Nominal Game Semantics.

Murawski, AS; Tzevelekos, N

Foundations and Trends® in Programming Languages
Vol. 2, No. 4 (2015) 191–269
© 2016 A. S. Murawski and N. Tzevelekos
DOI: 10.1561/2500000017

Nominal Game Semantics

Andrzej S. Murawski
University of Warwick

Nikos Tzevelekos
Queen Mary University of London

Contents

Abstract

These tutorial notes present *nominal game semantics*, a denotational technique for modelling higher-order programs.

A. S. Murawski and N. Tzevelekos. *Nominal Game Semantics*. Foundations and Trends® in Programming Languages, vol. 2, no. 4, pp. 191–269, 2015.
DOI: 10.1561/2500000017.

1

Introduction

Game semantics is a branch of denotational semantics that uses the metaphor of game playing to model computation. The game models of PCF [5, 21, 35] constructed in the 1990s have led to an unprecedented series of *full abstraction* results for a range of functional/imperative programming languages. A result of this kind characterises contextual equivalence between terms semantically, i.e. equality of denotations coincides with the fact that terms can be used interchangeably in any context. As such, full abstraction results can be said to capture the computational essence of programs.

The fully abstract game models from the 1990s covered a plethora of computational effects, contributing to a general picture referred to as *Abramsky's cube* [8]: by selectively weakening the combinatorial conditions on plays of the games, one was able to increase the expressivity of the games and capture desired computational effects.

Although those works successfully constructed models of state [7, 6, 4, 9], the techniques used to interpret reference types did not make them fully compatible with what constitutes the norm in languages such as ML or Java. In particular, references were modelled through a form of indirection originating in the work of Reynolds [39], namely by assuming that $\mathsf{ref}\,\theta = (\theta \to \mathsf{unit}) \times (\mathsf{unit} \to \theta)$. The approach led to identification of references with pairs of arbitrary reading $(\mathsf{unit} \to \theta)$ and writing $(\theta \to \mathsf{unit})$ functions. While this view is elegant and certainly comprises the range of behaviours corresponding to references, it does not enforce a relationship between reading and writing, as witnessed by the presence of the product type. This causes a significant strengthening of the semantic universe used for modelling references and, consequently, many desirable equivalences are not satisfied in the model. For example, the interpretation of $(x := 0; x := 1)$ is different from that of $x := 1$ and, similarly, for $x := !x$ and $()$. We list the interpretations below using the terminology of [6].

$x := 0; x := 1$	$\mathsf{run}\ \mathsf{write}(0)\ \mathsf{ok}\ \mathsf{write}(1)\ \mathsf{ok}\ \mathsf{done}$
$x := 1$	$\mathsf{run}\ \mathsf{write}(1)\ \mathsf{ok}\ \mathsf{done}$
$x := !x$	$\mathsf{run}\ \mathsf{read}\ i\ \mathsf{write}(i)\ \mathsf{ok}\ \mathsf{done}$
$()$	$\mathsf{run}\ \mathsf{done}$

Thus, for the first term, the semantic translation treats both updates as observable events and therefore both are recorded in the game play.[1] This immediately distinguishes semantically the first term from the second one, for which only a single update is recorded. On the other hand, the translation of the third term is more verbose, registering calls to both the read and write methods of x, even though the computational content of the term is in fact that of the skip command $()$ in the modelled language.

To prove full abstraction in this setting, it is then necessary to enrich the syntax with terms that will populate the whole semantic space of references. Such terms are often referred to as *bad variables*, because they are objects of reference type equipped with potentially unrelated reading and writing methods. These terms, if used by the context, can distinguish the pairs of terms discussed above. For instance, a context that instantiates x to a bad variable with divergent reading and writing capabilities will be able to distinguish $x := !x$ from $()$. Nonetheless, that solution is not entirely satisfactory as the bad-variable construct breaks standard expectations for references. Moreover, one would hope to be

[1] In effect, $\mathsf{write}(0)$ and $\mathsf{write}(1)$ represent calls to the write method of reference x, while ok's correspond to returns of that method.

able to carve the model in such a way that it matches the modelled language, instead of extending the language to match the model.

The bad-variable problem can be seen as the result of modelling a generative effect (the creation and use of references) by equating it with the product of its observable handling methods.[2] Nominal game semantics is a recent branch of game semantics that makes it possible to model generative effects in a more direct manner, by incorporating *names* (drawn from an infinite set) as atomic objects in its constructions. In particular, it can model reference types without bad variables by using names to interpret references. The names are embedded in moves and also feature in stores that are carried by moves in the game. Intuitively, the stores correspond to the observable part of program memory. For example, the two pairs of terms discussed above can be modelled by the following two nominal plays respectively.

$$a^{\{(a,i)\}}\ \star^{\{(a,1)\}} \qquad\qquad a^{\{(a,i)\}}\ \star^{\{(a,i)\}}$$

Here a stands for an arbitrary name, i.e. the collection of plays is stable with respect to name permutations. Formally, the objects studied in nominal game semantics (moves, plays, strategies) live in nominal sets [12].

Since 2004, the nominal approach has led to a series of new full abstraction results. The languages covered are the ν-calculus [3] (purely functional language with names), $\lambda\nu$ [25] (a higher-order language with storage of untyped names), Reduced ML [31] (a higher-order language with integer-valued storage), RefML [32] (higher-order references) and Middleweight Java [34]. Nominal game semantics has also been used to model Concurrent ML [26] and exceptions [34].

Structure of the tutorial

Our tutorial is meant to complement existing introductory literature to game semantics [1, 8, 19, 16], which highlighted the then new structural components necessary to model higher-order computation, e.g. arenas,

[2]Similar issues arise when modelling exceptions in this way, i.e. as products of raise/handle functions [24].

justification pointers, innocence. In contrast, we shall particularly focus on explaining the nominal content of our games. We hope the material has been written in a way that will make it accessible to readers familiar with standard denotational semantics and types, e.g. [10, 17, 40].

We begin our exposition with Chapter 2 covering the basics of nominal sets. In Chapter 3 we introduce the programming language of study, called GroundML. GroundML is a higher-order language with references capable of storing integers, reference to integers, references to references to integers and so on. In Chapter 5 we shall present the game model of GroundML in full detail. Before that, in Chapter 4, we focus on a fragment of GroundML that, for the sake of simplicity, features only integer-valued references and restricted higher-order types. Because ToyML is simpler, we can give a more direct and elementary presentation of its game semantics, which we hope will help the reader to make a transition to the full-blown model of the following section.

2

Elements of Nominal Set Theory

In this chapter we give a brief overview of notions from nominal sets which we shall be using in the sequel. What we present here is a tiny fragment of the theory — the reader is referred to [12, 13, 36] for thorough expositions.

Nominal sets provide a robust way to deal with computational entities (such as computations, traces, models, etc.) that involve *names*. For instance, in these notes, names will be used to model *references*, i.e. allocated mutable variables in programming languages with local state. The essence of using names is that:

- there is an infinite, but countable, amount of names;
- therefore, given a finite computation, we can always generate a *fresh name*, i.e. one that does not appear inside it;
- this fresh-name generation is non-deterministic: any of the cofinitely many fresh names can be generated at any point;
- a computation involving names should be closed under name permutation: if a different choice of fresh names is made, that should not affect the non-nominal component of the computation.

Above, when we mention computations we refer to objects coming either from the operational semantics of the modelled language or its denotational semantics.

Nominal sets

Let us fix a countably infinite set $\mathbb{A}$, the set of ***names***, the elements of which we denote by a, b, c and variants. Names are partitioned by

$$\mathbb{A} = \biguplus_{i\in\omega} \mathbb{A}_i$$

where each $\mathbb{A}_i$ is itself countably infinite. The indexing over natural numbers here is a comprehensive choice that captures specific scenarios where one caters for a countable collection of disjoint sets of names. For our needs, the indexing will be in fact over ground types of our language.

We write $\Pi(\mathbb{A})$ for the group of finite permutations of $\mathbb{A}$ which are component-preserving:

$$\Pi(\mathbb{A}) = \{\, \pi : \mathbb{A} \to \mathbb{A} \mid \mathsf{supp}(\pi) \text{ is finite} \land \forall i \in \omega, a \in \mathbb{A}_i.\, \pi(a) \in \mathbb{A}_i \,\}$$

where $\mathsf{supp}(\pi) = \{\, a \in \mathbb{A} \mid \pi(a) \neq a \,\}$. Recall that a group action of $\Pi(\mathbb{A})$ on some set X is a function

$$\_ \cdot \_ \ :\ \Pi(\mathbb{A}) \times X \to X$$

such that, for all $x \in X$ and $\pi, \pi' \in \Pi(\mathbb{A})$, $\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x$ and $\mathsf{id} \cdot x = x$, where id is the identity permutation.

Definition 2.1. A ***nominal set*** X is a set $|X|$ (usually written X) equipped with a group action of $\Pi(\mathbb{A})$ and such that all elements of X have *finite support*. That is, for each $x \in X$ there exists a finite set $\mathrm{U} \subseteq \mathbb{A}$ such that, for all $\pi \in \Pi(\mathbb{A})$, $(\forall a \in \mathrm{U}.\, \pi(a) = a) \implies \pi \cdot x = x$.

Finite support is closed under intersection, hence each element x of a nominal set has a least support, which we call ***the support of*** x:

$$\nu(x) \;=\; \bigcap\{\, \mathrm{U} \subseteq_{\mathrm{fin}} \mathbb{A} \mid (\forall a \in \mathrm{U}.\, \pi(a) = a) \implies \pi \cdot x = x \,\}\,.$$

Intuitively, $\nu(x)$ is the set of names "involved" in x. Accordingly, we say that a *is fresh for* x if $a \notin \nu(x)$.

Trivially, every set X can be seen as a nominal set by setting $\pi \cdot x = x$ for all $x \in X$. On the other hand, $\mathbb{A}$ is a nominal set by taking $\pi \cdot a = \pi(a)$, for each π and a. In the same fashion, each set $\mathbb{A}_i$ is a nominal set. The set $\mathcal{P}_{\mathrm{fin}}(\mathbb{A})$ of finite sets of names is also a nominal set, with action $\pi \cdot \mathrm{U} = \{\pi(a) \mid a \in \mathrm{U}\}$ for each permutation π and finite $\mathrm{U} \subseteq \mathbb{A}$. Moreover, $\nu(\mathrm{U}) = \mathrm{U}$. More interestingly, if X and Y are nominal sets then so are:

- their cartesian product $X \times Y$, with permutations acting componentwise: $\pi \cdot (x, y) = (\pi \cdot x, \pi \cdot y)$;
- their disjoint union $X \uplus Y$, with permutations acting as in X/Y: $\pi \cdot_{X \uplus Y} x = \pi \cdot_X x$ if $x \in X$, and $\pi \cdot_{X \uplus Y} x = \pi \cdot_Y x$ otherwise;
- the set X^* of finite sequences of elements of X, with permutations acting elementwise: $\pi \cdot (x_1 \ldots x_n) = (\pi \cdot x_1) \cdots (\pi \cdot x_n)$.

For example, the set $\mathbb{A}^*$ of finite sequences of atoms is a nominal set with $\pi \cdot (a_1 \ldots a_n) = (\pi \cdot a_1) \ldots (\pi \cdot a_n) = \pi(a_1) \ldots \pi(a_n)$ for each $\pi \in \Pi(\mathbb{A})$ and $a_1, \cdots, a_n \in \mathbb{A}$. Moreover, $\nu(a_1 \ldots a_n)$ contains precisely the names $a_1, \cdots, a_n$ (modulo repetitions).

We call $X' \subseteq X$ a *nominal subset* of X if X' is closed under permutations, these acting as on X. Accordingly, we say that $R \subseteq X \times Y$ is a ***nominal relation*** if R a nominal subset of $X \times Y$. Concretely, this means that:

$$(x, y) \in R \iff (\pi \cdot x, \pi \cdot y) \in R$$

for all permutations π. A ***nominal function*** is a function which is also a nominal relation, that is,

$$f(\pi \cdot x) = \pi \cdot f(x)$$

for all $x \in X$ and $\pi \in \Pi(\mathbb{A})$. As a consequence, $\nu(f(x)) \subseteq \nu(x)$.

Given a nominal set X, we can define an equivalence relation $\sim_X$ on its elements:

$$x \sim_X y \iff \exists \pi \in \Pi(\mathbb{A}).\, x = \pi \cdot y$$

specifying that two members of X are equal up to permutation. We shall denote $\sim_X$ by $\sim$ for simplicity. For each $x \in X$ we form its ***orbit*** under the permutation action by:

$$[x] = [x]_{/\sim} = \{y \in X \mid y \sim x\}$$

that is, $[x] = \{\pi \cdot x \mid \pi \in \Pi(\mathbb{A})\}$. Taking the orbit $[x]$ of x can be seen as blurring the specific choice of names within x while retaining its underlying structure. Note that $[x] \subseteq X$. For example, for any sequence $a_1 \dots a_n \in \mathbb{A}^*$, its orbit

$$[a_1 \dots a_n] = \{\pi \cdot (a_1 \dots a_n) \mid \pi \in \Pi(\mathbb{A})\}$$

contains all sequences of atoms $a'_1 \dots a'_n$ such that, for all $i, j \in \{1, ..., n\}$ and $k \in \omega$, we have $a'_i = a'_j$ if, and only if, $a_i = a_j$; and $a'_i \in \mathbb{A}_k$ if, and only if, $a_i \in \mathbb{A}_k$. On the other hand, for each i, in the nominal set $\mathbb{A}_i \times \mathbb{A}_i$ we can form just two orbits, namely $[(a, b)]$ and $[(a, a)]$, for some $a \neq b \in \mathbb{A}_i$. The former contains all pairs of distinct names in $\mathbb{A}_i$, while the latter all pairs made of the same name.

In game semantics a particular strengthening of the notion of support has turned out to be necessary to guarantee correct behaviour under strategy composition.[1] In particular, we say that an element x of nominal set X has ***strong support*** if, for all permutations π,

$$\pi \cdot x = x \implies \forall a \in \nu(x).\, \pi(a) = a$$

that is, the converse condition of that of (simple) support also holds. We call X a ***strong nominal set*** if all its elements have strong support.

The intuition behind strong support is that all names in the support of an element are distinguishable by some given order.[2] For example, the nominal set $\mathbb{A}^*$ is strong, whereas $\mathcal{P}_{\text{fin}}(\mathbb{A})$ is not. Note that a nominal subset of a strong nominal set is itself strong and, moreover, if X and Y are strong nominal sets then so are $X \times Y$, $X \uplus Y$ and X^*.

[1]See [42] for motivation and a detailed explanation of its significance. The same notion was defined by Schöpp [41] for nominal sets with an *essentially simple* action.

[2]Put otherwise, there are no *symmetries* in $\nu(x)$. A more operational reading is that, given the output of a computation involving names, we can distinguish between any two distinct names that were produced in the same computational step.

3

GroundML

Since these notes are intended as an introductory tutorial, our main focus will be a higher-order language with local state of a simple kind, namely state of ground type.[1] It is a functional language with the additional imperative effect of local store. The store is *full ground*, in the following sense: we can store integer values, we can also store references which themselves store integer values, and so on. We call this language GroundML.

3.1 Syntax

We start off with introducing the types of our language. Note that reference types are available for each ground type ζ. A reference type $\mathsf{ref}\,\zeta$ specifies locations storing values of type ζ.

Definition 3.1. The ***types*** of GroundML are generated according to the

[1]We briefly discuss extensions to more expressive, and more realistic, languages in Section 6 and give references to the relevant bibliography.

following grammar,

$$\begin{aligned} \theta &::= \zeta \mid \theta \times \theta \mid \theta \to \theta \\ \zeta &::= \mathsf{unit} \mid \mathsf{int} \mid \mathsf{ref}\,\zeta \end{aligned}$$

and types of the ζ kind are called *ground*.

Let us fix a set $\mathbb{A} = \biguplus_{\zeta} \mathbb{A}_{\zeta}$ of ***location names***, which precisely correspond to names as in nominal sets (cf. Chapter 2, where $\mathbb{A} = \biguplus_{i \in \omega} \mathbb{A}_i$). That is, we assume an enumeration $\zeta_1, \zeta_2, \cdots$ of ground types so that, for each i, the sets $\mathbb{A}_{\zeta_i}$ and $\mathbb{A}_i$ coincide. Note that each location name is associated with a unique ground type, namely the type indexing its originating name set.

Definition 3.2. The syntax of ***supported terms*** is given as follows,

$$\begin{aligned} M \ ::= \ & () \mid a \mid i \mid x \mid M \oplus M \mid \mathsf{while}(M) \mid \mathsf{if}\, M\, \mathsf{then}\, M\, \mathsf{else}\, M \mid \lambda x^{\theta}.M \\ & \mid MM \mid \langle M, M \rangle \mid \pi_1 M \mid \pi_2 M \mid \mathsf{ref}(M) \mid M = M \mid\, !M \mid M := M \end{aligned}$$

where $\oplus$ ranges over a set of arithmetic operators, x and i range over variables and integers respectively, and a over elements of $\mathbb{A}$. A ***term*** is a supported term featuring no locations (i.e. with empty support).

Supported terms are typed in environments U, Γ, where U is a finite set of location names and Γ is a variable typing context. The typing rules are given in Figure 3.1.

Remark 3.3. *Why are there distinct categories for terms and supported terms?* This distinction is meant to convey the difference between a syntactic phrase that a user can write, which cannot contain location names[2], and the syntactic phrases produced when a program is executed, where locations appear as soon as a $\mathsf{ref}(...)$ operator is evaluated.

Thus, the language is best described as the call-by-value λ-calculus with products over base types ζ, augmented with the do-nothing command, integer constants, arithmetic operations, looping and reference manipulation (allocation, dereferencing, assignment).

In what follows, we often use the following shorthand notation:

[2]in the sense that the set $\mathbb{A}$ is not accessible to the programmer (think of Java or ML programs).

$$\frac{}{\mathrm{U},\Gamma \vdash () : \mathsf{unit}} \qquad \frac{i \in \mathbb{Z}}{\mathrm{U},\Gamma \vdash i : \mathsf{int}} \qquad \frac{(x:\theta) \in \Gamma}{\mathrm{U},\Gamma \vdash x : \theta} \qquad \frac{a \in \mathrm{U} \cap \mathbb{A}_\zeta}{\mathrm{U},\Gamma \vdash a : \mathsf{ref}\zeta}$$

$$\frac{\mathrm{U},\Gamma \vdash M : \mathsf{int} \quad \mathrm{U},\Gamma \vdash N_0 : \theta \quad \mathrm{U},\Gamma \vdash N_1 : \theta}{\mathrm{U},\Gamma \vdash \mathsf{if}\, M \,\mathsf{then}\, N_1 \,\mathsf{else}\, N_0 : \theta} \qquad \frac{\mathrm{U},\Gamma \vdash M : \mathsf{int}}{\mathrm{U},\Gamma \vdash \mathsf{while}(M) : \mathsf{unit}}$$

$$\frac{\mathrm{U},\Gamma \uplus \{\, x:\theta \,\} \vdash M : \theta'}{\mathrm{U},\Gamma \vdash \lambda x^\theta.M : \theta \to \theta'} \qquad \frac{\mathrm{U},\Gamma \vdash M : \theta \to \theta' \quad \mathrm{U},\Gamma \vdash N : \theta}{\mathrm{U},\Gamma \vdash MN : \theta'}$$

$$\frac{\mathrm{U},\Gamma \vdash M : \theta \quad \mathrm{U},\Gamma \vdash N : \theta'}{\mathrm{U},\Gamma \vdash \langle M,N\rangle : \theta \times \theta'} \qquad \frac{\mathrm{U},\Gamma \vdash M : \theta_1 \times \theta_2}{\mathrm{U},\Gamma \vdash \pi_i M : \theta_i}\, {\scriptstyle i\in\{1,2\}}$$

$$\frac{\mathrm{U},\Gamma \vdash M : \mathsf{int} \quad \mathrm{U},\Gamma \vdash N : \mathsf{int}}{\mathrm{U},\Gamma \vdash M \oplus N : \mathsf{int}} \qquad \frac{\mathrm{U},\Gamma \vdash M : \mathsf{ref}\zeta \quad \mathrm{U},\Gamma \vdash N : \mathsf{ref}\zeta}{\mathrm{U},\Gamma \vdash M = N : \mathsf{int}}$$

$$\frac{\mathrm{U},\Gamma \vdash M : \zeta}{\mathrm{U},\Gamma \vdash \mathsf{ref}(M) : \mathsf{ref}\zeta} \qquad \frac{\mathrm{U},\Gamma \vdash M : \mathsf{ref}\zeta}{\mathrm{U},\Gamma \vdash\, !M : \zeta} \qquad \frac{\mathrm{U},\Gamma \vdash M : \mathsf{ref}\zeta \quad \mathrm{U},\Gamma \vdash N : \zeta}{\mathrm{U},\Gamma \vdash M := N : \mathsf{unit}}$$

Figure 3.1: Typing rules of GroundML.

- $\mathsf{let}\, x = M \,\mathsf{in}\, N$ for the term $(\lambda x^\theta.N)M$;
- $M;N$ for $\mathsf{let}\, x = M \,\mathsf{in}\, N$, where x does not occur in N;
- $\lambda\_^\theta.M$ for $\lambda x^\theta.M$, where x does not occur in M.

Observe that, for any type θ, we can define some closed term $\vdash M_\theta : \theta$. Hence, we can also define:

$$\mathsf{div}_\theta \;\equiv\; \mathsf{while}(1); M_\theta,$$

which will be used as a canonical divergent term of type θ.

3.2 Operational semantics

We give an operational semantics for GroundML by means of a small-step transition relation. It relates pairs consisting of (typed) supported terms together with a representation of the state of the memory. Formally, we define the set of ***stores*** as:

$$\text{Stores} \;=\; \{\, S : \mathbb{A} \rightharpoonup (\{\,\star\,\} \cup \mathbb{Z} \cup \mathbb{A}) \mid S \text{ finite and legal} \,\}$$

where a partial function $S : \mathbb{A} \rightharpoonup (\{\star\} \cup \mathbb{Z} \cup \mathbb{A})$ is called *legal* just if $\mathsf{cod}(S) \cap \mathbb{A} \subseteq \mathsf{dom}(S)$ and for all names $a \in \mathsf{dom}(S)$:

$$\begin{aligned}(a \in \mathbb{A}_{\mathsf{unit}} \implies S(a) = \star) \wedge (a \in \mathbb{A}_{\mathsf{int}} \implies S(a) \in \mathbb{Z})&\\ \wedge\, (a \in \mathbb{A}_{\mathsf{ref}\zeta} \implies S(a) \in \mathbb{A}_{\zeta})&\end{aligned}$$

Intuitively, a store is legal if it is well typed and, moreover, all the location names that are stored in it are also part of its domain. Put otherwise, there are no locations with undefined values in a store. Note here that we use $\star$ as the unique value of type unit as $()$ can be cumbersome when mixed with other brackets. In the sequel we follow the convention that $\star$ and $()$ are aliases of one another, with the former used in stores (and in game moves), and the latter in syntactic terms.

Given a store S, a name a and some $x \in \{\star\} \cup \mathbb{Z} \cup \mathbb{A}$, we define the update $S[a \mapsto x]$ by:

$$(S[a \mapsto x])(a') = \begin{cases} S(a') & \text{if } a' \in \mathsf{dom}(S) \setminus \{a\} \\ x & \text{if } a' = a \\ \text{undefined} & \text{otherwise} \end{cases}$$

with the proviso that the latter is still a store. We extend this notation to:

$$S[T] = S[a_1 \mapsto T(a_1)] \cdots [a_n \mapsto T(a_n)]$$

for any stores S and T with $\mathsf{dom}(T) = \{a_1, \cdots, a_n\}$. Finally, a ***configuration*** is a pair (M, S) of a supported term and a store such that $\mathsf{dom}(S)$ contains all the names which appear in M.

The transition relation is produced by the rules in Figure 3.2. The context rule uses *evaluation contexts*, which are given by the syntax:

$$\begin{aligned} E \ ::= \ & [\,] \mid E \oplus M \mid V \oplus E \mid \mathsf{if}\, E\, \mathsf{then}\, M\, \mathsf{else}\, M \mid EM \mid VE \mid \langle E, M\rangle \\ & \mid \langle V, E\rangle \mid \pi_i E \mid \mathsf{ref}(E) \mid E = M \mid x = E \mid\, !E \mid E := M \mid V := E \end{aligned}$$

where $[\,]$ denotes the hole of the context and M ranges over supported terms. On the other hand, V ranges over ***values***, which are given by:

$$V \ ::= \ () \mid i \mid x \mid a \mid \langle V, V\rangle \mid \lambda x^{\theta}.M$$

Note in particular that values are supported terms. For any term $\vdash M : \mathsf{unit}$, we write $M \Downarrow$ if $(\emptyset, M) \twoheadrightarrow (S, ())$, for some store S.

$$
\begin{array}{rcll}
(i \oplus j, S) & \longrightarrow & (k, S) & (k = i \oplus j) \\
((\lambda x.M)V, S) & \longrightarrow & (M[V/x], S) & \\
(\pi_1\langle V_1, V_2\rangle, S) & \longrightarrow & (V_1, S) & \\
(\pi_2\langle V_1, V_2\rangle, S) & \longrightarrow & (V_2, S) & \\
(\mathsf{if}\, 0\, \mathsf{then}\, M\, \mathsf{else}\, M', S) & \longrightarrow & (M', S) & \\
(\mathsf{if}\, i\, \mathsf{then}\, M\, \mathsf{else}\, M', S) & \longrightarrow & (M, S) & (i > 0) \\
(\mathsf{while}(M), S) & \longrightarrow & (\mathsf{if}\, M\, \mathsf{then}\, \mathsf{while}(M)\, \mathsf{else}\, (), S) & \\
(a = b, S) & \longrightarrow & (0, S) & (a \neq b) \\
(a = a, S) & \longrightarrow & (1, S) & \\
(!a, S) & \longrightarrow & (S(a), S) & \\
(a := V, S) & \longrightarrow & ((), S[a \mapsto V]) & \\
(\mathsf{ref}(V), S) & \longrightarrow & (a', S[a' \mapsto V]) & (a' \notin \mathsf{dom}(S))
\end{array}
$$

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

Figure 3.2: Operational semantics of GroundML.

Next we take a closer look at some examples of GroundML terms and their behaviour.

Example 3.4 (Name generators). Consider the following (closed) terms that generate integer references.

$$
\begin{aligned}
\mathsf{gen} \equiv{}& \lambda z^{\mathsf{int}}.\, \mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\, (x := z; x) \;:\; \mathsf{int} \to \mathsf{ref\ int} \\
\mathsf{gen}' \equiv{}& \mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\, \lambda z^{\mathsf{int}}.(x := z; x) \;:\; \mathsf{int} \to \mathsf{ref\ int}
\end{aligned}
$$

The two terms differ in one crucial aspect: while gen returns a fresh reference name each time it is called, gen′ always returns the same name (which is indeed fresh the first time it is called).

Example 3.5 (Name channels). The fact that names can be stored in GroundML enables us to simulate the behaviour of channels, in the style of the π-calculus. Consider the following term trmit.

$$
\begin{aligned}
&f : \mathsf{ref\ ref\ int} \to \mathsf{unit}, g : \mathsf{unit} \to \mathsf{ref\ ref\ int} \vdash \\
&\quad \mathsf{let}\, c_1 = \mathsf{ref}(\mathsf{ref}(0))\, \mathsf{in}\, (f c_1; \mathsf{let}\, c_2 = g()\, \mathsf{in}\, \lambda\_^{\mathsf{unit}}.\, c_2 := !c_1) : \mathsf{unit} \to \mathsf{unit}
\end{aligned}
$$

The term creates an input channel c_1 and passes it to its context (via $f c_1$); then receives an output channel c_2 from the context; and finally returns a process that listens at c_1 and transmits back to c_2.

The operational semantics allows us to evaluate terms to values. For closed terms of ground type, this will be sufficient to reveal all there is to their behaviour. However, the case of higher-order terms is much more complicated. In order to understand their computational potential, not only does one need to evaluate the term, but also consider the behaviour of the resultant value in future interactions. It is worth remarking that these subsequent uses cannot be restricted to single applications, because the behaviour of GroundML terms may evolve over time and different results can be returned for the same arguments if a function is applied to them repeatedly. This highlights the challenges inherent in analysing higher-order programs with state. In order to compare terms formally, one tests their behaviour in arbitrary contexts whose shape is given below.

$$\begin{aligned} C \;::=\;& [\,] \mid \mathsf{if}\, C\, \mathsf{then}\, M\, \mathsf{else}\, M \mid \mathsf{if}\, M\, \mathsf{then}\, C\, \mathsf{else}\, M \mid \mathsf{if}\, M\, \mathsf{then}\, M\, \mathsf{else}\, C \\ & \mid \mathsf{while}(C) \mid \lambda x^{\theta}.C \mid MC \mid CM \mid \langle C, M\rangle \mid \langle M, C\rangle \mid \pi_i C \mid C \oplus M \\ & \mid M \oplus C \mid C = M \mid M = C \mid \mathsf{ref}(C) \mid\, !C \mid C := M \mid M := C \end{aligned}$$

Contexts are also used to define what it means for the terms to be equivalent.

Definition 3.6. We say that the term-in-context $\Gamma \vdash M_1 : \theta$ ***approximates*** $\Gamma \vdash M_2 : \theta$ (written $\Gamma \vdash M_1 \sqsubseteq M_2$) if $C[M_1]\Downarrow$ implies $C[M_2]\Downarrow$ for any context C such that $\vdash C[M_1], C[M_2] : \mathsf{unit}$.

Two terms-in-context are ***equivalent*** if one approximates the other (written $\Gamma \vdash M_1 \cong M_2$).

For instance, the name generators of Example 3.4 are not equivalent, as they can be distinguished by any context that calls the generator twice and compares the results. For instance, setting

$$C \;\equiv\; (\lambda f^{\mathsf{int} \to \mathsf{ref\,int}}.\, \mathsf{if}\, (f0 = f0)\, \mathsf{then}\, ()\, \mathsf{else}\, \mathsf{div})\, [\,]$$

we obtain $C[\mathsf{gen}] \not\Downarrow$ and $C[\mathsf{gen}'] \Downarrow$. On the other hand, proving that two terms are equivalent is trickier since picking a specific context (or a bounded class of them) is not enough: rather, one needs to prove equi-termination in every enclosing context.

Game semantics sets out to provide a semantic characterisation of equivalence. We shall see that, for all terms $\Gamma \vdash M, N : \theta$:

$$\Gamma \vdash M \cong N \iff \mathsf{comp}([\![\Gamma \vdash M]\!]) = \mathsf{comp}([\![\Gamma \vdash N]\!])$$

that is, to show $\Gamma \vdash M \cong N$, it will suffice to calculate the game-semantic denotations of terms and compare them for equality on "complete" plays.

Example 3.7 (Equivalences)**.** Terms in GroundML can create reference names that are fresh and private (cannot be guessed by the environment). Next we consider three equivalences that explore these capabilities. Our first equivalence is between the terms:

$$\begin{aligned} M_1 &\equiv \mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\, \lambda y^{\mathsf{ref\,int}}.\, x = y \ : \ \mathsf{ref\,int} \to \mathsf{int}, \\ M_2 &\equiv \lambda y^{\mathsf{ref\,int}}.\, 0 \ : \ \mathsf{ref\,int} \to \mathsf{int}. \end{aligned}$$

In the former case, y has no chance of being equal to x as the latter is never exposed outside of M_1.

In the next example, the name x is wrapped into a function that is passed to the environment. However, the environment will still be unable to discover it, because it is protected by the fact that the value of c is 0 when the environment has access to the function:

$$\begin{aligned} M_3 &\equiv \mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\, \mathsf{let}\, c = \mathsf{ref}(0)\, \mathsf{in} \\ &\qquad f(\lambda\_.\, \mathsf{if}\, !c = 0\, \mathsf{then}\, \mathsf{div}\, \mathsf{else}\, x); c := 1; \lambda y^{\mathsf{ref\,int}}.\, x = y \\ M_4 &\equiv f(\lambda\_.\, \mathsf{div}); \lambda y^{\mathsf{ref\,int}}.\, 0 \end{aligned}$$

with types $f : (\mathsf{unit} \to \mathsf{ref\,int}) \to \mathsf{unit} \vdash M_3, M_4 : \mathsf{ref\,int} \to \mathsf{int}$. Note that this equivalence would not hold if our language supported higher-order references. The environment could then store the function and delay its use until the value of c becomes 1. In the game semantic setting, the equivalence will rely on a combinatorial condition called *visibility*.

In our final example, the term M_5 retains a secret name of type ref int in a private reference of type ref ref int. If the secret is guessed

$(y = z)$ then the term diverges. Otherwise, it reveals the secret but, at the same time, replaces it with another (fresh) name:

$$\begin{aligned}
M_5 &\equiv \mathsf{let}\, x = \mathsf{ref}(\mathsf{ref}(0))\,\mathsf{in} \\
&\qquad \lambda y^{\mathsf{ref\ int}}.\, \mathsf{let}\, z = !x\, \mathsf{in}\, (\mathsf{if}\ y = z\ \mathsf{then}\ \mathsf{div}\ \mathsf{else}\ (x := \mathsf{ref}(0); z)) \\
M_6 &\equiv \lambda y^{\mathsf{ref\ int}}.\, \mathsf{ref}(0)
\end{aligned}$$

where $\vdash M_5, M_6 : \mathsf{ref\ int} \to \mathsf{ref\ int}$. Hence, the secret names produced by M_5 can be revealed but, as soon as that happens, they will be replaced with fresh ones.

4

ToyML: A First-Order Language with Integer References

To give the reader a flavour of the nominal approach, we first consider ToyML, a minimalistic language with first-order procedures, looping and integer-valued references.

4.1 Types and terms

The types of ToyML are generated according to the following grammar.

$$\theta \ ::= \ \beta \mid \beta \to \beta \qquad \beta \ ::= \ \mathsf{unit} \mid \mathsf{int} \mid \mathsf{ref\,int}$$

ToyML terms have shapes defined below.

$$\begin{aligned} M \ ::= {} & () \mid i \mid x \mid M \oplus M \mid \mathsf{if}\, M \,\mathsf{then}\, M \,\mathsf{else}\, M \mid \lambda x^{\beta}.M \mid MM \mid \\ & \mathsf{while}(M) \mid \mathsf{ref}(M) \mid M = M \mid !M \mid M := M \end{aligned}$$

The corresponding typing rules are provided in Figure 4.1. For simplicity, we only consider (unsupported) terms and, hence, there is no need to keep track of the U sets. The syntax allows for the creation of anonymous ($\lambda x^{\beta}.M$) and undefined (x) first-order functions and their application (MM).

$$\frac{}{\Gamma \vdash () : \mathsf{unit}} \qquad \frac{i \in \mathbb{Z}}{\Gamma \vdash i : \mathsf{int}} \qquad \frac{(x : \theta) \in \Gamma}{\Gamma \vdash x : \theta} \qquad \frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash \mathsf{while}(M) : \mathsf{unit}}$$

$$\frac{\Gamma \vdash M : \mathsf{int} \quad \Gamma \vdash N : \mathsf{int}}{\Gamma \vdash M \oplus N : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \mathsf{int} \qquad \Gamma \vdash N, N' : \theta}{\Gamma \vdash \mathsf{if}\, M\, \mathsf{then}\, N\, \mathsf{else}\, N' : \theta}$$

$$\frac{\Gamma \uplus \{\, x : \beta \,\} \vdash M : \beta'}{\Gamma \vdash \lambda x^{\beta}.M : \beta \to \beta'} \qquad \frac{\Gamma \vdash M : \beta \to \beta' \quad \Gamma \vdash N : \beta}{\Gamma \vdash MN : \beta'}$$

$$\frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash \mathsf{ref}(M) : \mathsf{ref\,int}} \qquad \frac{\Gamma \vdash M : \mathsf{ref\,int} \quad \Gamma \vdash N : \mathsf{ref\,int}}{\Gamma \vdash M = N : \mathsf{int}}$$

$$\frac{\Gamma \vdash M : \mathsf{ref\,int}}{\Gamma \vdash !M : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \mathsf{ref\,int} \quad \Gamma \vdash N : \mathsf{int}}{\Gamma \vdash M := N : \mathsf{unit}}$$

Figure 4.1: Typing rules of ToyML.

4.2 Concrete games

This section presents a game model of ToyML formulated in a direct way that avoids references to (too much) game-semantic jargon. We trust it will facilitate the passage to the following chapter of the tutorial, in which a full-blown game model is presented for a higher-order language. We shall focus on presenting a special kind of play, called *complete*, as these are the plays that eventually deliver full abstraction for GroundML.

Let us recall that game semantics views computation as a dialogue between the environment (Opponent, O) and the program (Proponent, P). The game model we are about to sketch is based on sequences of moves that involve *names* drawn from the infinite set $\mathbb{A}$ and which are stable under name-invariance. Put otherwise, they form nominal sets.

We begin with several auxiliary definitions before specifying how complete plays look like in our setting. For every type θ, we first define the set $\mathcal{V}_\theta$ of associated ***semantic values*** as follows:

$$\mathcal{V}_{\mathsf{unit}} = \{\, \star \,\}, \qquad \mathcal{V}_{\mathsf{int}} = \mathbb{Z}, \qquad \mathcal{V}_{\mathsf{ref\,int}} = \mathbb{A}_{\mathsf{int}}, \qquad \mathcal{V}_{\beta \to \beta'} = \{\, \dagger \,\},$$

and write $\mathcal{V}$ for the set of all semantic values. We shall use ℓ and variants to range over semantic values.

In order to interpret ToyML typing judgments of the shape $\Gamma \vdash M : \theta$ we shall rely a special set of moves, defined next.

Definition 4.1. Let $\Gamma = \{x_1 : \theta_1, \cdots, x_m : \theta_m\}$ consist of ToyML types and let θ also be a ToyML type. The set $M_{\Gamma\vdash\theta}$ of ***moves associated with*** Γ ***and*** θ is defined to be

$$M_{\Gamma\vdash\theta} = I_\Gamma \cup M_\theta \cup \bigcup\nolimits_{1\leq i\leq m} M_{x_i}$$

where:

- I_Γ is the set of *initial moves* given by

$$I_\Gamma = \{\, (\ell_1, \cdots, \ell_m) \mid \ell_i \in \mathcal{V}_{\theta_i},\, 1 \leq i \leq m \,\};$$

- M_θ is the set of *output moves* defined by
  - $M_\theta = \{\, (\mathsf{r}_\downarrow, \ell) \mid \ell \in \mathcal{V}_\theta \,\}$ if θ is a base type,
  - $M_\theta = \{\, (\mathsf{r}_\downarrow, \dagger) \,\} \cup \{\, (\mathsf{c}, \ell) \mid \ell \in \mathcal{V}_{\theta'} \,\} \cup \{\, (\mathsf{r}, \ell) \mid \ell \in \mathcal{V}_{\theta''} \,\}$ if $\theta = \theta' \to \theta''$;

- M_{x_i} is the set of *variable moves*, taken to be empty if θ_i is a base type and, if $\theta_i = \theta_i' \to \theta_i''$, equal to

$$\{\, (\mathsf{c}_{x_i}, \ell) \mid \ell \in \mathcal{V}_{\theta_i'} \,\} \cup \{\, (\mathsf{r}_{x_i}, \ell) \mid \ell \in \mathcal{V}_{\theta_i''} \,\}.$$

Moves are ranged over by m and variants. We shall use i to range over I_Γ, and we shall often write i_{x_i} for ℓ_i.

Thus, each move in $M_{\Gamma\vdash\theta}$ consists of a pair (t, ℓ) of a *tag* and a *(semantic) value.* For each function-type identifier x in Γ, we have introduced tags c_x and r_x. They can be viewed as calls and returns related to that identifier. The accompanying value in e.g. a move (c_x, ℓ) corresponds to the value that identifier is called with. Similarly, $\mathsf{r}_\downarrow$ can be taken to correspond to the fact that our modelled term was successfully evaluated, and, if θ is a function type, c and r refer respectively to calling the corresponding value and obtaining a result.

Moves are assigned ***ownership*** as follows:

- i and those with tags r_x, c belong to O (environment);
- the rest (with tags $\mathsf{r}_\downarrow, \mathsf{c}_x, \mathsf{r}$) belong to P (program).

Using ownership of moves, we can extend the definition to *names* saying that a name a is owned by the owner of the first move m in which it occurs (i.e. such that $a \in \nu(m)$). We refer to moves owned by O as *O-moves*, and similarly for P-moves, O-names and P-names.

Remark 4.2 (Duality). Note the duality between calls and returns: c is an O-move, yet r is a P-move; on the other hand, c_x is a P-move, while r_x is an O-move. This extends to $\mathrm{i}/\mathsf{r}_\downarrow$, which belong to O/P respectively (this exactly matches the intuition that $\mathsf{r}_\downarrow$ is a return to calling the modelled term with input i).

Next we define the notion of a *complete play*, which will model the observable interactions of our terms. More precisely, the complete character of the plays stands for the fact that we shall be focussing on complete computations, i.e. those where all function calls have returned.[1] As suggested by the definition above, each non-empty complete play will begin with an initial move, to be followed by moves made from tags and labels.

Definition 4.3. A ***complete sequence*** over $\Gamma \vdash \theta$ is a (possibly empty) sequence of moves $\mathrm{i}\,(t_1, \ell_1) \cdots (t_k, \ell_k)$ such that the sequence $t_1 \cdots t_k$ of tags matches the grammar:

$$X\,\mathsf{r}_\downarrow\,(\mathsf{c}\,X\,\mathsf{r})^* \quad \text{where} \quad X \;=\; \Big(\sum\nolimits_{(x\,:\,\theta' \to \theta'') \in \Gamma} (\mathsf{c}_x\,\mathsf{r}_x)\Big)^*.$$

We assume that $X\mathsf{r}_\downarrow(\mathsf{c}X\mathsf{r})^*$ degenerates to $X\mathsf{r}_\downarrow$ when c, r are not available in $M_{\Gamma\vdash\theta}$, i.e. θ is a base type.

The shape of complete such sequences can be thought of as a record of interaction. First, calls can be made to the free identifiers of function type (expression X), then a value may be reached ($\mathsf{r}_\downarrow$) and, if the type of the value is a function type, a series of calls and returns is possible with further external calls in between ($(\mathsf{c}X\mathsf{r})^*$). Note that all calls have matching returns, which is why the term *complete* is used.

[1] In the next chapter, we will also consider general plays, which need not be complete.

Remark 4.4 (Alternation). Observe that Definition 4.3 imposes an alternation of ownership inside every complete sequence: the initial move is owned by O, and is followed by a move owned by P, and so on.

The semantic interpretation of ToyML will be based on a more complicated notion of play, in which each move is associated with a finite (and legal) integer-valued store. Domains of such stores cannot be arbitrary: they must contain all names that have been used during play. To make this precise, we make the following definitions.

Definition 4.5. A ***complete play*** over $\Gamma \vdash \theta$ is a sequence $m_1^{S_1} \cdots m_k^{S_k}$ of moves-with-store satisfying the conditions below.

- $m_1 \cdots m_k$ is a complete sequence over $\Gamma \vdash \theta$.
- For any $1 \leq i \leq k$, $\mathsf{dom}(S_i) = \nu(m_1 \cdots m_i)$.

The second clause of the above definition is specific to ToyML. For GroundML we will additionally want to take into account the names that are stored. Note that the initial move-with-store must be of the form i^S, where $\mathrm{i} \in I_\Gamma$ and $\mathsf{dom}(S) = \nu(\mathrm{i})$. The set of initial moves-with-store will be written I_Γ^{st}.

Next we discuss how to assign, to any ToyML term $\Gamma \vdash M : \theta$, a set of complete plays over $\Gamma \vdash \theta$ in order to achieve full abstraction. We shall write $(\!|\, \Gamma \vdash M : \theta \,|\!)$ for that set. This constitutes a very direct account of the game-semantic interpretation of GroundML (presented in the following chapter), specialised to ToyML. Overall the complete-play interpretation is guaranteed to yield the following result[2].

Theorem 4.6. Let $\Gamma \vdash M_1, M_2 : \theta$ be ToyML terms. Then $\Gamma \vdash M_1 \cong M_2$ if and only if $(\!|\, \Gamma \vdash M_1 \,|\!) = (\!|\, \Gamma \vdash M_2 \,|\!)$.

4.3 Interpretation of ToyML terms

We start off with a number of simple cases listed below before proceeding to more complicated ones.

$$() \mid i \mid x \mid x \oplus y \mid \mathsf{ref}(x) \mid x = y \mid\, !x \mid x := y \mid \mathsf{if}\, x\, \mathsf{then}\, N\, \mathsf{else}\, N'$$

[2]Note that $\cong$ is defined using contexts from GroundML.

Skip command $(())$

$(\!| \Gamma \vdash () : \mathsf{unit} |\!)$ is defined to contain all complete plays over $\Gamma \vdash \mathsf{unit}$ that have the shape $\mathrm{i}^S(\mathsf{r}_\downarrow, \star)^S$. Here P simply responds with the move $(\mathsf{r}_\downarrow, \star)$ without modifying the store.

Integer constant (i)

The defining complete plays for $(\!| \Gamma \vdash i : \mathsf{int} |\!)$ have the shape $\mathrm{i}^S(\mathsf{r}_\downarrow, i)^S$. This follows the same pattern as above, except that the value is i.

Variable (x)

We distinguish two cases depending on the type of x.

The complete plays in $(\!| \Gamma \vdash x : \beta |\!)$ all have the form $\mathrm{i}^S(\mathsf{r}_\downarrow, \mathrm{i}_x)^S$. P simply responds by copying the value of x provided by O in the initial move and pairing it with the tag $\mathsf{r}_\downarrow$.

For $(\!| \Gamma \vdash x : \beta \to \beta' |\!)$, the complete plays must have the form

$$\mathrm{i}^S(\mathsf{r}_\downarrow, \dagger)^S X_1 \cdots X_k$$

where $k \geq 0$ and

$$X_i = (\mathsf{c}, \ell_i)^{S_i}(\mathsf{c}_x, \ell_i)^{S_i}(\mathsf{r}_x, \ell'_i)^{S'_i}(\mathsf{r}, \ell'_i)^{S'_i}$$

for all $1 \leq i \leq k$. Intuitively, this corresponds to P first replying with the $\dagger$ value. Subsequently, a series of calls can be initiated by O. Each such call (tag c) is forwarded by P to x by changing the tag to c_x while copying the value. Similarly, return moves by O (tag r_x), which must happen right after the matching call moves, are copied by modifying the tag to r. Note that P never changes the stores played by O. In contrast, O is allowed to modify the stores insofaras the definition of complete play allows, i.e. the integer values in S'_i may be different from the corresponding values in S_i.

Arithmetic operations $(x \oplus y)$

$(\!| \Gamma \vdash x \oplus y : \mathsf{int} |\!)$ is given by complete plays of the form $\mathrm{i}^S(\mathsf{r}_\downarrow, \mathrm{i}_x \oplus \mathrm{i}_y)$.

Reference creation $(\mathsf{ref}(x))$

$(\!| \Gamma \vdash \mathsf{ref}(x) : \mathsf{ref\,int} |\!)$ is defined by complete plays of the form

$$\mathrm{i}^S(\mathsf{r}_\downarrow, a)^{S[a \mapsto \mathrm{i}_x]}$$

with $a \in \mathbb{A}_{\mathsf{int}} \setminus \mathsf{dom}(S)$. Here P provides a fresh name as part of his response (note the $a \notin \mathsf{dom}(S)$ requirement), and initialises the value of a in the store to i_x. Observe that, although many a's are possible, when we consider objects "up to name-permutation", there is only one complete play.[3]

Reference equality check ($x = y$)

We take $(\!|\,\Gamma \vdash x = y : \mathsf{int}\,|\!)$ to be

$$\{\mathrm{i}^S(\mathsf{r}_\downarrow, 0)^S \mid \mathrm{i}^S \in I_\Gamma^{\mathrm{st}},\ \mathrm{i}_x \neq \mathrm{i}_y\} \cup \{\mathrm{i}^S(\mathsf{r}_\downarrow, 1)^S \mid \mathrm{i}^S \in I_\Gamma^{\mathrm{st}},\ \mathrm{i}_x = \mathrm{i}_y\}.$$

In this case P responds with 0 or 1, depending on whether the values corresponding to x and y are the same in the initial move. Note here that $\mathrm{i}_x, \mathrm{i}_y$ are names from $\mathbb{A}_{\mathsf{int}}$.

Dereferencing ($!x$)

In this instance P simply returns the value provided in the store of the initial move:

$$(\!|\,\Gamma \vdash\, !x : \mathsf{int}\,|\!) = \{\mathrm{i}^S(\mathsf{r}_\downarrow, S(\mathrm{i}_x))^S \mid \mathrm{i}^S \in I_\Gamma^{\mathrm{st}}\}.$$

Note again that $\mathrm{i}_x \in \mathbb{A}_{\mathsf{int}}$.

Reference update ($x := y$)

$$(\!|\,\Gamma \vdash x := y : \mathsf{unit}\,|\!) = \{\mathrm{i}^S(\mathsf{r}_\downarrow, \star)^{S[\mathrm{i}_x \mapsto \mathrm{i}_y]} \mid \mathrm{i}^S \in I_\Gamma^{\mathrm{st}}\}$$

Here P modifies the store of the initial move using the value for y from the initial move. The types of x, y specify that i_x is a name and i_y is an integer. Observe that this is the only case encountered so far where the move that P makes has an effect on the initial store S. This reflects the fact that previous terms did not generate updates.

Conditionals ($\mathsf{if}\ x\ \mathsf{then}\ N\ \mathsf{else}\ N'$)

In this case we simply borrow plays from $(\!|\,\Gamma \vdash N\,|\!)$ or $(\!|\,\Gamma \vdash N'\,|\!)$ depending on the value of x inside the initial move, i.e. $(\!|\,\Gamma \vdash \mathsf{if}\ x\ \mathsf{then}\ N\ \mathsf{else}\ N'\,|\!)$

[3] I.e. for any $a' \in \mathbb{A}_{\mathsf{int}} \setminus \mathsf{dom}(S)$, we have that $\mathrm{i}^S(\mathsf{r}_\downarrow, a')^{S[a' \mapsto \mathrm{i}_x]} \sim \mathrm{i}^S(\mathsf{r}_\downarrow, a)^{S[a \mapsto \mathrm{i}_x]}$.

is equal to:

$$\{\mathrm{i}^S s \mid \mathrm{i}_x > 0,\ \mathrm{i}^S s \in (\!|\, \Gamma \vdash N \,|\!)\} \cup \{\mathrm{i}^S s \mid \mathrm{i}_x = 0,\ \mathrm{i}^S s \in (\!|\, \Gamma \vdash N' \,|\!)\}.$$

Observe that, in contrast to previous cases, here we allow for plays of length greater than 2. This is because N and N' may be terms of the more complicated shapes addressed below. More intuitively, though, we can say that the terms we had examined before did not engage in higher-order behaviours, which can lead to further exchange of moves between P and O.

Remark 4.7. In the above cases, most operations were restricted to being performed on variables. This allowed us to explain the essence of each operation in a minimalistic setting. However, if it is necessary to translate a term that involves more complex terms, we can follow the relevant recipe above and combine it with the way that application will be interpreted, which will be explained below. This relies on the fact that, for example, the complete-play interpretations of $M := N$ and $\mathsf{let}\, x = M \,\mathsf{in}\, (\mathsf{let}\, y = N \,\mathsf{in}\, x := y)$ are guaranteed to be the same.

Next we turn our attention to more complicated ways of extracting corresponding plays, involved in the remaining cases of our interpretation:

$$\lambda x^\beta.M \mid \mathsf{while}(M) \mid xy \mid (\lambda x^\beta.M)N$$

Remark 4.8. The cases we shall examine concern application. Note that in ToyML the argument has to be of type β and the function term of type $\beta \to \beta'$. Moreover, the scope for creating terms of function type is quite limited: they can be identifiers, λ-abstractions or branches of a conditional expression. Since the game interpretation of $(\mathsf{if}\, M \,\mathsf{then}\, M' \,\mathsf{else}\, M'')N$ is the same as that of $\mathsf{if}\, M \,\mathsf{then}\, (M'N) \,\mathsf{else}\, (M''N)$, we only cover the first two cases in this chapter. As before, the general approach to modelling application in game semantics will be defined in the next chapter.

Lambda abstraction ($\lambda x^\beta.M$)

$(\!|\, \Gamma \vdash \lambda x^\beta.M : \beta \to \beta' \,|\!)$ is defined to consist of all complete plays of the form

$$\mathrm{i}^S(\mathsf{r}_\downarrow, \dagger)^S X_1 \cdots X_k\,.$$

Each of the segments X_i corresponds to a single call to M and its format will be required to be compatible with $(\!|\,\Gamma, x : \beta \vdash M : \beta'\,|\!)$ in the sense we explain below.

Consequently, $X_1 \cdots X_k$ resembles an interleaving of plays from $(\!|\,\Gamma, x : \beta \vdash M : \beta'\,|\!)$ except that, due to multiple calls, more and more names can be generated than those participating in a single call. Such names have to be carried along by the play, even though they do not take part in the call. Accordingly, P will not be allowed to modify them. These inactive parts of the store will be referred to as $U_{i,j}$'s in the specification of the shape of each X_i. Overall, we shall require that each X_i be of the shape

$$(\mathsf{c}, \ell_{\mathsf{c}})^{S_0 \uplus U_{i,0}}\, m_{i,1}^{S_{i,1} \uplus U_{i,0}} \cdots m_{i,2k}^{S_{i,2k} \uplus U_{i,k}}\, (\mathsf{r}, \ell_{\mathsf{r}})^{S_{i,2k+1} \uplus U_{i,k}}$$

such that $(\mathrm{i}, \ell_{\mathsf{c}})^{S_0}\, m_{i,1}^{S_{i,1}} \cdots m_{2k}^{S_{i,2k}}\, (\mathsf{r}_{\downarrow}, \ell_{\mathsf{r}})^{S_{i,2k+1}}$, henceforth referred to as *thread* s_i, is a complete play from $(\!|\,\Gamma, x : \beta \vdash M\,|\!)$.

The presence of the $U_{i,j}$'s, the disjointness of their domains from those of the respective stores in s_i and the fact that P cannot make changes to $U_{i,j}$ guarantee that names introduced by P in different threads s_i have to be disjoint. Moreover, they will be different from any O-names introduced in earlier threads.

In summary, the defining complete plays for $\lambda x^{\beta}.M$ are obtained by interleaving complete plays from $(\!|\,\Gamma, x : \beta \vdash M\,|\!)$ in such a fashion that the names created in each thread by P are disjoint and fresh with respect to the preceding dialogue.

While loop (while(M))

Suppose $\Gamma \vdash M : \mathsf{int}$. We first calculate $(\!|\,\Gamma \vdash M : \mathsf{int}\,|\!)$ and observe that it must be equal to $(\!|\,\Gamma, x : \mathsf{unit} \vdash M : \mathsf{int}\,|\!)$ except that there is an extra $\star$ in the initial move. Note that the $\star$ has no bearing on names. $\mathsf{while}(M)$ will then be interpreted by restricting $(\!|\,\Gamma \vdash \lambda x^{\mathsf{unit}}.\, M : \mathsf{int}\,|\!)$. Recall that sequences from $(\!|\,\Gamma \vdash \lambda x^{\mathsf{unit}}.\, M : \mathsf{int}\,|\!)$ match the pattern

$$X\, (\mathsf{r}_{\downarrow}, \dagger)(\mathsf{c}, \star)\, X_1\, (\mathsf{r}, \ell_1)(\mathsf{c}, \star) \cdots (\mathsf{r}, \ell_{k-1})(\mathsf{c}, \star)\, X_k\, (\mathsf{r}, \ell_k).$$

The above represents interleavings of calls of $\lambda x^{\mathsf{unit}}.\, M : \mathsf{int}$ or, equivalently, of evaluations of M. To interpret $\Gamma \vdash \mathsf{while}(M) : \mathsf{unit}$ we select only those sequences above where the induced sequence $\ell_1 \cdots \ell_k$ satisfies $\ell_k = 0$ and $\ell_j > 0$ $(1 \leq j \leq k)$. Subsequently, we erase all moves with tags $\mathsf{c}, \mathsf{r}, \mathsf{r}_\downarrow$ and add the move $(\mathsf{r}_\downarrow, \star)$ at the end. This yields the sequence:

$$X X_1 \cdots X_k (\mathsf{r}_\downarrow, \star).$$

In the above we have omitted stores, which simply need to be copied over from one sequence to the other.

Application (xy)

$(\!|\, \Gamma \vdash xy : \beta' \,|\!)$ contains all complete plays of the shape

$$\mathrm{i}^S (\mathsf{c}_x, \mathrm{i}_y)^S (\mathsf{r}_x, \ell)^{S'} (\mathsf{r}_\downarrow, \ell)^{S'}.$$

Note that the second move is a move with a call-tag and a value corresponding to the value of y in the initial move. The third move is an O-move corresponding to a return from the call. The returned result is subsequently used in the last move. P does not change the store in any of the plays, but O can play a different store S'. We must have $\mathsf{dom}(S) \subseteq \mathsf{dom}(S')$ and the inclusion can be proper if $\ell \in \mathbb{A} \setminus \mathsf{dom}(S)$.

Application $((\lambda x^\beta . M) N)$

Observe that this case corresponds to $\mathsf{let}\, x = N \,\mathsf{in}\, M$, where N is of base type. The corresponding complete plays will be obtained by concatenating those from N with those from M. However, not all combinations can be used for that purpose. For a start, the respective initial moves must be compatible, i.e. identical for shared identifiers and, additionally, the value occurring in the last move of the complete play from N must be present in the initial move of the complete play from M. The latter will mimic parameter passing.

Accordingly, let us consider

$$\begin{aligned} s &= \mathrm{i}^{S_0}\, u\, (\mathsf{r}_\downarrow, \ell)^S \in (\!|\, \Gamma \vdash N : \beta \,|\!)\,, \\ t &= (\mathrm{i}, \ell)^{T_0}\, m_1^{T_1} \cdots m_{2k+1}^{T_{2k+1}} \in (\!|\, \Gamma, x : \beta \vdash M : \beta' \,|\!)\,. \end{aligned}$$

Note that the initial move of t contains ℓ, which also occurs in the last move of s. Still, not all s, t will be used to calculate $(\!|\, \Gamma \vdash \mathsf{let}\, x = N \,\mathsf{in}\, M \,|\!)$. In order to qualify, they will have to fulfil the three conditions given below:

1. $T_0 \subseteq S$
2. $\nu(s) \cap P(t) = \emptyset$, where we let $P(t)$ be the set of P-names of t: $P(t) = \{\, a \in \nu(t) \mid \exists t' m^T \sqsubseteq t.\, a \in \mathsf{dom}(T) \setminus \nu(t'),\ |t'| \text{ odd} \,\}$.
3. If $\ell \in \mathbb{A}$ and ℓ does not occur in $\mathrm{i}^{S_0}u$ then, if there is an occurrence of ℓ in some m_i $(1 \le i \le 2k{+}1)$, the first such occurrence (if any) must be in a P-move $m_{2k'+1}$ for some k'. Up to that moment, O may not change the stored value of ℓ, i.e. $T_{2l}(\ell) = T_{2l-1}(\ell)$ for $1 \le l < k'$. If ℓ does not occur in any m_i, we also insist on the above property with $k' = k+1$, i.e. O does not change ℓ at all.

Intuitively, the first condition ensures continuity between the evaluation of N and M: the first store in M must be consistent with the store left over after the evaluation of N. Note that $\mathsf{dom}(S)$ may properly contain $\mathsf{dom}(T_0)$. This will be the case if s contains more names than those in (i, ℓ).

The second and third conditions ensure privacy of freshly created names. The former insists that names generated by M be disjoint from any names interacting for N. The third condition is specific to the case where the last move of N generates a fresh name, which can only happen if $\beta = \mathsf{ref\,int}$, so that the last value ℓ be a (fresh) name. If that is the case, in the ensuing computation the name cannot be guessed by the environment. Thus, if it is to be revealed, this has to be done through a P-move $(m_1, m_3, \cdots$ or $m_{2k+1})$. Before that happens, the environment is forbidden from modifying the corresponding part of the store, to recognise the fact that it does not have access to that name.

When s, t satisfy all of the above conditions, they can be combined to form complete plays for $\mathsf{let}\, x = N \,\mathsf{in}\, M$. We distinguish two cases.

- If $\ell \notin \mathbb{A}$, or $\ell \in \mathbb{A} \cap \nu(\mathrm{i}^{S_0}u)$, we include in $(\!|\, \Gamma \vdash \mathsf{let}\, x = N \,\mathsf{in}\, M \,|\!)$ all complete plays over $\Gamma \vdash \beta'$ that match the following shape.
$$\mathrm{i}^{S_0} u\, m_1^{T_1 \uplus (S \setminus T_0)} m_2^{T_2 \uplus U_1} m_3^{T_3 \uplus U_1} \cdots m_{2k}^{T_{2k} \uplus U_l} m_{2k+1}^{T_{2k+1} \uplus U_l}$$
If $\ell \in \mathbb{A}$, in the above complete plays ℓ will feature in every store from the moment it appears in $\mathrm{i}^{S_0}u$. The stores U_i consist of names that are recognisable to N, but not to M. They can be

arbitrary as long as they give rise to a complete play. Observe that P will not be modifying the U_i's in his responses to reflect the fact that M has not access to these names and, hence, they cannot be modified. The same recipe as above applies to the case $\ell \notin \mathbb{A}$.

- If $\ell \in \mathbb{A}$ and the only occurrence of ℓ in s is the final move then we shall proceed differently. Recall that then, by condition 3, the first move of t in which ℓ may occur after the initial move must be odd-numbered. Let it be $m_{2k'+1}$. In order to reflect the fact that ℓ was freshly generated by N, we shall suppress it in stores coming from M until the name is eventually played by P. Let $T_i^\ell = T_i \upharpoonright (\mathsf{dom}(T_i) \setminus \ell)$. Then we include in $(\!| \Gamma \vdash \mathsf{let}\, x = N \,\mathsf{in}\, M \,|\!)$ all complete plays over $\Gamma \vdash \beta'$ of the shape:

$$\mathrm{i}^{S_0}\, u\, m_1^{T_1^\ell \uplus (S \setminus T_0)} m_2^{T_2^\ell \uplus U_1} \cdots m_{2k'}^{T_{2k'}^\ell \uplus U_{k'}} m_{2k'+1}^{T_{2k'+1} \uplus U_{k'}} \cdots m_{2k}^{T_{2k} \uplus U_k} m_{2k+1}^{T_{2k+1} \uplus U_k}$$

  Note that ℓ is being hidden in the stores before $m_{2k'+1}$ is played. From this point onwards it becomes part of the play and is carried on as in the previous case. Analogously, if ℓ does not occur in t (apart from the initial occurrence) then ℓ must be deleted from all stores.

This completes the semantic translation. We next look at some examples that demonstrate the above constructions.

Example 4.9. We calculate the complete plays corresponding to

$$x : \mathsf{ref\,int} \vdash (\lambda y^{\mathsf{ref\,int}}.\, x = y)(\mathsf{ref}(0)) : \mathsf{int}\,.$$

The complete plays corresponding to the term $x : \mathsf{ref\,int} \vdash \mathsf{ref}(0)$ have the shape $s = \mathrm{i}^{S_0}(\mathsf{r}_\downarrow, \ell)^S$, where $\mathrm{i} \neq \ell$, $S_0 = \{(\mathrm{i}, j)\}$, $S = S_0 \uplus \{(\ell, 0)\}$ for arbitrary $j \in \mathbb{Z}$. In turn, the term $x : \mathsf{ref\,int}, y : \mathsf{ref\,int} \vdash x = y : \mathsf{int}$ generates two kinds of complete plays:

- $t = (\mathrm{i}_x, \mathrm{i}_y)^{T_0}(\mathsf{r}_\downarrow, 0)^{T_0}$ with $\mathrm{i}_x \neq \mathrm{i}_y$ and $T_0 = \{(\mathrm{i}_x, j_x), (\mathrm{i}_y, j_y)\}$, where $j_x, j_y \in \mathbb{Z}$, or
- $t = (\mathrm{i}_x, \mathrm{i}_y)^{T_0}(\mathsf{r}_\downarrow, 1)^{T_0}$ with $\mathrm{i}_x = \mathrm{i}_y$, $T_0 = \{(\mathrm{i}_x, j_x)\}$ for any $j_x \in \mathbb{Z}$.

In order for s and t to be compatible and be able to yield a complete play for the original term, we must have $\mathrm{i} = \mathrm{i}_x$ and $\ell = \mathrm{i}_y$. Note that, because in s we must have $\mathrm{i} \neq \ell$, this implies $\mathrm{i}_x \neq \mathrm{i}_y$, i.e. only the first kind of t needs to be considered. Moreover, it must be the case that $T_0 \subseteq S$ (condition 1), i.e. $j_x = j$ and $j_y = 0$. As there are no P-names in t, condition 2 is satisfied vacuously. So is 3, as the only O-moves in t are initial. Because ℓ features only in the last move of s and in the initial move of t, it will not be present in complete plays corresponding to the whole term and also needs to be removed from stores. Consequently, the plays corresponding to $x : \mathsf{ref\,int} \vdash (\lambda y^{\mathsf{ref\,int}}.\, x = y)(\mathsf{ref}(0)) : \mathsf{int}$ will have the shape $\mathrm{i}_x^{\{(\mathrm{i}_x, j_x)\}}(\mathsf{r}_\downarrow, 0)^{\{(\mathrm{i}_x, j_x)\}}$, where $j_x \in \mathbb{Z}$.

Example 4.10. To illustrate condition 3 of the latter composition case at work, let us consider a slightly more complicated term, namely

$$f : \mathsf{unit} \to \mathsf{ref\,int} \vdash (\lambda y^{\mathsf{ref\,int}}.\, f() = y)(\mathsf{ref}(0)) : \mathsf{int}\,.$$

The complete plays for the term $f : \mathsf{unit} \to \mathsf{ref\,int} \vdash \mathsf{ref}(0)$ have the shape $s = \star^{\emptyset}(\mathsf{r}_\downarrow, \ell)^S$, where $S = \{(\ell, 0)\}$ and two shapes are possible for $f : \mathsf{unit} \to \mathsf{ref\,int}, y : \mathsf{ref\,int} \vdash f() = y : \mathsf{int}$, namely:

- $t = (\star, \mathrm{i}_y)^{T_0}(\mathsf{c}_f, \star)^{T_0}(\mathsf{r}_f, \ell')^{T_1}(\mathsf{r}_\downarrow, 0)^{T_1}$ with $T_0 = \{(\mathrm{i}_y, j_y)\}$, $\mathrm{i}_y \neq \ell'$, $T_1 = \{(\mathrm{i}_y, j'_y), (\ell', j')\}$, where $j_y, j'_y, j' \in \mathbb{Z}$, or
- $t = (\star, \mathrm{i}_y)^{T_0}(\mathsf{c}_f, \star)^{T_0}(\mathsf{r}_f, \mathrm{i}_y)^{T_1}(\mathsf{r}_\downarrow, 1)^{T_1}$ with $T_0 = \{(\mathrm{i}_y, j_y)\}$ and $T_1 = \{(\mathrm{i}_y, j'_y)\}$.

We must have $\ell = \mathrm{i}_y$ for compatibililty of s and t. Furthermore, condition 1 stipulates $T_0 \subseteq S$, i.e. $j_y = 0$. Because $P(t) = \emptyset$, condition 2 is satisfied. However, note that the second kind of t would break condition 3 because, after $\ell = \mathrm{i}_y$ occurs in the initial move, its first occurrence is in the third move, i.e. an O-move (referred to as m_2 in condition 3). Consequently, only the first kind of t can be considered when calculating the complete plays for $f : \mathsf{unit} \to \mathsf{ref\,int} \vdash (\lambda y^{\mathsf{ref\,int}}.f() = y)(\mathsf{ref}(0)) : \mathsf{int}$. Moreover, because ℓ occurs only in the last move of s and does not occur in non-initial moves of t, O should be prevented from modifying values stored at ℓ. Thus, the relevant shapes of t are of the form

$$(\star, \ell)^{\{(\ell,0)\}}(\mathsf{c}_f, \star)^{\{(\ell,0)\}}(\mathsf{r}_f, \ell')^{\{(\ell,0),(\ell',j')\}}(\mathsf{r}_\downarrow, 0)^{\{(\ell,0),(\ell',j')\}}$$

with $\ell \neq \ell'$. Note that, because no non-initial move in t contains an occurrence of ℓ, the resultant stores will not have values for ℓ. Thus, $(\!| f : \mathsf{unit} \to \mathsf{ref\,int} \vdash (\lambda y^{\mathsf{ref\,int}}.f() = y)(\mathsf{ref}(0)) : \mathsf{int} |\!)$ consists of all complete plays of the form

$$\star^{\emptyset}\,(\mathsf{c}_f,\star)^{\emptyset}\,(\mathsf{r}_f,\ell')^{\{(\ell',j')\}}\,(\mathsf{r}_{\downarrow},0)^{\{(\ell',j')\}}$$

with $j' \in \mathbb{Z}$.

In the following chapter we present a full game model for GroundML. It will tackle full ground storage (not just integers), so the structure of the stores will be necessarily more complicated. Secondly, we shall handle arbitrary higher-order types, which will make interactions between strategies more complicated. The last case above ($\mathsf{let}\,x = M\,\mathsf{in}\,N$) gives a flavour of what is to come, because the let construction corresponds to composition. From this point of view, the above account is simply the specialisation of strategy composition to the case of types β.

5

Game Model

In this section we focus on the fully abstract game semantics for GroundML. The framework amounts to a systematisation of the concepts introduced earlier for ToyML. In particular, we develop the whole necessary infrastructure for *playing games*, consisting of *prearenas* and *strategies*, which becomes the semantic universe, a category, in which GroundML can be modelled. The interpretation maps supported terms into strategies on suitable prearenas.

5.1 Moves, arenas, plays, strategies

Recall that our constructions will be in nominal sets over the set of location names: $\mathbb{A} = \biguplus_{\zeta} \mathbb{A}_{\zeta}$. More precisely, we shall be working in strong nominal sets as strong support will turn out to be essential for proving that game strategies remain deterministic when composed (cf. Proposition 5.27).

5.1.1 Nominal arenas

Our semantics involves games comprising a formal exchange of *moves* between two players: a Proponent, corresponding to the modelled term;

and an Opponent, corresponding to the computational environment of the term. The moves are selected from *(pre)arenas*, whose construction follows the structure of types. Arenas specify the set of available moves and the correlation between different moves within a game. Next we present these notions in detail. They are essentially the call-by-value arenas of Honda and Yoshida [18], cast inside the theory of nominal sets.

Definition 5.1. An ***arena*** $A = (M_A, I_A, \vdash_A, \lambda_A)$ is given by:

- a strong nominal set M_A of moves,
- a nominal subset $I_A \subseteq M_A$ of initial moves,
- a nominal relation $\vdash_A \subseteq M_A \times (M_A \setminus I_A)$,
- a nominal function $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$,

satisfying, for each $m, m' \in M_A$, the conditions:

- $m \in I_A \implies \lambda_A(m) = (P, A)\,,$
- $m \vdash_A m' \wedge \lambda_A^{QA}(m) = A \implies \lambda_A^{QA}(m') = Q\,,$
- $m \vdash_A m' \implies \lambda_A^{OP}(m) \neq \lambda_A^{OP}(m')\,.$

We call $\vdash_A$ the *justification relation* of A, and λ_A its *labelling function.*

The role of λ_A is to label moves as *Opponent* or *Proponent* moves, or *O-moves* and *P-moves* respectively, and as *Questions* or *Answers.* We write λ_A^{OP} and λ_A^{QA} for λ_A followed by a first and second projection functions respectively. We shall denote moves as m, n, etc. and write q for moves that are questions. Moreover, i and variants will be used to range over initial moves, and we will sometimes use o and p to denote O- and P-moves respectively. Finally, we will be depicting arenas as directed bipartite graphs, with nodes given by moves, edges by the justification relation, and in which the partition is defined by the OP-polarities. These graphs are rooted at initial moves.

Arenas are used to model the types of our language via a semantic translation that maps each type θ to an arena $[\![\theta]\!]$ and which is defined next. Consequently, an arena $A = [\![\theta]\!]$ is the domain of all values of type θ. The simplest arena is $(\emptyset, \emptyset, \emptyset, \emptyset)$ and could be used to represent the empty type, had we included it in our language. Other flat arenas are 1, $\mathbb{Z}$ and $\mathbb{A}_\zeta$, for each ground type ζ, defined by:

$$M_1 = I_1 = \{\star\}\,, \quad M_{\mathbb{Z}} = I_{\mathbb{Z}} = \mathbb{Z}\,, \quad M_{\mathbb{A}_\zeta} = I_{\mathbb{A}_\zeta} = \mathbb{A}_\zeta\,,$$

whereby “flat” means that the arenas only contain initial moves and, therefore, the justification relation is empty. These arenas allow us to model all ground types:

$$[\![\mathsf{unit}]\!] = 1 \qquad [\![\mathsf{int}]\!] = \mathbb{Z} \qquad [\![\mathsf{ref}\,\zeta]\!] = \mathbb{A}_\zeta.$$

Given arenas A and B, we can build more interesting arenas using the following constructions (depicted in Figure 5.1). The coproduct arena $A+B$ is constructed by simply taking the (disjoint) union of A and B:

$$\begin{aligned} M_{A+B} &= M_A + M_B & \lambda_{A+B} &= [\lambda_A, \lambda_B] \\ I_{A+B} &= I_A \cup I_B & \vdash_{A+B} &= \vdash_A \cup \vdash_B \end{aligned}$$

where the notation $[\lambda_A, \lambda_B]$ means $[\lambda_A, \lambda_B](m) = \lambda_A(m)$ if $m \in M_A$, and $\lambda_B(m)$ otherwise. Here $M_A + M_B$ stands for the disjoint union of M_A and M_B. For simplicity, in definitions we assume that M_A and M_B are disjoint, so that no indexing of their elements is required. In cases like $A + A$, such an indexing will be left implicit and we shall work under the assumption that moves from the left-hand-side A can be uniquely identified and in particular distinguished from those of the right-hand-side A, and vice versa.

The product arena of A and B has initial moves which are pairings of initial moves from A and B respectively. The non-initial moves are precisely those of A and B, and in particular moves in A that were justified by some initial move i_A are now justified by $(\mathrm{i}_A, \mathrm{i}_B)$, for all $\mathrm{i}_B \in I_B$ (and dually for B). That is, writing $\bar{I}_A$ for $M_A \setminus I_A$, we have:

$$M_{A\otimes B} = (I_A \times I_B) + \bar{I}_A + \bar{I}_B \qquad I_{A\otimes B} = I_A \times I_B$$

and, for all $m \in M_{A\otimes B}$,

$$\lambda_{A\otimes B}(m) = \begin{cases} PA & \text{if } m = (\mathrm{i}_A, \mathrm{i}_B) \in I_A \times I_B \\ \lambda_A(m) & \text{if } m \in \bar{I}_A \\ \lambda_B(m) & \text{if } m \in \bar{I}_B \end{cases}$$

and, for all $(m, n) \in M^2_{A\otimes B}$,

$$\begin{aligned} m \vdash_{A\otimes B} n \iff \quad & (m = (\mathrm{i}_A, \mathrm{i}_B) \wedge (\mathrm{i}_A \vdash_A n \vee \mathrm{i}_B \vdash_B n)) \vee \\ & (m \vdash_A n) \vee (m \vdash_B n) \end{aligned}$$

On the other hand, the arena $A \Rightarrow B$ contains functions, all of which start with an initial move of the form †, which we can think as a function handle or name. That initial move justifies initial moves i_A from A, seen as questions/calls to the represented function. In the same spirit, initial moves i_B from B are answers/returns to the i_A's. Formally,

$$M_{A\Rightarrow B} = \{\dagger\} + M_A + M_B \qquad\qquad I_{A\Rightarrow B} = \{\dagger\}$$

and, for all $m \in M_{A\Rightarrow B}$,

$$\lambda_{A\Rightarrow B}(m) = \begin{cases} PA & \text{if } m = \dagger \\ OQ & \text{if } m = \mathrm{i}_A \in I_A \\ \bar{\lambda}_A(m) & \text{if } m \in \bar{I}_A \\ \lambda_B(m) & \text{if } m \in M_B \end{cases}$$

and, for all $(m, n) \in M^2_{A\Rightarrow B}$,

$$\begin{aligned} m \vdash_{A\Rightarrow B} n \iff \quad & ((m, n) \in (\{\dagger\} \times I_A) \cup (I_A \times I_B)) \vee \\ & (m \vdash_A n) \vee (m \vdash_B n) \end{aligned}$$

where we write $\bar{\lambda}_A$ for the OP-complement of λ_A.

Example 5.2. Recall the ground types $\zeta ::= \mathsf{unit} \mid \mathsf{int} \mid \mathsf{ref}\,\zeta$. We have seen that their denotations are given by flat arenas 1, $\mathbb{Z}$ and $\mathbb{A}_\zeta$, each of which has as moves the semantic counterparts of syntactic values (locations and integers), including the move $\star$ which corresponds to $()$: unit. We now look into the translation of first-order types of the

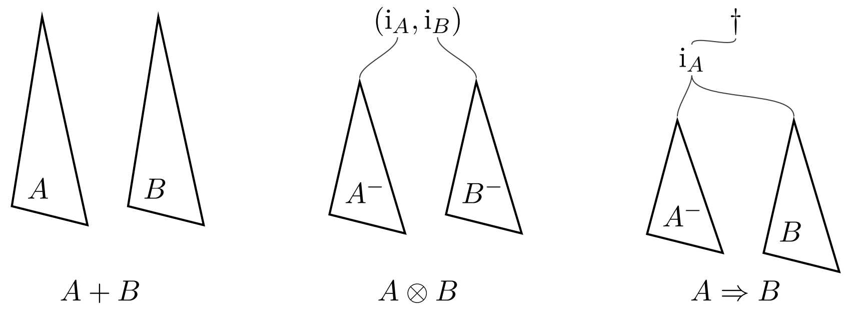
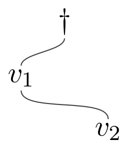


Figure 5.1: Arena constructions. The arena for $A + B$ is simply the combination of the arenas for A and B, seen as bipartite graphs. In the case of $A \otimes B$, the arena has as initial moves pairs of initial moves from A and B, from which the remainder sub-arenas of A (denoted A^-) and B (resp. B^-) are justified — we write A^- for A with its initial moves removed. The function arena $A \Rightarrow B$ has a unique initial move (†) which justifies the initial moves of the input arena A, the latter justifying A^- but also the initial moves of the output arena B.

form $\zeta_1 \to \zeta_2$. They translate to $[\![\zeta_1]\!] \Rightarrow [\![\zeta_2]\!]$, which are generally given by a diagram of the form:



where $v_1, v_2 \in \{\star\} \cup \mathbb{Z} \cup \mathbb{A}$ are moves from $[\![\zeta_1]\!]$ and $[\![\zeta_2]\!]$ respectively. The † is initial, and justifies (all) questions v_1, each of which justifies (all) answers v_2.

Games used to interpret terms are played *between* arenas, and in particular between the arenas denoting the context/input and the result/output type respectively. The structures that encode such combinations of input and output arenas are called ***prearenas***. They are defined in the same way as arenas with the exception that initial moves are O-questions.

Given arenas A and B, we define the prearena $A \to B$ by:

$$M_{A \to B} = M_A + M_B \qquad\qquad I_{A \to B} = I_A$$

and, for all $m \in M_{A\to B}$ and $(m,n) \in M^2_{A\to B}$:

$$\lambda_{A\to B}(m) = \begin{cases} OQ & \text{if } m = \mathrm{i}_A \in I_A \\ \bar\lambda_A(m) & \text{if } m \in \bar I_A \\ \lambda_B(m) & \text{if } m \in M_B \end{cases}$$

$$m \vdash_{A\Rightarrow B} n \iff ((m,n) \in I_A \times I_B) \vee (m \vdash_A n) \vee (m \vdash_B n)$$

Pictorially, $A \to B$ looks just like $A \Rightarrow B$ albeit having its initial $\dagger$ move removed.

Definition 5.3. The types of $\mathsf{GroundML}$ are interpreted into arenas by $[\![\mathsf{unit}]\!] = 1$, $[\![\mathsf{int}]\!] = \mathbb{Z}$, $[\![\mathsf{ref}\,\zeta]\!] = \mathbb{A}_\zeta$, and

$$[\![\theta_1 \times \theta_2]\!] = [\![\theta_1]\!] \otimes [\![\theta_2]\!], \qquad [\![\theta_1 \to \theta_2]\!] = [\![\theta_1]\!] \Rightarrow [\![\theta_2]\!].$$

Moreover, each finite $\mathrm{U} \subseteq \mathbb{A}$, say $\mathrm{U} = \{ a_1, \cdots, a_n \}$ with each name a_i belonging to the set $\mathbb{A}_{\zeta_i}$, is mapped to:

$$[\![\mathrm{U}]\!] = \{ (a'_1, \cdots, a'_n) \in \mathbb{A}_{\zeta_1} \times \cdots \times \mathbb{A}_{\zeta_n} \mid (a'_1, \cdots, a'_n) \sim (a_1, \cdots, a_n) \}$$

Finally, a typing environment $\mathrm{U}, \Gamma \vdash \theta$, with $\Gamma = \{ x_1 : \theta_1, \cdots, x_k : \theta_k \}$, is interpreted as the prearena[1]

$$[\![\mathrm{U}]\!] \otimes [\![\theta_1]\!] \otimes \cdots \otimes [\![\theta_k]\!] \;\to\; [\![\theta]\!]$$

which we shall denote by $[\![\mathrm{U}, \Gamma \vdash \theta]\!]$, or just $[\![\Gamma \vdash \theta]\!]$ if $\mathrm{U} = \emptyset$.

Remark 5.4. Let us look more closely at the structure of the prearena $[\![\mathrm{U}, \Gamma \vdash \theta]\!]$. Its initial moves are of the form $(a'_1, a'_2, \cdots, a'_n, \mathrm{i}_1, \mathrm{i}_2, \cdots, \mathrm{i}_k)$, where the $\mathrm{i}_1, \mathrm{i}_2, \cdots, \mathrm{i}_k$ part consists of initial moves from the denotation of Γ. Put otherwise, each i_i corresponds to an "opening" of the arena $[\![\theta_i]\!]$ with some semantic value provided by the Opponent (hence the polarity of the initial move is OQ). The rest of $[\![\mathrm{U}, \Gamma \vdash \theta]\!]$ comprises the output arena, $[\![\theta]\!]$, where Proponent himself provides a semantic value, and the remainders $[\![\theta_i]\!]^-$ of the input arenas. On the other hand, the $a'_1, a'_2, \cdots, a'_n$ part of the initial moves is a permutation variant of $(a_1, a_2, \cdots, a_n)$, assuming $\mathrm{U} = \{a_1, a_2, \cdots, a_n\}$. That is, there is

[1] if $k + |\mathrm{U}| = 0$ we take the left-hand side to be 1.

some permutation π such that, for all i, $a'_i = \pi \cdot a_i$. The reader might be puzzled by this choice — why not having the initial moves be just $(a_1, a_2, \cdots, a_n, \cdots)$? The answer to the latter is that we model terms modulo the choice of those names by Opponent, in the same sense that we model them modulo the choice of values for the open variables x_i.[2]

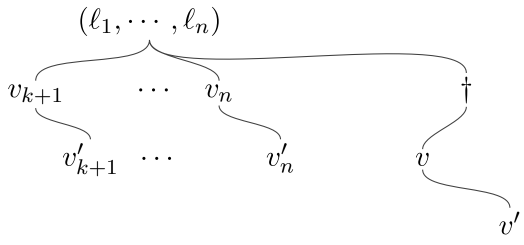
Example 5.5. We return to the arenas of Example 5.2 and expand our focus to prearenas which correspond to typing environments where the context is a mix of ground and ground-to-ground types:

$$x_1 : \zeta_1, \cdots, x_k : \zeta_k,\, x_{k+1} : \zeta_{k+1} \to \zeta'_{k+1}, \cdots, x_n : \zeta_n \to \zeta'_n \;\vdash\; \zeta \to \zeta'$$

These have denotations

$$(\,[\![\zeta_1]\!] \otimes \cdots \otimes [\![\zeta_k]\!] \otimes [\![\zeta_{k+1}]\!] {\Rightarrow} [\![\zeta'_{k+1}]\!] \otimes \cdots \otimes [\![\zeta_1]\!] {\Rightarrow} [\![\zeta'_1]\!]) \to [\![\zeta]\!] {\Rightarrow} [\![\zeta']\!]$$

which are generally of the following shape,



with $\ell_i = \dagger$ if $i > k$, and $\ell_i \in \{\star\} \cup \mathbb{Z} \cup \mathbb{A}$ otherwise (depending on ζ_i). The moves $(\ell_1, \cdots, \ell_n)$ and v are O-questions, the v_i's are P-questions, the v'_i's are O-answers, and $\dagger, v'$ are P-answers.

5.1.2 Plays

The plays of our games consist of moves equipped with stores which correspond to the stores present in the operational semantics, albeit with their domain restricted to the set of public (i.e. not private) names. Plays also have additional pointers to declare the *causality* of moves:

[2] A more committed answer is that, while it is possible to commit ourselves to the specific choice of initial names, $(a_1, a_2, \cdots, a_n)$, it makes the presentation unnecessarily complicated, cf. [3].

each question move must point to the function that it refers to; and each answer move must point to the question that it answers.[3]

Recall that a store is a finite partial function $S : \mathbb{A} \rightharpoonup (\mathbb{A} \cup \mathbb{Z} \cup \{\star\})$ that is legal in the following sense:

- if $S(a) = x$ and a has type $\mathsf{ref}\,\zeta$, then x has type ζ;
- if $S(a) = b$ and $b \in \mathbb{A}$, then $b \in \mathsf{dom}(S)$.

Given a prearena A, a ***move-with-store*** on A is a pair (m, S), written m^S, where $m \in M_A$ and $S \in$ Stores. For economy, we may be writing "move" when we really mean "move-with-store".

Definition 5.6. A ***justified sequence*** on a prearena A is a sequence s of moves-with-store on A such that:

- the first move is of the form i^S with $\mathrm{i} \in I_A$;
- every other (i.e. not first) move $n^{S'}$ in s is equipped with a pointer to an earlier move m^S such that $m \vdash_A n$.

In the latter case, m is called the *justifier* of n; if n is an answer, we also say that n *answers* m.

Plays are justified sequences obeying some further combinatorial conditions which arise from the restrictions present in $\mathsf{GroundML}$. A first such condition is *alternation*: the two players should play in turns. This reflects the fact that $\mathsf{GroundML}$ is a sequential language.

Another condition is *bracketing*: this stipulates that function calls and returns must be well-nested and is due to the fact that we do not have control mechanisms, such as call-cc or exceptions. For bracketing, we shall be needing the following notions. Given a justified sequence

[3]The use of pointers can be avoided by using names for marking initial moves of function type, instead of the † move, as done e.g. in [20, 27, 11, 22]. Note that such a solution requires some special hygiene as e.g. the same function name cannot be played twice in "introduction" position (i.e. even if we want to play the same function twice, we have to hide this non-visible fact under the use of different names). Herein we stick to pointers as they are standard in the literature and, moreover, their elimination would introduce an orthogonal level of nominal infrastructure and conditions that would go beyond the purposes of this tutorial.

s, we say that s in ***complete*** if all questions in s justify an answer in s. If s is not complete, then there is a last (i.e. rightmost) question in it that justifies no answer — we call this the ***pending question*** of s. Bracketing would then amount to follow the rule of answering the pending question.

The absence of higher-order state in the language is captured by *visibility*: this disallows function invocations to jump outside their calling context and e.g. interact with functions that are no longer in scope. The calling context of a move is described by the following notion. We let the ***view*** $\ulcorner s \urcorner$ of a justified sequence s to be given by:

$$\ulcorner \varepsilon \urcorner = \varepsilon \quad \text{and} \quad \ulcorner s\, \overparen{m^S t\, n}^{S'} \urcorner = \ulcorner s \urcorner m^S n^{S'}.$$

Finally, the *frugality* condition controls the flow of names and in particular restricts the store to its public/available part. For each $X \subseteq \mathbb{A}$ and store S we define $S^*(X) = \bigcup_{i\in\omega} S_i(X)$, where

$$S_0(X) = X \quad \text{and} \quad S_{i+1}(X) = S(S_i(X)) \cap \mathbb{A},$$

to be the set of names that can be reached from X through S. Then, the set of ***available*** names of a justified sequence is defined inductively by:

$$\mathsf{Av}(\varepsilon) = \emptyset \quad \text{and} \quad \mathsf{Av}(sm^S) = S^*(\mathsf{Av}(s) \cup \nu(m)).$$

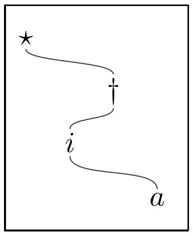
Note below that we write $s' \sqsubseteq s$ if s' is a prefix of s.

Definition 5.7. A justified sequence s is a ***play*** if it satisfies the following conditions.

- No two adjacent moves belong to the same player (*Alternation*).
- For all $tm^S \sqsubseteq s$ with m an answer, the justifier of m is the pending question of t (*Bracketing*).
- For all $tm^S \sqsubseteq s$ with non-empty t, the justifier of m is in $\ulcorner t \urcorner$ (*Visibility*).
- For all $tm^S \sqsubseteq s$, $\mathsf{dom}(S) = \mathsf{Av}(tm^S)$ (*Frugality*).

The set of plays on A is denoted by P_A.

Example 5.8. Recall the name generators $\vdash \mathsf{gen}, \mathsf{gen}' : \mathsf{int} \to \mathsf{ref\,int}$ from Example 3.4. Their typing context is translated into the prearena $[\![\vdash \mathsf{int} \to \mathsf{ref\,int}]\!] = 1 \to (\mathbb{Z} \Rightarrow \mathbb{A}_{\mathsf{int}})$, which has the shape depicted on the right (all $i \in \mathbb{Z}$ and $a \in \mathbb{A}_{\mathsf{int}}$). The plays produced by the always-same-name generator gen' shall have the following form (where there are no explicit pointers, assume the move points to its predecessor).



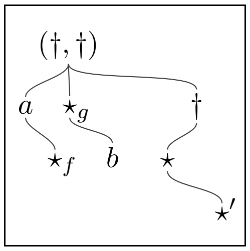
$$\begin{array}{cccccccc} \star & \dagger & i_1 & a^{(a,i_1)} & i_2{}^{(a,i_1')} & a^{(a,i_2)} & i_3{}^{(a,i_2')} & a^{(a,i_3)} \ \cdots \\ OQ & PQ & OA & PQ & OA & PA & OQ & PA \end{array}$$

On the other hand, the term gen will produce a different name a_i after each call:

$$\begin{array}{cccccccc} \star & \dagger & i_1 & a_1^{(a_1,i_1)} & i_2{}^{(a_1,i_1')} & a_2^{(a_2,i_2),(a_1,i_1')} & i_3{}^{(a_2,i_2'),(a_1,i_1'')} & a_3^{(a_3,i_3),(a_2,i_2'),(a_1,i_1'')} \ \cdots \\ OQ & PQ & OA & PQ & OA & PA & OQ & PA \end{array}$$

In the latter play, observe the saturating effect of frugality: the condition stipulates that all old names be carried along in the store, even when they are no more relevant to the term, and therefore the domain of the store always grows.

Example 5.9. Going back to Example 3.5, we look into the channel transmitter $f\colon \mathsf{ref\,ref\,int} \to \mathsf{unit}, g\colon \mathsf{unit} \to \mathsf{ref\,ref\,int} \vdash \mathsf{trmit}\colon \mathsf{unit} \to \mathsf{unit}$. The typing context corresponds to the prearena $(\mathbb{A}_{\mathsf{ref\,int}} \Rightarrow 1) \otimes (1 \Rightarrow \mathbb{A}_{\mathsf{ref\,int}}) \to (1 \Rightarrow 1)$, which we depict as on the right (all $a, b \in \mathbb{A}_{\mathsf{ref\,int}}$). The plays produced by the transmitter will be as follows, where for economy we only depict for each store S its $\mathsf{ref\,int}$ references (i.e. $S \upharpoonright \mathbb{A}_{\mathsf{ref\,int}}$), and suppress the values for $\mathbb{A}_{\mathsf{int}}$ names



$$\begin{array}{cccccccc} (\dagger,\dagger) & a_1^{(a_1,b_1)} & \star_f^{(a_1,b_1')} & \star_g^{(a_1,b_1')} & a_2^{(a_2,b_2),(a_1,b_1'')} & \dagger^{(a_2,b_2),(a_1,b_1'')} & \star^{(a_2,a'),(a_1,a)} & \star'^{(a_2,a),(a_1,a)} \ \cdots \\ OQ & PQ & OA & PQ & OA & PA & OQ & PA \end{array}$$

that have been encountered. Thus, such a play starts with an initial move $(\dagger, \dagger)$ presenting the functions f and g. P then plays a question a_1 (plus a store, that is not important at this point) thus supplying the name for the input channel c_1. O answers with $\star_f$ (and a possibly updated store), at which point P asks for the output channel name by playing $\star_g$ (plus store). O answers with the channel name a_2 and P returns his function via $\dagger$. From there on, any question $\star^{(a_2,a'),(a_1,a)}$ is answered by a move $\star'^{(a_2,a),(a_1,a)}$, which acknowledges that a transmission from a_1 to a_2 was made. Finally, note that this is not the only play shape for trmit: there are also plays where a_1 and a_2 coincide since O is allowed to return $a_2 = a_1$ in the fifth move above.

When defining play composition in the next section it will be important to be strict on retaining name privacy during composition, and for the latter we shall rely on the notion of *name ownership.* Given a play s and some $a \in \nu(s)$, we say that a is a ***P-name*** in s, written $a \in \mathsf{P}(s)$, if a is first introduced in s in a P-move. Put otherwise, if there is some even-length $s'm^S \sqsubseteq s$ such that $a \in \nu(m^S) \setminus \nu(s')$. The set $\mathsf{O}(s)$ of ***O-names*** of s is defined in a similar manner. Clearly,

$$\nu(s) = \mathsf{P}(s) \uplus \mathsf{O}(s)$$

for any play s.

We conclude this section on plays by looking at two properties related to visibility. First, the visibility imposed on plays, say, on $A \to B$, restricts them in a crucial aspect: only P is allowed to switch between the A and B component.

Lemma 5.10 (Switching). Let $s \in P_{A\to B}$ and $s'm^S n^{S'} \sqsubseteq s$ be such that $(m, n) \in (M_A \times M_B) \cup (M_B \times M_A)$. Then, $\lambda^{OP}_{A\to B} = P$.

Proof. By induction on the length of s, using also the fact that whenever the hypothesis holds for s then, for all odd-length $s' \sqsubseteq s$ ending in a move in A, all moves of $\ulcorner s' \urcorner$ belong to A; and if s' ends in B then all moves $\ulcorner s' \urcorner$ belong to B apart from the initial move. $\square$

Finally, we can see that visibility and bracketing both restrict, in different ways, what moves can be played at the end of a play. One may therefore imagine a scenario where a play has a pending question that is not in its view. At that point, an answer to that move would not be playable. The next lemma declares such scenarios impossible.

Lemma 5.11. Let $s \in P_A$ be a play that is not complete. Then, its pending question is in $\ulcorner s \urcorner$.

Proof. We argue by induction on s. Let q be the pending question of s and m_1 the last move of s (we eliminate stores for economy). If m_1 is a question then $m_1 = q$ and we are done. Otherwise, by bracketing, s must have the form $s = s'q_1 s_1 m_1$ with q_1 justifying m_1 and all questions of s_1 answered inside s_1. Thus, q is the pending question of s' as well and $\ulcorner s \urcorner = \ulcorner s' \urcorner q_1 m_1$, so we can apply the IH on s'. □

5.1.3 Strategies and composition

Game semantics interprets types as arenas. Strategies are then used to denote terms.

Definition 5.12. A ***strategy*** σ on a prearena A is a non-empty set of even-length plays of A satisfying:

- If $so^S p^{S'} \in \sigma$ then $s \in \sigma$ (*Even-prefix closure*).
- If $s \in \sigma$ then, for all permutations π, $\pi \cdot s \in \sigma$ (*Equivariance*).
- If $sp_1^{S_1}, sp_2^{S_2} \in \sigma$ then $sp_1^{S_1} = \pi \cdot sp_2^{S_2}$ for some permutation π (*Determinacy*).

We write $\sigma : A$ to declare that σ is a strategy on A.

Note in particular that $\epsilon \in \sigma$ for all strategies σ. Since strategies are even-prefix closed, they are generally specified by plays of maximal length. We thus introduce the following notation to save space in definitions. For any set X of even-length plays, we shall write X_{cl} for its even-prefix closure: $X_{cl} = \{ s \mid \exists s' \in X.\, s \sqsubseteq_{even} s' \}$.

Example 5.13. Let us revisit the terms examined in Examples 5.8, 5.9 and give formal definitions for the strategies corresponding to each term. While we have yet to define the semantic translation of terms, we shall nonetheless write $[\![M]\!]$ for the strategy denoting the term M; the explanation of this is the subject of the next sections. We start with the name generators:

$$[\![\mathsf{gen}']\!] = \{\, \star \dagger\, i_1\, a^{(a,i_1)} \cdots i_n\, a^{(a,i_n)} \mid n \geq 0 \wedge i_1, \cdots, i_n \in \mathbb{Z} \,\}_{cl}$$
$$[\![\mathsf{gen}]\!] = \{\, \star \dagger\, i_1^{S_1} a_1^{S_1 \uplus \{(a_1,i_1)\}} \cdots i_n^{S_n} a_n^{S_n \uplus \{(a_n,i_n)\}} \mid n \geq 0 \wedge i_1, \cdots, i_n \in \mathbb{Z} \,\}_{cl}$$

where, for each j, $\mathsf{dom}(S_j) = \{a_1, \cdots, a_{j-1}\}$ and $a_j \notin \mathsf{dom}(S_j)$.[4] On the other hand, for trmit we obtain the strategy:

$$[\![\mathsf{trmit}]\!] = \{\, (\dagger,\dagger)\, a_1^S \star_f^{S'} \star_g^{S'} a_2^{S_0} \dagger^{S_0} \star^{S_1} \star'^{S'_1} \cdots \star^{S_n} \star'^{S'_n} \mid n \geq 0 \,\}_{cl}$$

where stores satisfy the conditions:

$$\begin{aligned}
\mathsf{dom}(S) &= \{\, a_1, S(a_1) \,\} \\
\mathsf{dom}(S') &= \mathsf{dom}(S) \cup \{\, S'(a_1) \,\} \\
\mathsf{dom}(S_0) &= \mathsf{dom}(S') \cup \{\, S_0(a_1), a_2, S_0(a_2) \,\} \\
\mathsf{dom}(S_i) &= \mathsf{dom}(S_{i-1}) \cup \{\, S_i(a_1), S_i(a_2) \,\} \quad (i > 0)
\end{aligned}$$

and, for each j, $S'_j = S_j[a_2 \mapsto S_j(a_1)]$.

Remark 5.14. In the previous example we defined strategies by describing their candidate plays and also providing conditions ensuring that those where indeed plays of the examined prearenas. In the sequel we will be more economic and, for instance, when writing

$$[\![\mathsf{trmit}]\!] = \{\, (\dagger,\dagger)\, a_1^S \star_f^{S'} \star_g^{S'} a_2^{S_0} \dagger^{S_0} \star^{S_1} \star'^{S'_1} \cdots \star^{S_n} \star'^{S'_n} \mid n \geq 0 \,\}_{cl}$$

we shall implicitly assume that the defined expression is a play in $(\mathbb{A}_{\mathsf{ref\,int}} \Rightarrow 1) \otimes (1 \Rightarrow \mathbb{A}_{\mathsf{ref\,int}}) \to (1 \Rightarrow 1)$, a requirement which in this case implies the conditions on store domains we specified above.

We next show how plays and strategies are composed. Let us set γ to be an endofunction on justified sequences which restricts any justified sequence to a frugal one by removing from the stores the names violating frugality. Formally,

$$\gamma(\epsilon) = \epsilon\,, \quad \gamma(tm^S) = \gamma(t)\, m^{S'}$$

[4]The fact that $a_j \notin \mathsf{dom}(S_j)$ is already implicit in the notation $(S_j \uplus \{(a_j, i_j)\})$. Note in particular that $S_1 = \emptyset$.

where $S' = S \upharpoonright \mathsf{Av}(tm^S)$.

Play composition is defined extensionally, as follows. First, we define a notion of "3-arena play": these are plays played between three arenas, say A, B and C, and represent compositions of plays between $A \to B$ and $B \to C$. The way we compose plays is by matching moves in the common arena B.

We now turn to defining a suitable notion of interaction between plays. Given arenas A, B, C, we define the prearena $A \to B \to C$ by setting:

$$M_{A\to B\to C} = M_{A\to B} + M_C \qquad \lambda_{A\to B\to C} = [\lambda_{A\to B}[\mathrm{i}_B \mapsto PQ], \overline{\lambda}_C]$$
$$I_{A\to B\to C} = I_A \qquad \vdash_{A\to B\to C} = \vdash_{A\to B} \cup \{(\mathrm{i}_B, \mathrm{i}_C)\} \cup \vdash_C$$

Let u be a justified sequence on $A \to B \to C$. We define $u \upharpoonright AB$ to be u in which all C-moves are suppressed, along with associated pointers. $u \upharpoonright BC$ is defined in an analogous manner. $u \upharpoonright AC$ is defined similarly with the caveat that, if there was a pointer from an initial C-move to an initial B-move which in turn had a pointer to an initial A-move, we add a pointer from the C-move to the A-move. Moreover, we write $u \upharpoonright_\gamma AB$ for the result of applying γ to the projection $u \upharpoonright AB$, i.e. $u \upharpoonright_\gamma AB = \gamma(u \upharpoonright AB)$, and similarly for the other projections. Note that such a projection may alter move polarities: e.g. each i_B is a P-move in $A \to B \to C$, yet an O-move in $B \to C$.

Below we shall often say that a move is an O- or a P-move in X, for $X \in \{AB, BC, AC\}$, meaning ownership in the associated prearena ($A \to B$, $B \to C$ or $A \to C$). The sets of P- and O-names in $u \upharpoonright X$ are defined using the same convention.

Definition 5.15. A justified sequence u on $A \to B \to C$ is an ***interaction sequence*** on ABC if $\gamma(u \upharpoonright_\gamma AB) \in P_{A\to B}$, $\gamma(u \upharpoonright_\gamma BC) \in P_{B\to C}$ and the following conditions are satisfied (*Laird conditions* [28]):

- u is frugal, that is, $\gamma(u) = u$;
- $\mathsf{P}(u \upharpoonright_\gamma AB) \cap \mathsf{P}(u \upharpoonright_\gamma BC) = \emptyset$;
- $\mathsf{O}(u \upharpoonright_\gamma AC) \cap (\mathsf{P}(u \upharpoonright_\gamma AB) \cup \mathsf{P}(u \upharpoonright_\gamma BC)) = \emptyset$;
- for each $u' \sqsubseteq u$ ending in $m^S m'^{S'}$ and $a \in \mathsf{dom}(S')$ if

- m' is a P-move in AB and $a \notin \mathsf{Av}(u' \restriction AB)$,
- or m' is a P-move in BC and $a \notin \mathsf{Av}(u' \restriction BC)$,
- or m' is an O-move in AC and $a \notin \mathsf{Av}(u' \restriction AC)$,

then $S(a) = S'(a)$.

We write $\mathit{Int}(ABC)$ for the set of interaction sequences on ABC.

Thus, an interaction sequence on ABC must project well on AB and BC and, moreover, satisfy a set of conditions pertaining to frugality and name privacy (last three conditions). The latter conditions ensure that no party in the interaction can guess the other parties' private names or touch their private store. Note that, in an interaction as above, there are three parties interacting:[5] the Proponent of AB, the Proponent of BC, and the Opponent of AC. The following lemma shows that every name in u can be traced back to one of these parties.

Lemma 5.16. For any $u \in \mathit{Int}(ABC)$,

$$\nu(u) = \mathsf{O}(u \restriction_\gamma AC) \uplus \mathsf{P}(u \restriction_\gamma AB) \uplus \mathsf{P}(u \restriction_\gamma BC).$$

Proof. By definition of interaction sequences, the three RHS components have no common elements. Thus, it suffices to show that every element of $\nu(u)$ belongs in one of these components. So take any $a \in \nu(u)$ and let $u'm^S \sqsubseteq u$ be such that m^S introduces a in u. Let us assume that m is a P-move in AB — the other two cases (P-move in BC or O-move in AC) are dealt with in the same manner. We claim that $a \in \mathsf{Av}(u'm^S \restriction AB)$. For suppose this is not the case. Then $a \notin \nu(m)$ and, moreover, there is no $b \in \mathsf{Av}(u' \restriction AB) \cup \nu(m)$ such that a be reachable from b via S. But, by frugality of u, there must be some $b \in \mathsf{Av}(u') \cup \nu(m)$ such that a be reachable from b via S. Since $b \notin \mathsf{Av}(u' \restriction AB)$, the Laird conditions on stores imply that a is reachable from b already in the store of the last move of u', which contradicts the fact that $a \notin \nu(u')$. Hence, $a \in \mathsf{Av}(u'm^S \restriction AB)$ and, consequently, $a \in \mathsf{P}(u \restriction_\gamma AB)$. ☐

[5]There seem to be three roles missing from the above description: the Opponent of AB and BC, and the Proponent of AC. But these are already present here: e.g. the O of AB is played by the P of BC in B, and by the O of AC in A. Similarly for the other roles.

In order for interactions to lead to a well-defined notion of composition, we need to show that projecting an ABC-interaction sequence on AC yields a play in $A \to C$ (modulo frugality). We obtain this result after a series of technical lemmas. Although these results are obtained from standard game semantic analysis of call-by-value computation (e.g. [18]), for completeness, we include proofs for the first two in a technical appendix in Section 5.3.

Lemma 5.17 (Interaction switching)**.** Let $u \in \mathit{Int}(ABC)$, $u'm^S n^{S'} \sqsubseteq u$ be such that $(m,n) \notin M_A^2 \cup M_B^2 \cup M_C^2$. Then there is $X \in \{AB, BC\}$ such that $(m,n) \in M_X^2$, m is an O-move in X and n is a P-move in X.

Lemma 5.18 (Interaction bracketing)**.** Let $u \in \mathit{Int}(ABC)$ and $u'm^S \sqsubseteq u$ with m an answer. Then, m^S answers the last open question of u'.

Lemma 5.19 (Interaction visibility)**.** Given $u \in \mathit{Int}(ABC)$, define the view of u inductively by: $\ulcorner \epsilon \urcorner = \epsilon$; $\ulcorner u m^S u' n^{S'} \urcorner = \ulcorner u \urcorner m^S n^{S'}$, if n a move in AC justified by m; and $\ulcorner u m^S \urcorner = \ulcorner u \urcorner m^S$ otherwise. Then, for all $u'm^S \sqsubseteq u$, the justifier of m^S appears in $\ulcorner u' \urcorner$.

We can now prove that composition is well defined.

Proposition 5.20. If $u \in \mathit{Int}(ABC)$ then $(u \upharpoonright_\gamma AC) \in P_{A \to C}$.

Proof. The fact that $u \upharpoonright AC$ satisfies alternation, bracketing and visibility follows respectively from the previous three lemmata. Frugality of $\gamma(u \upharpoonright AC)$ is by definition of γ. $\square$

The above proposition gives us a way to compose plays: given $s \in P_{A \to B}$ and $t \in P_{B \to C}$, these can be composed to a play in $P_{A \to C}$ if there is some $u \in \mathit{Int}(ABC)$ such that $(u \upharpoonright_\gamma AB) = s$ and $(u \upharpoonright_\gamma BC) = t$. In such a case, the composite of s and t is $u \upharpoonright_\gamma AC$. We now extend this notion of composition to strategies.

Definition 5.21. For each pair of strategies $\sigma : A \to B$ and $\tau : B \to C$, their ***composition*** $\sigma;\tau \subseteq P_{A \to C}$ is given by:

$$\sigma;\tau = \{\, u \upharpoonright_\gamma AC \mid u \in \mathit{Int}(ABC) \wedge (u \upharpoonright_\gamma AB) \in \sigma \wedge (u \upharpoonright_\gamma BC) \in \tau \,\}\,.$$

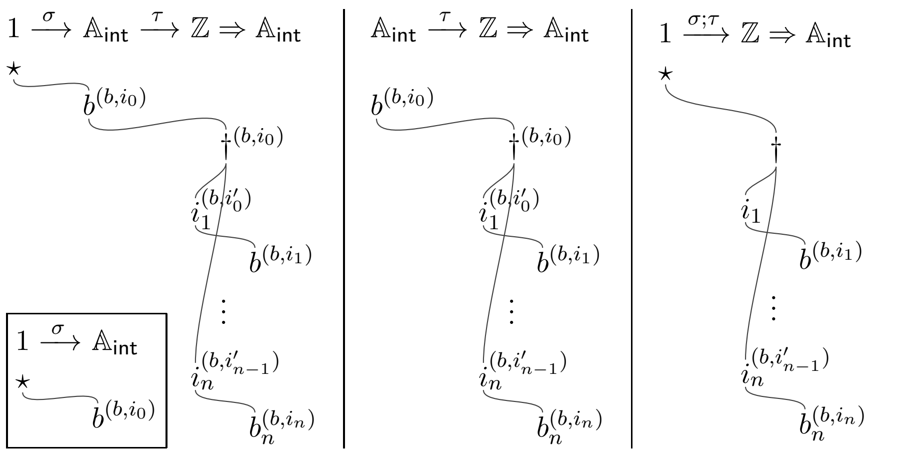

$1 \xrightarrow{\sigma} \mathbb{A}_{\mathsf{int}} \xrightarrow{\tau} \mathbb{Z} \Rightarrow \mathbb{A}_{\mathsf{int}}$
$\mathbb{A}_{\mathsf{int}} \xrightarrow{\tau} \mathbb{Z} \Rightarrow \mathbb{A}_{\mathsf{int}}$
$1 \xrightarrow{\sigma;\tau} \mathbb{Z} \Rightarrow \mathbb{A}_{\mathsf{int}}$
$1 \xrightarrow{\sigma} \mathbb{A}_{\mathsf{int}}$

Figure 5.2: Composition example: interaction sequence in ABC (left), projection on BC (middle), on AC (right), and on AB (bottom left). Here the composing strategies are $\sigma = [\![\mathsf{ref}(0)]\!] : A \to B$ and $\tau = [\![x : \mathsf{ref\,int} \vdash \lambda z^{\mathsf{int}}.\, x := z; x]\!] : B \to C$. The composition conditions enforce that $i_0 = i'_0 = 0$.

Example 5.22. Recall the first name generator from Example 3.4:

$$\mathsf{gen}' \equiv \ \mathsf{let}\, x = \mathsf{ref}(0)\,\mathsf{in}\, \lambda z^{\mathsf{int}}.\, x := z; x \ : \ \mathsf{int} \to \mathsf{ref\,int}.$$

$[\![\mathsf{gen}']\!]$ will be given via the composition:

$$[\![\mathsf{gen}']\!] = 1 \xrightarrow{[\![\mathsf{ref}(0)]\!]} \mathbb{A}_{\mathsf{int}} \xrightarrow{[\![x:\mathsf{ref\,int} \vdash \lambda z^{\mathsf{int}}.\, x:=z;x]\!]} \mathbb{Z} \Rightarrow \mathbb{A}_{\mathsf{int}}$$

with the semantics of each component being:

$$[\![\mathsf{ref}(0)]\!] = \{\, \star\, a^{(a,0)} \mid a \in \mathbb{A}_{\mathsf{int}} \,\}_{cl}$$

$$[\![\lambda z.x := z; x]\!] =$$

$$\{\, b^{(b,i_0)}\, \dagger^{(b,i_0)}\, i_1^{(b,i'_0)}\, b^{(b,i_1)} \cdots i_n^{(b,i'_{n-1})}\, b^{(b,i_n)} \mid b \in \mathbb{A}_{\mathsf{int}} \wedge n \geq 0 \,\}_{cl}$$

Composing these strategies amounts to considering interaction sequences u over $1\mathbb{A}_{\mathsf{int}}(\mathbb{Z} \Rightarrow \mathbb{A}_{\mathsf{int}})$ that project in $[\![\mathsf{ref}(0)]\!]$ and $[\![\lambda z.x := z; x]\!]$ on $1\mathbb{A}_{\mathsf{int}}$ and $\mathbb{A}_{\mathsf{int}}(\mathbb{Z} \Rightarrow \mathbb{A}_{\mathsf{int}})$ respectively, i.e. u are of the form

$$\star\, b^{(b,i_0)}\, \dagger^{(b,i_0)}\, i_1^{(b,i'_0)}\, b^{(b,i_1)} \cdots i_n^{(b,i'_{n-1})}\, b^{(b,i_n)}$$

with $i_0 = 0$ (see also Figure 5.2). Note that the above projects as $\star\, b^{(b,0)}$ on $1\mathbb{A}_{\mathsf{int}}$, which is in $[\![\mathsf{ref}(0)]\!]$ as the latter is closed under permutation. Moreover, by Laird conditions on composition, $i'_0 = i_0$. This is because the move $i_1^{(b,i'_0)}$ is an O-move in $1(\mathbb{Z} \Rightarrow \mathbb{A}_{\mathsf{int}})$ and, if we project on $1(\mathbb{Z} \Rightarrow \mathbb{A}_{\mathsf{int}})$ and apply γ we see that b disappears, i.e. is not available. Since b is a P-name in $1\mathbb{A}_{\mathsf{int}}$, O cannot change its value in the move $i_1^{(b,i'_0)}$. Therefore, i'_0 must be the same as i_0. Projecting these interaction sequences on $1(\mathbb{Z} \Rightarrow \mathbb{A}_{\mathsf{int}})$, we obtain:

$$[\![\mathsf{gen}']\!] = \{\, \star\ \dagger\ i_1\, b^{(b,i_1)} \cdots i_n^{(b,i'_{n-1})}\ b^{(b,i_n)} \mid b \in \mathbb{A}_{\mathsf{int}} \wedge n \geq 0 \,\}_{cl}$$

which is the strategy given in Example 5.8.

Example 5.23. While the Laird conditions did influence composition in the previous example, this was done in a way that did not affect the resulting strategy. We next look at a term where the role of these conditions is crucial. Let us consider the term:

$$M_1 \ \equiv\ \mathsf{let}\, s = \mathsf{ref}(0)\, \mathsf{in}\, \lambda y^{\mathsf{ref\,int}}.\, s = y \ :\ \mathsf{ref\,int} \to \mathsf{int}$$

from Example 3.7. Again, this results from a composition:

$$[\![M_1]\!] = 1 \xrightarrow{[\![\mathsf{ref}(0)]\!]} \mathbb{A}_{\mathsf{int}} \xrightarrow{[\![s:\mathsf{ref\,int} \vdash \lambda y^{\mathsf{ref\,int}}.\, s=y]\!]} \mathbb{A}_{\mathsf{int}} \Rightarrow \mathbb{Z}$$

$$[\![\mathsf{ref}(0)]\!] = \{\, \star\, a^{(a,0)}\, \}_{cl}$$

$$[\![\lambda y.s = y]\!] = \{\, b^{S_0}\ \dagger^{S_0}\ a_1^{S_1}\ i_1^{S_1} \cdots a_n^{S_n}\ i_n^{S_n} \mid b, a_1, \cdots, a_n \in \mathbb{A}_{\mathsf{int}} \wedge n \geq 0 \,\}_{cl}$$

where, for each j, $i_j = 1$ if $a_j = b$, and 0 otherwise, and S_j ranges over all the stores S such that $\mathsf{dom}(S_j) = \{b, a_1, \cdots, a_j\}$.[6] For composition, we consider interaction sequences u over $1\mathbb{A}_{\mathsf{int}}(\mathbb{A}_{\mathsf{int}} \Rightarrow \mathbb{Z})$:

$$\star\ b^{S_0}\ \dagger^{S_0}\ a_1^{S_1}\ i_1^{S_1} \cdots a_n^{S_n}\ i_n^{S_n}$$

with $S_0 = \{(b, 0)\}$. We now observe the following fact. The projection of $\star\ b^{S_0} \dagger^{S_0}$ on $1(\mathbb{A}_{\mathsf{int}} \Rightarrow \mathbb{Z})$, after applying γ, is $\star \dagger$, and b is a P-name in $1\mathbb{A}_{\mathsf{int}}$ while the next move, $a_1^{S_1}$, is an O-move in $1(\mathbb{A}_{\mathsf{int}} \Rightarrow \mathbb{Z})$. This

[6]For example, $S_0 = \{(b, i)\}$ for any $i \in \mathbb{Z}$. Note there may be repetitions in the sequence $b, a_1, \cdots, a_j$.

means two things: first, O cannot play $a_1 = b$ and, moreover, O cannot change the value of b. That is, $a_1 \neq b$ (thus $i_1 = 0$) and $S_1(b) = 0$. This reasoning extends to all $a_j^{S_j}$'s, and therefore u must be of the form:

$$\star\, b^{S_0}\, \dagger^{S_0}\, a_1^{S_1}\, 0^{S_1} \cdots a_n^{S_n}\, 0^{S_n}$$

with $S_j(b) = 0$. Projecting onto $1(\mathbb{A}_{\mathsf{int}} \Rightarrow \mathbb{Z})$, we obtain that $[\![M_1]\!] = \{\, \star\, \dagger\, a_1^{S'_1}\, 0^{S'_1} \cdots a_n^{S'_n}\, 0^{S'_n} \mid a_1, \cdots, a_n \in \mathbb{A}_{\mathsf{int}} \wedge n \geq 0 \,\}_{cl}$ where, for each j, $\mathsf{dom}(S'_j) = \{a_1, \cdots, a_j\}$. This is precisely $[\![M_2]\!]$, from Example 3.7.

5.1.4 Strategy closure

While our previous analysis ensures that the elements of $\sigma;\tau$ are plays, we would actually like $\sigma;\tau$ to be a strategy in $A \to C$. Of the strategy conditions, the one that presents the greatest difficulty is that of determinacy. First, we examine the following "stronger" determinacy condition for candidate strategies $\sigma \subseteq P_A$:

- For all $s_1 m_1^{S_1}, s_2 m_2^{S_2} \in \sigma$, if $s_1 \sim s_2$ then $s_1 m_1^{S_1} \sim s_2 m_2^{S_2}$ (*Strong determinacy*).

Lemma 5.24. Any equivariant set of even-length plays $\sigma \subseteq P_A$ satisfies determinacy iff it satisfies strong determinacy.

Proof. Strong determinacy clearly implies determinacy. For the converse, assume σ is deterministic, $s_1 m_1^{S_1}, s_2 m_2^{S_2} \in \sigma$ and $s_1 \sim s_2$, say with $s_1 = \pi \cdot s_2$. By equivariance, $\pi \cdot (s_2 m_2^{S_2}) = s_1 m_2'^{S'_2} \in \sigma$, where $m_2'^{S'_2} = \pi \cdot m_2^{S_2}$. Hence, by determinacy, $s_1 m_1^{S_1} \sim s_1 m_2'^{S'_2}$ and so, using $s_2 m_2^{S_2} \sim s_1 m_2'^{S'_2}$, we get $s_1 m_1^{S_1} \sim s_2 m_2^{S_2}$. $\square$

We shall use the following lemmas, the proofs of which are deferred to Section 5.3. Suppose $\sigma : A \to B$ and $\tau : B \to C$ are strategies.

Lemma 5.25. Let $u_1 m_1^{S_1}, u_2 m_2^{S_2} \in \mathit{Int}(ABC)$ and $u_1 \sim u_2$. If m_1 is a P-move in AB (a P-move in BC, an O-move in AC) and $(u_1 m_1^{S_1} \restriction_\gamma AB) \sim (u_2 m_2^{S_2} \restriction_\gamma AB)$ (resp. $\restriction_\gamma BC$, $\restriction_\gamma AC$) then $u_1 m_1^{S_1} \sim u_2 m_2^{S_2}$.

Lemma 5.26. Let $u_1, u_2 \in \mathit{Int}(ABC)$ with $(u_i \restriction_\gamma AB) \in \sigma$ and $(u_i \restriction_\gamma BC) \in \tau$, for $i = 1, 2$, and suppose there is $u'_1 \sqsubseteq u_1$ such that $(u'_1 \restriction_\gamma AC) \sqsubseteq (u_2 \restriction_\gamma AC)$. Then, there is $u'_2 \sqsubseteq u_2$ such that $u'_1 \sim u'_2$.

Proposition 5.27. $\sigma;\tau$ is a strategy in $A \to C$.

Proof. Even-prefix closure of $\sigma;\tau$ follows from the respective property for σ and τ and the switching conditions of Lemma 5.17. Equivariance follows from the fact that all operations applied for composition are equivariant. Now let $sm_1^{S_1}, sm_2^{S_2} \in \sigma;\tau$ with $sm_i^{S_i} = u_i m_i^{S_i'} \upharpoonright_\gamma AC$. By previous lemma, $u_1 \sim\sqsubseteq u_2 \sim\sqsubseteq u_1$ and hence $u_1 \sim u_2$. Suppose WLOG that m_1 is in A. Then, by switching, u_1, u_2 end in AB and m_2 is also in A. By nominal determinacy of σ, $(u_1 m_1^{S_1'} \upharpoonright_\gamma AB) \sim (u_2 m_2^{S_2'} \upharpoonright_\gamma AB)$ and thus, by penultimate lemma, $u_1 m_1^{S_1'} \sim u_2 m_2^{S_2'}$. $\square$

5.1.5 The category of games

We have seen the ingredients of games, namely arenas, plays and strategies, and how the latter two notions can support a natural notion of composition. Our modelling approach amounts to translating terms of the language to a universe of games: more precisely, a category with arenas as objects and strategies as morphisms.

The identity morphisms of our category of games are given by:

$$\mathsf{id}_A = \{\, s \in P_{A\to A} \mid s \upharpoonright A_l = s \upharpoonright A_r \,\}$$

where A_l above denotes the left A in $A \to A$ (and dually for A_r). This strategy behaviour, whereby P copies moves from one sub-areana A to another, is called a *copycat*.

We can immediately verify the following.

Proposition 5.28. For any $\sigma : A \to B$, we have that $\sigma = \mathsf{id}_A;\sigma = \sigma;\mathsf{id}_B$.

To show that strategy composition is associative and, hence, games as above indeed form a category, is significantly more intricate. Starting from a triple of strategies $A \xrightarrow{\sigma} B \xrightarrow{\tau} C \xrightarrow{\rho} D$, one defines interactions over the 4 arenas $ABCD$ and shows that plays in $(\sigma;\tau);\rho$ and $\sigma;(\tau;\rho)$ can be seen as projections on AD of the same interactions on $ABCD$. As a full proof would need to go into rather tedious lengths, we shall omit it and refer the reader to [28] for a similar account.

Proposition 5.29. Given strategies $\sigma : A \to B$, $\tau : B \to C$ and $\rho : C \to D$, we have $(\sigma;\tau);\rho = \sigma;(\tau;\rho)$.

With compositionality in hand, we have our well-defined category.

Definition 5.30. Let $\mathcal{G}$ be the category whose objects are arenas, and morphisms from A to B are the strategies on $A \to B$.

To gain some first insight on the structure that is available in $\mathcal{G}$, we shall probe the $A + B$ construction in order to obtain coproduct. For any pair of strategies $\sigma : A \to C$ and $\tau : B \to C$, we define:

$$[\sigma,\tau] : A + B \to C \;=\; \sigma \cup \tau$$

where the union on the right-hand side takes care of any re-indexing of A and B moves within $A + B$.

Adding the injection morphisms (with appropriate re-indexing)

$$\mathsf{in}_l : A \to A + B \;=\; \mathsf{id}_A \quad \text{and} \quad \mathsf{in}_r : B \to A + B \;=\; \mathsf{id}_B$$

we can straighforwardly verify the coproduct equations:

- $\sigma = A \xrightarrow{\mathsf{in_l}} A + B \xrightarrow{[\sigma,\tau]} C$ and $\tau = B \xrightarrow{\mathsf{in_r}} A + B \xrightarrow{[\sigma,\tau]} C$,
- for any $\rho : A + B \to C$, $\rho = [\mathsf{in}_l;\rho, \mathsf{in}_r;\rho]$.

Hence, $A + B$ is the coproduct of A and B in $\mathcal{G}$.

Finally, note that $+$ distributes over $\otimes$ via the isomorphism:

$$\mathsf{dist}_{A,B,C} : (A + B) \otimes C \xrightarrow{\cong} (A \otimes C) + (B \otimes C)$$

which plays like $\mathsf{id}_{A\otimes C}$ or $\mathsf{id}_{B\otimes C}$ respectively depending on whether the initial move is of the form $(\mathrm{i}_A, \mathrm{i}_C)$ or $(\mathrm{i}_B, \mathrm{i}_C)$.

5.1.6 Modelling value terms

Having defined the category $\mathcal{G}$ of games, we start looking at the translation of (supported) terms into $\mathcal{G}$. We shall first examine values. Let us recall that they are given by the grammar

$$V \;::=\; () \mid i \mid x \mid a \mid \langle V,V\rangle \mid \lambda x^{\theta}.M.$$

These will be identified with strategies over a subcategory of $\mathcal{G}$, written $\mathcal{G}_{tti}$, obeying some further conditions which we shall discuss next.

Consider a value term $\mathrm{U}, \Gamma \vdash V : \theta$ and its denotation $[\![V]\!] : [\![\mathrm{U}, \Gamma]\!] \to [\![\theta]\!]$. Since V is already a value, for any initial move i^S played by O, it must be the case that P replies with an initial move in $[\![\theta]\!]$: there is no question to be played here, we only need to return the value V. This leads us to the first condition for value terms.

Definition 5.31. Given a strategy $\sigma : A$, we call σ ***total*** if for all $\mathrm{i}^S \in P_A$ there is $\mathrm{i}^S m^{S'} \in \sigma$, with m an answer (to i).

Let us restrict for a moment our scenario to the typing context $a, x : \mathsf{int} \vdash \theta \to \theta'$ with $a \in \mathbb{A}_{\mathsf{int}}$. Any value term of this type must be of the form $\lambda x^{\theta}.M$ and corresponds to some strategy:

$$\sigma : [\![\theta]\!] \otimes \mathbb{A}_{\mathsf{int}} \longrightarrow [\![\theta]\!] \Rightarrow [\![\theta']\!]$$

Suppose O plays an initial move $(a, \mathrm{i}_\theta)^S$ where, say, $S = \{(a, 5)\}$. As we argued above, σ replies to the initial move with $\dagger$, which is the initial move of $[\![\theta]\!] \Rightarrow [\![\theta']\!]$ and signifies that P plays a function of that type. Passing from $(a, \mathrm{i}_\theta)^S$ to $\dagger$, P is not going to make any changes to the store: as $\lambda x.M$ is already a value, no store update is involved in returning it. Hence, P should in fact play $\dagger^S$.

While our last observation bans store updates when moving from $(a, \mathrm{i}_\theta)^S$ to $\dagger$, there is more behaviour that needs to be excluded here: there is also no *reading* from the store. The latter is more subtle to formulate as it essentially describes a uniformity condition: P should have the same behaviour for any other initial value of a other than 5. Put otherwise, P bases his reaction to $(a, \mathrm{i})^S$ just on (a, i) — the initial store S may as well be considered as invisible at that point. An even subtler restriction we need to impose on σ is the following. The modelled value is not only free of observable stateful behaviour in its second move (i.e. interaction with S) but, moreover, does not exert any private stateful effect. That is to say, the different calls of $\lambda x.M$ are independent, in that each call has a behaviour that does not depend on other calls, and its own private names as well.

To model the notion of "call" in such a situation, we introduce the notion of *thread*. Suppose $s \in P_A$ is a play of the form $s = \mathrm{i}^S m^{S'} s'$

with s' non-empty and m an answer. We call each move n in s' which is justified by m a *threader move.* Each move in s' is assigned a unique threader move from s' as follows. For each $s = \mathrm{i}^S m^{S'} s'$ with m an answer, define $\mathsf{thrr}(s)$ inductively by

$$\begin{aligned} \mathsf{thrr}(\mathrm{i}^S \overparen{m^{S'} s_1 n}^T) &= n \\ \mathsf{thrr}(\mathrm{i}^S m^{S'} s_1 \overparen{p^{S''} s_2 o}^T) = \mathsf{thrr}(\mathrm{i}^S m^{S'} s_1\, p^{S''}) &= \mathsf{thrr}(\mathrm{i}^S m^{S'} s_1) \end{aligned}$$

and, for any move m' in s', let its threader move be $\mathsf{thrr}(\mathrm{i}^S m^{S'} s'_{m'})$, where $s'_{m'}$ is the prefix of s' ending in m'. We now define the ***thread-view*** of s, written $\lceil s \rceil$, as:

$$\lceil \mathrm{i}^S m^{S'} s' \rceil = \gamma(\mathrm{i}\, m\, s'')$$

where s'' is the subsequence of s' containing all of its moves whose threader move is $\mathsf{thrr}(\mathrm{i}^S m^{S'} s')$. Thus, the thread-view of s contains all the moves of the *current thread* of s, where threads are specified by threader moves. Note in particular that only O is allowed to switch between threads.

We can now define the strategies that capture value behaviour.

Definition 5.32. A strategy $\sigma : A$ is called ***thread-independent*** if:

- for each $\mathrm{i} \in I_A$ there is a unique answer $m_\mathrm{i} \in M_A$ such that whenever $\mathrm{i}^S m^{S'} \in \sigma$ then $S' = S$ and $m = m_\mathrm{i}$;
- for all $s_1 m_1^{S_1} n_1^{T_1}, s_2 \in \sigma$ and $s_2 m_2^{S_2} \in P_A$ such that $\lceil s_1 m_1^{S_1} \rceil = \lceil s_2 m_2^{S_2} \rceil$, there is $s_2 m_2^{S_2} n_2^{T_2} \in \sigma$ such that $\lceil s_1 m_1^{S_1} n_1^{T_1} \rceil \sim \lceil s_2 m_2^{S_2} n_2^{T_2} \rceil$;
- for all $s m^S n^T \in \sigma$ and $a \in \mathsf{dom}(S) \setminus \nu(\lceil s m^S \rceil)$ we have $a \notin \nu(\lceil s m^S n^T \rceil)$ and $T(a) = S(a)$.

It is called ***total thread-independent (tti)*** if it is both total and thread-independent.

We can show that total strategies and thread-independent strategies compose.[7] We shall write $\mathcal{G}_{tti}$ for the wide subcategory of $\mathcal{G}$ of tti

[7]Closure under totality is straightforward. The case of thread-independence is more demanding and one uses the fact that such strategies can actually be represented by interleavings of single threads.

strategies. We can already see that the following strategies in $[\![\mathrm{U}, \Gamma \vdash \theta]\!]$ are tti.

$$[\![\mathrm{U}, \Gamma \vdash ()]\!] = \{\, \mathrm{i}^S \star^S \,\}_{cl} \qquad [\![\mathrm{U}, \Gamma \vdash x_i : \theta_i]\!] = \{\, \mathrm{i}^S \, \mathrm{i}_{\theta_i}^S \,\}_{cl}$$
$$[\![\mathrm{U}, \Gamma \vdash i]\!] = \{\, \mathrm{i}^S i^S \,\}_{cl} \qquad [\![\mathrm{U}, \Gamma \vdash a_i : \mathbb{A}_{\zeta_i}]\!] = \{\, \mathrm{i}^S a_i'^S \,\}_{cl}$$

Here i ranges over initial moves of $[\![\mathrm{U}]\!] \otimes [\![\theta_1]\!] \otimes \cdots \otimes [\![\theta_n]\!]$ and is of the form $(a_1', \cdots, a_m', \mathrm{i}_{\theta_1}, \cdots, \mathrm{i}_{\theta_k})$, assuming $\Gamma = \{\, x_1 : \theta_1, \cdots, x_k : \theta_k \,\}$ and $\mathrm{U} = \{\, a_1, \cdots, a_n \,\}$.[8] In the rest of this section we shall focus on products and exponentials in order to model the remaining classes of values.

For any pair of arenas A and B, define the projection strategy

$$\pi_1 : A \otimes B \to A = \{\, (\mathrm{i}_A, \mathrm{i}_B)^S \, \mathrm{i}_A^S \, s \mid \mathrm{i}_A^S \, \mathrm{i}_A^S \, s \in \mathsf{id}_A \,\}_{cl}$$

and its dual $\pi_2 : A \otimes B \to B$. We also define strategies for rearranging product components:

$$\begin{aligned} \mathsf{sy}_{A,B} &: A \otimes B \to B \otimes A \\ &= \{\, (\mathrm{i}_A, \mathrm{i}_B)^S \, (\mathrm{i}_B, \mathrm{i}_A)^S \, s \mid (\mathrm{i}_A, \mathrm{i}_B)^S \, (\mathrm{i}_A, \mathrm{i}_B)^S \, s \in \mathsf{id}_{A \otimes B} \,\}_{cl} \\ \mathsf{as}_{A,B,C} &: A \otimes (B \otimes C) \to (A \otimes B) \otimes C \\ &= \{\, (\mathrm{i}_A, m)^S ((\mathrm{i}_A, \mathrm{i}_B), \mathrm{i}_C)^S s \mid (\mathrm{i}_A, m)^S (\mathrm{i}_A, m)^S s \in \mathsf{id}_{A \otimes (B \otimes C)} \,\}_{cl} \end{aligned}$$

with $m = (\mathrm{i}_B, \mathrm{i}_C)$. These are in fact isomorphisms (e.g. $\mathsf{sy}_{A,B}; \mathsf{sy}_{B,A} = \mathsf{id}_{A \otimes B}$) and therefore we will usually leave them implicit as "$\cong$" in diagrams. Given a strategy $\sigma : A \to B$, we can build the strategy:

$$\begin{aligned} \langle \sigma, \mathsf{id}_A \rangle : A \to B \otimes A \; = \; &\{\, \epsilon \,\} \cup \{\, \mathrm{i}_A^S \, s \in \sigma \mid \forall m^T \,\mathrm{in}\, s.\, m \in M_A \,\} \\ &\cup \{\, \mathrm{i}_A^S \, s \, (\mathrm{i}_B, \mathrm{i}_A)^{S'} s' \mid \mathrm{i}_A^S \, s \, \mathrm{i}_B^{S'} (s' @ B) \in \sigma, \gamma(\mathrm{i}_A^S \, \mathrm{i}_A^S (s' @ A)) \in \mathsf{id}_A \,\} \end{aligned}$$

where $s'@B$ is the subsequence of s' of moves whose threader move comes from the B component of $B \otimes A$, and similarly for $s'@A$. This defines a trivial strategy pairing which we can extend to arbitrary strategies $\sigma : A \to B$ and $\tau : A \to C$ as follows,

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

[8]Recall from Remark 5.4 that, while $\mathrm{U} = \{\, a_1, \cdots, a_n \,\}$ contains specific choices of names for each of its components, its semantic interpretation $[\![\mathrm{U}]\!]$ includes all nominal orbits of the sequence $(a_1, \cdots, a_n)$, i.e. all $(a_1', \cdots, a_n') \sim (a_1, \cdots, a_n)$.

and, for any $\sigma' : A' \to B'$, we let $\sigma \otimes \sigma' = A \otimes A' \xrightarrow{\langle \pi_1;\sigma,\pi_2;\sigma'\rangle} B \otimes B'$.

The defined structure suffices for capturing the call-by-value lambda-calculus fragment of $\mathsf{GroundML}$ (cf. [29]). The category $\mathcal{G}$ is symmetric premonoidal [37] under $\otimes$, and tti morphisms are *central*: for all $\sigma : A \to B, \sigma' : A' \to B'$, if σ is tti then $\sigma \otimes \sigma' = \sigma \otimes \mathsf{id}; \mathsf{id} \otimes \sigma' = \mathsf{id} \otimes \sigma'; \sigma \otimes \mathsf{id}$. In fact, we can show $(\mathcal{G}, \mathcal{G}_{tti})$ to be a closed Freyd category [38].

Proposition 5.33. Projections and pairings, defined as above, yield products in $\mathcal{G}_{tti}$. Moreover, for all A, B, C, there is an isomorphism $\Lambda : \mathcal{G}(A \otimes B, C) \;\cong\; \mathcal{G}_{tti}(A, B \Rightarrow C)$ natural in B, C.

For instance, for all A, B, we have an *evaluation morphism*

$$\mathsf{ev}_{A,B} : (A \Rightarrow B) \otimes A \to B \;=\; \Lambda^{-1}(\mathsf{id}_{A\Rightarrow B})\,.$$

We can now complete the translation of values by setting:

$$\begin{aligned}
[\![\langle V, V'\rangle]\!] &= \; [\![\mathrm{U}, \Gamma]\!] \xrightarrow{\langle [\![V]\!], [\![V']\!]\rangle} [\![\theta]\!] \otimes [\![\theta']\!] \\
[\![\lambda x^{\theta}.M]\!] &= \; [\![\mathrm{U}, \Gamma]\!] \xrightarrow{\Lambda([\![M]\!])} [\![\theta]\!] \Rightarrow [\![\theta']\!]
\end{aligned}$$

given $[\![M]\!] : [\![\mathrm{U}, \Gamma, x : \theta]\!] \to [\![\theta']\!]$. Of course, this assumes the translation of general terms has been established, which we examine next.

5.1.7 The model for GroundML

The developments of the previous section allow us to model the call-by-value λ-calculus fragment of $\mathsf{GroundML}$. What remains to be modelled is the stateful part of it. More concretely, we need strategies for reference creation, dereferencing and update. We define these below,

$$\begin{aligned}
\mathsf{new}_{\zeta} : [\![\zeta]\!] \to \mathbb{A}_{\zeta} &= \{\, v^{S}\, a^{S[a\mapsto v]} \mid a \in \mathbb{A}_{\zeta} \setminus \nu(v, S) \,\}_{cl} \\
\mathsf{get}_{\zeta} : \mathbb{A}_{\zeta} \to [\![\zeta]\!] &= \{\, a^{S}\, v^{S} \mid v = S(a) \,\}_{cl} \\
\mathsf{set}_{\zeta} : \mathbb{A}_{\zeta} \otimes [\![\zeta]\!] \to 1 &= \{\, (a, v)^{S} \star^{S[a\mapsto v]} \,\}_{cl}
\end{aligned}$$

for any ground type ζ.

The full translation of $\mathsf{GroundML}$ into $\mathcal{G}$ is given inductively as below. Suppose that $|\mathrm{U}| = n$ and $\Gamma = \{\, x_1 : \theta_1, \cdots, x_k : \theta_k \,\}$. We write A for the arena $[\![\mathrm{U}]\!] \otimes [\![\theta_1]\!] \otimes \cdots \otimes [\![\theta_k]\!]$.

- $[\![\mathrm{U},\Gamma \vdash () : \mathsf{unit}]\!] = A \xrightarrow{\mathsf{t}} 1$, where $\mathsf{t} = \{\, \mathrm{i}_A^S \star^S \mid \mathrm{i}_A^S \in P_{A\to 1} \,\}_{cl}$.
- $[\![\mathrm{U},\Gamma \vdash i : \mathsf{int}]\!] = A \xrightarrow{\mathsf{t}} 1 \xrightarrow{\mathsf{i}} \mathbb{Z}$, where $\mathsf{i} = \{\, \star\, i \,\}_{cl}$.
- $[\![\mathrm{U},\Gamma \vdash x_j : \theta_j]\!] = A \xrightarrow{\pi_{n+j}} [\![\theta_j]\!]$.
- $[\![\mathrm{U},\Gamma \vdash M_1 \oplus M_2 : \mathsf{int}]\!] = A \xrightarrow{\langle [\![M_1]\!],[\![M_2]\!]\rangle} \mathbb{Z}\otimes\mathbb{Z} \xrightarrow{\sigma_\oplus} \mathbb{Z}$, where $\sigma_\oplus = \{\, (i_1,i_2)\,(i_1\oplus i_2) \mid i_1,i_2 \in \mathbb{Z} \,\}_{cl}$.
- $[\![\mathrm{U},\Gamma \vdash \mathsf{if}\, M\, \mathsf{then}\, N_1\, \mathsf{else}\, N_0 : \theta]\!] = A \xrightarrow{\langle [\![M]\!],\mathsf{id}\rangle} \mathbb{Z}\otimes A \xrightarrow{\mathsf{if}\otimes\mathsf{id}} (1+1)\otimes A \xrightarrow{\cong} A + A \xrightarrow{[[\![N_1]\!],[\![N_0]\!]]} [\![\theta]\!]$, where $\mathsf{if} = \{\, 0 \star_r \,\}_{cl} \cup \{\, i \star_l \mid i \neq 0 \,\}$.
- $[\![\mathrm{U},\Gamma \vdash M_1 = M_2 : \mathsf{int}]\!] = A \xrightarrow{\langle [\![M_1]\!],[\![M_2]\!]\rangle} \mathbb{A}_\zeta \otimes \mathbb{A}_\zeta \xrightarrow{\sigma_=} \mathbb{Z}$, where $\sigma_= = \{\, (a,a)\,1 \mid a \in \mathbb{A}_\zeta \,\}_{cl} \cup \{\, (a,b)\,0 \mid a,b \in \mathbb{A}_\zeta, a \neq b \,\}$.
- $[\![\mathrm{U},\Gamma \vdash \mathsf{ref}(M) : \mathsf{ref}\zeta]\!] = A \xrightarrow{[\![M]\!]} [\![\zeta]\!] \xrightarrow{\mathsf{new}} \mathbb{A}_\zeta$.
- $[\![\mathrm{U},\Gamma \vdash\, !M : \zeta]\!] = A \xrightarrow{[\![M]\!]} \mathbb{A}_\zeta \xrightarrow{\mathsf{get}} [\![\zeta]\!]$.
- $[\![\mathrm{U},\Gamma \vdash M := N : \mathsf{unit}]\!] = A \xrightarrow{\langle [\![M]\!],[\![N]\!]\rangle} \mathbb{A}_\zeta \otimes [\![\zeta]\!] \xrightarrow{\mathsf{set}} 1$.
- $[\![\mathrm{U},\Gamma \vdash MN : \theta']\!] = A \xrightarrow{\langle [\![M]\!],[\![N]\!]\rangle} ([\![\theta]\!] \Rightarrow [\![\theta']\!]) \otimes [\![\theta]\!] \xrightarrow{\mathsf{ev}} [\![\theta']\!]$.
- $[\![\mathrm{U},\Gamma \vdash \lambda x^\theta.M : \theta \to \theta']\!] = \Lambda([\![M]\!] : A \otimes [\![\theta]\!] \to [\![\theta']\!])$.
- $[\![\mathrm{U},\Gamma \vdash \mathsf{while}(M) : \mathsf{unit}]\!] = A \xrightarrow{\Lambda(\cong;[\![M]\!])} 1 \Rightarrow \mathbb{Z} \xrightarrow{\mathsf{wh}} 1$, where $\mathsf{wh} = \{\, \dagger \star i_1 \star i_2 \cdots \star i_n \star 0 \star' \mid n \geq 0, i_1, \cdots, i_n > 0 \,\}_{cl}$.

It is worth commenting briefly on the function of $\mathsf{wh} : 1 \Rightarrow \mathbb{Z} \to 1$. The strategy replies to the initial $\dagger$ move by a question $\star$, thus querying the 1 component of $1 \Rightarrow \mathbb{Z}$. If O answers 0 then P replies with the answer $\star'$, which is played in the right-hand-side 1 and answers the initial question $\dagger$. On the other hand, if O answers some $i_1 > 0$ then P asks another $\star$, and so on, until O plays the answer 0.

Example 5.34. We have already given the denotations of example terms in Examples 5.8, 5.9, 5.22, 5.23. We show how to derive the one for gen. Omitting the defined $\mathsf{let\text{-}in}$ notation, gen can be written as:

$$\mathsf{gen} \;\equiv\; \lambda z^{\mathsf{int}}.\,(\lambda x^{\mathsf{ref\,int}}.\,x := z; x)(\mathsf{ref}(0))$$

We construct its denotation inductively as follows.

- Let $\sigma_0 = [\![\mathsf{ref}(0)]\!] = \; 1 \xrightarrow{\mathsf{0}} \mathbb{Z} \xrightarrow{\mathsf{new}} \mathbb{A}_{\mathsf{int}}$.
- $\sigma_1 = [\![x := z; x]\!] = \; \mathbb{Z} \otimes \mathbb{A}_{\mathsf{int}} \xrightarrow{\langle \cong ; \mathsf{set}, \pi_2 \rangle} 1 \otimes \mathbb{A}_{\mathsf{int}} \xrightarrow{\cong} \mathbb{A}_{\mathsf{int}}$.
- $\sigma_2 = [\![\lambda x.\, x := z; x]\!] = \; \mathbb{Z} \xrightarrow{\Lambda(\sigma_1)} \mathbb{A}_{\mathsf{int}} \Rightarrow \mathbb{A}_{\mathsf{int}}$.
- $\sigma_3 = [\![(\lambda x.\, x := z; x)(\mathsf{ref}(0))]\!] = \; \mathbb{Z} \xrightarrow{\langle \sigma_2, \mathsf{t}; \sigma_0 \rangle} (\mathbb{A}_{\mathsf{int}} \Rightarrow \mathbb{A}_{\mathsf{int}}) \otimes \mathbb{A}_{\mathsf{int}} \xrightarrow{\mathsf{ev}} \mathbb{A}_{\mathsf{int}} \overset{(*)}{=} \mathbb{Z} \xrightarrow{\langle \mathsf{id}, \mathsf{t}; \sigma_0 \rangle} \mathbb{Z} \otimes \mathbb{A}_{\mathsf{int}} \xrightarrow{\sigma_1} \mathbb{A}_{\mathsf{int}}$, using $\sigma_2 = \Lambda(\sigma_1)$ in $(*)$.
- $[\![\mathsf{gen}]\!] = \; 1 \xrightarrow{\Lambda(\cong ; \sigma_3)} \mathbb{Z} \Rightarrow \mathbb{A}_{\mathsf{int}}$.

The other terms are translated in a similar manner. We leave them as an exercise.

5.2 Full abstraction

In this section we culminate with our main result: two GroundML terms are equivalent if, and only if, their strategy denotations contain the same complete plays:

$$M \cong N \iff \mathsf{comp}([\![M]\!]) = \mathsf{comp}([\![N]\!])$$

where a play $s \in P_A$ is called ***complete*** if all questions in s have an answer in s (i.e. there is no pending question), and $\mathsf{comp}(\sigma) = \{\, s \in \sigma \mid s \text{ complete} \,\}$.

5.2.1 Correctness, adequacy and soundness

For the soundness direction ($\Leftarrow$), the argument navigates through two sub-results:

- Correctness: the model is coherent with the operational semantics — every term has the same denotation as its reducts.

- Adequacy: the model reflects divergence — if M diverges then its denotation is the empty strategy $\{\epsilon\}$.

We therefore start with correctness. Since the result involves configurations from the operational semantics, which are pairs of stores and supported terms, we need to be able to give denotations of such pairs.

For any store S and supported term M such that $\nu(M) \subseteq \mathsf{dom}(S)$, we define the terms:

$$\begin{aligned} S \rhd M &\equiv a_1 := S(a_1); \cdots ; a_n := S(a_n); M \\ \mathsf{new}\, S \,\mathsf{in}\, M &\equiv \mathsf{let}\, x_1 = \mathsf{ref}(S(a_1))\,\mathsf{in} \\ &\qquad \cdots\ \mathsf{let}\, x_n = \mathsf{ref}(S(a_n)[\vec{x}/\vec{a}]_1^{n-1})\ \mathsf{in}\ M[\vec{x}/\vec{a}]_1^n \end{aligned}$$

where $a_1, \cdots, a_n$ some ordering of $\mathsf{dom}(S)$ that is type-increasing (i.e. all $a \in \mathbb{A}_{\mathsf{int}}$ come first, followed by $a \in \mathbb{A}_{\mathsf{ref\,int}}$, etc.), and $M[\vec{x}/\vec{a}]_1^k$ is the term obtained by replacing each a_i with the fresh variable x_i, for $1 \leq i \leq k$.

Stores are important only in some reduction rules (Figure 3.2), which we single out as they will need special treatment. We do the same for the context rule. Let us list the rules below.

$$\begin{array}{rcll} (!a, S) & \longrightarrow & (S(a), S) & & (\textsc{drf}) \\ (a := V, S) & \longrightarrow & ((), S[a \mapsto V]) & & (\textsc{asn}) \\ (\mathsf{ref}(V), S) & \longrightarrow & (a', S[a' \mapsto V]) & a' \notin \mathsf{dom}(S) & (\textsc{new}) \end{array}$$

$$\frac{(M, S) \longrightarrow (M', S')}{(E[M], S) \longrightarrow (E[M'], S')} \qquad (\textsc{ctx})$$

For each reduction step $(M, S) \to (M', S')$, let us write $(M, S) \xrightarrow{\kappa} (M', S')$ if the axiom rule used for that step is κ (all reduction rules are axioms apart from $(\textsc{ctx})$). The following lemma is then pivotal for proving correctness.

Lemma 5.35. Suppose $(M, S) \xrightarrow{\kappa} (M', S')$.

1. If $\kappa \notin \{\textsc{drf}, \textsc{asn}, \textsc{new}\}$ then $[\![M]\!] = [\![M']\!]$.
2. If $\kappa \in \{\textsc{drf}, \textsc{asn}\}$ then $[\![S \rhd M]\!] = [\![S' \rhd M']\!]$.
3. If $\kappa = \textsc{new}$ then $[\![M]\!] = [\![\mathsf{new}\, \{(a', V)\}\, \mathsf{in}\, M']\!]$.

Proof. The proof is by induction on the derivation of $(S, M) \xrightarrow{\kappa} (S', M')$. For the base case, we need to check that the axiom rules satisfy the above conditions. For point 1, $[\![M]\!] = [\![M']\!]$ follows from the properties established for $\mathcal{G}$, e.g. for β-reduction rules we use Proposition 5.33, and the only interesting case is the one for $\mathsf{while}(M)$, which is shown using the equality:

$$\mathsf{wh} = 1 \Rightarrow \mathbb{Z} \xrightarrow{\langle\cong;\mathsf{ev};\mathsf{if},\mathsf{id}\rangle} (1+1) \otimes (1{\Rightarrow}\mathbb{Z}) \xrightarrow{\cong} (1{\Rightarrow}\mathbb{Z}) \otimes (1{\Rightarrow}\mathbb{Z}) \xrightarrow{[\mathsf{wh},\mathsf{t}]} 1$$

The equalities for points 2 and 3 are straightforward from the definitions of $\mathsf{get}, \mathsf{set}$ and new.

For the induction step, we need to show that the above equalities are preserved through evaluation contexts E. For 1, the fact that $[\![M]\!] = [\![M']\!]$ implies $[\![E[M]]\!] = [\![E[M']]\!]$ is clear from compositionality. For 2, we use the equality $[\![S \triangleright E[M]]\!] = [\![S \triangleright E[S \triangleright M]]\!]$ and compositionality. For 3, since a' is fresh for E, we can show $[\![\mathsf{new}\,\{(a', V)\}\,\mathsf{in}\,E[M]]\!] = [\![E[\mathsf{new}\,\{(a', V)\}\,\mathsf{in}\,M]]\!]$. □

Proposition 5.36 (Correctness)**.** For any supported term $\mathrm{U}, \Gamma \vdash M : \theta$ and store S with support U, if $(M, S) \longrightarrow (M', S')$ then $[\![\mathsf{new}\,S\,\mathsf{in}\,M]\!] = [\![\mathsf{new}\,S'\,\mathsf{in}\,M']\!]$.

Proof. The claim directly follows from the previous lemma. We only need to check that, in case 2, if $[\![S \triangleright M]\!] = [\![S' \triangleright M']\!]$ then $[\![\mathsf{new}\,S\,\mathsf{in}\,M]\!] = [\![\mathsf{new}\,S'\,\mathsf{in}\,M']\!]$ and, for case 3, that the order of assignments in the $\mathsf{new}\,S\,\mathsf{in}\,M$ construct does not matter semantically. □

We next look into adequacy. We shall argue as follows. Given a diverging term $\vdash M : \mathsf{unit}$, we know that its diverging computation must involve an infinite number of unfoldings of while loops, as the calculus is otherwise strongly normalising. Hence, if we instrument such a term so that in each loop unfolding it increases an internal counter then the value of that counter cannot be bounded. Making that counter visible in the semantics would then enforce that the semantics of M cannot contain the play $\{\star\star\}$.

Proposition 5.37 (Computational adequacy)**.** For any term $\vdash M : \mathsf{unit}$ if $M \not\Downarrow$ then $[\![M]\!] = \{\epsilon\}$.

Proof. Suppose, for the sake of contradiction, that $M \not\Downarrow$ and $[\![\vdash M]\!] = \{\epsilon, \star\star\}$. For any term $\mathrm{U}, \Gamma \vdash N : \theta$ and $a \in \mathbb{A}_{\mathsf{int}} \setminus \mathrm{U}$ construct $\mathrm{U}, a, \Gamma \vdash N_a$ by recursively replacing each subterm of N of the shape $\mathsf{while}(N')$ with $\mathsf{while}(a := (!a + 1); N')$. Observe that each $s \in [\![\mathrm{U}, \Gamma \vdash N]\!]$ induces some $s' \in [\![\mathrm{U}, a, \Gamma \vdash N_a]\!]$ 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. Then, for each $i \in \mathbb{Z}$ take N_i to be the term $\mathsf{let}\, x = \mathsf{ref}(i) \,\mathsf{in}\, N_a[x/a]; !x$, for some fresh variable x. Because $\star\star \in [\![\vdash M]\!]$, we shall have $\star j \in [\![\vdash M_0]\!]$ for some $j \in \mathbb{Z}$.

On the other hand, each play corresponding to N_i is obtained from a play s' for N_a such that the initial value of a is i, a appears in s' only in stores (and in a single place of the initial move) and O never changes the value of a. Moreover, P never decreases the value of a in s'. Thus, if si'^S is a complete play of $[\![N_i]\!]$, then $i' \geq i$. We shall find a term contradicting this by considering the infinite reduction sequence of $(\emptyset, M)$. It must have infinitely many while-loop unfoldings, so suppose $(\emptyset, M) \twoheadrightarrow (S, M')$ in $j+1$ such unfoldings. Then, we obtain $(\emptyset, M_0) \twoheadrightarrow (S_a, (M')_a; !a)$ with $S_a = \{(a, j+1)\} \uplus S$. By Proposition 5.36 and $\star j \in [\![\vdash M_0]\!]$, we have $\star j \in [\![\vdash \mathsf{new}\, S_a \,\mathsf{in}\, (M')_a; !a]\!] = [\![\vdash (\mathsf{new}\, S \,\mathsf{in}\, M')_{j+1}]\!]$, a contradiction. □

We can therefore prove the main result of this section.

Theorem 5.38 (Soundness)**.** For any pair of terms $\Gamma \vdash M, N : \theta$ if $\mathsf{comp}([\![M]\!]) = \mathsf{comp}([\![N]\!])$ then $M \cong N$.

Proof. We show the contrapositive. Suppose C is some context such that $C[M] \Downarrow$ but $C[N] \not\Downarrow$. Then, by Propositions 5.36 and 5.37 respectively, $[\![C[M]]\!] = \{\epsilon, \star\star\}$ and $[\![C[N]]\!] = \{\epsilon\}$. Now observe that $[\![C[M]]\!] = [\![(\lambda f. C[f\langle x_1, \cdots, x_n\rangle])(\lambda x^{\vec{\theta}}.(\lambda \vec{x}.M)(\pi_1 x)\cdots(\pi_n x))]\!] = \Lambda([\![M]\!]); [\![C[f\langle\vec{x}\rangle]]\!]$, assuming $\Gamma = \{x_1 : \theta_1, \cdots, x_n : \theta_n\}$, so in particular there is some interaction sequence u in $1([\![\vec{\theta}]\!] \Rightarrow [\![\theta]\!])1$ such that $s = (u \restriction_\gamma 1([\![\vec{\theta}]\!] \Rightarrow [\![\theta]\!])) \in \Lambda([\![M]\!])$, $(u \restriction_\gamma ([\![\vec{\theta}]\!] \Rightarrow [\![\theta]\!])1) \in [\![C[f\langle\vec{x}\rangle]]\!]$, and $(u \restriction_\gamma 11) = \star\star$. By bracketing, u is complete and, hence, so is s. This means $s \in \mathsf{comp}(\Lambda([\![M]\!])) \setminus \mathsf{comp}(\Lambda([\![N]\!]))$, therefore $\mathsf{comp}([\![M]\!]) \neq \mathsf{comp}([\![N]\!])$. □

5.2.2 Definability and full abstraction

To show the converse of Theorem 5.38 we need to show that, whenever a term M produces a complete play s that term N cannot replicate, there is always an appropriate context C which can take the opportunity and interact with s to produce a complete play in $1 \to 1$. The latter will be shown via a stronger result which essentially is a *compact completeness property*: every finite strategy (up to name permutations) is in fact the denotation of some GroundML term.

The proof of completeness consists of two stages. First we show a decomposition lemma which demonstrates how a finitary strategy can be canonically decomposed into smaller ones. We will then use such decompositions to prove definability of finitary strategies by induction on the length of their longest plays.

Given a strategy $\sigma : A$ we let its set of *maximal orbits* be:

$$\mathcal{MO}(\sigma) = \{\, [s] \mid s \in \sigma \wedge \forall s' \in \sigma.\, s \sqsubseteq s' \implies s = s' \,\}$$

Conversely, for an even-length play $s \in P_A$, we let $\mathsf{epref}(s)$ be the set of plays:

$$\mathsf{epref}(s) = \bigcup \{\, [t] \mid t \in P_A \wedge t \sqsubseteq_{even} s \,\}$$

which is clearly a strategy in A. We call σ ***finitary*** if $\mathcal{MO}(\sigma)$ is finite and, in such a case, we let its size be:

$$|\sigma| = \max\{\, |s| \mid s \in \sigma \,\}$$

where $|s|$ is the length of the sequence s.

Lemma 5.39 (Decomposition). Let $\sigma : A \to B$ be a finitary strategy such that $A = A_1 \otimes \cdots \otimes A_k$ and $A_i = [\![\theta_i]\!]$ for each i (for some θ_i).

1. If $\mathcal{MO}(\sigma) = \{\, [s_1], \cdots, [s_n] \,\}$ then $\sigma = \bigcup_{i=1}^{n} \mathsf{epref}(s_i)$.
2. If $\sigma = \mathsf{epref}(\mathrm{i}^S m^{S'} s)$ with m a question in $A_i = C_1 \to C_2$ then:

$$\sigma = A \xrightarrow{\langle \pi_i, \sigma' \rangle} (C_1 \Rightarrow C_2) \otimes C_1 \otimes (C_2 \Rightarrow B) \xrightarrow{\mathsf{ev} \otimes \mathsf{id}} C_2 \otimes (C_2 \Rightarrow B) \xrightarrow{\cong ; \mathsf{ev}} B$$

   where $\pi_i : A \to C_1 \Rightarrow C_2$ is the i-th projection and $\sigma' : A \to C_1 \otimes (C_2 \Rightarrow B) = \mathsf{epref}(\mathrm{i}^S (m, \dagger)^{S'} s')$ whereby:

- m is in answer position (initial in C_1);
- assuming that $s = s_1 n^T s_2\, n'^{T'} s_3$ with n and n' being the answers of m and i respectively, $s' = s_1 n^T s_2\, n'^{T'} s_3$ with the proviso that n is now a question justified by † in $(m, \dagger)$, and n' answers n.[9]

3. If $\sigma = \mathsf{epref}(\mathrm{i}^S m^{S'} s)$ with m an answer and $a_1, \cdots, a_n$ is an enumeration of $\mathsf{P}(\mathrm{i}^S m^{S'} s)$ then
$$\sigma = A \xrightarrow{\langle \mathsf{id}, \tau\rangle} A \otimes \mathbb{A}_{\zeta_1} \otimes \cdots \otimes \mathbb{A}_{\zeta_n} \xrightarrow{\sigma'} B$$
where $\tau = \mathsf{epref}(\mathrm{i}^S\,(a_1, \cdots, a_n)^{S \uplus S'})$ with S' some minimal (fresh) store with $\{a_1, \cdots, a_n\} \subseteq \mathsf{dom}(S')$, and $\sigma' = \mathsf{epref}((\mathrm{i}, a_1, \cdots, a_n)^{S \uplus S'} S'[s])$ where $S'[s]$ is s with each move n^T replaced by $n^{S'[T]}$.

4. If $\sigma = \mathsf{epref}(\mathrm{i}^S m^{S'} s)$ with m an answer, $\mathsf{P}(\mathrm{i}^S m^{S'} s) = \emptyset$ and $a_1, \cdots, a_n$ is an enumeration of $\mathsf{O}(\mathrm{i}^S m^{S'} s) \setminus \nu(\mathrm{i}^S)$ with $a_i \in \mathbb{A}_{\zeta_i}$ for each i, then
$$\sigma = A \xrightarrow{\langle \mathsf{id}, \tau\rangle} A \otimes \mathbb{A}_{\mathsf{ref}\zeta_1} \otimes \cdots \otimes \mathbb{A}_{\mathsf{ref}\zeta_n} \xrightarrow{\sigma'} B$$
where $\tau = \mathsf{epref}(\mathrm{i}^S\,(a'_1, \cdots, a'_n)^{S \uplus S'})$, a'_i some fresh name in $\mathbb{A}_{\mathsf{ref}\zeta_i}$, S' is some minimal (fresh) store with $\{a'_1, \cdots, a'_n\} \subseteq \mathsf{dom}(S')$, and $\sigma' = \mathsf{epref}((\mathrm{i}, a'_1, \cdots, a'_n)^{S \uplus S'} S'[s[\vec{a}' \mapsto \vec{a}]])$ where $s[\vec{a}' \mapsto \vec{a}]$ is s with each move n^T replaced by $n^{T \uplus \{(a'_i, a_i)\,|\,a_i \in \mathsf{dom}(T)\}}$.

5. If $\sigma = \mathsf{epref}(\mathrm{i}^S m^{S'} s)$ with m an answer, $\mathsf{P}(\mathrm{i}^S m^{S'} s) = \emptyset$, $a_1, \cdots, a_n$ an enumeration of $\mathsf{dom}(S)$ and $\forall s' n^T \sqsubseteq_{odd} \mathrm{i}^S m^{S'} s.\ \gamma(\mathrm{i}^S n^T) = \mathrm{i}^S n^T$, then
$$\sigma = A \xrightarrow{\langle \mathsf{id}, \tau\rangle} A \otimes [\![\zeta_1]\!] \otimes \cdots \otimes [\![\zeta_n]\!] \otimes \mathbb{A}_{\mathsf{int}} \xrightarrow{\sigma'} B$$
where $\tau = \mathsf{epref}(\mathrm{i}^S(S(a_1), \cdots, S(a_n), a)^{S' \uplus \{(a,0)\}})$ with $a \in \mathbb{A}_{\mathsf{int}}$ some fresh name, and
$$\sigma' = \mathsf{epref}((\mathrm{i}, S(a_1), \cdots, S(a_n), a)^{S' \uplus \{(a,0)\}} m^{S' \uplus \{(a,0)\}} s')$$

[9] If m or i are not answered in s then s' is the respective prefix of the above s'.

is thread-independent, where s' is obtained from s by replacing each move n^T of its i-th thread by $n^{T \uplus \{(a,i)\}}$, except if n^T is the i-th threader move in which case it is replaced by $n^{T \uplus \{(a,i-1)\}}$.

6. If $\sigma = \mathsf{epref}(\mathrm{i}^S (m_1, m_2)^S s)$ is thread-independent then $\sigma = \langle \sigma; \pi_1, \sigma; \pi_2 \rangle$.

7. If $\sigma = \mathsf{epref}(\mathrm{i}^S \dagger^S s)$ is thread-independent then $\sigma = \Lambda(\Lambda^{-1}(\overline{\sigma})) \upharpoonright [\mathrm{i}^S]$, where $\overline{\sigma} = \sigma \cup \{ \mathrm{i}'^{S'} m^{S'} \in P_{A \to B} \mid m = \dagger \}$.

Proof. We concentrate on the more interesting cases, 2-5, as the others follow from general properties of $\mathcal{G}$. For 2, following the copycat links imposed by the ev strategies, we can see that indeed $\sigma = \langle \pi_i, \sigma' \rangle; \mathsf{ev} \otimes \mathsf{id}; \cong \mathsf{ev}$. We need also check that σ' is well-defined and for the latter it suffices to ensure that $t' = \mathrm{i}^S (m, \dagger)^{S'} s_1 n^T s_2\, n'^{T'} s_3$ is a valid play. Given that $t = \mathrm{i}^S m^{S'} s_1 n^T s_2\, n'^{T'} s_3$ is a play, it is easy to see that t' satisfies justification, alternation and bracketing. For visibility, we note that the pointer structure before the move $n'^{T'}$ is common in t and t'. Moreover, by Lemma 5.11 we have that n^T is in $\ulcorner \mathrm{i}^S m^{S'} s_1 n^T s_2 \urcorner$, thus $n'^{T'}$ can point to it. Finally, since $n'^{T'}$ points to n^T in t', there are more moves in the view for s_3-moves in t', than there are in t.

In cases 3 and 4 the equalities are straightforward. In 3, we delegate all P-name creation to τ, so σ' does not need to create any names. In 4, τ creates (private) locations where all O-names of s are stored, as soon as they are created, and remain there for the whole play. Note that σ' of 3 satisfies $\mathsf{P}(t) = \emptyset$ for all $t \in \sigma'$, whereas for the one from 4 we have $\forall t' n^{T'} \sqsubseteq_{odd} \mathrm{i}'^T t \in \sigma'.\ \gamma(\mathrm{i}'^T n^{T'}) = \mathrm{i}'^T n^{T'}$. Finally, for 5, we observe that the composition indeed yields σ. For thread-independence, the initial move is answered without any store changes, and each thread is indexed via the name a, which renders the second condition of Definition 5.32 trivial. Finally, the given conditions for σ ensure that, for all $t \in \sigma'$, $\nu(t) = \nu(\lceil t \rceil)$; hence, there is no interference of private names between different threads, simply because there are no thread-private names. □

Proposition 5.40 (Definability). Let $\sigma : [\![\mathrm{U}, \Gamma \vdash \theta]\!]$ be a finitary strategy. There exists $\mathrm{U}, \Gamma \vdash M_\sigma : \theta$ such that $\mathsf{comp}([\![\mathrm{U}, \Gamma \vdash M_\sigma : \theta]\!]) =$

$\mathsf{comp}(\sigma)$.

Proof. We construct M_σ by induction on $\|\sigma\| = (|\sigma|, 8-\phi(\sigma))$, ordered lexicographically, where $\phi(\sigma)$ the greatest index i such that σ satisfies the conditions of case i of the previous lemma, unless $\sigma = \mathsf{epref}(\mathrm{i}^S v^S)$ with $\nu(v) \subseteq \nu(\mathrm{i})$ and $v \in \{\star\} \cup \mathbb{A} \cup \mathbb{Z}$ in which case we set $\phi(\sigma) = 8$. Let us write $[\![\mathrm{U}, \Gamma \vdash \theta]\!]$ as $A \to B$. We do a case analysis on $\phi(\sigma)$.

In case 1, we can decompose $\sigma \neq \{\epsilon\}$ as $\bigcup_{i=1}^{n} \sigma_i$, $n > 1$, where each $\sigma_i = \mathsf{epref}(\mathrm{i}_i^{S_i} s_i)$. We use the induction hypothesis on the σ_i's to obtain terms M_{σ_i}. For each $\mathrm{i}_i^{S_i}$ we can construct a characteristic term $\mathrm{U}, \Gamma \vdash M_i : \mathsf{int}$ such that $[\![\mathrm{U}, \Gamma \vdash M_i : \mathsf{int}]\!] = \mathsf{epref}(\mathrm{i}_i^{S_i} 1) \cup \{\, \mathrm{i}^S 0 \mid \mathrm{i}^S \not\sim \mathrm{i}_i^{S_i} \,\}$ — M_i simply checks that the elements of Γ and U satisfy the specifications dictated by i_i and S_i. We can therefore take:

$$M_\sigma \;\equiv\; \mathsf{if}\, M_1 \,\mathsf{then}\, M_{\sigma_1} \,\mathsf{else\,if}\, M_2 \,\mathsf{then}\, M_{\sigma_2} \,\mathsf{else}\, \cdots \mathsf{if}\, M_n \,\mathsf{then}\, M_{\sigma_n} \,\mathsf{else}\, \mathsf{div}_\theta$$

where recall that div_θ is a divergent term of type θ. If $\sigma = \{\epsilon\}$ then $M_\sigma \equiv \mathsf{div}_\theta$.

For case 2, suppose $\sigma = \mathsf{epref}(\mathrm{i}^S m^{S'} s)$, m is a question in $[\![\theta']\!]$ with $(f : \theta') \in \Gamma$ and $\theta' = \theta_1 \to \theta_2$. Then, writing C_i for $[\![\theta_i]\!]$,

$$\sigma = A \xrightarrow{\langle \pi_i, \sigma' \rangle} (C_1 \Rightarrow C_2) \otimes C_1 \otimes (C_2 \Rightarrow B) \xrightarrow{\mathsf{ev} \otimes \mathsf{id}} C_2 \otimes (C_2 \Rightarrow B) \xrightarrow{\cong ; \mathsf{ev}} B$$

and, by definition, $\|\sigma'\| < \|\sigma\|$. Thus, taking the $M_{\sigma'}$ given by the IH, we set: $M_\sigma \;\equiv\; \mathsf{let}\, x = M_{\sigma'} \,\mathsf{in\,let}\, y = f(\pi_1 x) \,\mathsf{in}\, (\pi_2 x) y$.

For case 3, taking M' to be some term creating the fresh names $a_1, \cdots, a_n$ with default values and $\mathrm{U}, \Gamma, y_1 : \mathsf{ref}\zeta_1, \cdots, y_n : \mathsf{ref}\zeta_n \vdash M_{\sigma'} : \theta$ the one given by the IH for σ' (noting $\|\sigma'\| < \|\sigma\|$ as there are no P-names in σ'), we set: $M_\sigma \;\equiv\; \mathsf{let}\, x = M' \,\mathsf{in}\, M_{\sigma'}[\pi_1 x / y_1, \cdots, \pi_n x / y_n]$. Case 4 is similar.

For case 5, we take:

$$M \;\equiv\; \mathsf{let}\, x = \langle !a_1, \cdots, !a_n \rangle \,\mathsf{in\,let}\, y = \mathsf{ref}(0) \,\mathsf{in}\, M_{\sigma'}[\pi_1 x / y_1, \cdots, \pi_n x / y_n]$$

assuming $\mathrm{U}, \Gamma, y_1 : \zeta_1, \cdots, y_n : \zeta_n, y : \mathsf{ref\,int} \vdash M_{\sigma'} : \theta$ given by the IH.

For case 6, we apply the lemma repetitively and obtain $\sigma = \langle \sigma; \pi_1, \cdots, \sigma; \pi_n \rangle$, given $\theta = \theta_1 \times \cdots \times \theta_n$. We can apply the IH to each $\sigma; \pi_i$ to obtain some $\mathrm{U}, \Gamma \vdash M_i : \theta_i$, and take $M \;\equiv\; \langle M_1, \cdots, M_n \rangle$. For case 7, since $|\Lambda^{-1}(\overline{\sigma})| < |\sigma|$, and given $\theta = \theta_1 \to \theta_2$, we apply the IH on $\Lambda^{-1}(\overline{\sigma})$ to obtain $\mathrm{U}, \Gamma, x : \theta_1 \vdash M' : \theta_2$. Now, as in case 1, we define $\mathrm{U}, \Gamma \vdash M_1 : \mathsf{int}$ which checks if the initial move is of the form i^S, and set: $M \equiv \mathsf{if}\, M_1\, \mathsf{then}\, \lambda x.M'\, \mathsf{else}\, \mathsf{div}$. Finally, in case $\sigma = \mathsf{epref}(\mathrm{i}^s v^S)$ with v a singleton value with names from i, we set $M \equiv \mathsf{if}\, M_1\, \mathsf{then}\, \hat{v}\, \mathsf{else}\, \mathsf{div}$, where $\hat{v}$ the syntactic counterpart of v (given i) and M_1 as above. □

Theorem 5.41 (Full abstraction)**.** For any pair of terms $\Gamma \vdash M, N : \theta$, $M \cong N \iff \mathsf{comp}([\![M]\!]) = \mathsf{comp}([\![N]\!])$.

Proof. We only need to show completeness ($\Rightarrow$), and for that we show the contrapositive. Suppose s is some play in $\mathsf{comp}([\![M]\!]) \setminus \mathsf{comp}([\![N]\!])$. Assuming $\Gamma = \{x_1{:}\theta_1, \cdots, x_n{:}\theta_n\}$, we have $\star \dagger s \in \mathsf{comp}([\![M']\!]) \setminus \mathsf{comp}([\![N']\!])$, with $M' \equiv \lambda x^{\vec{\theta}}.(\lambda \vec{x}.M)(\pi_1 x) \cdots (\pi_n x)$ and similarly for N'. Moreover, $\dagger s \star$ is a complete play in $([\![\vec{\theta}]\!] \Rightarrow [\![\theta]\!]) \to 1$. By definability, there is some term $f : \vec{\theta} \to \theta \vdash M_{\dagger s \star} : 1$ such that $[\![M_{\dagger s\star}]\!] = \mathsf{epref}(\dagger s \star)$. Hence, $\star\star \in [\![M']\!]; [\![M_{\dagger s\star}]\!] = [\![C[M]]\!]$, given $C[-] \equiv \mathsf{let}\, f = \lambda x.(\lambda \vec{x}.\, -)(\pi_1 x) \cdots (\pi_n x)\, \mathsf{in}\, M_{\dagger s\star}$, and $[\![C[N]]\!] = \{\, \epsilon \,\}$. Then, adequacy implies $C[M] \Downarrow$ while by correctness we have $C[N] \not\Downarrow$. □

This concludes the technical component of the chapter. We can now use full abstraction to prove the equivalences from Example 3.7. We recall the pairs of equivalent terms below.

$$\begin{aligned}
M_1 &\equiv \mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\, \lambda y^{\mathsf{ref\,int}}.\, x = y \\
M_2 &\equiv \lambda y^{\mathsf{ref\,int}}.\, 0
\end{aligned}$$

$$\begin{aligned}
M_3 &\equiv \mathsf{let}\, x = \mathsf{ref}(0)\, \mathsf{in}\, \mathsf{let}\, c = \mathsf{ref}(0)\, \mathsf{in} \\
&\qquad\quad f(\lambda\_.\, \mathsf{if}\, !c = 0\, \mathsf{then}\, \mathsf{div}\, \mathsf{else}\, x); c := 1; \lambda y^{\mathsf{ref\,int}}.\, x = y \\
M_4 &\equiv f(\lambda\_.\, \mathsf{div}); \lambda y^{\mathsf{ref\,int}}.\, 0
\end{aligned}$$

$$\begin{aligned}
M_5 &\equiv \mathsf{let}\, x = \mathsf{ref}(\mathsf{ref}(0))\, \mathsf{in} \\
&\qquad\quad \lambda y^{\mathsf{ref\,int}}.\, \mathsf{let}\, z = !x\, \mathsf{in}\;\, \mathsf{if}\, y = z\, \mathsf{then}\, \mathsf{div}\, \mathsf{else}\, (x := \mathsf{ref}(0); z) \\
M_6 &\equiv \lambda y^{\mathsf{ref\,int}}.\, \mathsf{ref}(0)
\end{aligned}$$

Example 5.42. We already saw, when studying strategy composition, that $[\![M_1]\!] = [\![M_2]\!]$, which trivially implies $\mathsf{comp}([\![M_1]\!]) = \mathsf{comp}([\![M_2]\!])$ and hence $M_1 \cong M_2$ by the previous result.

For M_3, M_4 the typing context is interpreted as the prearena $((1 \Rightarrow \mathbb{A}_{\mathsf{int}}) \Rightarrow 1) \to (\mathbb{A}_{\mathsf{int}} \Rightarrow \mathbb{Z})$, which is depicted on the right (all $a, b \in \mathbb{A}_{\mathsf{int}}$, $i \in \mathbb{Z}$). We shall start from the denotation of M_3 and argue that it equals that of M_4. Consider first the open term (with context $f : (\mathsf{unit} \to \mathsf{ref\,int}) \to \mathsf{unit}, x : \mathsf{ref\,int}, c : \mathsf{ref\,int}$):

$$\begin{tikzcd}
 & \dagger \arrow[d, dash] \arrow[rr, dash] & & \dagger'' \arrow[d, dash] \\
 & \dagger' \arrow[dl, dash] \arrow[dr, dash] & & b \arrow[d, dash] \\
\star \arrow[dr, dash] & & \star' & i \\
 & a & &
\end{tikzcd}$$

$$M_3' \;\equiv\; f(\lambda\_.\,\mathsf{if}\,!c = 0\,\mathsf{then\,div\,else}\,x); c := 1; \lambda y^{\mathsf{ref\,int}}.\, x = y$$

Its denotation consists of even-length prefixes of plays of the form:[10]

$$\begin{tikzcd}[column sep=small]
(\dagger, a_s, a_c)^{S_0} & \dagger'^{S_0} & \star^{S_1} \arrow[l, bend right] & a_s^{S_1} & \cdots & \star^{S_n} \arrow[llll, bend right] & a_s^{S_n} & \star'^{T_0} \arrow[llllll, bend right] & \dagger''^{T_0'} \arrow[llllllll, bend right] & a_1^{T_1} \arrow[l, bend right] & i_1^{T_1} & \cdots & a_m^{T_m} \arrow[llll, bend right] & i_m^{T_m} & \cdots \\
OQ & PQ & OQ & PA & & OQ & PA & OA & PA & OQ & PA & & OQ & PA &
\end{tikzcd}$$

where $T_0' = T_0[a_c \mapsto 1]$ and, for all j, $i_j = 1$ if $a_j = a_s$ (and 0 otherwise), and $S_j(c) > 0$. We moreover have that

$$[\![M_3]\!] \;=\; A \xrightarrow{\langle \mathsf{id}, \mathsf{t}; \langle [\![\mathsf{ref}(0)]\!], [\![\mathsf{ref}(0)]\!] \rangle\rangle} A \otimes \mathbb{A}_{\mathsf{int}} \otimes \mathbb{A}_{\mathsf{int}} \xrightarrow{[\![M_3']\!]} \mathbb{A}_{\mathsf{int}} \Rightarrow \mathbb{Z}$$

where $A = (1 \Rightarrow \mathbb{A}_{\mathsf{int}}) \Rightarrow 1$, and the interaction sequences produced are of the form:

$$\dagger\ (\dagger, a_s, a_c)^{S_0}\ \dagger'^{S_0}\ \star^{S_1}\ a_s^{S_1}\ \cdots\ \star^{S_n}\ a_s^{S_n}\ \star'^{T_0}\ \dagger''^{T_0'}\ a_1^{T_1}\ i_1^{T_1}\ \cdots\ a_m^{T_m}\ i_m^{T_m}\ \cdots$$

The initial values of S_0 are thus determined by $[\![\mathsf{ref}(0)]\!]$ and, therefore, $S_0(a_s) = S_0(a_c) = 0$. Moreover, $\star^{S_1}$ is an O-move in $A(\mathbb{A}_{\mathsf{int}} \Rightarrow \mathbb{Z})$ and a_c a P-name for the LHS strategy, so by Laird conditions we have that $S_1(a_c) = S_0(a_c) = 0$. Nonetheless, in the denotation of $[\![M_3']\!]$ we

[10] Here we can observe the crucial role of visibility. As soon as $\dagger''^{T_0'}$ is played, the move $\dagger'^{S_0}$ is hidden from the view of O for the remaining play. That is, for each even-length play $t = (\dagger, a_s, a_c)^{S_0} \cdots \dagger''^{T_0'} \cdots$, $\ulcorner t \urcorner = (\dagger, a_s, a_c)^{S_0}\ \dagger''^{T_0'} \cdots$. Hence, O cannot ask a question $\star^S$ under $\dagger'^{S_0}$ anymore, which would allow him to gain access to the secret name a_s (as, at this point, $a_c = 1$).

stipulated that $S_1(a_c) > 0$, as otherwise P cannot play $a_s^{S_1}$ (the term diverges in this case). This contradiction means that $n = 0$, i.e. O never plays $\star^{S_1}$ but, rather, immediately answers $\dagger'^{S_0}$ with $\star'^{T_0}$. The latter also implies that a_s remains unavailable to O for the whole interaction and in particular $a_j \neq a$, for all j. Thus, the interaction actually looks like:

$$\dagger\ (\dagger, a_s, a_c)^{S_0}\ \dagger'^{S_0}\ \star'^{T_0}\ \dagger''^{T_0'}\ a_1^{T_1}\ 0^{T_1}\ \cdots\ a_m^{T_m}\ 0^{T_m}\ \cdots$$

and projects on $A(\mathbb{A}_{\mathsf{int}} \Rightarrow \mathbb{Z})$ as:

$$\dagger\ \dagger'\ \star'\ \dagger''\ a_1^{T_1'}\ 0^{T_1'}\ \cdots\ a_m^{T_m'}\ 0^{T_m'}\ \cdots$$

which are precisely the plays forming $[\![M_4]\!]$.

Now, following a very similar argument, we invite the reader to check that $M_5 \cong M_6$.

5.3 Chapter Appendix: deferred proofs

5.3.1 Lemmas for Proposition 5.20

Proof of Lemma 5.17. By mutual induction on the length of u'. Suppose m is a O-move in AB (the BC case treated similarly). We first exclude the case that n be a move in C. If that were the case, by the projection condition on BC and Lemma 5.10, m must be a move in A. Then, by the IH, the last BC-move in u', say n'^T, must be a P-move in BC. Using again the projection condition on BC and Lemma 5.10, n' must in particular be in C, contradicting the IH as n'^T is necessarily followed by a move in A. Thus, n is in AB and so, by projection on AB, it must be a P-move in AB. □

Proof of Lemma 5.18. We argue by contradiction, and we also suppress stores from this argument from brevity. So, let u' be the least prefix of u breaking bracketing and in particular suppose m answers a question q in u', and that the last open question of u' is instead a move q'. Thus, u' has the form: $u' = u_1 q u_2 q' u_3$. Because the projections of u on AB and BC are both bracketed by definition, we cannot have all of q, m, q' be in the same component. Suppose q and m are in AB, in which case q' must be in C (the case of q, m being in BC is similar). If m is a move in B then, since q, m, q' cannot all be in BC, q would need to be in A, but this would imply that both q, m are initial moves in their arenas and therefore q' cannot precede m in u (in case q, m are in BC and q' in A, m cannot be in B as then q would necessarily be in B as well). Thus, m is in A. Since u' is bracketed, it must then have the form: $u' = u_1 q u_2 q' q_1 \cdots m_1 q_2 \cdots m_2 \cdots q_k \cdots m_k$, for some k, with each m_i answering q_i. Let i be the greatest index such that $m_1, \cdots, m_i$ be all in BC. As m_i is followed by an A-move, by the previous lemma, m_i is a P-move in BC (and an O-move in AB), and is in particular a B-move. By alternation in BC, q' must also be a P-move in BC. But then, again from the previous lemma, we have that m_i is in C, a contradiction. □

5.3.2 Lemmas for Proposition 5.27

We use the following auxiliary result.

Lemma 5.43 (Strong support [42])**.** Let X be a strong nominal set and let $x_1, x_2, y_1, y_2, z_1, z_2 \in X$. Suppose also that $\nu(y_i) \cap \nu(z_i) \subseteq \nu(x_i)$, for $i = 1, 2$, and that $(x_1, y_1) \sim (x_2, y_2)$ and $(x_1, z_1) \sim (x_2, z_2)$. Then, $(x_1, y_1, z_1) \sim (x_2, y_2, z_2)$.

Proof of Lemma 5.25. Suppose m_1 is a P-move in AB — the other cases are proven similarly. The move preceding it is an O-move in AB and, by $u_1 \sim u_2$, so is the last move in u_2. Let $u_i m_i^{S_i} \restriction_\gamma AB = u_i' m_i^{S_i'}$ for $i = 1, 2$. Now, $u_1 \sim u_2$ implies that $(u_1', u_1) \sim (u_2', u_2)$, and we also have $(u_1', m_1^{S_1'}) \sim (u_2', m_2^{S_2'})$ by hypothesis. Moreover, $\nu(u_i) \cap \nu(m_i^{S_i'}) \subseteq \nu(u_i')$, for $i = 1, 2$. Because if $a \in \nu(m_i^{S_i'}) \setminus \nu(u_i')$ then a is introduced by $m_i^{S_i'}$ in $u_i' m_i^{S_i'}$, and therefore introduced by $m_i^{S_i}$ in $u_i m_i^{S_i}$, by Lemma 5.16, so $a \notin \nu(u_i)$. Thus, by strong support lemma, $(u_1', u_1, m_1^{S_1'}) \sim (u_2', u_2, m_2^{S_2'})$. Finally, for all $a \in \mathsf{dom}(S_i) \setminus \mathsf{dom}(S_i')$ we have $a \notin \nu(u_i' m_i^{S_i'})$ and therefore $S_i(a) = S_i''(a)$, where S_i'' the store of the last move of u_i. We thus get $u_1 m_1^{S_1} \sim u_2 m_2^{S_2}$. □

Proof of Lemma 5.26. By induction on $|u_1'|$. Suppose $u_1' = u_1'' m^S$. By IH there is $u_2'' \sqsubseteq u_2$ with $u_1'' \sim u_2''$. Suppose m is a P-move in AB. Then

u_2'' is strictly smaller than u_2 (as u_2'' ends in an O-move in AB), so let $u_2''n^T \sqsubseteq u_2$. By strong determinacy of σ, $(u_1''m^S \upharpoonright_\gamma AB) \sim (u_2''n^T \upharpoonright_\gamma AB)$ and thus, by the previous lemma, $u_1''m^S \sim u_2''n^T$. Similarly if m is a P-move in BC. Finally, if m is an O-move in AC then, again, there is $u_2''n^T \sqsubseteq u_2$. By hypothesis we have $(u_1''m^S \upharpoonright AC) = (u_2''n^T \upharpoonright AC)$ and so, by previous lemma, $u_1''m^S \sim u_2''n^T$. $\square$

6

Conclusions

We have presented a game model of a higher-order language with ground references. We looked into the notion of *names* and how it lends itself to give a semantics for modelling the creation, private use and flow of references in a real-life programming paradigm. The reach of nominal game semantics is much broader, though. As mentioned at the beginning of these notes, it can be used for modelling generative dynamic effects, such as general references, objects or exceptions. Moreover, the trace-like flavour of nominal games allows for automata-driven compositional representations of term denotations, leading to what is called *algorithmic game semantics*. The connection to trace semantics has recently been further explored and the correspondence has been formalised. We briefly discuss these directions next.

6.1 Other paradigms: higher-order references

Higher-order references can be captured in the same fashion as ground references via a full abstraction result [32]. This comes at a cost of complicating the shape of plays, because the functional storage needs to be taken into account. Consequently, reference names have to be included in stores carried by moves. The corresponding values cannot be revealed, though, as this would jeopardise full abstraction. After all, in equivalent terms the values could be different, yet still equivalent! Accordingly, we simply use † to mark function values inside stores. On the other hand, we need to provide for exploration for these values by both players. To that end, in addition to the usual justification pointers, we introduce pointers that can point to the store and at specific functional values. This corresponds to using such values during interaction. In drawings we distinguish the second kind of pointers with double lines.

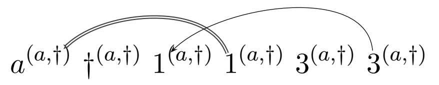
Example 6.1. We illustrate the semantics of higher-order references using two inequivalent terms:

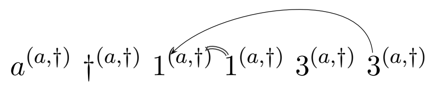
$$x : \mathsf{ref}(\mathsf{int} \to \mathsf{int}) \vdash\, !x : \mathsf{int} \to \mathsf{int}$$

and

$$x : \mathsf{ref}(\mathsf{int} \to \mathsf{int}) \vdash \lambda h^{\mathsf{int}}.(!x)h : \mathsf{int} \to \mathsf{int}$$

respectively. Note that the former refers to the initial content of x, while the latter accesses the current content each time the function is applied. This is reflected by the two plays below, namely, the different targets of pointers from the respectively fourth moves.


$a^{(a,\dagger)}\ \dagger^{(a,\dagger)}\ 1^{(a,\dagger)}\ 1^{(a,\dagger)}\ 3^{(a,\dagger)}\ 3^{(a,\dagger)}$


$a^{(a,\dagger)}\ \dagger^{(a,\dagger)}\ 1^{(a,\dagger)}\ 1^{(a,\dagger)}\ 3^{(a,\dagger)}\ 3^{(a,\dagger)}$

Similarly to the results for GroundML, such generalized plays can be used to prove a full abstraction result. The two terms discussed above are indeed inequivalent,

$$\mathsf{let}\, x = \mathsf{ref}(\lambda\_.\mathsf{div})\, \mathsf{in}\, \mathsf{let}\, f = [\,]\, \mathsf{in}\, (x := (\lambda x^{\mathsf{int}}.0); f(0)).$$

as they can be distinguished by the context above.

6.2 From references to Java objects

Insights from nominal games for higher-order references can be applied to model Java objects taken from the idealised fragment called *Interface Middleweight Java (IMJ)* [34]. We highlight two crucial differences.

• Firstly, Java programs feature functions in the form of methods only. Accordingly, functions can only be accessed by reference to the underlying object: there is no mechanism for anonymous method passing. This makes the use of † moves redundant. However, †'s are still needed inside stores, because all higher-order behaviour is now delegated to moves pointing to stores.

• The first observation leaves us with plays where all pointers are to †'s residing inside stores. However, since a method cannot be updated after it has been included in an object, we can safely remove pointers altogether. This is because the associated name uniquely identifies the method. In order to demonstrate this, given an object a with a single method $\mathsf{mth} : \mathsf{int} \to \mathsf{int}$, let us assume that each store S containing a satisfies $(a.\mathsf{mth}, \dagger) \in S$. Suppose we have a play where move m_1 introduces a, some moves m_2, m_3 follow, and finally a move calling the last $a.\mathsf{mth}$ is played:

$$\begin{tikzcd}[column sep=tiny] \cdots & m_1^{(a.\mathsf{mth},\dagger)} & m_2^{(a.\mathsf{mth},\dagger)} & m_3^{(a.\mathsf{mth},\dagger)} & 1^{(a.\mathsf{mth},\dagger)} \arrow[l, bend right, dash] \end{tikzcd}$$

The last move represents a call $\mathsf{mth}(1)$. The player who played m_3 can only resolve what to return by propagating the call $\mathsf{mth}(1)$ to m_2, and the same applies to m_2 for m_1. Hence, the play can only evolve in the following way:

$$\begin{tikzcd}[column sep=tiny] \cdots & m_1^{(a.\mathsf{mth},\dagger)} & m_2^{(a.\mathsf{mth},\dagger)} & m_3^{(a.\mathsf{mth},\dagger)} & 1^{(a.\mathsf{mth},\dagger)} \arrow[l, bend right, dash] & 1^{(a.\mathsf{mth},\dagger)} \arrow[lll, bend right, dash] & 1^{(a.\mathsf{mth},\dagger)} \arrow[lllll, bend right, dash] \end{tikzcd}$$

Once m_1 returns with an answer to the call, the above copycat behaviour has to be repeated in a dual way to propagate the answer back to m_3. This presentation can be optimised by removing the intermediate calls/returns, as they do not add anything essential to the observable behaviour. Instead, all method calls of player O (resp. P): should be to methods of objects introduced by player P (O); should point at the move introducing the object enclosing the called method. Thus, our example simplifies to:

$$\begin{tikzcd}
\cdots & m_1^{(a.\mathsf{mth},\dagger)} & m_2^{(a.\mathsf{mth},\dagger)} & m_3^{(a.\mathsf{mth},\dagger)} & 1^{(a.\mathsf{mth},\dagger)} \arrow[lll, bend right, dash]
\end{tikzcd}$$

Based on these observations, one can construct a more direct model of IMJ, in which the pointer structure is omitted. On the other hand, the absence of this structure makes simple properties, like name ownership or play-to-arena projection, significantly subtler and this subtlety affects the definitions of play and strategy composition [34].

6.3 Algorithmic game semantics

Game semantics provides a handle for attacking equivalence problems. One might wonder to what extent it can be used to automate the process. This direction was initiated and first pursued in classical game models, for fragments of Idealized Algol, where it was observed that by restricting type disciplines to low orders it was possible to concretely represent term strategies by means of finite-state or pushdown automata [14, 2]. We have investigated this problem for GroundML [33] as well as Reduced ML (the restriction of GroundML with integer but not full ground references) [32] and Interface Middleweight Java [30]. Our technique is based on automata over infinite alphabets. This class is a natural fit for representing nominal plays due to the presence of names, drawn from an infinite alphabet.

Of course, in order to obtain decidability results, the language has to be restricted. For a start, the type of integers has to be limited to a finite subset. But even then the presence of higher-order types makes contextual equivalence undecidable. The smallest GroundML term types of this kind are

$$\mathsf{unit} \to \mathsf{unit} \to \mathsf{unit} \quad \text{and} \quad ((\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}) \to \mathsf{unit}$$

with $\mathsf{unit} \to \mathsf{unit}$ and $(\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$ remaining decidable. There are also restrictions on the allowable types of free identifiers, but once they are met, contextual equivalence can be checked in a fully automated fashion.

In addition, the structure of game plays in typing contexts where these restrictions are not met is rich enough to render the problem of comparing sets of complete plays undecidable. The complete classification of decidable cases for GroundML can be found in [33].

6.4 Operational game semantics

The operational flavour of nominal games, combined with its concrete representation of names, leads one to consider connections to trace models for higher-order programs with names as examined e.g. in [23, 27, 15]. It turns out that games can be given a fully operational presentation and can be seen essentially as open trace models with composition as a primitive operation. This has been done in the specific case of games for higher-order references [22], but we can now envisage a general applicability of the approach.

In this presentation, names here take on the additional role of encoding justification pointers: † moves are replaced by moves from a designated set of function names and, accordingly, every move is accompanied by a name representing the position of its justifier. While this may seem obscure at first, and involves the additional burden of obscuring function identities and an ensuing function-name accumulation, it offers a more operational understanding of games as a sequence of calls and returns of functions whose names are supplied during the game.

References

[1] S. Abramsky. Semantics of interaction. In A. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, pages 1–32. Cambridge University Press, Cambridge, 1997.

[2] S. Abramsky, D. R. Ghica, A. S. Murawski, and C.-H. L. Ong. Algorithmic game semantics and component-based verification. In *Proceedings of SAVBCS: Specification and Verification of Component-Based Systems, Workshop at ESEC/FASE*, Technical Report 03-11, pages 66–74. Department of Computer Science, Iowa State University, 2003.

[3] S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and I. D. B. Stark. Nominal games and full abstraction for the nu-calculus. In *Proceedings of LICS*, pages 150–159. IEEE Computer Society Press, 2004.

[4] S. Abramsky, K. Honda, and G. McCusker. Fully abstract game semantics for general references. In *Proceedings of IEEE Symposium on Logic in Computer Science*, pages 334–344. Computer Society Press, 1998.

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

[6] S. Abramsky and G. McCusker. Call-by-value games. In *Proceedings of CSL*, volume 1414 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, 1997.

[7] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In P. W. O'Hearn and R. D. Tennent, editors, *Algol-like languages*, pages 297–329. Birkhaüser, 1997.

[8] S. Abramsky and G. McCusker. Game semantics. In H. Schwichtenberg and U. Berger, editors, *Logic and Computation*. Springer-Verlag, 1998. Proceedings of the 1997 Marktoberdorf Summer School.

[9] S. Abramsky and G. McCusker. Full abstraction for Idealized Algol with passive expressions. *Theoretical Computer Science*, 227:3–42, 1999.

[10] R. L. Crole. *Categories for Types*. Cambridge University Press, 1993.

[11] M. Gabbay and D. R. Ghica. Game semantics in the nominal model. *Electr. Notes Theor. Comput. Sci.*, 286:173–189, 2012.

[12] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.

[13] Murdoch James Gabbay. *A Theory of Inductive Definitions with Alpha-Equivalence*. PhD thesis, University of Cambridge, 2001.

[14] D. R. Ghica and G. McCusker. Reasoning about Idealized Algol using regular expressions. In *Proceedings of ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 103–115. Springer-Verlag, 2000.

[15] D. R. Ghica and N. Tzevelekos. A system-level game semantics. *Electr. Notes Theor. Comput. Sci.*, 286:191–211, 2012.

[16] Dan R. Ghica. Applications of game semantics: From program analysis to hardware synthesis. In *Proceedings of LICS*, pages 17–26. IEEE Computer Society, 2009.

[17] C. A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. MIT Press, 1992.

[18] K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation. *Theoretical Computer Science*, 221(1–2):393–456, 1999.

[19] J. M. E. Hyland. Game semantics. In A. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, pages 131–182. Cambridge Univ. Press, 1997.

[20] J. M. E. Hyland and C.-H. L. Ong. Pi-calculus, dialogue games and PCF. In *Proc. 7th* ACM *Conf. Functional Prog. Lang. Comp. Architecture*, pages 96 – 107. ACM Press, 1995.

[21] J. M. E. Hyland and C.-H. L. Ong. 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):285–408, 2000.

[22] Guilhem Jaber. Operational nominal game semantics. In *Proceedings of FOSSACS*, pages 264–278. Springer, 2015.

[23] A. Jeffrey and J. Rathke. Java Jr: Fully abstract trace semantics for a core Java language. In *Proceedings of ESOP*, volume 3444 of *Lecture Notes in Computer Science*, pages 423–438. Springer, 2003.

[24] J. Laird. A fully abstract games semantics of local exceptions. In *Proceedings of 16th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 2001.

[25] J. Laird. A game semantics of local names and good variables. In *Proceedings of FOSSACS*, volume 2987 of *Lecture Notes in Computer Science*, pages 289–303. Springer-Verlag, 2004.

[26] J. Laird. Game semantics for higher-order concurrency. In *FSTTCS*, volume 4337 of *Lecture Notes in Computer Science*, pages 417–428, 2006.

[27] J. Laird. A fully abstract trace semantics for general references. In *Proceedings of ICALP*, volume 4596 of *Lecture Notes in Computer Science*, pages 667–679. Springer, 2007.

[28] J. Laird. A game semantics of names and pointers. *Annals of Pure and Applied Logic*, 151:151–169, 2008.

[29] E. Moggi. Computational lambda-calculus and monads. In *Proceedings of IEEE Symposium on Logic in Computer Science*, pages 14–23. Computer Society Press, 1989.

[30] A. S. Murawski, S. J. Ramsay, and N. Tzevelekos. Game semantic analysis of equivalence in IMJ. submitted, 2015.

[31] A. S. Murawski and N. Tzevelekos. Full abstraction for Reduced ML. In *Proceedings of FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 32–47. Springer-Verlag, 2009.

[32] A. S. Murawski and N. Tzevelekos. Algorithmic nominal game semantics. In *Proceedings of ESOP*, volume 6602 of *Lecture Notes in Computer Science*, pages 419–438. Springer-Verlag, 2011.

[33] A. S. Murawski and N. Tzevelekos. Algorithmic games for full ground references. In *Proceedings of ICALP*, volume 7392 of *Lecture Notes in Computer Science*, pages 312–324. Springer, 2012.

[34] A. S. Murawski and N. Tzevelekos. Game semantics for interface middleweight java. In *POPL*, pages 517–528, 2014.

[35] H. Nickau. *Hereditarily Sequential Functionals: A Game-Theoretic Approach to Sequentiality.* PhD thesis, Universität-Gesamthochschule-Siegen, 1996.

[36] Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science.* Cambridge University Press, New York, NY, USA, 2013.

[37] J. Power and E. Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7:453–468, 10 1997.

[38] John Power and Hayo Thielecke. Closed Freyd- and kappa-categories. In *Proceedings of ICALP*, pages 625–634. Springer, 1999.

[39] J. C. Reynolds. The essence of Algol. In J. W. de Bakker and J.C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North Holland, 1981.

[40] J. C. Reynolds. *Theories of Programming Languages*. Cambridge University Press, 1998.

[41] Ulrich Schöpp. *Names and Binding in Type Theory*. PhD thesis, University of Edinburgh, 2006.

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