover, let $\mathcal{I}$ be an interface such that

$$\Delta(\mathcal{I}) \upharpoonright Meths = \{\mathsf{m}_1 : \vec{\theta}_1 \to \theta_1, \cdots, \mathsf{m}_n : \vec{\theta}_n \to \theta_n\}$$

where $\vec{\theta}_i = \theta_{i1}, \cdots, \theta_{im_i}$, for each i. For any arena A, given single-threaded strategies $\phi_1, \cdots, \phi_n : A \to \mathcal{I}$ such that, for each i, if $m_A^\Sigma a^{\Sigma \uplus T} s \in \phi_i$ then

$$a \notin \nu(\Sigma) \wedge T(a) : \mathcal{I} \wedge (\mathsf{call}\, a.\mathsf{m}(\vec{v}) \in s \implies \mathsf{m} = \mathsf{m}_i),$$

we can group them into one single-threaded strategy:

$$\langle\langle \phi_1, \ldots, \phi_n\rangle\rangle : A \to \mathcal{I} = \bigcup\nolimits_{i=1}^n \phi_i\,.$$

Let now $\sigma_1, \cdots, \sigma_n$ be strategies with $\sigma_i : A \times [\![\vec{\theta}_i]\!] \to [\![\theta_i]\!]$. For each i, we define the strategy $\Lambda^{\mathcal{I}}(\sigma_i) : A \to \mathcal{I}$:

$$\begin{aligned}\Lambda^{\mathcal{I}}(\sigma_i) = {} & \{m_A^\Sigma (a^\Sigma \mathsf{call}\, a.m_i(\vec{v})^{\Sigma'} s)^{\uplus T} \in P^{\mathsf{t}}_{A\mathcal{I}} \mid \gamma((m_A, \vec{v})^{\Sigma'} s) \in \sigma_i\} \\ & \cup \{m_A^\Sigma (a^\Sigma \mathsf{call}\, a.m_i(\vec{v})^{\Sigma'} s\ \mathsf{ret}\, a.\mathsf{m}_i(v)^{T'} s')^{\uplus T} \in P^{\mathsf{t}}_{A\mathcal{I}} \mid \gamma((m_A, \vec{v})^{\Sigma'} s\, v^{T'} s') \in \sigma_i\} \\ & \cup \{m_A^\Sigma a^{\Sigma \uplus T} \in P^{\mathsf{t}}_{A\mathcal{I}}\}\end{aligned}$$

where $a \notin \nu(\Sigma, \vec{v}, v, s, s', \Sigma', T')$, $\mathrm{dom}(T) = \{a\}$ and we write $s^{\uplus T}$ for the sequence obtained by replacing each move m^Σ in s with $m^{\Sigma \uplus T}$. By definition, $\Lambda^{\mathcal{I}}(\sigma_i)$ is single-threaded.

Definition 4.9 (Exponentiation). Let $\sigma_1, \cdots, \sigma_n, A, \mathcal{I}$ be as above. We define a thread-independent strategy implementing the *simultaneous currying* thereof by:

$$\Lambda^{\mathcal{I}}(\sigma_1, \ldots, \sigma_n) = \langle\langle \Lambda^{\mathcal{I}}(\sigma_1), \ldots, \Lambda^{\mathcal{I}}(\sigma_n)\rangle\rangle^{\dagger} : A \to \mathcal{I}.$$

Moreover, we define *evaluation strategies* $\mathsf{ev}^{\mathcal{I}}_{\mathsf{m}_i} : \mathcal{I} \times [\![\vec{\theta}_i]\!] \to [\![\theta_i]\!]$ by taking:

$$\mathsf{ev}^{\mathcal{I}}_{\mathsf{m}_i} = \{\, s \mid s \sqsubseteq_{\text{even}} (a, \vec{v})^{\Sigma} \mathsf{call}\, a.\mathsf{m}_i(\vec{v})^{\Sigma} \mathsf{ret}\, a.\mathsf{m}_i(v)^{T} v^{T} \in P_{(\mathcal{I} \times [\![\vec{\theta}_i]\!])[\![\theta_i]\!]} \,\}.$$

In the sequel, we will be frequently dropping the superscript $\mathcal{I}$ from $\Lambda^{\mathcal{I}}$ and $\mathsf{ev}^{\mathcal{I}}_{\mathsf{m}_i}$ for brevity. In some cases, we might drop the subscript m_i as well.

Remark 4.10. In defining $\Lambda^{\mathcal{I}}(\sigma_i)$ above, we essentially simulated the higher-order semantic value (the arrow) needed for exponentiation by means of a fresh name a. This design has no deeper semantic meaning. It is worth noting that the name a is not allowed to participate in the interactions of $\Lambda^{\mathcal{I}}(\sigma_i)$ in any other way than providing the arrow. In Section 4.4, when defining the semantics of the new-object constructor, we will use a construction that hides away this name a from the environment. In the meantime we can see that, given translations $[\![M_i]\!]$ for each method in a method-set implementation $\mathcal{M}$, we can construct: $[\![\mathcal{M}]\!] : [\![\Gamma; u]\!] \to \mathcal{I} = \Lambda^{\mathcal{I}}([\![M_1]\!], \cdots, [\![M_n]\!])$.

We can show the following properties for Λ and ev, which will suffice to prove our game model sound. In terms of Freyd-closedness [41], these properties are too weak for obtaining an adjunction. The main obstacle is that in our setting Λ leads to creating a fresh name, which is incompatible with the required universality (e.g. $\Lambda(\mathsf{ev}_{\mathsf{m}_1}, \cdots, \mathsf{ev}_{\mathsf{m}_n}) \neq \mathsf{id}$).

Lemma 4.11. *Let $\sigma_1, \cdots, \sigma_n$ be as above, and let $\tau : A' \to A$ be evaluated. Then:*

- $(\Lambda(\sigma_1, \ldots, \sigma_n) \times \mathsf{id}); \mathsf{ev}_{\mathsf{m}_i} = \sigma_i$,
- $\tau; \Lambda(\sigma_1, \ldots, \sigma_n) = \Lambda((\tau \times \mathsf{id}); \sigma_1, \ldots, (\tau \times \mathsf{id}); \sigma_n)$.

4.3 Modelling store

Store modelling is embedded in our games by means of the stores carried along with each move. What remains is to define strategies for assignment and reading (dereferencing) from the store. These are given as follows. Assuming an interface table Δ such that $\Delta(\mathcal{I}).\mathsf{f} = \theta$, we define

$$\mathsf{asn}_{\mathsf{f}} : \mathcal{I} \times [\![\theta]\!] \to 1 = \{(a, v)^{\Sigma} *^{\Sigma[a.\mathsf{f} \mapsto v]} \in P_{(\mathcal{I} \times [\![\theta]\!]) 1}\}$$
$$\mathsf{drf}_{\mathsf{f}} : \mathcal{I} \to [\![\theta]\!] = \{a^{\Sigma} v^{\Sigma} \in P_{\mathcal{I}[\![\theta]\!]} \mid \Sigma(a).\mathsf{f} = v\}$$

for respectively assigning to and reading from field f in $\mathcal{I}$-objects.

We can check that the strategies satisfy the intended read/write discipline: assigning a value to a field and accessing it results in the same value; moreover, two assignments in a row have the same effect as just the last one.

Lemma 4.12. *The following equations hold.*

$$\langle \mathsf{asn}_{\mathsf{f}}, \pi_1 \rangle; \pi_2; \mathsf{drf}_{\mathsf{f}} = \langle \mathsf{asn}_{\mathsf{f}}, \pi_2 \rangle; \pi_2 : \mathcal{I} \times [\![\theta]\!] \to [\![\theta]\!]$$
$$(\langle \mathsf{asn}_{\mathsf{f}}, \pi_1 \rangle \times \mathsf{id}); \pi_2; \mathsf{asn}_{\mathsf{f}} = (\mathsf{id} \times \pi_2); \mathsf{asn}_{\mathsf{f}} : \mathcal{I} \times [\![\theta]\!] \times [\![\theta]\!] \to 1$$

4.4 Modelling self-reference

Apart from dealing with exponentials, in order to complete our translation we need also to address the appearance of $x : \mathcal{I}$ in the rule

$$\frac{\Delta|\Gamma, x : \mathcal{I}; u \vdash \mathcal{M} : \Theta}{\Delta|\Gamma; u \vdash \mathsf{new}(x : \mathcal{I}; \mathcal{M}) : \mathcal{I}}\ \Delta(\mathcal{I}) \restriction \mathit{Meths} = \Theta$$

on left-hand side ($x : \mathcal{I}$) and on the right-hand side (inside $\mathcal{M}$) – this is similar to the use of the keyword this in Java.

Recall that

$$[\![\mathcal{M}]\!] : [\![\Gamma; u]\!] \times \mathcal{I} \to \mathcal{I} \tag{2}$$

is obtained using exponentiation. Thus, the second move of $[\![\mathcal{M}]\!]$ will appear in the right-hand occurrence of $\mathcal{I}$ above and will be a fresh name b, which will serve as a handle to the methods of $\mathcal{M}$: in order to invoke $\mathsf{m} : \lambda\vec{x}.M$ on input $\vec{v}$, the Opponent would have to call $b.\mathsf{m}(\vec{v})$. The remaining challenge is to merge the two occurrences of $\mathcal{I}$ in (2). We achieve this as follows. Let us assume a well-formed extension Δ' of Δ:

$$\Delta' = (\mathcal{I}' : (\mathsf{f}' : \mathcal{I})), \Delta$$

i.e. $\mathcal{I}'$ contains a single field f' of type $\mathcal{I}$. We next define the strategy $\kappa_{\mathcal{I}} : 1 \to \mathcal{I}' \times \mathcal{I}$ of $\mathcal{G}_{\Delta'}$ that simply *copycats* between the calls and returns of methods from $\mathcal{I}$ and those from the stored field f' of $\mathcal{I}'$:

$$\kappa_{\mathcal{I}} = \mathsf{epref}(\{*\,(a', a)^{\Sigma_0}\mathsf{call}\,a.\mathsf{m}(\vec{v})^{\Sigma}\mathsf{call}\,b.\mathsf{m}(\vec{v})^{\Sigma}\mathsf{ret}\,b.\mathsf{m}(v)^{T}\mathsf{ret}\,a.\mathsf{m}(v)^{T} \in P_{1(\mathcal{I}'\times\mathcal{I})} \mid \Sigma_0 \in \mathit{Sto}_0 \wedge b = \Sigma(a').\mathsf{f}'\})^{\dagger} \tag{3}$$

where $\mathsf{epref}(\phi) = \{s' \mid \exists s \in \phi.\, s' \sqsubseteq_{\mathsf{even}} s\}$. Thus, upon receiving a request $\mathsf{call}\,a.\mathsf{m}(\vec{v})^{\Sigma}$, $\kappa_{\mathcal{I}}$ forwards it to the respective method of $a'.\mathsf{f}'$ and, once it receives a return value, copies it back as the return value of the original call.

Given the $\kappa_{\mathcal{I}}$ strategy, we shall let $[\![\mathsf{new}(x : \mathcal{I}; \mathcal{M})]\!] : [\![\Gamma; u]\!] \to \mathcal{I}$ be the strategy:

$$\Delta'/\Delta\left([\![\Gamma; u]\!] \xrightarrow{\langle \mathsf{id},!;\kappa_{\mathcal{I}}\rangle;\cong} \mathcal{I}'\times[\![\Gamma; u]\!]\times\mathcal{I} \xrightarrow{\mathsf{id}\times\langle[\![\Delta/\Delta'(\mathcal{M})]\!],\pi_2\rangle} \mathcal{I}'\times\mathcal{I}\times\mathcal{I} \xrightarrow{(\mathsf{asn}_{\mathsf{f}'}\times\mathsf{id});\pi_2} \mathcal{I}\right).$$

As the application of the functors Δ/Δ' and Δ'/Δ above acts as identity on the respective strategies, we write the above simply as:

$$[\![\Gamma; u]\!] \xrightarrow{\langle \mathsf{id},!;\kappa_{\mathcal{I}}\rangle;\cong} \mathcal{I}'\times[\![\Gamma; u]\!]\times\mathcal{I} \xrightarrow{\mathsf{id}\times\langle[\![\mathcal{M}]\!],\pi_2\rangle} \mathcal{I}'\times\mathcal{I}\times\mathcal{I} \xrightarrow{(\mathsf{asn}_{\mathsf{f}'}\times\mathsf{id});\pi_2} \mathcal{I}.$$

Thus, object creation involves creating a pair of names (a', a) with $a : \mathcal{I}$ and $a' : \mathcal{I}'$, 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, i.e. the name created by the second move of $[\![\mathcal{M}]\!]$, will be passed.

5 SOUND MODEL FOR IMJ

Here we take stock of the structure defined in the previous section and show how to translate IMJ terms into strategies. Then we prove that the model is sound.

5.1 Interpretation of IMJ

For each sequence of interfaces $\overrightarrow{\mathcal{I}}$, let $\#(\vec{\mathcal{I}}) : \overrightarrow{\mathcal{I}} \to \#(\overrightarrow{\mathcal{I}}) = \{\vec{a}^{\Sigma}\vec{a}^{\Sigma} \mid a_i\text{s distinct}\}$. The strategies have right inverses $\#(\vec{\mathcal{I}})^{-r} : \#(\overrightarrow{\mathcal{I}}) \to \overrightarrow{\mathcal{I}}$, containing the same plays. We can now define the semantic translation of terms.

Definition 5.1. The semantic translation is given as follows.

$[\![\Gamma; u \vdash x_i : \theta_i]\!] = [\![\Gamma; u]\!] \xrightarrow{\pi_i} [\![\theta_i]\!]$
$[\![\Gamma; u \vdash a_i : \mathcal{I}_i]\!] = [\![\Gamma; u]\!] \xrightarrow{\pi_{n+1}} \#(\overrightarrow{\mathcal{I}}) \xrightarrow{\#(\vec{\mathcal{I}})^{-r}} \overrightarrow{\mathcal{I}} \xrightarrow{\pi_i} \mathcal{I}_i$
$[\![\Gamma; u \vdash \mathsf{skip} : \mathsf{void}]\!] = [\![\Gamma; u]\!] \xrightarrow{!} 1$
$[\![\Gamma; u \vdash \mathsf{null} : \mathcal{I}]\!] = [\![\Gamma; u]\!] \xrightarrow{!;\hat{\mathsf{nul}}} \mathcal{I}$, where $\hat{\mathsf{nul}} : 1 \to \mathcal{I} = \{* \, \mathsf{nul}\}$
$[\![\Gamma; u \vdash i : \mathsf{int}]\!] = [\![\Gamma; u]\!] \xrightarrow{!;\hat{i}} \mathbb{Z}$
$[\![\Gamma; u \vdash \mathsf{let}\, x = M' \,\mathsf{in}\, M : \theta]\!] = [\![\Gamma; u]\!] \xrightarrow{\langle \mathsf{id}, [\![M']\!] \rangle} [\![\Gamma; u]\!] \times [\![\theta']\!] \xrightarrow{[\![M]\!]} [\![\theta]\!]$
$[\![\Gamma; u \vdash (\mathcal{I})M : \mathcal{I}]\!] = [\![\Gamma; u]\!] \xrightarrow{[\![M]\!]} \mathcal{I}' \xrightarrow{\mathsf{stp}_{\mathcal{I}'\mathcal{I}}} \mathcal{I}$, where $\mathsf{stp}_{\mathcal{I}'\mathcal{I}} : \mathcal{I}' \to \mathcal{I} = \{\mathsf{nul}\,\mathsf{nul}\} \cup \{a^{\Sigma} a^{\Sigma} \in P_{\mathcal{I}'\mathcal{I}} \mid \Sigma(a) \leq \mathcal{I}\}$
$[\![\Gamma; u \vdash M \oplus M' : \mathsf{int}]\!] = [\![\Gamma; u]\!] \xrightarrow{\langle [\![M]\!], [\![M']\!] \rangle} \mathbb{Z} \times \mathbb{Z} \xrightarrow{\oplus} \mathbb{Z}$, where $\oplus : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} = \{(i, j)\ (i \oplus j)\}$
$[\![\Gamma; u \vdash M = M' : \mathsf{int}]\!] = [\![\Gamma; u]\!] \xrightarrow{\langle [\![M]\!], [\![M']\!] \rangle} \mathcal{I} \times \mathcal{I} \xrightarrow{\mathsf{eq}} \mathbb{Z}$, where $\mathsf{eq} = \{(a, a)^{\Sigma}\, 1^{\Sigma} \in P_{(\mathcal{I} \times \mathcal{I})\mathbb{Z}}\} \cup \{(a, b)^{\Sigma}\, 0^{\Sigma} \in P_{(\mathcal{I} \times \mathcal{I})\mathbb{Z}} \mid a \neq b\}$
$[\![\Gamma; u \vdash \mathsf{if}\ M\ \mathsf{then}\ M'\ \mathsf{else}\ M'' : \theta]\!] = [\![\Gamma; u]\!] \xrightarrow{\langle [\![M]\!], \mathsf{id} \rangle} \mathbb{Z} \times [\![\Gamma; u]\!] \xrightarrow{[[\![M']\!], [\![M'']\!]]} [\![\theta]\!]$
$[\![\Gamma; u \vdash \mathsf{new}(x : \mathcal{I}; \mathcal{M}) : \mathcal{I}]\!] = [\![\Gamma; u]\!] \xrightarrow{\langle \mathsf{id}, !; \kappa_{\mathcal{I}} \rangle; \cong} \mathcal{I}' \times [\![\Gamma; u]\!] \times \mathcal{I} \xrightarrow{\mathsf{id} \times \langle [\![\mathcal{M}]\!], \pi_2 \rangle} \mathcal{I}' \times \mathcal{I} \times \mathcal{I} \xrightarrow{\mathsf{asn}_{\mathsf{f}'} \times \mathsf{id}} 1 \times \mathcal{I} \xrightarrow{\pi_2} \mathcal{I}$, where $\mathcal{M} = \{\mathsf{m}_1 : \lambda \vec{x}_1.M_1, \cdots, \mathsf{m}_n : \lambda \vec{x}_n.M_n\}$ and $[\![\mathcal{M}]\!] = [\![\Gamma; u]\!] \times \mathcal{I} \xrightarrow{\Lambda([\![M_1]\!], \ldots, [\![M_n]\!])} \mathcal{I}$
$[\![\Gamma; u \vdash M.\mathsf{f} := M' : \mathsf{void}]\!] = [\![\Gamma; u]\!] \xrightarrow{\langle [\![M]\!], [\![M']\!] \rangle} \mathcal{I} \times [\![\theta]\!] \xrightarrow{\mathsf{asn}_{\mathsf{f}}} 1$
$[\![\Gamma; u \vdash M.\mathsf{f} : \theta]\!] = [\![\Gamma; u]\!] \xrightarrow{[\![M]\!]} \mathcal{I} \xrightarrow{\mathsf{drf}_{\mathsf{f}}} [\![\theta]\!]$
$[\![\Gamma; u \vdash M.\mathsf{m}(\overrightarrow{M}) : \theta]\!] = [\![\Gamma; u]\!] \xrightarrow{\langle [\![M]\!], [\![\overrightarrow{M}]\!] \rangle} \mathcal{I} \times [\![\vec{\theta}]\!] \xrightarrow{\mathsf{ev}_{\mathsf{m}}} [\![\theta]\!]$, where $[\![\overrightarrow{M}]\!] = \langle\langle\langle [\![M_1]\!], [\![M_2]\!] \rangle, \cdots \rangle, [\![M_n]\!] \rangle$

Fig. 6. The semantic translation of IMJ.

- Contexts $\Gamma = \{x_1 : \theta_1, \cdots, x_n : \theta_n\}$, $u = \{a_1 : \mathcal{I}_1, \cdots, a_m : \mathcal{I}_m\}$ are translated into arenas by
$$[\![\Gamma; u]\!] = [\![\theta_1]\!] \times \cdots \times [\![\theta_n]\!] \times \#(\mathcal{I}_1, \cdots, \mathcal{I}_m)$$
where $[\![\mathsf{void}]\!] = 1$, $[\![\mathsf{int}]\!] = \mathbb{Z}$ and $[\![\mathcal{I}]\!] = \mathcal{I}$.
- Terms are translated as in Figure 6.

Example 5.2. We discuss the shape of strategies arising from terms discussed in Example 2.8, assuming $\mathcal{I} = \mathsf{Empty}$. For brevity, let us set $\mathcal{I}' = \mathsf{HashFun}_{\mathcal{I}}$ and $\mathcal{I}'' = \mathsf{Var}_{\mathsf{HashFun}_{\mathcal{I}}}$. In what follows, we use a (and variants) to range over $\mathcal{I}$ objects, o for $\mathcal{I}'$ objects, and r for $\mathcal{I}''$ objects. We write $\Sigma \cup \{o\}$ for $\Sigma[o \mapsto (\mathcal{I}', \emptyset)]$, where $\Sigma(o)$ may or may not be defined, and similarly $\Sigma \cup \{a\}$ for $\Sigma[a \mapsto (\mathcal{I}, \emptyset)]$. We use the variants $\Sigma \uplus \{o\}$ and $\Sigma \uplus \{a\}$ when o and a respectively are not in the domain of Σ. We extend this notation to sequences of moves, by applying it move-by-move.

(1) We first look at translating $\mathcal{M}_0$ and $\mathcal{M}_1$, following Section 4.2.

$$\mathcal{M}_0 = (\mathsf{hash} : \ \lambda\_.\, 0,\ \ \mathsf{reset} : \ \lambda\_.\, \mathsf{skip})$$
$$\mathcal{M}_1 = (\mathsf{hash} : \ \lambda z.\ \mathit{priv}.\mathsf{val}.\mathsf{hash}(z),\ \ \mathsf{reset} : \ \lambda h.\ \mathit{priv}.\mathsf{val} := h)$$

The translation recipe requires that we first model the bodies of these methods. For $\mathcal{M}_0$, this is trivial:

$$[\![\, \_ : \mathcal{I} \vdash 0]\!] = \{a^{\Sigma}\ 0^{\Sigma}\} \quad \text{and} \quad [\![\, \_ : \mathcal{I} \vdash \mathsf{skip}]\!] = \{a^{\Sigma}\ *^{\Sigma}\},$$

for any appropriate a^{Σ}. By exponentiation we get:

$$[\![\mathcal{M}_0]\!] : 1 \to \mathcal{I}' = (\{*\, o^{\Sigma_0}\ \mathsf{call}\, o.\mathsf{hash}(a)^{\Sigma_0 \uplus \{a\}}\ \mathsf{ret}\, o.\mathsf{hash}(0)^{\Sigma_0 \uplus \{a\}}\} \cup \{*\, o^{\Sigma_0}\ \mathsf{call}\, o.\mathsf{reset}(o')^{\Sigma_0 \uplus \{o'\}}\ \mathsf{ret}\, o.\mathsf{reset}(*)^{\Sigma_0 \uplus \{o'\}}\})^{\dagger}$$

where $\Sigma_0 = \{o \mapsto (\mathcal{I}', \emptyset)\}$. For $\mathcal{M}_1$, the method bodies are translated as follows:

$$[\![\mathit{priv} : \mathcal{I}'', z : \mathcal{I} \vdash \mathit{priv}.\mathsf{val}.\mathsf{hash}(z)\ ]\!] = \{(r, a)^{\Sigma}\ \mathsf{call}\, o'.\mathsf{hash}(a)^{\Sigma}\ \mathsf{ret}\, o'.\mathsf{hash}(i)^{\Sigma'}\ i^{\Sigma'} \mid i \in \mathbb{Z} \wedge o' = \Sigma(p).\mathsf{val}\}$$
$$[\![\mathit{priv} : \mathcal{I}'', h : \mathcal{I}' \vdash \mathit{priv}.\mathsf{val} := h\ ]\!] = \{(r, o')^{\Sigma}\ *^{\Sigma[r.\mathsf{val} \mapsto o']}\}$$

and by exponentiation we get (noting the typing $\mathit{priv} : \mathcal{I}'' \vdash \mathcal{M}_1$):

$$[\![\mathcal{M}_1]\!] : \mathcal{I}'' \to \mathcal{I}' = \Big(\{r^{\Sigma}\ (o^{\Sigma}\ \mathsf{call}\, o.\mathsf{hash}(a)^{\Sigma'}\ \mathsf{call}\, o'.\mathsf{hash}(a)^{\Sigma'}\ \mathsf{ret}\, o'.\mathsf{hash}(i)^{\Sigma''}\ \mathsf{ret}\, o.\mathsf{hash}(i)^{\Sigma''})^{\uplus\{o\}} \mid o' = \Sigma'(r).\mathsf{val}\}$$
$$\cup\ \{r^{\Sigma}\ (o^{\Sigma}\ \mathsf{call}\, o.\mathsf{reset}(o')^{\Sigma'}\ \mathsf{ret}\, o.\mathsf{reset}(*)^{\Sigma'[r.\mathsf{val} \mapsto o']})^{\uplus\{o\}}\ \}\Big)^{\dagger}.$$

(2) We next look at $\sigma_0 = [\![\vdash \mathsf{new}(\_ : \mathcal{I}'; \mathcal{M}_0) : \mathcal{I}']\!]$ and $\sigma_1 = [\![\mathit{priv} : \mathcal{I}'' \vdash \mathsf{new}(\_ : \mathcal{I}'; \mathcal{M}_1) : \mathcal{I}']\!]$. Since no self-reference is used in the term corresponding to σ_0, we have $\sigma_0 = [\![\mathcal{M}_0]\!]$. One is also tempted to say $\sigma_1 = [\![\mathcal{M}_1]\!]$, but there is a slight difference: the name o that we used for exponentiation is the actual returned object, and Opponent can use this object, e.g. by storing it in *priv*, something that is not allowed in $[\![\mathcal{M}_0]\!]$. Instead, we shall have:

$$\sigma_1 : \mathcal{I}'' \to \mathcal{I}' = \Big(\{r^{\Sigma}\ o^{\Sigma \uplus \{o\}}\ \mathsf{call}\, o.\mathsf{hash}(a)^{\Sigma'}\ \mathsf{call}\, o'.\mathsf{hash}(a)^{\Sigma'}\ \mathsf{ret}\, o'.\mathsf{hash}(i)^{\Sigma''}\ \mathsf{ret}\, o.\mathsf{hash}(i)^{\Sigma''} \mid o' = \Sigma'(r).\mathsf{val} \neq o\}$$
$$\cup\ \{r^{\Sigma}\ o^{\Sigma \uplus \{o\}}\ \mathsf{call}\, o.\mathsf{reset}(o')^{\Sigma'}\ \mathsf{ret}\, o.\mathsf{reset}(*)^{\Sigma'[r.\mathsf{val} \mapsto o']}\}\Big)^{\dagger}.$$

Let us spell out σ_1 in more detail.

- For a start, we have $r^{\Sigma} o^{\Sigma \uplus \{o\}} \in \sigma_1$, where Σ maps r to a pair $(\mathcal{I}'', \{\mathsf{val} \mapsto v\})$ and v can be a name or nul.
- Next we describe the remainder of the strategy inductively by examining all possible future O-moves and giving responses due to σ_1. Suppose $\epsilon \neq s \in \sigma_1$ and let Σ be the store of the last move from s. Observe that O can change $\Sigma(r).\mathsf{val}$ at every step, to a fresh value or a value that has already been seen in play. We shall write $\Sigma[o']$ to stand for $\Sigma[r.\mathsf{val} \mapsto o']$. Then, as long as the O move is valid, we have:

$$s\ \mathsf{call}\, o.\mathsf{reset}(o'')^{\Sigma[o'] \cup \{o', o''\}}\ \mathsf{ret}\, o.\mathsf{reset}(*)^{\Sigma[o''] \cup \{o', o''\}} \in \sigma_1,$$
$$s\ \mathsf{call}\, o.\mathsf{hash}(a)^{\Sigma[o'] \cup \{o', a\}}\ \mathsf{call}\, o'.\mathsf{hash}(a)^{\Sigma[o'] \cup \{o', a\}} \in \sigma_1 \ \ (\text{if } o' \neq o),$$
$$s\ \mathsf{ret}\, o'.\mathsf{hash}(i)^{\Sigma[o''] \cup \{o''\}}\ \mathsf{ret}\, o.\mathsf{hash}(i)^{\Sigma[o''] \cup \{o''\}} \in \sigma_1.$$

In the second case above, note that the strategy has no response to $\mathsf{call}\, o.\mathsf{hash}(a)^{\Sigma[o] \cup \{a\}}$.

(3) The strategy $\sigma' = [\![priv : \mathcal{I}'' \vdash priv.\mathsf{val} := \mathsf{new}(\_ : \mathcal{I}'; \mathcal{M}_0);\ \mathsf{new}(\_ : \mathcal{I}'; \mathcal{M}_1) : \mathcal{I}']\!]$ is given by:

$$\sigma' = \mathcal{I}'' \xrightarrow{\langle \mathsf{id}, !; \sigma_0 \rangle} \mathcal{I}'' \times \mathcal{I}' \xrightarrow{\langle \pi_1, \mathsf{asn} \rangle; \cong} \mathcal{I}'' \xrightarrow{\sigma_1} \mathcal{I}$$

Concretely, σ' is defined by the same clauses as σ_1 except that:

- for a start, we have $r^{\Sigma} o^{\Sigma[r.\mathsf{val} \mapsto o_0] \uplus \{o_0, o\}} \in \sigma'$;
- in the second clause above, we need to strengthen $o' \neq o$ to $o' \neq o, o_0$ and add

$$s\ \mathsf{call}\, o.\mathsf{hash}(a)^{\Sigma[o_0] \cup \{a\}}\ \mathsf{ret}\, o.\mathsf{hash}(0)^{\Sigma[o_0] \cup \{a\}} \in \sigma'.$$

In other words, we initialise $r.\mathsf{val}$ to an object o_0 that is defined by $[\![\mathcal{M}_0]\!]$.

(4) Finally, to interpret the whole term and compute

$$\sigma = [\![\mathsf{let}\ priv = \mathsf{new}(\_ : \mathcal{I}'';)\ \mathsf{in}\ (priv.\mathsf{val} := \mathsf{new}(\_ : \mathcal{I}'; \mathcal{M}_0));\ \mathsf{new}(\_ : \mathcal{I}; \mathcal{M}_1)]\!]$$

we need to pre-compose σ' with the strategy $\sigma'' = [\![\mathsf{new}(\_ : \mathcal{I}'';)]\!] = \{ * \, r^{\{r \mapsto (\mathcal{I}'', \{\mathsf{val} \mapsto \mathsf{nul}\})\}} \}$. The interactions hide the object r that stands for $priv$ and have the effect of preventing O from changing $r.\mathsf{val}$ in every move, though O can still do this indirectly by calling reset (though note that it is P who makes the change). Consequently, $\mathsf{call}\, o.\mathsf{hash}(a)$ will now trigger $\mathsf{call}\, o'.\mathsf{hash}(a)$, where o' originates from the most recent call-move (by O) to reset (i.e. $\mathsf{call}\, o.\mathsf{reset}(o')$). If no such move has been played yet, $\mathsf{ret}\, o.\mathsf{hash}(0)$ is played in line with the case for o_0 for σ'.

5.2 Soundness

In order to prove that the semantics is sound, we will also need to interpret terms inside state contexts. Formally, let us assume Γ, u, M, θ, S be such that:

- $\Gamma; u \vdash M : \theta$ and $\mathsf{dom}(u) = \mathsf{dom}(S) = \{a_1, \cdots, a_n\}$,
- for each $a_i \in \mathsf{dom}(S)$, and setting $S(a_i) = (\mathcal{I}_i, (F_i, \mathcal{M}_i))$, we have $u(a_i) = \mathcal{I}_i$ and $\Gamma; u \vdash (F, \mathcal{M}) : \Delta(\mathcal{I}_i)$.

Then, the term-in-state (S, M) is translated into the strategy (recall $[\![\Gamma; u]\!]$ from (1)):

$$[\![\Gamma \vdash (S, M)]\!] = [\![\Gamma]\!] \xrightarrow{[\![\Gamma \vdash S]\!]} [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}} \xrightarrow{\mathsf{id} \times \#(\vec{\mathcal{I}})} [\![\Gamma; u]\!] \xrightarrow{[\![M]\!]} [\![\theta]\!]$$

where we write $\overrightarrow{\mathcal{I}} = \mathcal{I}_1 \times \cdots \times \mathcal{I}_n$. The semantic translation of state $[\![\Gamma \vdash S]\!]$ is given in two stages: the first stage, $[\![S]\!]_1$, creates the objects in $\mathsf{dom}(S)$ and implements their methods; the second stage of the translation, $[\![S]\!]_2$, initialises the fields of the newly created objects:

$$[\![\Gamma \vdash S]\!] = [\![\Gamma]\!] \xrightarrow{[\![S]\!]_1} [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}} \xrightarrow{[\![S]\!]_2} [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}}.$$

We look at $[\![S]\!]_1$ and $[\![S]\!]_2$ next.

To implement the methods in S we can start by setting $\overrightarrow{\mathcal{M}} = (\mathcal{M}_1, \cdots, \mathcal{M}_n)$ and

$$[\![\overrightarrow{\mathcal{M}}]\!] = [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}} \xrightarrow{\mathsf{id} \times \#(\vec{\mathcal{I}})} [\![\Gamma]\!] \times \#(\overrightarrow{\mathcal{I}}) \xrightarrow{\langle [\![\mathcal{M}_1]\!], \cdots, [\![\mathcal{M}_n]\!] \rangle} \overrightarrow{\mathcal{I}}.$$

We next need to merge the left- and right-hand-side occurrence of each $\mathcal{I}_i$ above. This corresponds to identifying each created object's self-reference with the object itself, and is accomplished by employing the strategies κ from (3). We

take, for each i, $\kappa_{\mathcal{I}_i} : 1 \to \mathcal{I}_i' \times \mathcal{I}_i$, and set:

$$\kappa_S = 1 \xrightarrow{\langle \kappa_{\mathcal{I}_1}, \cdots, \kappa_{\mathcal{I}_n} \rangle} \mathcal{I}_1' \times \mathcal{I}_1 \times \cdots \times \mathcal{I}_n' \times \mathcal{I}_n \xrightarrow{\cong} \overrightarrow{\mathcal{I}} \times \overrightarrow{\mathcal{I}'}.$$

Thus, κ_S creates pairs of names (a_i', a_i) and, for each such pair, copycats between the calls and returns to the methods of a_i and those of the unique field of a_i'. To obtain $[\![S]\!]_1$ we prepend $[\![\overrightarrow{\mathcal{M}}]\!]$ with κ_S, and append the assignment strategy $\overrightarrow{\mathsf{asn}_{\mathsf{f}'}} = \mathsf{asn}_{\mathsf{f}_1'} \times \cdots \times \mathsf{asn}_{\mathsf{f}_n'}$ (recall that each f_i' is the unique field in $\mathcal{I}_i'$, storing a value of type $\mathcal{I}_i$):

$$[\![S]\!]_1 = [\![\Gamma]\!] \xrightarrow{\langle \mathsf{id}, !; \kappa_S \rangle} ([\![\Gamma]\!] \times \overrightarrow{\mathcal{I}}) \times \overrightarrow{\mathcal{I}'} \xrightarrow{\langle \mathsf{id}, [\![\overrightarrow{\mathcal{M}}]\!] \rangle \times \mathsf{id}} ([\![\Gamma]\!] \times \overrightarrow{\mathcal{I}}) \times \overrightarrow{\mathcal{I}} \times \overrightarrow{\mathcal{I}'} \xrightarrow{\mathsf{id} \times \cong} ([\![\Gamma]\!] \times \overrightarrow{\mathcal{I}}) \times \overrightarrow{(\mathcal{I}' \times \mathcal{I})} \xrightarrow{(\mathsf{id} \times \overrightarrow{\mathsf{asn}_{\mathsf{f}'}}); \pi_1} [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}}.$$

We move on to $[\![S]\!]_2$. This should be a multiple assignment of values to all fields of all objects in S, i.e. those objects created by $[\![S]\!]_1$. Assuming that $F_i = (\mathsf{f}_i^1 : v_i^1, \cdots, \mathsf{f}_i^{k_i} : v_i^{k_i})$, with each $\Gamma; u \vdash v_i^j : \theta_i^j$ (and $[\![v_i^j]\!] : [\![\Gamma]\!] \times \#(\overrightarrow{\mathcal{I}}) \to [\![\theta_i^j]\!]$), and setting $\overrightarrow{F} = (F_1, \cdots, F_n)$, we first build:

$$[\![F_i]\!] = \mathsf{id} \times \#(\overrightarrow{\mathcal{I}}); \langle [\![v_i^1]\!], \cdots, [\![v_i^{k_i}]\!] \rangle : [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}} \to \overrightarrow{[\![\theta_i]\!]} \quad \text{and} \quad [\![\overrightarrow{F}]\!] = \langle [\![F_1]\!], \cdots, [\![F_n]\!] \rangle : [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}} \to \overrightarrow{\overrightarrow{[\![\theta]\!]}}$$

where $\overrightarrow{[\![\theta_i]\!]} = [\![\theta_i^1]\!] \times \cdots \times [\![\theta_i^{k_i}]\!]$ and $\overrightarrow{\overrightarrow{[\![\theta]\!]}} = \overrightarrow{[\![\theta_1]\!]} \times \cdots \times \overrightarrow{[\![\theta_n]\!]}$. We can now assign all fields by:

$$[\![S]\!]_2 = [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}} \xrightarrow{\langle \mathsf{id}, \langle \pi_2, [\![\overrightarrow{F}]\!] \rangle \rangle} ([\![\Gamma]\!] \times \overrightarrow{\mathcal{I}}) \times \overrightarrow{\mathcal{I}} \times \overrightarrow{\overrightarrow{[\![\theta]\!]}} \xrightarrow{\mathsf{id} \times \mathsf{copy}} ([\![\Gamma]\!] \times \overrightarrow{\mathcal{I}}) \times \overrightarrow{(\mathcal{I} \times [\![\theta]\!])} \xrightarrow{(\mathsf{id} \times \overrightarrow{\mathsf{asn}_{\mathsf{f}}}); \pi_1} [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}}$$

where we take $\overrightarrow{\mathsf{asn}_{\mathsf{f}}} = \mathsf{asn}_{\mathsf{f}_i} \times \cdots \times \mathsf{asn}_{\mathsf{f}_n}$ and $\mathsf{asn}_{\mathsf{f}_i} = \mathsf{asn}_{\mathsf{f}_i^1} \times \cdots \times \mathsf{asn}_{\mathsf{f}_i^{k_i}}$. The strategy copy above makes several copies of each $\mathcal{I}_i$, one for each field f_i^j of $\mathcal{I}_i$, and places each such copy near the corresponding $[\![\theta_i^j]\!]$:

$$\mathsf{copy} = \mathcal{I}_1 \times \cdots \times \mathcal{I}_n \times \overrightarrow{\overrightarrow{[\![\theta]\!]}} \xrightarrow{\delta^{k_1} \times \cdots \times \delta^{k_n} \times \mathsf{id}} \mathcal{I}_1^{k_1} \times \cdots \times \mathcal{I}_n^{k_n} \times \overrightarrow{\overrightarrow{[\![\theta]\!]}} \xrightarrow{\cong} (\mathcal{I}_1 \times [\![\theta_1^1]\!]) \times \cdots \times (\mathcal{I}_1 \times [\![\theta_1^{k_1}]\!]) \times \cdots \times (\mathcal{I}_n \times [\![\theta_n^1]\!]) \times \cdots \times (\mathcal{I}_n \times [\![\theta_n^{k_n}]\!])$$

and, in general, δ^j is the diagonal strategy $A \to A^j$.

Thus, $[\![S]\!]_1$ is charged with setting up the methods in S, whereas $[\![S]\!]_2$ sets its field values. Setting up the methods $\mathcal{M}$ involves three stages: name creation (via κ_S), the thread-independent strategy $[\![\mathcal{M}]\!]$, and the assignments $\overrightarrow{\mathsf{asn}_{\mathsf{f}'}}$. The two latter stages commute with setting the fields of S, because of Lemma 4.12 (assignments of different fields commute) and Lemma 4.8 (thread-independent strategies commute with any strategy), as the next lemma states.

Lemma 5.3. *For Γ, S as above, let us write $[\![S]\!]_1$ as $[\![S]\!]_1 = [\![\Gamma]\!] \xrightarrow{\langle \mathsf{id}, !; \kappa_S \rangle} [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}} \times \overrightarrow{\mathcal{I}'} \xrightarrow{[\![S]\!]_1'} [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}}$. Then:*

$$[\![\Gamma \vdash S]\!] = [\![\Gamma]\!] \xrightarrow{\langle \mathsf{id}, !; \kappa_S \rangle} [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}} \times \overrightarrow{\mathcal{I}'} \xrightarrow{[\![S]\!]_2 \times \mathsf{id}} [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}} \times \overrightarrow{\mathcal{I}'} \xrightarrow{[\![S]\!]_1'} [\![\Gamma]\!] \times \overrightarrow{\mathcal{I}}.$$

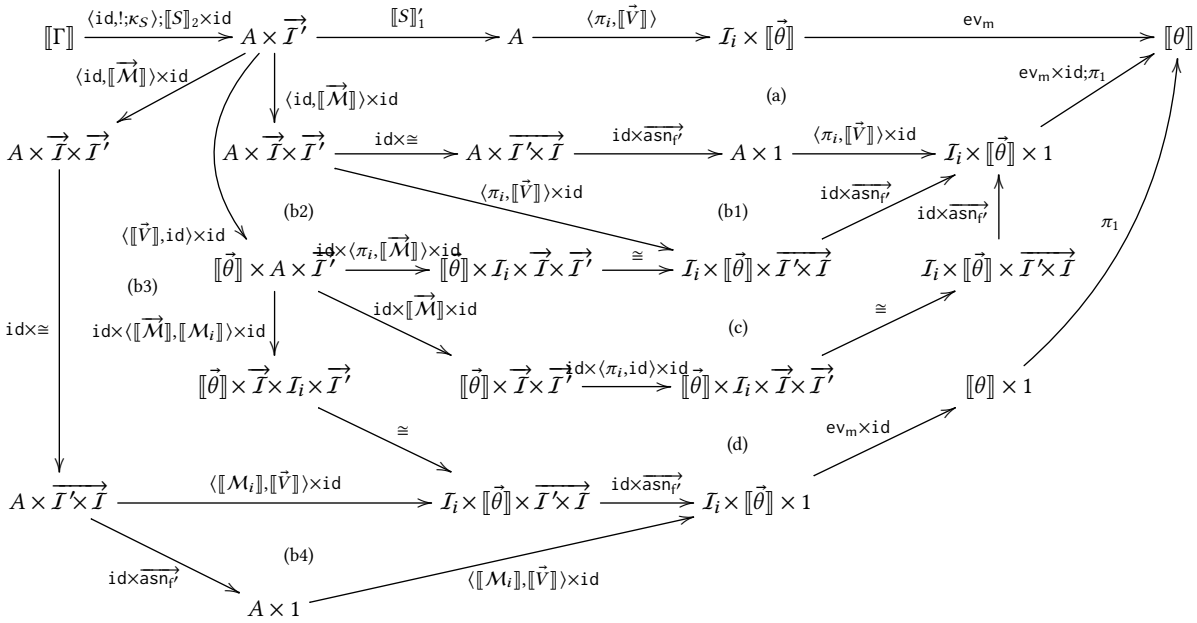
In the rest of this section we show soundness of the semantics. Let us call New, FieldUp, FieldAc and MethodCl respectively the transition rules in Figure 3 which involve state. Given a rule r, we write $(S, M) \xrightarrow{\mathsf{r}} (S', M')$ if the transition $(S, M) \longrightarrow (S', M')$ involves applying r and context rules.

Proposition 5.4 (Correctness). *Let (S, M) be a term-in-state-context and suppose $(S, M) \xrightarrow{\mathsf{r}} (S', M')$.*

(1) *If the transition r is not stateful then $[\![M]\!] = [\![M']\!]$.*
(2) *If r is one of* FieldAc *or* FieldUp *then $[\![S]\!]_2; (\mathsf{id} \times \#(\vec{\mathcal{I}})); [\![M]\!] = [\![S']\!]_2; (\mathsf{id} \times \#(\vec{\mathcal{I}})); [\![M']\!]$.*
(3) *If r is one of* MethodCl *or* New *then $[\![(S, M)]\!] = [\![(S', M')]\!]$.*

Thus, in every case, $[\![(S, M)]\!] = [\![(S', M')]\!]$.

Proof. Claim 1 is proved by using the naturality results of Sections 4.1 and 4.2. For the let construct, we show by induction on M that $[\![M[v/x]]\!] = \langle \mathtt{id}, [\![v]\!]\rangle; [\![M]\!]$. For 2 we use Lemma 4.12. For claim 3, the case for MethodCl is shown by diagram chasing, as follows (we write A for $[\![\Gamma]\!] \times \overrightarrow{\mathcal{I}}$).



The path at the top of the diagram (going from $[\![\Gamma]\!]$ to $[\![\theta]\!]$) is a decomposition of $[\![\Gamma \vdash (S, a_i.\mathsf{m}(\vec{V}))]\!]$ using Lemma 5.3; while the one passing from the bottom of the diagram is $[\![\Gamma \vdash M_i(\vec{V}/\vec{x})]\!]$. Diagram (a) trivially commutes, by definition of $[\![S]\!]_1$. Diagrams (b1)-(b4) commute because of naturality of thread-independent strategies (Lemma 4.8). Diagram (d) says that, assigning method implementations $\overrightarrow{\mathcal{M}}$ to object stores $\vec{a}'$ and calling $\mathcal{M}_i$ on some method m is the same as assigning $\overrightarrow{\mathcal{M}}$ to $\vec{a}'$ and evaluating instead a new copy of $\mathcal{M}_i$ on m. The reason the diagram commutes is that the copy of $\mathcal{M}_i$ differs from the original just in the handle name (the one returned in the codomain of $[\![\mathcal{M}_i]\!]$), but the latter is hidden via composition with ev_m. Diagram (d) commutes after pre-pending with $\langle \mathtt{id}, !; \kappa_S\rangle; [\![S]\!]_2 \times \mathtt{id}; \langle [\![\vec{V}]\!], \mathtt{id}\rangle \times \mathtt{id}$. It stipulates that if we create $\vec{a}$ with methods $\overrightarrow{\mathcal{M}}$, then calling a_i on m is the same as calling $\mathcal{M}_i$ on m. The latter holds because of the way that $\kappa_{\mathcal{I}_i}$ manipulates calls inside the interaction, by delegating calls to methods of a_i to $\mathcal{M}_i$.

Finally, for New we simply need to re-arrange the κ maps so that the one corresponding to the newly created object is pulled at the front and included in $[\![S']\!]$. □

Proposition 5.5 (Computational Soundness). *For all* $\vdash M : \mathsf{void}$, *if* $M \Downarrow$ *then* $[\![M]\!] = \{**\}$ *(i.e.* $[\![M]\!] = [\![\mathsf{skip}]\!]$*).*

Proof. This directly follows from Correctness. □

Proposition 5.6 (Computational Adequacy). *For all* $\vdash M : \mathsf{void}$, *if* $[\![M]\!] = \{**\}$ *then* $M \Downarrow$.

Proof. Suppose, for the sake of contradiction, that $[\![M]\!] = \{**\}$ and $M \not\Downarrow$. We notice that, by definition of the translation for blocking constructs (casts 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 $[\![M]\!] = \{\epsilon\}$.

For any term $\Gamma \vdash N : \theta$ and $a \in \mathsf{Names} \setminus \mathrm{dom}(\Gamma)$, construct $\Gamma_a \vdash N_a$, where $\Gamma_a = \Gamma \uplus \{a : \mathsf{Var}_I\}$, by recursively replacing each subterm of N of the shape $N'.\mathsf{m}(\vec{N})$ with $a.\mathsf{f} := (a.\mathsf{f} + 1); N'.\mathsf{m}(\vec{N})$. Var_I is an interface with a sole field $\mathsf{f} : \mathsf{int}$. Observe that each $s \in [\![\Gamma \vdash N]\!]$ induces some $s' \in [\![\Gamma_a \vdash N_a]\!]$ 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 $a.\mathsf{f}$, while P never decreases the value of $a.\mathsf{f}$. We write $[\![\Gamma_a \vdash N_a]\!]_a$ for the subset of $[\![\Gamma_a \vdash N_a]\!]$ containing precisely these plays. Then, take M_0 to be the term $\mathsf{let}\, x = \mathsf{new}(x : \mathsf{Var}_I;)\, \mathsf{in}\, (M_a[x/a]; x.\mathsf{f})$, where x a fresh variable. Because $** \in [\![M]\!]$, we get $*j \in [\![M_0]\!]$ for some $j \in \mathbb{Z}$. Consider now the infinite reduction sequence of $(\emptyset, M)$. It must have infinitely many MethodCl steps, so suppose $(\emptyset, M) \longrightarrow^* (S, M')$ contains $j+1$ such steps. Then, we obtain $(\emptyset, M_0) \longrightarrow^* (S_a, (M')_a; a.\mathsf{f})$, with $S_a(a).\mathsf{f} = j+1$. By Correctness, we have that $*j \in [\![S_a, (M')_a; a.\mathsf{f}]\!] = [\![S_a]\!]; (\mathtt{id} \times \#); [\![(M')_a; a.\mathsf{f}]\!]_a$. Since in $[\![(M')_a]\!]_a$ the value of a cannot decrease, and its initial value is $j+1$ (as stipulated by S_a), we reach a contradiction. □

6 FULL ABSTRACTION

In this section, we finally show that our game model for IMJ is fully abstract. To that end, we shall develop a suitable definability result (Lemma 6.1). Combined with Propositions 5.5 and 5.6, this will lead to our first full abstraction result for contextual approximation (Theorem 6.3). To conclude, we show that complete plays characterise contextual equivalence, i.e. they provide an equationally fully abstract model for IMJ (Theorem 6.5).

Recall that, given plays s, s', we call s an O-extension of s' (written $s \leq_O s'$) 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 $s \leq_P s'$ for the dual notion involving P-names, i.e., $s \leq_P s'$ if s, s' are the same, but the types of P-names in s' may be subtypes of those in s. Then, given $X \in \{O, P\}$ and fixed A, B, let us define $\mathsf{cl}_X(s) = \{s' \in P_{AB} \mid s' \leq_X s\}$ and $\mathsf{cl}_X(\sigma) = \bigcup_{s \in \sigma} \mathsf{cl}_X(s)$. We write $P_{\Delta|\Gamma\vdash\theta}$ for $P_{[\![\Gamma]\!][\![\theta]\!]}$. A play will be called *complete* if it is of the form $m_A Y m_B Y$.

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

Lemma 6.1 (Definability). *Let $s \in P_{\Delta|\Gamma\vdash\theta}$ be a complete play. There exists $\Delta' \supseteq \Delta$ and $\Delta'|\Gamma \vdash M : \theta$ such that $[\![\Delta'|\Gamma \vdash M : \theta]\!] = \mathsf{cl}_O(s)$.*

Proof. The argument proceeds by induction on $|s|$. For $s = \epsilon$, any divergent term suffices.

Suppose $s \neq \epsilon$. 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 handled.

Suppose

$$s = q^{\Sigma_q}\ \mathsf{call}\, o.\mathsf{m}(\vec{u})^{\Sigma_1}\ s_1\ \mathsf{ret}\, o.\mathsf{m}(v)^{\Sigma_2}\ s_2\ w^{\Sigma_3}\ s_3,$$

where $o : \mathcal{I}'$ and $\Delta(\mathcal{I}')(\mathsf{m}) : \overrightarrow{\mathcal{I}_L} \to \mathcal{I}_R$. Consider $\Delta' = \Delta \oplus \{\mathcal{I}'' \mapsto (\overrightarrow{\mathsf{f}' : \mathcal{I}_L},\ \mathsf{m}' : \mathcal{I}_R \to \theta)\}$ and the following play from $P_{\Delta'|\Gamma\vdash\mathcal{I}''}$:

$$s' = q^{\Sigma_q}\ p^{\Sigma'_1}\ s'_1\ \mathsf{call}\, p.\mathsf{m}'(v)^{\Sigma'_2}\ s'_2\ \mathsf{ret}\, p.\mathsf{m}'(v)^{\Sigma'_3}\ s'_3,$$

where $p \notin \nu(s)$, $\Sigma'_i = \Sigma_i \oplus \Sigma$, $\Sigma = \{p \mapsto (\mathcal{I}'', \overrightarrow{\mathsf{f}' \mapsto u})\}$ and s'_j is the same as s_j except that each store is extended by Σ. If $\Delta'|\Gamma \vdash M' : \mathcal{I}'$ satisfies the Lemma for s' then, for s, one can take $\mathsf{let}\, x_p = M'\, \mathsf{in}\, x_p.\mathsf{m}'(y.\mathsf{m}(\overrightarrow{x_p.\mathsf{f}'}))$, where y refers to o, i.e., y is of the shape $x.\overrightarrow{f}$, where $x \in \mathrm{dom}\,\Gamma$ and $\overrightarrow{f}$ is a sequence of fields that points at o in Σ_q.

Thanks to the reduction given above we can now assume that $s \in P_{\Delta|\Gamma\vdash\theta}$ is non-empty and

$$s = q^{\Sigma_q}\ m_0^{\Sigma_0}\ m_1^{\Sigma_1} \cdots m_{2k}^{\Sigma_{2k}},$$

where m_0 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 $m_1^{\Sigma_1} \cdots m_{2k}^{\Sigma_{2k}}$ 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: m_{2i} belongs to the thread of m_{2i-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 them without facing the problem of object fusion.
  Suppose $P(s) = \overrightarrow{p_i}$ and $p_i : \mathcal{I}_{p_i}$. Let $\Delta' = \Delta \oplus \{\mathcal{I}_P \mapsto \overrightarrow{f_i : \mathcal{I}_{p_i}}\}$. Consider $s' = (n, q)^{\Sigma'_q}\ m_0^{\Sigma'_0}\ m_1^{\Sigma'_1} \cdots m_{2k}^{\Sigma'_{2k}}$, where $\Sigma'_q = \Sigma_q \oplus \{n \mapsto (\mathcal{I}_P, \overrightarrow{\mathsf{null}})\}$ and $\Sigma'_i = \Sigma_i \oplus \{n \mapsto (\mathcal{I}_P, \overrightarrow{p_i})\} \oplus \{p_i \mapsto (\mathcal{I}_{p_i}, \overrightarrow{\mathsf{null}}) \mid \Sigma_i(p_i) \text{ undefined},\ p_i \in P(s)\}$. Let $\Gamma' = \{x_n : \mathcal{I}_P\} \oplus \Gamma$. Observe that $s' \in P_{\Delta'|\Gamma'\vdash\theta}$.
- 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 $\mathcal{I}_{play}$ 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 [6]). Let $\Delta'' = \Delta' \oplus \{\mathsf{History} \mapsto \mathsf{play} : I_{play}\}$ and $\Gamma'' = \{x_h : \mathsf{History}\} \oplus \Gamma$. Consider

  $$s'' = (h, n, q)^{\Sigma''_q}\ m_0^{\Sigma''_0}\ m_1^{\Sigma''_1} \cdots m_{2k}^{\Sigma''_{2k}}$$

  with

  $$\begin{aligned} \Sigma''_q &= \Sigma'_q \oplus \{h \mapsto (\mathsf{History}, \mathsf{play} \mapsto \mathsf{null})\}, \\ \Sigma''_{2i} &= \Sigma'_{2i} \oplus \{h \mapsto (\mathsf{History}, \mathsf{play} \mapsto s'_{\leq m_{2i}})\}, \\ \Sigma''_{2i+1} &= \Sigma'_{2i+1} \oplus \{h \mapsto (\mathsf{History}, \mathsf{play} \mapsto s'_{\leq m_{2i}})\}. \end{aligned}$$

  Observe that $s'' \in P_{\Delta''|\Gamma'\vdash\theta}$.

Now we shall decompose $m_1^{\Sigma''_1} \cdots m_{2k}^{\Sigma''_{2k}}$ into threads. Recall that each of them is a subsequence of s'' of the form

$$\mathsf{call}\, p.\mathsf{m}(\overrightarrow{u})^{\Sigma_c} \quad t \quad \mathsf{ret}\, p.\mathsf{m}(v)^{\Sigma_r}$$

where the segment t contains moves of the form call o or ret o for some $o \in O(s)$. 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 $(h, n, q, \overrightarrow{u})^{\Sigma_c}\ t'\ v^{\Sigma_r}$. Let us call the resultant term $M_{p,\mathsf{m},\overrightarrow{u},\Sigma_c}$. Next we combine terms related to the same $p : \mathcal{I}_p$ into an object definition by

$$M_p \equiv \mathsf{new}(x : I_p; \mathsf{m} : \lambda\overrightarrow{u}.\mathsf{case}(\overrightarrow{u}, \Sigma_c)[M_{p,\mathsf{m},\overrightarrow{u},\Sigma_c}]).$$

The case statement, which can be implemented in IMJ using nested ifs, is needed to recognize instances of $\overrightarrow{u}$ and Σ_c that really occur in threads related to p. In such cases the corresponding term $M_{p,\mathsf{m},\overrightarrow{u},\Sigma_c}$ will be run. Otherwise, the statement leads to divergence.

The term M for s can now be obtained by taking

$$\begin{array}{l}\mathsf{let}\ x_n = \mathsf{new}(x : \mathcal{I}_P;)\ \mathsf{in}\\ \mathsf{let}\ x_h = \mathsf{new}(x : \mathsf{History};)\ \mathsf{in}\\ \mathsf{let}\ \overrightarrow{x_{p_i} = M_{p_i}}\ \mathsf{in}\\ \qquad \mathsf{assert}(q^{\Sigma_q}); \overrightarrow{x_n.f_i = x_{p_i}}; \mathsf{make}(\Sigma_0''); \mathsf{play}(m_0)\end{array}$$

where $\overrightarrow{x_{p_i} = M_{p_i}}$ represents a series of bindings (one for each P-name $p_i \in P(s)$), $\mathsf{assert}((h,n,q)^{\Sigma_q''})$ is a conditional that converges if and only if the initial values of free Γ identifiers as well as values accessible through them are consistent with q and Σ_q respectively, $\mathsf{make}(\Sigma_0'')$ is a sequence of assignments that set values to those specified in Σ_0'' (up-casts need to be performed to ensure typability) and $\mathsf{play}(m_0)$ is skip, i, null or, if m_0 is a name, it is a term of the form $(\theta)y.\vec{f}$, where y is x_n or $(x : I_x) \in \Gamma$ such that $y.\vec{f}$ gives an access path to m_0 in Σ_0''. □

We conclude with full abstraction results both in inequational and equational forms. For technical convenience, we shall use a modified (but equivalent) definition of contextual approximation.

Lemma 6.2. *Let* $\Gamma = \{x_1 : \mathcal{I}_1, \cdots, x_k : \mathcal{I}_k\}$, $\Delta|\Gamma \vdash M_i : \theta$ $(i = 1,2)$, *and* $\Delta' = \Delta \cup \{\mathsf{Wrap}_{\Gamma,\mathcal{I}} \mapsto (f : (\mathcal{I}_1, \cdots, \mathcal{I}_k) \to \theta)\}$. *Then* $\Delta|\Gamma \vdash M_1 \sqsubseteq_{\approx} M_2$ *if and only if, for all* $\Delta'' \supseteq \Delta'$ *and* $\Delta'', z : \mathsf{Wrap}_{\Gamma,\mathcal{I}} \vdash \mathsf{test} : \mathsf{void}$, *if* $C_{\mathsf{test}}[M_1] \Downarrow$ *then* $C_{\mathsf{test}}[M_2] \Downarrow$, *where* $C_{\mathsf{test}}[-] \equiv \mathsf{let}\ z = \mathsf{new}(x : \mathsf{Wrap}_{\Gamma,\mathcal{I}}; f : \lambda\vec{x_i}.[-])\ \mathsf{in}\ \mathsf{test}$.

Proof. The Lemma holds because, on the one hand, it relies on contexts of a specific shape and, on the other hand, any closing context $C[-]$ for M_i can be presented in the above form with $\mathsf{test} \equiv C[z.f(x_1, \cdots, x_k)]$. □

Given a term $\Delta|\Gamma \vdash M : \theta$, let us write $[\![\Delta|\Gamma \vdash M : \theta]\!]_{comp}$ for the set of complete plays from $[\![\Delta|\Gamma \vdash M : \theta]\!]$. In what follows, we shall often omit $\Delta|\Gamma, \vdash$ for brevity.

Theorem 6.3 (Inequational full abstraction). *Given* $\Delta|\Gamma \vdash M_i : \theta$ $(i = 1,2)$, *we have* $\Delta|\Gamma \vdash M_1 \sqsubseteq_{\approx} M_2 : \theta$ *if and only if*

$$\mathsf{cl}_P([\![\Delta|\Gamma \vdash M_1 : \theta]\!]_{comp}) \subseteq \mathsf{cl}_P([\![\Delta|\Gamma \vdash M_2 : \theta]\!]_{comp}).$$

Proof. The proof uses the following play transformation. Given $t = q^{\Sigma_q} s_1 a^{\Sigma_a} s_2 \in P_{\Delta|\Gamma\vdash\theta}$, we define $\overline{t} \in P_{\Delta',\mathsf{Wrap}_{\Gamma,\mathcal{I}}\vdash\mathsf{void}}$ as

$$n^{\Sigma_n}\ \mathsf{call}\, n.f(q)^{\Sigma_q \oplus \Sigma_n}\ s_1^{\oplus\Sigma_n}\ \mathsf{ret}\, n.f(a)^{\Sigma_a\oplus\Sigma_n}\ s_2^{\oplus\Sigma_n}\ *^{\Sigma\oplus\Sigma_n},$$

where $\Delta', \mathsf{Wrap}_{\Gamma,\mathcal{I}}$ are the same as in the above Lemma, $\Sigma_n = \{n \mapsto (\mathsf{Wrap}_{\Gamma,\mathcal{I}}, \emptyset)\}$, $s^{\oplus\Sigma_n}$ stands for s in which each store was augmented by Σ_n and Σ is the store of the last move in t. Intuitively, $\overline{t}$ is the play that $C_{\mathsf{test}}[-]$ needs to provide for a terminating interaction with t.

($\Rightarrow$). Let $s \in \mathsf{cl}_P([\![M_1]\!]_{comp})$. Then there exists $s' \in [\![M_1]\!]_{comp}$ with $s \in \mathsf{cl}_P(s')$. Apply Definability to $\overline{s'}$ to obtain $\Delta'', z : \mathsf{Wrap}_{\Gamma,\mathcal{I}} \vdash \mathsf{test} : \mathsf{void}$ such that $[\![\mathsf{test}]\!] = \mathsf{cl}_O(\overline{s'})$. Because $s' \in [\![M_1]\!]_{comp}$ and Adequacy holds, we must have $C_{\mathsf{test}}[M_1] \Downarrow$. From $M_1 \sqsubseteq_{\approx} M_2$ we obtain $C_{\mathsf{test}}[M_2] \Downarrow$. Hence, because of Soundness, there exists $s'' \in [\![M_2]\!]_{comp}$ such that $\overline{s''} \in [\![\mathsf{test}]\!]$. Since $[\![\mathsf{test}]\!] = \mathsf{cl}_O(\overline{s'})$, it follows that $\overline{s''} \in \mathsf{cl}_O(\overline{s'})$ and, consequently, $s' \in \mathsf{cl}_P(s'')$. Thus, $s \in \mathsf{cl}_P(s')$ and $s' \in \mathsf{cl}_P(s'')$. Hence, $s \in \mathsf{cl}_P(s'')$ and, because $s'' \in [\![M_2]\!]_{comp}$, we can conclude $s \in \mathsf{cl}_P([\![M_2]\!]_{comp})$.

(⇐). Let $C_{\text{test}}[-]$ be such that $C_{\text{test}}[M_1] \Downarrow$. By Soundness, there exists $s \in [\![M_1]\!]_{comp}$ such that $\overline{s} \in [\![\text{test}]\!]$. Because $[\![M_1]\!]_{comp} \subseteq \mathsf{cl}_P([\![M_1]\!]_{comp})$ and $\mathsf{cl}_P([\![M_1]\!]_{comp}) \subseteq \mathsf{cl}_P([\![M_2]\!]_{comp})$, we also have $s \in \mathsf{cl}_P([\![M_2]\!]_{comp})$. Thus, there exists $s' \in [\![M_2]\!]_{comp}$ such that $s \in \mathsf{cl}_P(s')$. Consequently, $\overline{s'} \in \mathsf{cl}_O(\overline{s})$. Since $\overline{s} \in [\![\text{test}]\!]$, we also have $\overline{s'} \in [\![\text{test}]\!]$. Because $s' \in [\![M_2]\!]_{comp}$ and $\overline{s'} \in [\![\text{test}]\!]$, by Adequacy, we can conclude that $C_{\text{test}}[M_2] \Downarrow$. □

Example 6.4. Let us revisit Example 2.12. We have $\mathsf{cl}_P(\sigma_1) = \sigma_1$ and $\mathsf{cl}_P(\sigma_2) = \sigma_2 \cup \{*^{\emptyset}, n^{\{n \mapsto (\mathsf{Empty}, \emptyset)\}}\}$, i.e. $\mathsf{cl}_P(\sigma_1) \subsetneq \mathsf{cl}_P(\sigma_2)$. Thus, it follows from Theorem 6.3 that $\Delta|\emptyset \vdash M_1 \sqsubsetneq M_2$ and $\Delta|\emptyset \vdash M_1 \not\cong M_2$.

Theorem 6.5 (Equational full abstraction). *Given $\Delta|\Gamma \vdash M_i : \theta$ $(i = 1, 2)$, $\Delta|\Gamma \vdash M_1 \cong M_2 : \theta$ if and only if*

$$[\![\Delta|\Gamma \vdash M_1 : \theta]\!]_{comp} = [\![\Delta|\Gamma \vdash M_2 : \theta]\!]_{comp}.$$

Proof. The preceding result implies that $M_1 \cong M_2$ if and only if $\mathsf{cl}_P([\![M_1]\!]_{comp}) = \mathsf{cl}_P([\![M_2]\!]_{comp})$. We show that this implies $[\![M_1]\!]_{comp} = [\![M_2]\!]_{comp}$. Let $s \in [\![M_1]\!]_{comp}$. By $\mathsf{cl}_P([\![M_1]\!]_{comp}) = \mathsf{cl}_P([\![M_2]\!]_{comp})$, it must be the case that $s \in \mathsf{cl}_P([\![M_2]\!]_{comp})$, i.e., there exists $s' \in [\![M_2]\!]_{comp}$ such that $s \in \mathsf{cl}_P(s')$. Again, by $\mathsf{cl}_P([\![M_1]\!]_{comp}) = \mathsf{cl}_P([\![M_2]\!]_{comp})$, it follows that $s' \in \mathsf{cl}_P([\![M_1]\!]_{comp})$, i.e., there exists $s'' \in [\![M_1]\!]_{comp}$ such that $s' \in \mathsf{cl}_P(s'')$. So, we have $s \in \mathsf{cl}_P(s')$ and $s' \in \mathsf{cl}_P(s'')$, which implies $s \in \mathsf{cl}_P(s'')$. However, $s, s'' \in [\![M_1]\!]_{comp}$, so $s \in \mathsf{cl}_P(s'')$ entails $s = s''$. Hence, $s \in \mathsf{cl}_P(s')$ and $s' \in \mathsf{cl}_P(s)$, and $s = s'$ follows. Because $s' \in [\![M_2]\!]_{comp}$, we showed $s \in [\![M_2]\!]_{comp}$. The other inclusion is derived analogously. □

7 CONCLUSIONS

We have presented a game model of Java-style objects. We see it as a stepping stone towards advancing game semantics to more and more realistic programming languages, in order to catch up with the fast evolution of programming paradigms. The advantage of game semantics over other denotational approaches is its concrete, low-level nature, which has allowed for full abstraction results like the one presented in this paper. Moreover, compared to operational approaches, game semantics is designed to accommodate open programs as first-class citizens and in a compositional manner. As such, it can be seen as combining the best of two worlds: denotational and operational. Apart from the foundational value found in the latter statement, there is practical value, namely that game semantics could play the role of a more generally applicable semantics for open code: be it programs-in-context, libraries with external dependencies, code distributed over a network, etc. Evidence of this can be located in open trace models used for low-level languages which are based, each to a different extent, on the game semantics approach [14, 19, 42, 44].

As mentioned in the Introduction, we are also pursuing a more applied strand of work, which relies on semantic insights to inform the design of verification tools. We believe that in the long run game semantics can provide a fruitful methodology for dealing with a variety of verification tasks in a compositional manner [30, 31].

ACKNOWLEDGMENTS

This work was supported by the Engineering and Physical Sciences Research Council (EP/J019577/1) and the Royal Academy of Engineering (Research Fellowship: Tzevelekos).

REFERENCES

[1] M. Abadi and L. Cardelli. 1996. *A theory of objects.* Springer Verlag.
[2] E. Ábraham, M. M. Bonsangue, F. S. de Boer, A. Gruener, and M. Steffen. 2004. Observability, Connectivity, and Replay in a Sequential Calculus of Classes. In *FMCO (Lecture Notes in Computer Science)*, Vol. 3657. Springer, 296–316.

[3] S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and I. D. B. Stark. 2004. Nominal Games and Full Abstraction for the Nu-Calculus. In *Proceedings of LICS*. IEEE Computer Society Press, 150–159.

[4] S. Abramsky, K. Honda, and G. McCusker. 1998. Fully Abstract Game Semantics for General References. In *Proceedings of IEEE Symposium on Logic in Computer Science*. Computer Society Press, 334–344.

[5] S. Abramsky, R. Jagadeesan, and P. Malacaria. 2000. Full Abstraction for PCF. *Information and Computation* 163 (2000), 409–470.

[6] S. Abramsky and G. McCusker. 1997. Linearity, Sharing and State: a fully abstract game semantics for Idealized Algol with active expressions. In *Algol-like languages*, P. W. O'Hearn and R. D. Tennent (Eds.). Birkhaüser, 297–329.

[7] S. Abramsky and G. McCusker. 1998. Game semantics. In *Logic and Computation*, H. Schwichtenberg and U. Berger (Eds.). Springer-Verlag. Proceedings of the NATO Advanced Study Institute, Marktoberdorf.

[8] J. Alves-Foss (Ed.). 1999. *Formal Syntax and Semantics of Java*. Lecture Notes in Computer Science, Vol. 1523. Springer.

[9] J. Alves-Foss and F. S. Lam. 1999. Dynamic Denotational Semantics of Java. In *Formal Syntax and Semantics of Java (Lecture Notes in Computer Science)*, Vol. 1523. Springer, 201–240.

[10] G.M. Bierman, M.J. Parkinson, and A.M. Pitts. 2002. *MJ: An imperative core calculus for Java and Java with effects.* Technical Report 563. Computer Laboratory, University of Cambridge.

[11] H. Björklund and T. Schwentick. 2010. On notions of regularity for data languages. *Theor. Comput. Sci.* 411, 4-5 (2010), 702–715.

[12] V. Danos and R. Harmer. 2002. Probabilistic game semantics. *ACM Transactions on Computational Logic* 3(3) (2002), 359–382.

[13] M. J. Gabbay and A. M. Pitts. 2002. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing* 13 (2002), 341–363.

[14] D. R. Ghica and N. Tzevelekos. 2012. A System-Level Game Semantics. *Electr. Notes Theor. Comput. Sci.* 286 (2012), 191–211.

[15] R. Harmer and G. McCusker. 1999. A fully abstract game semantics for finite nondeterminism. In *Proceedings of Fourteenth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 422–430.

[16] K. Honda and N. Yoshida. 1999. Game-Theoretic Analysis of Call-by-Value Computation. *Theor. Comput. Sci.* 221, 1-2 (1999), 393–456.

[17] J. M. E. Hyland and C.-H. L. Ong. 2000. On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. *Information and Computation* 163(2) (2000), 285–408.

[18] G. Jaber. 2015. Operational Nominal Game Semantics. In *Proceedings of FOSSACS*. 264–278.

[19] R. Jagadeesan, C. Pitcher, J. Rathke, and J. Riely. 2011. Local Memory via Layout Randomization. In *Proceedings of CSF*. 161–174.

[20] A. Jeffrey and J. Rathke. 2003. Java Jr: Fully Abstract Trace Semantics for a Core Java Language. In *Proceedings of ESOP*. Lecture Notes in Computer Science, Vol. 3444. Springer, 423–438.

[21] A. Jeffrey and J. Rathke. 2005. A fully abstract may testing semantics for concurrent objects. *Theor. Comput. Sci.* 338, 1-3 (2005), 17–63.

[22] S. N. Kamin and U. S. Reddy. 1994. Two semantic models of object-oriented languages. In *Theoretical Aspects of Object Oriented Programming*. MIT Press.

[23] V. Koutavas and M. Wand. 2007. Reasoning About Class Behavior. In *Proceedings of FOOL/WOOD*.

[24] J. Laird. 1997. Full Abstraction for Functional Languages with Control. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science*. 58–67.

[25] J. Laird. 2004. A Game Semantics of Local Names and Good Variables. In *Proceedings of FOSSACS*. Lecture Notes in Computer Science, Vol. 2987. Springer-Verlag, 289–303.

[26] J. Laird. 2006. Game Semantics for Higher-Order Concurrency. In *FSTTCS (Lecture Notes in Computer Science)*, Vol. 4337. 417–428.

[27] J. Laird. 2010. Game Semantics for Call-by-Value Polymorphism. In *Proceedings of ICALP (Lecture Notes in Computer Science)*, Vol. 6199. Springer, 187–198.

[28] J. Laird. 2013. Game semantics for a polymorphic programming language. *J. ACM* 60, 4 (2013), 29:1–29:27.

[29] S. B. Lassen and P. B. Levy. 2008. Typed Normal Form Bisimulation for Parametric Polymorphism. In *Proceedings of LICS*. IEEE Computer Society, 341–352.

[30] A. S. Murawski, S. J. Ramsay, and N. Tzevelekos. 2015a. A Contextual Equivalence Checker for IMJ*. In *Proceedings of ATVA (Lecture Notes in Computer Science)*, Vol. 9364. Springer, 234–240.

[31] A. S. Murawski, S. J. Ramsay, and N. Tzevelekos. 2015b. Game Semantic Analysis of Equivalence in IMJ. In *Proceedings of ATVA (Lecture Notes in Computer Science)*, Vol. 9364. Springer, 411–428.

[32] A. S. Murawski, S. J. Ramsay, and N. Tzevelekos. 2017. Reachability in pushdown register automata. *J. Comput. Syst. Sci.* 87 (2017), 58–83.

[33] A. S. Murawski and N. Tzevelekos. 2011. Game Semantics for Good General References. In *Proceedings of LICS*. IEEE Computer Society Press, 75–84.

[34] A. S. Murawski and N. Tzevelekos. 2013. Full abstraction for Reduced ML. *Ann. Pure Appl. Log.* 164, 11 (2013), 1118–1143.

[35] A. S. Murawski and N. Tzevelekos. 2014a. Game Semantics for Interface Middleweight Java. In *Proceedings of POPL*. 517–528.

[36] A. S. Murawski and N. Tzevelekos. 2014b. Game Semantics for Nominal Exceptions. In *Proceedings of FOSSACS (Lecture Notes in Computer Science)*, Vol. 8412. 164–179.

[37] A. S. Murawski and N. Tzevelekos. 2016a. An invitation to game semantics. *SIGLOG News* 3, 2 (2016), 56–67.

[38] A. S. Murawski and N. Tzevelekos. 2016b. Nominal Game Semantics. *Foundations and Trends in Programming Languages* 2, 4 (2016), 191–269.

[39] H. Nickau. 1996. *Hereditarily Sequential Functionals: A Game-Theoretic Approach to Sequentiality.* Ph.D. Dissertation. Universität-Gesamthochschule-Siegen.

[40] J. Power and E. Robinson. 1997. Premonoidal Categories and Notions of Computation. *Math. Struct. Comput. Sci.* 7, 5 (1997), 453–468.

[41] J. Power and H. Thielecke. 1999. Closed Freyd- and kappa-categories. In *Proceedings of ICALP (Lecture Notes in Computer Science)*, Jirí Wiedermann,

Peter van Emde Boas, and Mogens Nielsen (Eds.), Vol. 1644. Springer, 625–634. DOI: http://dx.doi.org/10.1007/3-540-48523-6_59

[42] T. Ramananandro, Z. Shao, S.-C. Weng, J. Koenig, and Y. Fu. 2015. A Compositional Semantics for Verified Separate Compilation and Linking. In *Proceedings of CPP*. 3–14.

[43] I. D. B. Stark. 1995. *Names and Higher-Order Functions.* Ph.D. Dissertation. University of Cambridge Computing Laboratory. Technical Report No. 363.

[44] G. Stewart, L. Beringer, S. Cuellar, and A. W. Appel. 2015. Compositional CompCert. In *Proceedings of POPL*. 275–287.

[45] N. Tzevelekos. 2009. Full abstraction for nominal general references. *Logical Methods in Computer Science* 5, 3 (2009).

[46] N. Tzevelekos. 2011. Fresh-register automata. In *Proceedings of POPL*. ACM Press, 295–306.

A ASSOCIATIVITY

Here we show that strategy composition is associative. That is, if $\rho : A \to B$, $\sigma : B \to C$ and $\tau : C \to D$ then $(\rho;\sigma);\tau = \rho;(\sigma;\tau)$. We first need to accommodate for composing three strategies. The set of polarities for such extended interactions is given by (M stands for "middle"):

$$Pol_3 = \{X_L, X_L\overline{X}_M, X_M\overline{X}_R, X_L\overline{X}_R, X_R \mid X \in \{O,P\}\}$$

Thus, for ρ, σ, τ as above,

- polarities of the form X_L will represent moves played only by ρ;
- $X_L Y_M$ represent moves played between ρ and σ;
- $X_M Y_R$ represent moves played between σ and τ;
- while $X_L Y_R$ are moves played between ρ and τ.

For example, the latter kind of moves are used in scenarios where ρ calls a method of a name introduced by τ, or viceversa. Thus, the polarities of each binary projection of $ABCD$ are:

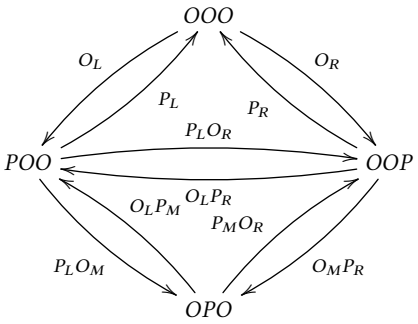
$$p(AB) = \{X_L, X_L\overline{X}_M, X_L\overline{X}_R \mid X \in \{O,P\}\}$$
$$p(BC) = \{X_L\overline{X}_M, X_M\overline{X}_R \mid X \in \{O,P\}\}$$
$$p(AC) = \{X_L, X_L\overline{X}_R, X_M\overline{X}_R \mid X \in \{O,P\}\}$$
$$p(CD) = \{X_L\overline{X}_R, X_M\overline{X}_R, X_R \mid X \in \{O,P\}\}$$
$$p(BD) = \{X_L\overline{X}_M, X_L\overline{X}_R, X_R \mid X \in \{O,P\}\}$$
$$p(AD) = \{X_L, X_R \mid X \in \{O,P\}\}$$

Given a sequence of moves from M_{ABCD} (which is defined analogously to M_{ABC}) and an $X \in \{AB, BC, AC, CD, BD, AD\}$, we let $s \upharpoonright X$ be the subsequence of s containing those moves m^Σ of s such that $p(m) \in p(X)$.

Pseudo-polarities and complementation are defined by:

$$\overline{P_R} = \overline{P_L} = OOO = \{O_L, O_R\}$$
$$\overline{O_L} = \overline{O_L P_M} = \overline{O_L P_R} = POO = \{P_L, P_L O_M, P_L O_R\}$$
$$\overline{P_L O_M} = \overline{O_M P_R} = OPO = \{O_L P_M, P_M O_R\}$$
$$\overline{P_M O_R} = \overline{P_L O_R} = \overline{O_R} = OOP = \{O_M P_R, O_L P_R, P_R\}$$

and the interaction diagram is depicted below.



An ***interaction sequence*** in $ABCD$ is a sequence s of moves-with-store in $ABCD$ satisfying the following conditions.

- For each $s'm^{\Sigma} \sqsubseteq s$ we have $\mathrm{dom}(\Sigma) = \mathsf{Av}(s'm^{\Sigma})$. (*Frugality*)
- If $a \in \mathrm{dom}(\Sigma)$ with $\Sigma(a) : \mathcal{I}$ then: (*Well-classing*)
  - if $m \in M_X$, for $X \in \{A, B, C, D\}$, then $\mathcal{I} \leq \xi_X(m, a)$;
  - for all n^T in s', if $a \in \mathrm{dom}(T)$ then $T(a) : \mathcal{I}$;
  - if $\Delta(\mathcal{I}).\mathsf{m} = \vec{\theta} \to \theta$ then:
    - $m = \mathrm{call}\, a.\mathsf{m}(\vec{v})$ with $\Sigma \vdash \vec{v} : \vec{\theta'}$ implies $\vec{\theta'} \leq \vec{\theta}$,
    - $m = \mathrm{ret}\, a.\mathsf{m}(v)$ with $\Sigma \vdash v : \theta'$ implies $\theta' \leq \theta$.
- There is a polarity function p from move occurrences in $\underline{s}$ to Pol_3 such that:
  - For all $m_X \in M_X$ ($X = A, B, C, D$) occurring in $\underline{s}$ we have $p(m_A) = O_L$, $p(m_B) = P_L O_M$, $p(m_C) = P_M O_R$ and $p(m_D) = P_R$;
  - If mn are consecutive moves in $\underline{s}$ then $p(n) \in \overline{p(m)}$. (*Alternation*)
- If $s'm^{\Sigma} \sqsubseteq s$ then $m = \mathrm{call}\, a.\mathsf{m}(\overline{v})$ implies $o(a) \in \overline{p(m)}$. (*Well-calling*)
- For each $X \in \{AB, BC, AC, CD, BD, AD\}$ we have $\underline{s \upharpoonright X} \in L_X$. (*Projecting*)
- If $s'm^{\Sigma} \sqsubseteq s$ and $m = \mathrm{ret}\, a.\mathsf{m}(v)$ then there is a move n^T in s' such that, for all X such that $p(m) \in p(X)$, n is the justifier of m in $\underline{s \upharpoonright X}$. (*Well-returning*)
- For all $X, Y \in \{AB, BC, CD\}$: (*Laird's conditions*)
  - $P(s \upharpoonright_Y X) \cap O(s \upharpoonright_Y AD) = \emptyset$, and if $X \neq Y$ then $P(s \upharpoonright_Y X) \cap P(s \upharpoonright_Y Y) = \emptyset$;
  - for all $s' \sqsubseteq s$ ending in $m^{\Sigma} n^T$ and $(a) \in \mathrm{dom}(T)$, if
    - $p(m) \in X \setminus OOO$ and $a \notin v(s' \upharpoonright_Y X)$,
    - or $p(m) \in OOO$ and $a \notin v(s' \upharpoonright_Y AD)$,

    then $\Sigma(a) = T(a)$.

We write $Int(ABCD)$ for the set of interaction sequences in $ABCD$.

Lemma A.1. *Each $s \in Int(ABCD)$ has a unique polarity function p.*

Proof. Suppose $s \in Int(ABCD)$. We show by induction on $|s|$ that $\underline{s}$ has a unique run in the interaction diagram. The base case is trivial, so suppose $\underline{s} = s'm$. By induction hypothesis, s' has a unique run, which reaches some state X. We do a case analysis on m. If $m \in M_A \cup M_B \cup M_C \cup M_D$ then there is a unique edge accepting m and, by alternation, this edge must depart from X. If, on the other hand, $m = \mathrm{call}\, a.\mathsf{m}(\vec{v})$ then $o(a) \in \overline{p(m)}$ gives the following possible cases.

- $o(a) \in OOO$ and $p(m) \in \{P_L, P_R\}$;

- $o(a) \in POO$ and $p(m) \in \{O_L, O_L P_R, O_L P_M\}$;
- $o(a) \in OPO$ and $p(m) \in \{P_L O_M, O_M P_R\}$;
- $o(a) \in OOP$ and $p(m) \in \{P_M O_R, P_L O_R, O_R\}$.

Now observe that, in each case, at most one choice for $p(m)$ can be available from X and, by alternation, exactly one is. Finally, let $m = \mathrm{ret}\, a.\mathrm{m}(v)$ 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. □

We define polarity projections for each component as follows,

$$\begin{aligned}
\pi_{AB}(X_L) &= X & \pi_{AB}(X_L Y_M) &= X & \pi_{AB}(X_L Y_R) &= X \\
\pi_{BC}(X_L Y_M) &= Y & \pi_{BC}(X_M Y_R) &= X & & \\
\pi_{AC}(X_L) &= X & \pi_{AC}(X_M Y_R) &= X & \pi_{AC}(X_L Y_R) &= X \\
\pi_{CD}(X_M Y_R) &= Y & \pi_{CD}(X_L Y_R) &= Y & \pi_{CD}(Y_R) &= Y \\
\pi_{BD}(X_L Y_M) &= Y & \pi_{BD}(X_L Y_R) &= Y & \pi_{BD}(Y_R) &= Y \\
\pi_{AD}(X_L) &= X & \pi_{AD}(Y_R) &= Y & &
\end{aligned}$$

where $X, Y \in \{O, P\}$. We can now show the following.

Lemma A.2. *Let $s \in Int(ABCD)$. Then, for all $X \in \{AB, BC, AC, CD, BD, AD\}$ and m^{Σ} in s, if $p(m) \in p(X)$ then $\pi_X(p(m)) = p_X(m)$, where p_X is the polarity function in $s \upharpoonright X$.*

Proof. Proved similarly to Lemma 3.21.

□

Lemma A.3. *Let $s \in Int(ABCD)$.*

(1) *If m is a move in s introducing some name a in it, then:*
   - *if $p_X(m) = P$, for some $X \in \{AB, BC, CD\}$, then m introduces a in $s \upharpoonright_\gamma X$;*
   - *if $p_{AD}(m) = O$ then m introduces a in $s \upharpoonright_\gamma AD$.*

(2) $\nu(s) = P(s \upharpoonright_\gamma AB) \uplus P(s \upharpoonright_\gamma BC) \uplus P(s \upharpoonright_\gamma CD) \uplus O(s \upharpoonright_\gamma AD)$.

Proof. For 1, suppose m introduces a in s, say $s' m^{\Sigma} \sqsubseteq s$, and $\pi_{AD}(m) = O$. Let $s' m^{\Sigma} \upharpoonright_\gamma AD = s'' m^{\Sigma'}$. If $a \in \nu(m^{\Sigma'})$ then we are done. Otherwise, since $p(m) \in OOO$, by Laird's conditions we have that the values of $\Sigma \setminus \Sigma'$ are copied from the last move in s', which contradicts a being introduced at m^{Σ}. The other cases are similar. Item 2 then follows, using also disjointness conditions from the definition of $Int(ABCD)$. □

We next proceed to show that each interaction sequence in $ABCD$ projects into interaction sequences in ABC and ACD. First, we let

$$\begin{aligned}
p(ABC) &= p(AB) \cup p(BC) \cup p(AC) \\
&= \{X_L, X_L \overline{X}_M, X_M \overline{X}_R, X_L \overline{X}_R \mid X \in \{O, P\}\} \\
p(ACD) &= p(AC) \cup p(CD) \cup p(AD) \\
&= \{X_L, X_M \overline{X}_R, X_L \overline{X}_R, X_R \mid X \in \{O, P\}\}
\end{aligned}$$

and, for each $s \in Int(ABCD)$ and $X \in \{ABC, ACD\}$, we define $s \upharpoonright X$ to be the subsequence of X comprising those of its moves with polarities in X.

Lemma A.4. *Let $s \in Int(ABCD)$, m^{Σ} an element of s and $a \in$ Names.*

(1) *(a) If m introduces a in $s \upharpoonright_\gamma ABC$ and $p_{AB}(m) = P$ (resp. $p_{BC}(m) = P$) then m introduces a in $s \upharpoonright_\gamma AB$ ($s \upharpoonright_\gamma BC$).*
(b) If m introduces a in $s \upharpoonright_\gamma ACD$ and $p_{CD}(m) = P$ (resp. $p_{AD}(m) = O$) then m introduces a in $s \upharpoonright_\gamma CD$ ($s \upharpoonright_\gamma AD$).

(2) $\nu(s \upharpoonright_\gamma ABC) \cap \nu(s \upharpoonright_\gamma ACD) \subseteq \nu(s \upharpoonright_\gamma AC)$.

(3) *(a) If m introduces a in $s \upharpoonright_\gamma ABC$ and $p_{AC}(m) = O$ then m introduces a in $s \upharpoonright_\gamma AC$.*
(b) If m introduces a in $s \upharpoonright_\gamma ACD$ and $p_{AC}(m) = P$ then m introduces a in $s \upharpoonright_\gamma AC$.

(4) *(a) $\nu(s \upharpoonright_\gamma ABC) = P(s \upharpoonright_\gamma AB) \uplus P(s \upharpoonright_\gamma BC) \uplus O(s \upharpoonright_\gamma AC)$.*
(b) $\nu(s \upharpoonright_\gamma ACD) = P(s \upharpoonright_\gamma AC) \uplus P(s \upharpoonright_\gamma CD) \uplus O(s \upharpoonright_\gamma AD)$.

Proof. For 1, we show (b), and (a) is shown similarly. Suppose WLOG that $s = s'm^{\Sigma}$ and let $s \upharpoonright_\gamma CD = s''m^{\Sigma'}$. If $a \in \nu(m^{\Sigma'})$ then we are done. Otherwise, let n^T be the last move in s'. By the interaction diagram, $p(n) \in p(CD)$. Setting $X = \{b \in \text{Names} \mid a \in \Sigma^*(\{b\})\}$, and since $a \notin \nu(s \upharpoonright_\gamma CD) = \mathsf{Av}(s \upharpoonright CD)$, we have that $\Sigma(b) = T(b)$ for all $b \in X$. Now, since m introduces a in $s \upharpoonright_\gamma ACD$, we have $a \in \nu(s \upharpoonright_\gamma ACD) = \mathsf{Av}(s \upharpoonright ACD)$. Using the definition of Av, we get:

$$\begin{aligned}\mathsf{Av}(s \upharpoonright_\gamma ACD) &= \Sigma^*(\mathsf{Av}(s' \upharpoonright ACD) \cup \nu(m)) \\ &= \Sigma^*(\mathsf{Av}(s' \upharpoonright ACD)) \cup \Sigma^*(\nu(m))\end{aligned}$$

By hypothesis, $a \notin \Sigma^*(\nu(m))$, thus $a \in \Sigma^*(\mathsf{Av}(s' \upharpoonright ACD))$. So let $b' \in \mathsf{Av}(s' \upharpoonright ACD)$ with $a \in \Sigma^*(\{b'\})$, and suppose $s' \upharpoonright_\gamma ACD = \cdots n^{T'}$. As $b' \in \mathsf{Av}(s' \upharpoonright ACD)$, we have $b' \in \mathrm{dom}(T')$. But $b' \in X$ and $\forall b \in X.T(b) = \Sigma(b)$, therefore $T'^*(\{b'\}) = \Sigma^*(\{b'\}) \ni a$, contradiction to m introducing a in $s \upharpoonright_\gamma ACD$. Similarly for $\pi_{AD}(p(m)) = O$.

For 2, we do induction on $|s|$, with base case clear. So let $s = tm^{\Sigma}$ and pick some $a \in \nu(s)$. We show that if $a \in \nu(s \upharpoonright_\gamma ABC) \cap \nu(s \upharpoonright_\gamma ACD)$ then $a \in \nu(s \upharpoonright_\gamma AC)$. Let us assume that a is introduced in s in some move m' with $p(m') \in POO \cup OPO$. Then, $p_{AB}(m') = P$ or $p_{BC}(m') = P$ so, by first part of Lemma A.3, m' introduces a in $s \upharpoonright_\gamma ABC$. If $a \in \nu(t \upharpoonright_\gamma ACD)$ then, by IH, $a \in \nu(t \upharpoonright_\gamma AC)$ so $a \in \nu(s \upharpoonright_\gamma AC)$. Suppose now $s \upharpoonright_\gamma ACD = t'm^{\Sigma'}$ and $a \in \nu(m^{\Sigma'}) \setminus \nu(t')$. By item 1 and Lemma A.3 we have that $\pi_{CD}(m) \neq P$ and $\pi_{AD}(m) \neq O$ so, since $p(m) \in p(ACD)$, we have that $p(m) \in \{P_L, P_M O_R, P_L O_R\}$, and in particular $p(m) \in p(AC)$ so let $s \upharpoonright_\gamma AC = t''m^{\Sigma''}$. We claim that $a \in \nu(m^{\Sigma''})$. For, suppose otherwise. Then, $a \in \nu(\Sigma')$ implies

$$\begin{aligned}a \in \mathsf{Av}(s \upharpoonright_\gamma ACD) &= \Sigma^*(\mathsf{Av}(t \upharpoonright_\gamma ACD) \cup \nu(m)) \\ &= \Sigma^*(\mathsf{Av}(t \upharpoonright_\gamma ACD)) \cup \Sigma^*(\nu(m))\end{aligned}$$

and, since by hypothesis $a \in \Sigma^*(\nu(m))$, we have $a \in \Sigma^*(\mathsf{Av}(t \upharpoonright_\gamma ACD))$. Let $b \in \mathsf{Av}(t \upharpoonright_\gamma ACD)$ be such that $a \in \Sigma^*(\{b\})$, and let n^T be the last move in t such that $p(n) \in p(ACD)$, i.e. $t = t_1 n^T t_2$ with all moves in t_2 having polarities from $\{O_L P_M, P_L O_M\}$. Then, $t \upharpoonright_\gamma ACD = t_1' n^{T'}$ and $b \in \mathrm{dom}(T')$. We claim that then $\Sigma(b) = T(b)$ and therefore $a \in T'^*(\{b\})$, a contradiction to $a \notin \nu(t \upharpoonright_\gamma ACD)$. Note first that $b \in \nu(t \upharpoonright_\gamma ACD) \setminus \nu(t \upharpoonright_\gamma AC)$ so, by IH, $b \notin \nu(t \upharpoonright_\gamma ABC)$. Then, by Laird's conditions for AB and BC, $T(b)$ is copied throughout t_2 and, thus, $T(b) = \Sigma(b)$. Finally, if $p(m') \in OOO \cup OOP$ then work dually as above.

For 3, we show (b), and (a) is shown similarly. Suppose WLOG that $s = s'm^{\Sigma}$ and $s \upharpoonright_\gamma ACD = s''m^{\Sigma'}$, so $a \in \nu(m^{\Sigma'}) \setminus \nu(s'')$. If $a \notin \nu(s \upharpoonright_\gamma AC)$ then, by item 2 and Lemma A.3, $a \in P(s \upharpoonright_\gamma CD) \cup O(s \upharpoonright_\gamma AD)$, which implies

$a \in \nu(s'')$, a contradiction.

For 4, we first show the equalities, starting from (b). Suppose a is introduced in $s \upharpoonright_\gamma ACD$ by a move m. If $p_{CD}(m) = P$ or $p_{AD}(m) = O$ then, by item 1, $a \in P(s \upharpoonright_\gamma CD) \cup O(s \upharpoonright_\gamma AD)$. Otherwise, it must be the case that $p_{AC}(m) = P$, so $a \in P(s \upharpoonright_\gamma AC)$ by item 3. Thus, $\nu(s \upharpoonright_\gamma ACD) = P(s \upharpoonright_\gamma AC) \cup P(s \upharpoonright_\gamma CD) \cup O(s \upharpoonright_\gamma AD)$ and, similarly, $\nu(s \upharpoonright_\gamma ABC) = P(s \upharpoonright_\gamma AB) \cup P(s \upharpoonright_\gamma BC) \cup O(s \upharpoonright_\gamma AC)$.

Finally, $P(s \upharpoonright_\gamma CD) \cap O(s \upharpoonright_\gamma AD) = \emptyset$ is by definition, while $(P(s \upharpoonright_\gamma AC) \cup P(s \upharpoonright_\gamma CD)) \cap O(s \upharpoonright_\gamma AD) = \emptyset$ follows from $P(s \upharpoonright_\gamma AC) \subseteq \nu(s \upharpoonright_\gamma ABC) \setminus O(s \upharpoonright_\gamma AC) \subseteq P(s \upharpoonright_\gamma AB) \cup P(s \upharpoonright_\gamma BC)$ and Lemma A.3. Similarly for (a). □

We can now show the following.

Lemma A.5. *If $s \in Int(ABCD)$ then $s \upharpoonright_\gamma ABC \in Int(ABC)$ and $s \upharpoonright_\gamma ACD \in Int(ACD)$.*

Proof. We show that $s' = s \upharpoonright_\gamma ACD \in Int(ACD)$, and the case for $s \upharpoonright_\gamma ABC$ is shown similarly. First, using the polarity p of s, we define a polarity function p' for s'. For each $X, Y \in \{O, P\}$, we set:

$$\pi_{ACD}(X_L) = X_L \qquad \pi_{ACD}(X_M Y_R) = X_L Y_R$$
$$\pi_{ACD}(X_L Y_R) = X_L Y_R \qquad \pi_{ACD}(Y_R) = Y_R$$

and define $p'(m) = \pi_{ACD}(p(m))$. We next verify that s' is alternating. Let m be a move in s'.

- If $m \in M_A$ then $p(m) = O_L$ so $p'(m) = O_L$.
- If $m \in M_C$ then $p(m) = P_M O_R$ so $p'(m) = P_L O_R$.
- If $m \in M_D$ then $p(m) = P_R$ so $p'(m) = P_R$.

Now, let mn be consecutive in s'. Then, $\underline{s} = \cdots mtn \cdots$ for some t containing moves with polarities in $\{O_L P_M, P_L O_M\}$. By the interaction diagram, one of the following must be the case.

- $p(n) \in \overline{p(m)}$. In this case, observe that, for all $X, Y \in p(ACD)$, if $X \in \overline{Y}$ then $\pi_{ACD}(X) \in \overline{\pi_{ACD}(Y)}$.
- $p(n) = P_M O_R$ and $p(m) \in \{O_L, O_L P_R\}$. Then, $p'(n) = P_L O_R$ and $\overline{p'(m)} = PO$.
- $p(m) = O_M P_R$ and $p(n) \in \{P_L, P_L O_R\}$. Then, $p'(n) = O_L P_R$ and $\overline{p'(m)} = OP$.

In every case, $p'(n) \in \overline{p'(m)}$.

Well-classing, projecting and well-returning conditions are directly inherited from s, while frugality is ensured by application of γ.

For well-calling, let $tm^\Sigma \sqsubseteq s'$ with $m = \mathrm{call}\, a.\mathsf{m}(\vec{v})$, and let n^T be introducing a in s. By well-calling for s, we have that $p(n) \in \overline{p(m)}$. If $p_{CD}(n) = P$ or $p_{AD}(n) = O$ then n introduces a in s' and, as above, $p'(n) \in \overline{p'(m)}$. Suppose now $p_{AB}(n) = P$. Then, $a \in P(s \upharpoonright_\gamma AB)$ so $a \notin P(s \upharpoonright_\gamma CD) \cup O(s \upharpoonright_\gamma AD)$, and therefore $a \in P(s \upharpoonright_\gamma AC)$ by Lemma A.4 (4b). The latter implies that $o_{ACD}(a) \in \{P_L, P_L O_R\}$. Since $p_{AB}(n) = P$ and $p(n) \in \overline{p(m)}$, we have that $p(m) \in \{O_L, O_L P_R\}$ and therefore $\overline{p'(m)} = \{P_L, P_L O_R\}$. We work similarly for the case of $p_{BC}(n) = P$.

Laird's disjointness conditions follow from the definition of $Int(ABCD)$ and Lemma A.4 (4b). Finally, let $t \sqsubseteq s'$ end in $m^\Sigma n^T$ and $(a) \in \mathrm{dom}(T)$. If $p_{CD}(n) = P$ and $a \notin \nu(s \upharpoonright_\gamma CD)$ (or $p_{AD}(n) = O$ and $a \notin \nu(s \upharpoonright_\gamma AD)$) then, by definition of $Int(ABCD)$, we have that $\Sigma(a) = T(a)$. If $p_{AC}(n) = P$ and $a \notin \nu(s \upharpoonright_\gamma AC)$ then, by Lemma A.4 (2), $a \notin \nu(s \upharpoonright_\gamma AB) \cup \nu(s \upharpoonright_\gamma BC)$ and $p_{AB}(n) = P$ or $p_{BC}(n) = P$ so $\Sigma(a) = T(a)$. □

Conversely, we want to show that interaction sequences in ABD and BCD with common projection in BD can be themselves obtained as projections of interaction sequences in $ABCD$. We let

$$\begin{aligned} p(ABD) &= p(AB) \cup p(BD) \cup p(AD) \\ &= \{X_L, X_L\overline{X}_M, X_L\overline{X}_R, X_R \mid X \in \{O, P\}\} \\ p(BCD) &= p(BC) \cup p(CD) \cup p(BD) \\ &= \{X_L\overline{X}_M, X_M\overline{X}_R, X_L\overline{X}_R, X_R \mid X \in \{O, P\}\} \end{aligned}$$

and, for each $s \in Int(ABCD)$ and $X \in \{ABD, BCD\}$, we define $s \upharpoonright X$ to be the subsequence of X comprising those of its moves with polarities in X.

Lemma A.6. *Let $s \in Int(ABD)$ and $t \in Int(BCD)$ with $s \upharpoonright_\gamma BD = t \upharpoonright_\gamma BD$ and $v(s) \cap v(t) \subseteq v(s \upharpoonright_\gamma BD)$. Then, there is $u \in Int(ABCD)$ such that $u \upharpoonright_\gamma ABD = s$ and $u \upharpoonright_\gamma BCD = t$.*

Proof. We do induction on $|s| + |t|$. Suppose $s = s'm^\Sigma$. If $p_{ABD}(m) = P_L$ then, by IH, there is $u \in Int(ABCD)$ such that $s' = u \upharpoonright_\gamma ABD$ and $t = u \upharpoonright_\gamma BCD$. We claim that u and s' end in the same move. Indeed, let $s' = s''n^{T'}$ and $u'n^T \sqsubseteq u$. By alternation of s, we have $p_{ABD}(n) \in \{O_L, O_LP_R\}$, hence $p_{ABCD}(n) \in \{O_L, O_LP_M, O_LP_R\}$. If n is not the last move in u then the move following it in u, say n', will have polarity in $\{P_L, P_LO_M, P_LO_R\}$. But then $p(n') \in p(ABD)$, contradicting $u \upharpoonright_\gamma ABD = s'$. We can now see that $um^{\Sigma'} \in Int(ABCD)$, with $um^{\Sigma'} \upharpoonright_\gamma ABD = s$ and $um^{\Sigma'} \upharpoonright_\gamma BCD = t$, where

$$\Sigma' = \Sigma \cup \{(a, T(a)) \mid a \in v(um^\Sigma) \setminus v(s)\}$$

is T updated with the values of Σ. Note that if m introduces some name a in $um^{\Sigma'} \upharpoonright_\gamma AB$ then m introduces a in s and therefore, by hypothesis, $a \notin v(t)$. This ensures that Laird's disjointness conditions are satisfied, while the definition of Σ' ensures the value-copying conditions.

If $p_{ABD}(m) = O_L$ then, by IH, there is $u \in Int(ABCD)$ such that $s' = u \upharpoonright_\gamma ABD$ and $t = u \upharpoonright_\gamma BCD$ and, working as above, we can show that s' and u end in the same move and construct the required $um^{\Sigma'} \in Int(ABCD)$.

The cases of $t = tm^\Sigma$ with $p_{BCD}(m) \in \{O_LP_R, P_LO_R\}$ are dealt with similarly to the ones above.

Finally, let $s = s'm^\Sigma$ and $t = t'm^T$ with $p_{ABD}(m) \in \{O_LP_R, P_LO_R, O_R, P_R\}$ and $p_{BCD}(m) \in \{O_L, P_L, O_R, P_R\}$. Note that $s' \upharpoonright_\gamma BD = t' \upharpoonright_\gamma BD$ by hypothesis. We claim that $v(s') \cap v(t') \subseteq v(s' \upharpoonright_\gamma BD)$. Indeed, if $a \in v(s') \cap v(t')$ then, by hypothesis, $a \in v(s \upharpoonright_\gamma BD)$. Thus, if $a \notin v(s' \upharpoonright_\gamma BD)$ then m would be introducing a in $s \upharpoonright_\gamma BD$ so, by Lemma 3.22, it would be also introducing a in either s or t. Hence, we can apply the IH on s', t' and obtain $u \in Int(ABCD)$ such that $u \upharpoonright_\gamma ABD = s'$ and $u \upharpoonright_\gamma BCD = t'$. We can now form $um^{\Sigma \cup T} \in Int(ABCD)$, which projects as s and t. □

We can now prove associativity of strategy composition.

Proposition 3.37. For all $\rho : A \to B$, $\sigma : B \to C$ and $\tau : C \to D$, $(\rho; \sigma); \tau = \rho; (\sigma; \tau)$.

Proof. The lemmata we produced above are for proving the right-to-left inclusion, which is what we show here. The other inclusion is proved symmetrically. So let $s \in \rho; (\sigma; \tau)$. We have that $s = v \upharpoonright_\gamma AD$ for some $v \in \rho \| (\sigma; \tau)$, while $v \upharpoonright_\gamma BD = w \upharpoonright_\gamma BD$ for some $w \in \sigma \| \tau$. By equivariance of σ, τ, we can assume that $v(v) \cap v(w) \subseteq v(w \upharpoonright_\gamma BD)$. Thus, by Lemma A.6, there is a $u \in Int(ABCD)$ with $u \upharpoonright_\gamma ABD = v$ and $u \upharpoonright_\gamma BCD = w$. From Lemma A.5 we get $u \upharpoonright_\gamma ABC \in Int(ABC)$, so in particular $u \upharpoonright_\gamma ABC \in \rho \| \sigma$ and $u \upharpoonright_\gamma AC \in \rho; \sigma$. By the same lemma, $u \upharpoonright_\gamma ACD \in Int(ABCD)$, so in particular $u \upharpoonright_\gamma ACD \in (\rho; \sigma) \| \tau$. Thus, $s = u \upharpoonright_\gamma AD \in (\rho; \sigma); \tau$. □