Nominal Game Semantics

Andrzej S. Murawski  Nikos Tzevelekos
University of Warwick  Queen Mary University of London
# Contents

1 Introduction 192

2 Elements of Nominal Set Theory 196

3 GroundML 200
   3.1 Syntax 200
   3.2 Operational semantics 202

4 ToyML: A First-Order Language with Integer References 208
   4.1 Types and terms 208
   4.2 Concrete games 209
   4.3 Interpretation of ToyML terms 212

5 Game Model 222
   5.1 Moves, arenas, plays, strategies 222
   5.2 Full abstraction 248
   5.3 Chapter Appendix: deferred proofs 258

6 Conclusions 261
   6.1 Other paradigms: higher-order references 261
   6.2 From references to Java objects 263
   6.3 Algorithmic game semantics 264
6.4 Operational game semantics .......................... 265

References .......................... 266
Abstract

These tutorial notes present nominal game semantics, a denotational technique for modelling higher-order programs.
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 \( \text{ref} \theta = (\theta \to \text{unit}) \times (\text{unit} \to \theta) \). The approach led to identification of references with pairs of arbitrary reading (unit \( \to \theta \)) and writing (\( \theta \to \text{unit} \)) 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].

\[
\begin{align*}
    x := 0; x := 1 & \quad \text{run write}(0) \ \text{ok write}(1) \ \text{ok done} \\
    x := 1 & \quad \text{run write}(1) \ \text{ok done} \\
    x := !x & \quad \text{run read } i \ \text{write}(i) \ \text{ok done} \\
    () & \quad \text{run done}
\end{align*}
\]

Thus, for the first term, the semantic translation treats both updates as observable events and therefore both are recorded in the game play. 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

\[\text{In effect, write}(0) \text{ and write}(1) \text{ represent calls to the write method of reference } x, \text{ while ok’s correspond to returns of that method.}\]
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.\footnote{Similar issues arise when modelling exceptions in this way, i.e. as products of raise/handle functions.} 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.

\[
a^{\{(a,i)\}} \star \{(a,1)\} \quad a^{\{(a,i)\}} \star \{(a,i)\}
\]

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\footnote{Similar issues arise when modelling exceptions in this way, i.e. as products of raise/handle functions.}.

Since 2004, the nominal approach has led to a series of new full abstraction results. The languages covered are the \(\nu\)-calculus\footnote{Similar issues arise when modelling exceptions in this way, i.e. as products of raise/handle functions.} (purely functional language with names), \(\lambda\nu\)\footnote{Similar issues arise when modelling exceptions in this way, i.e. as products of raise/handle functions.} (a higher-order language with storage of untyped names), Reduced ML\footnote{Similar issues arise when modelling exceptions in this way, i.e. as products of raise/handle functions.} (a higher-order language with integer-valued storage), RefML\footnote{Similar issues arise when modelling exceptions in this way, i.e. as products of raise/handle functions.} (higher-order references) and Middleweight Java\footnote{Similar issues arise when modelling exceptions in this way, i.e. as products of raise/handle functions.}. Nominal game semantics has also been used to model Concurrent ML\footnote{Similar issues arise when modelling exceptions in this way, i.e. as products of raise/handle functions.} and exceptions\footnote{Similar issues arise when modelling exceptions in this way, i.e. as products of raise/handle functions.}.

**Structure of the tutorial**

Our tutorial is meant to complement existing introductory literature to game semantics\footnote{Similar issues arise when modelling exceptions in this way, i.e. as products of raise/handle functions.}, which highlighted the then new structural components necessary to model higher-order computation, e.g. arenas,
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.
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 $A$, the set of *names*, the elements of which we denote by $a, b, c$ and variants. Names are partitioned by

$$A = \biguplus_{i \in \omega} A_i$$

where each $A_i$ 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 $\Pi(A)$ for the group of finite permutations of $A$ which are component-preserving:

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

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

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

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

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

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

$$\nu(x) = \bigcap\{ U \subseteq \text{fin } A \mid (\forall a \in U. \pi(a) = a) \implies \pi \cdot x = x \}.$$ 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 \( \pi \) and \( a \). In the same fashion, each set \( A_i \) is a nominal set. The set \( \mathcal{P}_{\text{fin}}(\mathbb{A}) \) of finite sets of names is also a nominal set, with action \( \pi \cdot U = \{ \pi(a) \mid a \in U \} \) for each permutation \( \pi \) and finite \( U \subseteq \mathbb{A} \). Moreover, \( \nu(U) = U \). 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 \( a_1, \ldots, a_n \in \mathbb{A} \). Moreover, \( \nu(a_1 \ldots a_n) \) contains precisely the names \( a_1, \ldots, a_n \) (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 \( \pi \). 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 $a_1 \ldots a_n \in A^*$, its orbit $[a_1 \ldots a_n] = \{\pi \cdot (a_1 \ldots a_n) \mid \pi \in \Pi(\mathbb{A})\}$ contains all sequences of atoms $a_1' \ldots a_n'$ such that, for all $i, j \in \{1, \ldots, n\}$ 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.\footnote{See \cite{42} for motivation and a detailed explanation of its significance. The same notion was defined by Schöpp \cite{41} for nominal sets with an essentially simple action.} In particular, we say that an element $x$ of nominal set $X$ has \textbf{strong support} if, for all permutations $\pi$,

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

that is, the converse condition of that of (simple) support also holds. We call $X$ a \textbf{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.\footnote{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.} 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 $X^*$.
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 $\zeta$. 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 $\zeta$. A reference type $\text{ref}\, \zeta$ specifies locations storing values of type $\zeta$.

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

---

1We briefly discuss extensions to more expressive, and more realistic, languages in Section 6 and give references to the relevant bibliography.
following grammar,
\[
\theta ::= \zeta | \theta \times \theta | \theta \to \theta
\]
\[
\zeta ::= \text{unit} | \text{int} | \text{ref} \zeta
\]
and types of the \(\zeta\) kind are called ground.

Let us fix a set \(\mathcal{A} = \biguplus_{\zeta} \mathcal{A}_{\zeta}\) of location names, which precisely correspond to names as in nominal sets (cf. Chapter \(\text{2}\)) where \(\mathcal{A} = \biguplus_{i \in \omega} \mathcal{A}_i\). That is, we assume an enumeration \(\zeta_1, \zeta_2, \ldots\) of ground types so that, for each \(i\), the sets \(\mathcal{A}_{\zeta_i}\) and \(\mathcal{A}_i\) 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,
\[
M ::= () | a | i | x | M \oplus M | \text{while}(M) | \text{if } M \text{ then } M \text{ else } M | \lambda x^\theta . M
\]
\[
| \langle M, M \rangle | \pi_1 M | \pi_2 M | \text{ref}(M) | M = M | !M | M := M
\]
where \(\oplus\) ranges over a set of arithmetic operators, \(x\) and \(i\) range over variables and integers respectively, and \(a\) over elements of \(\mathcal{A}\). A term is a supported term featuring no locations (i.e. with empty support).

Supported terms are typed in environments \(U, \Gamma\), where \(U\) is a finite set of location names and \(\Gamma\) 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 \(\text{ref}(\ldots)\) operator is evaluated.

Thus, the language is best described as the call-by-value \(\lambda\)-calculus with products over base types \(\zeta\), 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 \(\mathcal{A}\) is not accessible to the programmer (think of Java or ML programs).
GroundML

\[ U, \Gamma \vdash i : \text{int} \]
\[ U, \Gamma \vdash x : \theta \]
\[ U, \Gamma \vdash a : \text{ref} \]
\[ U, \Gamma \vdash M : \text{int} \]
\[ U, \Gamma \vdash N_0 : \theta \]
\[ U, \Gamma \vdash N_1 : \theta \]
\[ U, \Gamma \vdash M : \text{int} \]
\[ U, \Gamma \vdash \text{if } M \text{ then } N_1 \text{ else } N_0 : \theta \]
\[ U, \Gamma \vdash \text{while}(M) : \text{unit} \]
\[ U, \Gamma \vdash \lambda x^\theta.M : \theta \rightarrow \theta' \]
\[ U, \Gamma \vdash \pi_i M : \theta_i \]
\[ U, \Gamma \vdash M \oplus N : \text{int} \]
\[ U, \Gamma \vdash M = N : \text{unit} \]

Figure 3.1: Typing rules of GroundML.

- let \( x = M \) in \( N \) for the term \( \lambda x^\theta.N ) M \);
- \( M; N \) for \( \text{let } x = M \text{ in } N \), 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 \( \theta \), we can define some closed term \( \vdash M_\theta : \theta \). Hence, we can also define:

\[ \text{div}_\theta \equiv \text{while}(1); M_\theta, \]

which will be used as a canonical divergent term of type \( \theta \).

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:

\[ \text{Stores} = \{ S : A \rightarrow (\{ \star \} \cup \mathbb{Z} \cup A) \mid S \text{ finite and legal} \} \]
where a partial function \( S : A \rightarrow (\{ \star \} \cup Z \cup A) \) is called legal just if 
\[
\text{cod}(S) \cap A \subseteq \text{dom}(S) \quad \text{and for all names } a \in \text{dom}(S): \\
(a \in A_{\text{unit}} \implies S(a) = \star) \land (a \in A_{\text{int}} \implies S(a) \in Z) \\
\land (a \in A_{\text{ref}} \implies S(a) \in A_z)
\]
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 \( \text{unit} \) 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 \( x \in \{ \star \} \cup Z \cup A \), we define the update \( S[a \mapsto x] \) by:
\[
(S[a \mapsto x])(a') = \\
\begin{cases} 
S(a') & \text{if } a' \in \text{dom}(S) \setminus \{a\} \\
x & \text{if } a' = a \\
\text{undefined} & \text{otherwise}
\end{cases}
\]
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 \( \text{dom}(T) = \{a_1, \ldots, a_n\} \). Finally, a configuration is a pair \((M, S)\) of a supported term and a store such that \( \text{dom}(S) \) 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:
\[
E ::= [] \mid E \oplus M \mid V \oplus E \mid \text{if } E \text{ then } M \text{ else } M \mid EM \mid VE \mid \langle E, M \rangle \\
\mid \langle V, E \rangle \mid \pi_i E \mid \text{ref}(E) \mid E = M \mid x = E \mid !E \mid E := M \mid V := E
\]
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:
\[
V ::= () \mid i \mid x \mid a \mid \langle V, V \rangle \mid \lambda x^0.M
\]
Note in particular that values are supported terms. For any term \( \vdash M : \text{unit} \), we write \( M \Downarrow \) if \( (\emptyset, M) \rightarrow (S, ()) \), for some store \( S \).
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.

\[ \text{gen} \equiv \lambda z. \text{int}. \text{let} \ x = \text{ref}(0) \ \text{in} \ (x := z; x) : \text{int} \rightarrow \text{ref int} \]

\[ \text{gen}' \equiv \lambda z. \text{int}. \text{let} \ x = \text{ref}(0) \ \text{in} \ \lambda x. \text{int}. (x := z; x) : \text{int} \rightarrow \text{ref int} \]

The two terms differ in one crucial aspect: while \( \text{gen} \) returns a fresh reference name each time it is called, \( \text{gen}' \) 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 \( \pi \)-calculus. Consider the following term \( \text{trmit} \).

\[ f : \text{ref ref int} \rightarrow \text{unit}, \ g : \text{unit} \rightarrow \text{ref ref int} \]

\[ \text{let} \ c_1 = \text{ref}(\text{ref}(0)) \ \text{in} \ (f \ c_1; \text{let} \ c_2 = g() \ \text{in} \ \lambda \_ \text{unit}. \ c_2 := !c_1) : \text{unit} \rightarrow \text{unit} \]
The term creates an input channel $c_1$ and passes it to its context (via $fc_1$); 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.

\[
C ::= []; \text{ if } C \text{ then } M \text{ else } M \mid \text{ if } M \text{ then } C \text{ else } M \mid \text{ if } M \text{ then } M \text{ else } C \\
\mid \text{ while}(C) \mid \lambda x^\theta.C \mid MC \mid CM \mid \langle C, M \rangle \mid \langle M, C \rangle \mid \pi C \mid C \oplus M \\
\mid M \oplus C \mid C = M \mid M = C \mid \text{ref}(C) \mid !C \mid C := M \mid M := C
\]

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 $\Gamma[M_1] \Downarrow$ implies $\Gamma[M_2] \Downarrow$ for any context $C$ such that $\vdash C[M_1], C[M_2] : \text{unit}$.

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

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

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

we obtain $C[\text{gen}] \not\Downarrow$ and $C[\text{gen}'] \Downarrow$. 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 $\Gamma \vdash M, N : \theta$:

$$\Gamma \vdash M \equiv N \iff \text{comp}(\llbracket \Gamma \vdash M \rrbracket) = \text{comp}(\llbracket \Gamma \vdash N \rrbracket)$$

that is, to show $\Gamma \vdash M \equiv N$, 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:

$$M_1 \equiv \text{let } x = \text{ref}(0) \text{ in } \lambda y. \text{ref int}. x = y : \text{ref int} \rightarrow \text{int},$$

$$M_2 \equiv \lambda y. \text{ref int}. 0 : \text{ref int} \rightarrow \text{int}.$$

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:

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

$$M_4 \equiv f(\lambda _, \text{div}); \lambda y. \text{ref int}. 0$$

with types $f : (\text{unit} \rightarrow \text{ref int}) \rightarrow \text{unit} \vdash M_3, M_4 : \text{ref int} \rightarrow \text{int}$. 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 $\text{ref int}$ in a private reference of type $\text{ref ref int}$. 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:

\[ M_5 \equiv \text{let } x = \text{ref(ref(0)) in} \]
\[ \lambda y^{\text{ref int}}. \text{let } z = \text{!x in (if } y = z \text{ then div else (x := ref(0); z))} \]
\[ M_6 \equiv \lambda y^{\text{ref int}}. \text{ref(0)} \]

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

\[
\theta ::= \beta | \beta \to \beta \\
\beta ::= \text{unit} | \text{int} | \text{ref int}
\]

ToyML terms have shapes defined below.

\[
M ::= () | i | x | M \oplus M | \text{if } M \text{ then } M \text{ else } M | \lambda x^\beta.M | MM |
\]

\[
\text{while}(M) | \text{ref}(M) | M = M | !M | M := M
\]

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 \((MM)\).
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 $A$ 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 $\theta$, we first define the set $V_\theta$ of associated semantic values as follows:

$V_{\text{unit}} = \{ \star \}$,  
$V_{\text{int}} = \mathbb{Z}$,  
$V_{\text{ref int}} = A_{\text{int}}$,  
$V_{\beta \rightarrow \beta'} = \{ \dagger \}$,

and write $V$ for the set of all semantic values. We shall use $\ell$ and variants

\[
\begin{align*}
\Gamma \vdash () : \text{unit} & \quad \Gamma \vdash i : \text{int} & \quad (x : \theta) \in \Gamma & \quad \Gamma \vdash M : \text{int} & \quad \Gamma \vdash \text{while}(M) : \text{unit} \\
\Gamma \vdash M : \text{int} & \quad \Gamma \vdash N : \text{int} & \quad \Gamma \vdash M \oplus N : \text{int} & \quad \Gamma \vdash \text{if } M \text{ then } N \text{ else } N' : \theta \\
\Gamma \uplus \{ x : \beta \} \vdash M : \beta' & \quad \Gamma \vdash M : \beta \rightarrow \beta' & \quad \Gamma \vdash N : \beta & \quad \Gamma \vdash MN : \beta' \\
\Gamma \vdash M : \text{int} & \quad \Gamma \vdash \text{ref}(M) : \text{ref int} & \quad \Gamma \vdash M = N : \text{int} \\
\Gamma \vdash M : \text{ref int} & \quad \Gamma \vdash M : \text{ref int} & \quad \Gamma \vdash N : \text{ref int} & \quad \Gamma \vdash M := N : \text{unit} \\
\end{align*}
\]

Figure 4.1: Typing rules of ToyML.
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 $\Gamma = \{ x_1 : \theta_1, \ldots , x_m : \theta_m \}$ consist of ToyML types and let $\theta$ also be a ToyML type. The set $M_{\Gamma \vdash \theta}$ of moves associated with $\Gamma$ and $\theta$ is defined to be

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

where:

- $I_{\Gamma}$ is the set of initial moves given by
  $$I_{\Gamma} = \{ (\ell_1, \ldots , \ell_m) \mid \ell_i \in V_{\theta_i}, 1 \leq i \leq m \};$$

- $M_{\theta}$ is the set of output moves defined by
  - $M_{\theta} = \{ (r_\downarrow, \ell) \mid \ell \in V_\theta \}$ if $\theta$ is a base type,
  - $M_{\theta} = \{ (r_\downarrow, \uparrow) \} \cup \{ (c, \ell) \mid \ell \in V_{\theta'} \} \cup \{ (r, \ell) \mid \ell \in V_{\theta''} \}$ if $\theta = \theta' \rightarrow \theta''$;

- $M_{x_i}$ is the set of variable moves, taken to be empty if $\theta_i$ is a base type and, if $\theta_i = \theta_i' \rightarrow \theta_i''$, equal to
  $$\{ (c_{x_i}, \ell) \mid \ell \in V_{\theta_i'} \} \cup \{ (r_{x_i}, \ell) \mid \ell \in V_{\theta_i''} \}.$$

Moves are ranged over by $m$ and variants. We shall use $i$ to range over $I_{\Gamma}$, and we shall often write $i_{x_i}$ for $\ell_i$.

Thus, each move in $M_{\Gamma \vdash \theta}$ consists of a pair $(t, \ell)$ of a tag and a (semantic) value. For each function-type identifier $x$ in $\Gamma$, 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 $(c_x, \ell)$ corresponds to the value that identifier is called with. Similarly, $r_\downarrow$ can be taken to correspond to the fact that our modelled term was successfully evaluated, and, if $\theta$ is a function type, $c$ and $r$ refer respectively to calling the corresponding value and obtaining a result.

Moves are assigned ownership as follows:
4.2. Concrete games

- i and those with tags \( r_x, c \) belong to \( O \) (environment);
- the rest (with tags \( r_i, c_x, r \)) 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 \( i/r_i \), which belong to \( O/P \) respectively (this exactly matches the intuition that \( r_i \) 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.

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 \( i(t_1, \ell_1) \cdots (t_k, \ell_k) \) such that the sequence \( t_1 \cdots t_k \) of tags matches the grammar:

\[
X r_i (c X r)^* \quad \text{where} \quad X = \left( \sum_{(x: \theta' \rightarrow \theta'') \in \Gamma} (c_x r_x) \right)^*.
\]

We assume that \( X r_i (c X r)^* \) degenerates to \( X r_i \) when \( c, r \) are not available in \( M_{\Gamma \vdash \theta} \), i.e. \( \theta \) 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 \( (r_i) \) 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 \( ((c X r)^*) \). 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 \( 1 \leq i \leq k \), \( \text{dom}(S_i) = \nu(m_1 \cdots m_i) \).

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 \( i \in I_{\Gamma} \) and \( \text{dom}(S) = \nu(i) \). The set of initial moves-with-store will be written \( I_{\Gamma}^{\text{st}} \).

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 \( \langle \Gamma \vdash M : \theta \rangle \) 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.

4.3 Interpretation of ToyML terms

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

\[
() \mid i \mid x \mid x \oplus y \mid \text{ref}(x) \mid x = y \mid !x \mid x := y \mid \text{if } x \text{ then } N \text{ else } N'
\]

\(^2\)Note that \( \cong \) is defined using contexts from GroundML.
4.3. Interpretation of ToyML terms

**Skip command (())**

$(\Gamma \vdash ()) : \text{unit}$ is defined to contain all complete plays over $\Gamma \vdash \text{unit}$ that have the shape $i^S(r_\downarrow, *)^S$. Here $P$ simply responds with the move $(r_\downarrow, *)$ without modifying the store.

**Integer constant ($i$)**

The defining complete plays for $(\Gamma \vdash i : \text{int})$ have the shape $i^S(r_\downarrow, i)^S$. 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 $i^S(r_\downarrow, i_x)^S$. $P$ simply responds by copying the value of $x$ provided by $O$ in the initial move and pairing it with the tag $r_\downarrow$.

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

$$i^S(r_\downarrow, \uparrow)^S X_1 \ldots X_k$$

where $k \geq 0$ and

$$X_i = (c, \ell_i)^S_i(c_x, \ell_x)^S_i(r_x, \ell_x')^S_i r, \ell_x')^S_i$$

for all $1 \leq i \leq k$. Intuitively, this corresponds to $P$ first replying with the $\uparrow$ 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 insofar as 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 ($x \oplus y$)**

$(\Gamma \vdash x \oplus y : \text{int})$ is given by complete plays of the form $i^S(r_\downarrow, i_x \oplus i_y)$.

**Reference creation ($\text{ref}(x)$)**

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

$$i^S(r_\downarrow, a)^{S[0 \rightarrow i_x]}$$
with \( a \in \mathcal{A}_{\text{int}} \setminus \text{dom}(S) \). Here \( P \) provides a fresh name as part of his response (note the \( a \not\in \text{dom}(S) \) 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 \( \langle \Gamma \vdash x = y : \text{int} \rangle \) to be

\[
\{i^S(\bot_0, 0)^S | i^S \in I^\Gamma_x, \; i_x \neq i_y \} \cup \{i^S(\bot_1, 1)^S | i^S \in I^\Gamma_x, \; i_x = i_y \}.
\]

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 \( i_x, i_y \) are names from \( \mathcal{A}_{\text{int}} \).

**Dereferencing (\(!x\))**

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

\[
\langle \Gamma \vdash !x : \text{int} \rangle = \{i^S(\bot_x, S(i_x))^S | i^S \in I^\Gamma_x \}.
\]

Note again that \( i_x \in \mathcal{A}_{\text{int}} \).

**Reference update (\( x := y \))**

\[
\langle \Gamma \vdash x := y : \text{unit} \rangle = \{i^S(\bot_x, *)^{S[i_x = i_y]} | i^S \in I^\Gamma_x \}
\]

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 (if \( x \) then \( N \) else \( N' \))**

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

\(^3\)I.e. for any \( a' \in \mathcal{A}_{\text{int}} \setminus \text{dom}(S) \), we have that \( i^S(\bot_0, a')^{S[a' \mapsto i_x]} \sim i^S(\bot_1, a)^{S[a' \mapsto i_x]} \).
4.3. Interpretation of ToyML terms

is equal to:

\[ \{ i^S s \mid i_x > 0, \ i^S s \in \{ \Gamma \vdash N \} \} \cup \{ i^S s \mid i_x = 0, \ i^S s \in \{ \Gamma \vdash N' \} \} \].

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 let \( x = M \) in (let \( y = N \) 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:

\[ \lambda x^\beta . M \mid \text{while}(M) \mid xy \mid (\lambda x^\beta . M)N \]

**Remark 4.8.** The cases we shall examine concern application. Note that in ToyML the argument has to be of type \( \beta \) 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, \( \lambda \)-abstractions or branches of a conditional expression. Since the game interpretation of \( (\text{if } M \text{ then } M' \text{ else } M'')N \) is the same as that of \( \text{if } M \text{ then } (M'N) \text{ else } (M''N) \), 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

\[ i^S(r_i, \hat{i})^S X_1 \cdots X_k . \]
Each of the segments $X_i$ corresponds to a single call to $M$ and its format will be required to be compatible with $\langle \Gamma, x : \beta \vdash M : \beta' \rangle$ in the sense we explain below.

Consequently, $X_1 \cdots X_k$ resembles an interleaving of plays from $\langle \Gamma, x : \beta \vdash M : \beta' \rangle$ 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

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

such that $(i, \ell_c)^{S_0} m_{i,1}^{S_{i,1}} \cdots m_{2k}^{S_{i,2k}} (r, \ell_r)^{S_{i,2k+1}}$, henceforth referred to as thread $s_i$, is a complete play from $\langle \Gamma, x : \beta \vdash M \rangle$.

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

$$
X (r_i, \dagger)(c, *) X_1 (r, \ell_1)(c, *) \cdots (r, \ell_{k-1})(c, *) X_k (r, \ell_k).
$$

The above represents interleaveings of calls of $\lambda x^\text{unit} . M : \text{int}$ or, equivalently, of evaluations of $M$. To interpret $\Gamma \vdash \text{while}(M) : \text{unit}$ 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 \( c, r, r_\downarrow \) and add the move \((r_\downarrow, \star)\) at the end. This yields the sequence:

\[ XX_1 \cdots X_k (r_\downarrow, \star). \]

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

**Application** \((xy)\)

\(\llbracket \Gamma \vdash xy : \beta' \rrbracket\) contains all complete plays of the shape

\[ i^S(c_x, i_y)^S(r_x, \ell)^S(r_\downarrow, \ell)^S'. \]

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 \( \text{dom}(S) \subseteq \text{dom}(S') \) and the inclusion can be proper if \( \ell \in \mathbb{A} \setminus \text{dom}(S) \).

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

Observe that this case corresponds to \( \text{let } x = N \text{ in } M \), 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

\[
\begin{align*}
s & = i^{S_0} u (r_1, \ell)^S \in \llbracket \Gamma \vdash N : \beta \rrbracket , \\
t & = (i, \ell)^{T_0} m_1^{T_1} \cdots m_{2k+1}^{T_{2k+1}} \in \llbracket \Gamma, x : \beta \vdash M : \beta' \rrbracket .
\end{align*}
\]

Note that the initial move of \( t \) contains \( \ell \), which also occurs in the last move of \( s \). Still, not all \( s, t \) will be used to calculate \( \llbracket \Gamma \vdash \text{let } x = N \text{ in } M \rrbracket \). 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 \( t \):
\[
P(t) = \{ a \in \nu(t) \mid \exists t' m^T \subseteq t. a \in \text{dom}(T) \setminus \nu(t'), |t'| \text{ odd} \}.
\]

3. If \( \ell \in \mathcal{A} \) and \( \ell \) does not occur in \( i_{S_0}u \) then, if there is an occurrence of \( \ell \) in some \( m_i \) \((1 \leq i \leq 2k+1)\), 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 \( \ell \), i.e. \( T_{2l}(\ell) = T_{2l-1}(\ell) \) for \( 1 \leq l < k' \). If \( \ell \) does not occur in any \( m_i \), we also insist on the above property with \( k' = k + 1 \), i.e. \( O \) does not change \( \ell \) 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 \( \text{dom}(S) \) may properly contain \( \text{dom}(T_0) \). This will be the case if \( s \) contains more names than those in \( (i, \ell) \).

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 \( \beta = \text{ref int} \), so that the last value \( \ell \) 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 \((m_1, m_3, \cdots \text{ or } m_{2k+1})\). 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 \( \text{let } x = N \text{ in } M \). We distinguish two cases.

- If \( \ell \notin \mathcal{A} \), or \( \ell \in \mathcal{A} \cap \nu(i_{S_0}u) \), we include in \( \{ \Gamma \vdash \text{let } x = N \text{ in } M \} \) all complete plays over \( \Gamma \vdash \beta' \) that match the following shape.
\[
i_{S_0}u \ x_1 \ T_{1\cup(S\setminus T_0)} x_2 \ T_{2\cup U_1} x_3 \ T_{3\cup U_1} \cdots x_{2k} \ T_{2k\cup U_1} x_{2k+1} \ T_{2k+1\cup U_1}
\]

If \( \ell \in \mathcal{A} \), in the above complete plays \( \ell \) will feature in every store from the moment it appears in \( i_{S_0}u \). The stores \( U_i \) consist of names that are recognisable to \( N \), but not to \( M \). They can be
4.3. Interpretation of ToyML terms

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 \( \ell \notin \mathcal{A} \).

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

\[
iS_0 \cup m_1 T^i_1 u(S \setminus T_0) m_2 T^i_2 uU_1 \cdots m_{2k'} T^i_{2k'} uU_{k'} m_{2k'+1} uU_{k+1} \cdots m_{2k} uU_k m_{2k+1} uU_k
\]

Note that \( \ell \) 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 \( \ell \) does not occur in \( t \) (apart from the initial occurrence) then \( \ell \) 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

\[
x : \text{ref int} \vdash (\lambda y. \text{ref int}. x = y)(\text{ref}(0)) : \text{int}.
\]

The complete plays corresponding to the term \( x : \text{ref int} \vdash \text{ref}(0) \) have the shape \( s = iS_0(r_i, \ell)^S \), where \( i \neq \ell \), \( S_0 = \{(i, j)\} \), \( S = S_0 \uplus \{(\ell, 0)\} \) for arbitrary \( j \in \mathbb{Z} \). In turn, the term \( x : \text{ref int}, y : \text{ref int} \vdash x = y : \text{int} \) generates two kinds of complete plays:

- \( t = (i_x, i_y)^{T_0}(r_i, 0)^{T_0} \) with \( i_x \neq i_y \) and \( T_0 = \{(i_x, j_x), (i_y, j_y)\} \), where \( j_x, j_y \in \mathbb{Z} \), or

- \( t = (i_x, i_y)^{T_0}(r_i, 1)^{T_0} \) with \( i_x = i_y \), \( T_0 = \{(i_x, j_x)\} \) 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 $i = i_x$ and $\ell = i_y$. Note that, because in $s$ we must have $i \neq \ell$, this implies $i_x \neq i_y$, 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 $\ell$ 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 $x : \text{ref int} \vdash (\lambda y \text{ref int}. x = y)(\text{ref}(0)) : \text{int}$ will have the shape $i_x^{((i_x,j_x))}(r_x,0)^{(i_x,j_x)}$, 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

$$f : \text{unit} \rightarrow \text{ref int} \vdash (\lambda y \text{ref int}. f() = y)(\text{ref}(0)) : \text{int}.$$ 

The complete plays for the term $f : \text{unit} \rightarrow \text{ref int} \vdash \text{ref}(0)$ have the shape $s = x^0(r_1,\ell)^S$, where $S = \{(\ell,0)\}$ and two shapes are possible for $f : \text{unit} \rightarrow \text{ref int}, y : \text{ref int} \vdash f() = y : \text{int}$, namely:

- $t = (\ast, i_y)^{T_0}(c_f, \ast)^{T_0}(r_f, \ell')^{T_1}(r_1,0)^{T_1}$ with $T_0 = \{(i_y, j_y)\}$, $i_y \neq \ell'$, $T_1 = \{(i_y, j_y'), (\ell', j')\}$, where $j_y, j_y', j' \in \mathbb{Z}$, or
- $t = (\ast, i_y)^{T_0}(c_f, \ast)^{T_0}(r_f, i_y)^{T_1}(r_1,1)^{T_1}$ with $T_0 = \{(i_y, j_y)\}$ and $T_1 = \{(i_y, j_y')\}$.

We must have $\ell = i_y$ for compatibility 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 $\ell = i_y$ 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 $f : \text{unit} \rightarrow \text{ref int} \vdash (\lambda y \text{ref int}. f() = y)(\text{ref}(0)) : \text{int}$. Moreover, because $\ell$ 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 $\ell$. Thus, the relevant shapes of $t$ are of the form

$$(\ast, \ell)^{(\ell,0)}(c_f, \ast)^{(\ell,0)}(r_f, \ell')^{\{(\ell,0),(\ell',j')\}}(r_1,0)^{\{(\ell,0),(\ell',j')\}}$$
with $\ell \neq \ell'$. Note that, because no non-initial move in $t$ contains an occurrence of $\ell$, the resultant stores will not have values for $\ell$. Thus, \( \llbracket f : \text{unit} \rightarrow \text{ref int} \rrbracket \vdash (\lambda y^{\text{ref int}}. f() = y)(\text{ref}(0)) : \text{int} \) consists of all complete plays of the form

\[ \star^0 (c_f, \star)^0 (r_f, \ell')^{(\ell', j')} (r_\downarrow, 0)^{(\ell', j')} \]

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 (\texttt{let} $x = M \text{ in } N$) gives a flavour of what is to come, because the \texttt{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 $\beta$. 

---

4.3. 

\textit{Interpretation of ToyML terms} 

221
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: $A = \bigcup \zeta A_\zeta$. 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 \( \lambda_A : M_A \rightarrow \{O, P\} \times \{Q, A\} \),

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

- \( m \in I_A \implies \lambda_A(m) = (P, A) \),
- \( m \vdash_A m' \land \lambda_A^{Q_A}(m) = A \implies \lambda_A^{Q_A}(m') = Q \),
- \( m \vdash_A m' \implies \lambda_A^{O_P}(m) \neq \lambda_A^{O_P}(m') \).

We call \( \vdash_A \) the justification relation of \( A \), and \( \lambda_A \) its labelling function.

The role of \( \lambda_A \) is to label moves as Opponent or Proponent moves, or O-moves and P-moves respectively, and as Questions or Answers. We write \( \lambda_A^{O_P} \) and \( \lambda_A^{Q_A} \) for \( \lambda_A \) 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 \( \theta \) to an arena \( \llbracket \theta \rrbracket \) and which is defined
next. Consequently, an arena $A = \llbracket \theta \rrbracket$ is the domain of all values of type $\theta$. 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 $\Lambda_\zeta$, for each ground type $\zeta$, defined by:

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

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:

$$\llbracket \text{unit} \rrbracket = 1 \quad \llbracket \text{int} \rrbracket = \mathbb{Z} \quad \llbracket \text{ref } \zeta \rrbracket = \Lambda_\zeta.$$

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 $B$:

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

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 $(i_A, i_B)$, for all $i_B \in I_B$ (and dually for $B$). That is, writing $\bar{I}_A$ for $M_A \setminus I_A$, we have:

$$M_{A \otimes B} = (I_A \times I_B) + \bar{I}_A + \bar{I}_B \quad I_{A \otimes B} = I_A \times I_B$$
and, for all $m \in M_{A \otimes B}$,
\[ \lambda_{A \otimes B}(m) = \begin{cases} PA & \text{if } m = (i_A, i_B) \in I_A \times I_B \\ \lambda_A(m) & \text{if } m \in \bar{I}_A \\ \lambda_B(m) & \text{if } m \in \bar{I}_B \end{cases} \]

and, for all $(m, n) \in M^2_{A \otimes B}$,
\[ m \vdash_{A \otimes B} n \iff (m = (i_A, i_B) \land (i_A \vdash_A n \lor i_B \vdash_B n)) \lor (m \vdash_A n) \lor (m \vdash_B n) \]

On the other hand, the arena $A \Rightarrow B$ contains functions, all of which start with an initial move of the form $\ddagger$, 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,
\[ M_{A \Rightarrow B} = \{\ddagger\} + M_A + M_B \quad I_{A \Rightarrow B} = \{\ddagger\} \]

and, for all $m \in M_{A \Rightarrow B}$,
\[ \lambda_{A \Rightarrow B}(m) = \begin{cases} PA & \text{if } m = \ddagger \\ OQ & \text{if } m = i_A \in I_A \\ \bar{\lambda}_A(m) & \text{if } m \in \bar{I}_A \\ \lambda_B(m) & \text{if } m \in M_B \end{cases} \]

and, for all $(m, n) \in M^2_{A \Rightarrow B}$,
\[ m \vdash_{A \Rightarrow B} n \iff ((m, n) \in (\{\ddagger\} \times I_A) \cup (I_A \times I_B)) \lor (m \vdash_A n) \lor (m \vdash_B n) \]

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

Example 5.2. Recall the ground types $\zeta ::= \text{unit} \mid \text{int} \mid \text{ref} \zeta$. We have seen that their denotations are given by flat arenas $1$, $\mathbb{Z}$ and $\Lambda_{\zeta}$, each of which has as moves the semantic counterparts of syntactic values (locations and integers), including the move $\star$ which corresponds to $() : \text{unit}$. 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 $\zeta_1 \rightarrow \zeta_2$. They translate to $[[\zeta_1]] \Rightarrow [[\zeta_2]]$, which are generally given by a diagram of the form:

$$
\begin{array}{c}
\hspace{0.5cm}
\vdash
\
v_1
\end{array}
\begin{array}{c}
\hspace{0.5cm}
\vdash
\
v_2
\end{array}
$$

where $v_1, v_2 \in \{\ast\} \cup \mathbb{Z} \cup A$ 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 $A \rightarrow B$ by:

$$
M_{A \rightarrow B} = M_A + M_B \\
I_{A \rightarrow B} = I_A
$$
and, for all \( m \in M_{A \to B} \) and \((m, n) \in M^2_{A \to B}\):

\[
\lambda_{A \to B}(m) = \begin{cases} \ OQ \quad & \text{if } m = i_A \in I_A \\
\lambda_A(m) \quad & \text{if } m \in I_A \\
\lambda_B(m) \quad & \text{if } m \in M_B \\
\end{cases}
\]

\( m \vdash_{A \to B} n \iff ((m, n) \in I_A \times I_B) \lor (m \vdash_A n) \lor (m \vdash_B n) \)

Pictorially, \( A \to B \) looks just like \( A \Rightarrow B \) albeit having its initial \( \uparrow \) move removed.

**Definition 5.3.** The types of \( \text{GroundML} \) are interpreted into arenas by

\[
\llbracket \text{unit} \rrbracket = 1, \quad \llbracket \text{int} \rrbracket = \mathbb{Z}, \quad \llbracket \text{ref} \zeta \rrbracket = \mathbb{K}_\zeta, \quad \text{and}
\]

\[
\llbracket \theta_1 \times \theta_2 \rrbracket = \llbracket \theta_1 \rrbracket \otimes \llbracket \theta_2 \rrbracket, \quad \llbracket \theta_1 \rightarrow \theta_2 \rrbracket = \llbracket \theta_1 \rrbracket \Rightarrow \llbracket \theta_2 \rrbracket.
\]

Moreover, each finite \( U \subseteq \mathbb{A} \), say \( U = \{ a_1, \cdots, a_n \} \) with each name \( a_i \) belonging to the set \( \mathbb{K}_\zeta \), is mapped to:

\[
\llbracket U \rrbracket = \{ (a'_1, \cdots, a'_n) \in \mathbb{K}_{\zeta_1} \times \cdots \times \mathbb{K}_{\zeta_n} \mid (a_1, \cdots, a_n) \sim (a_1', \cdots, a_n') \}
\]

Finally, a typing environment \( U, \Gamma \vdash \theta \), with \( \Gamma = \{ x_1 : \theta_1, \cdots, x_k : \theta_k \} \), is interpreted as the prearena\footnote{If \( k + |U| = 0 \) we take the left-hand side to be 1.}

\[
\llbracket U \rrbracket \otimes \llbracket \theta_1 \rrbracket \otimes \cdots \otimes \llbracket \theta_k \rrbracket \rightarrow \llbracket \theta \rrbracket
\]

which we shall denote by \( \llbracket U, \Gamma \vdash \theta \rrbracket \), or just \( \llbracket \Gamma \vdash \theta \rrbracket \) if \( U = \emptyset \).

**Remark 5.4.** Let us look more closely at the structure of the prearena \( \llbracket U, \Gamma \vdash \theta \rrbracket \). Its initial moves are of the form \((a'_1, a'_2, \cdots, a'_n, i_1, i_2, \cdots, i_k)\), where the \( i_1, i_2, \cdots, i_k \) part consists of initial moves from the denotation of \( \Gamma \). Put otherwise, each \( i_i \) corresponds to an “opening” of the arena \( \llbracket \theta_i \rrbracket \) with some semantic value provided by the Opponent (hence the polarity of the initial move is \( OQ \)). The rest of \( \llbracket U, \Gamma \vdash \theta \rrbracket \) comprises the output arena, \( \llbracket \theta \rrbracket \), where Proponent himself provides a semantic value, and the remainders \( \llbracket \theta_i \rrbracket^- \) 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 \( U = \{a_1, a_2, \cdots, a_n\} \). That is, there is
some permutation $\pi$ 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:

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

These have denotations

$$(\llbracket \zeta_1 \rrbracket \otimes \cdots \otimes \llbracket \zeta_k \rrbracket \otimes \llbracket \zeta_{k+1} \rrbracket \Rightarrow \llbracket \zeta'_{k+1} \rrbracket \otimes \cdots \otimes \llbracket \zeta_1 \rrbracket \Rightarrow \llbracket \zeta'_1 \rrbracket) \rightarrow \llbracket \zeta \rrbracket \Rightarrow \llbracket \zeta' \rrbracket$$

which are generally of the following shape,

$$\begin{tikzpicture}
  \node (L1) at (0,0) {$(\ell_1, \cdots, \ell_n)$};
  \node (V1) at (-1,-1) {$v_{k+1}$};
  \node (V2) at (0,-1) {$v_n$};
  \node (V3) at (1,-1) {$v_{k+1}'$};
  \node (V4) at (0,-2) {$v_n'$};
  \node (V5) at (2,-2) {$v'$};
  \draw (L1) -- (V1);
  \draw (L1) -- (V2);
  \draw (L1) -- (V3);
  \draw (V3) -- (V4);
  \draw (V4) -- (V5);
\end{tikzpicture}$$

with $\ell_i = \dagger$ if $i > k$, and $\ell_i \in \{\star\} \cup \mathbb{Z} \cup \mathbb{A}$ otherwise (depending on $\zeta_i$). 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:

²A 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.\footnote{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 \cite{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.}

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

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

Given a prearena \( A \), a \textit{move-with-store} on \( A \) is a pair \((m, S)\), written \( m^S \), where \( m \in M_A \) and \( S \in \text{Stores} \). For economy, we may be writing “move” when we really mean “move-with-store”.

**Definition 5.6.** A \textit{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 \( i \in I_A \);
- 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 \textit{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 \textsc{GroundML}. A first such condition is \textit{alternation}: the two players should play in turns. This reflects the fact that \textsc{GroundML} is a sequential language.

Another condition is \textit{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
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 $\mathsf{view} \gamma s \gamma$ of a justified sequence s to be given by:

$$
\begin{align*}
\mathsf{view} \gamma \varepsilon \gamma & = \varepsilon \\
\mathsf{view} \gamma s \tau n \gamma & = \mathsf{view} \gamma s \gamma \cup \nu(n).
\end{align*}
$$

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

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

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:

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

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 $\mathsf{view} \gamma t \gamma$ (Visibility).
- For all $tm^s \sqsubseteq s$, $\mathsf{dom}(S) = \mathsf{Av}(tm^s)$ (Frugality).

The set of plays on $A$ is denoted by $P_A$. 
5.1. Moves, arenas, plays, strategies

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

\[
\begin{align*}
\star \quad \uparrow & \quad i_1 \quad a^{(a,i_1)} \quad i_2^{(a,i_1')} \quad a^{(a,i_2)} \quad i_3^{(a,i_2')} \quad a^{(a,i_3)} \quad \ldots \\
OQ & \quad PQ & \quad OA & \quad PQ & \quad OA & \quad PA & \quad OQ & \quad PA
\end{align*}
\]

On the other hand, the term \( \text{gen} \) will produce a different name \( a_i \) after each call:

\[
\begin{align*}
\star \quad \uparrow & \quad i_1 \quad a_1^{(a_1,i_1)} \quad i_2^{(a_1,i_1')} \quad a_2^{(a_2,i_1),(a_1,i_1')} \quad i_3^{(a_2,i_1'),(a_1,i_1'')} \quad a_3^{(a_3,i_3),(a_2,i_1''),(a_1,i_1'''')} \quad \ldots \\
OQ & \quad PQ & \quad OA & \quad PQ & \quad OA & \quad PA & \quad OQ & \quad PA
\end{align*}
\]

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 \( f: \text{ref ref int} \rightarrow \text{unit}, g: \text{unit} \rightarrow \text{ref ref int} \vdash \text{trmit}: \text{unit} \rightarrow \text{unit} \). The typing context corresponds to the prearena \((A_{\text{ref int}} \Rightarrow 1) \otimes (1 \Rightarrow A_{\text{ref int}}) \rightarrow (1 \Rightarrow 1)\), which we depict as on the right (all \( a, b \in A_{\text{ref int}} \)). The plays produced by the transmitter will be as follows, where for economy we only depict for each store \( S \) its \text{ref int} references (i.e. \( S \upharpoonright A_{\text{ref int}} \)), and suppress the values for \( A_{\text{int}} \) names.

\[
\begin{align*}
\uparrow & \quad f_1 \quad a_1^{(a_1,b_1)} \quad f \quad (a_1,b_1') \quad g \quad (a_1,b_1'') \quad a_2^{(a_2,b_2)} \quad i \quad (a_2,b_2') \quad (a_2,a') \quad (a_2,a), \quad \ldots \\
OQ & \quad PQ & \quad OA & \quad PQ & \quad OA & \quad PA & \quad OQ & \quad PA
\end{align*}
\]
that have been encountered. Thus, such a play starts with an initial move \((†, †)\) 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 \(⋆_f\) (and a possibly updated store), at which point \(P\) asks for the output channel name by playing \(⋆_g\) (plus store). \(O\) answers with the channel name \(a_2\) and \(P\) returns his function via \(†\). From there on, any question \(⋆(a_2,a')\), \((a_1,a)\) is answered by a move \(⋆(a_2,a),(a_1,a)\), which acknowledges that a transmission from \(a_1\) to \(a_2\) was made. Finally, note that this is not the only play shape for \(\text{transmit}\): 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 \textit{name ownership}. Given a play \(s\) and some \(a ∈ ν(s)\), we say that \(a\) is a \(P\)-name in \(s\), written \(a ∈ P(s)\), if \(a\) is first introduced in \(s\) in a \(P\)-move. Put otherwise, if there is some even-length \(s'm^S ⊑ s\) such that \(a ∈ ν(m^S) \setminus ν(s')\). The set \(O(s)\) of \(O\)-names of \(s\) is defined in a similar manner. Clearly,

\[
ν(s) = P(s) ∪ O(s)
\]

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 \(A → B\), restricts them in a crucial aspect: only \(P\) is allowed to switch between the \(A\) and \(B\) component.

\textbf{Lemma 5.10 (Switching).} Let \(s ∈ P_{A→B}\) and \(s'm^Sn^S' ⊑ s\) be such that \((m,n) ∈ (M_A × M_B) ∪ (M_B × M_A)\). Then, \(λ_{A→B}^{OP} = P\).

\textit{Proof.} By induction on the length of \(s\), using also the fact that whenever the hypothesis holds for \(s\) then, for all odd-length \(s' ⊑ s\) ending in a move in \(A\), all moves of \(ρs'\) belong to \(A\); and if \(s'\) ends in \(B\) then all moves \(ρs'\) belong to \(B\) apart from the initial move.

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 $\lbrack s \rbrack$.

*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 $s = s_1q_1s_1m_1$ with $q_1$ justifying $m_1$ and all questions of $s_1$ answered inside $s_1$. Thus, $q$ is the pending question of $s_1$ as well and $\lbrack s \rbrack = \lbrack s' \rbrack q_1m_1$, 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** $\sigma$ on a prearena $A$ is a non-empty set of even-length plays of $A$ satisfying:

- If $so^S p o^S \in \sigma$ then $s \in \sigma$ (Even-prefix closure).
- If $s \in \sigma$ then, for all permutations $\pi$, $\pi \cdot s \in \sigma$ (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 $\pi$ (Determinacy).

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

Note in particular that $\epsilon \in \sigma$ for all strategies $\sigma$. 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: $X_{cl} = \{ s \mid \exists s' \in X. s \sqsubseteq_{\text{even}} s' \}$.

**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:

$$\llbracket \text{gen} \rrbracket = \{ \star \uparrow i_1 a^{(a,i_1)} \cdots i_n a^{(a,i_n)} \mid n \geq 0 \wedge i_1, \ldots, i_n \in \mathbb{Z} \}_c l$$

$$\llbracket \text{gen} \rrbracket = \{ \star \uparrow i_1 a^{S_1 a \{\{(a_1,i_1)\}\}} \cdots i_n a^{S_n a \{\{(a_n,i_n)\}\}} \mid n \geq 0 \wedge i_1, \ldots, i_n \in \mathbb{Z} \}_c l$$

where, for each $j$, $\text{dom}(S_j) = \{ a_1, \ldots, a_{j-1} \}$ and $a_j \notin \text{dom}(S_j)$.

On the other hand, for $\text{trmit}$ we obtain the strategy:

$$\llbracket \text{trmit} \rrbracket = \{ \star \uparrow S_1 \star^{S'_1} \star^{S_2} \star^{S'_2} \star^{S_n} \star^{S'_n} \uparrow S_0 \star^{S_1} \star^{S'_1} \star^{S_2} \star^{S'_2} \star^{S_n} \star^{S'_n} \mid n \geq 0 \}_c l$$

where stores satisfy the conditions:

$$\text{dom}(S) = \{ a_1, S(a_1) \}$$

$$\text{dom}(S') = \text{dom}(S) \cup \{ S'(a_1) \}$$

$$\text{dom}(S_0) = \text{dom}(S') \cup \{ S_0(a_1), a_2, S_0(a_2) \}$$

$$\text{dom}(S_i) = \text{dom}(S_{i-1}) \cup \{ S_i(a_1), S_i(a_2) \} \quad (i > 0)$$

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 were indeed plays of the examined prearenas. In the sequel we will be more economic and, for instance, when writing

$$\llbracket \text{trmit} \rrbracket = \{ \star \uparrow S_1 \star^{S'_1} \star^{S_2} \star^{S'_2} \star^{S_n} \star^{S'_n} \uparrow S_0 \star^{S_1} \star^{S'_1} \star^{S_2} \star^{S'_2} \star^{S_n} \star^{S'_n} \mid n \geq 0 \}_c l$$

we shall implicitly assume that the defined expression is a play in $(A_{\text{ref int}} \Rightarrow 1) \otimes (1 \Rightarrow A_{\text{ref int}}) \rightarrow (1 \Rightarrow 1)$, 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 $\gamma$ 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,

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

4The fact that $a_j \notin \text{dom}(S_j)$ is already implicit in the notation $(S_j \cup \{(a_j,i_j)\})$. Note in particular that $S_1 = \emptyset$. 
where $S' = S \upharpoonright \text{Av}(tm^S)$.

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 $A \rightarrow B$ and $B \rightarrow C$. 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 $A \rightarrow B \rightarrow C$ by

\[
M_{A \rightarrow B \rightarrow C} = M_{A \rightarrow B} + M_C \quad \lambda_{A \rightarrow B \rightarrow C} = [\lambda_{A \rightarrow B}[i_B \mapsto PQ], \lambda_C] \\
I_{A \rightarrow B \rightarrow C} = I_A \quad \vdash_{A \rightarrow B \rightarrow C} = \vdash_{A \rightarrow B} \cup \{(i_B, i_C)\} \cup \vdash_C
\]

Let $u$ be a justified sequence on $A \rightarrow B \rightarrow C$. 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 $\gamma$ 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 $A \rightarrow B \rightarrow C$, yet an O-move in $B \rightarrow C$.

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

- $u$ is frugal, that is, $\gamma(u) = u$;
- $P(u \upharpoonright \gamma AB) \cap P(u \upharpoonright \gamma BC) = \emptyset$;
- $O(u \upharpoonright \gamma AC) \cap (P(u \upharpoonright \gamma AB) \cup P(u \upharpoonright \gamma BC)) = \emptyset$;
- for each $u' \subseteq u$ ending in $m^u m'^{S'}$ and $a \in \text{dom}(S')$ if
– $m'$ is a P-move in $AB$ and $a \notin \text{Av}(u' \upharpoonright AB)$,
– or $m'$ is a P-move in $BC$ and $a \notin \text{Av}(u' \upharpoonright BC)$,
– or $m'$ is an O-move in $AC$ and $a \notin \text{Av}(u' \upharpoonright AC)$,
then $S(a) = S'(a)$.

We write $\text{Int}(ABC)$ 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: 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 $u \in \text{Int}(ABC)$,

$$\nu(u) = \text{O}(u \upharpoonright_{\gamma} AC) \sqcup \text{P}(u \upharpoonright_{\gamma} AB) \sqcup \text{P}(u \upharpoonright_{\gamma} BC).$$

*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 $u'm^S \subseteq u$ 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 $a \in \text{Av}(u'm^S \upharpoonright AB)$. For suppose this is not the case. Then $a \notin \nu(m)$ and, moreover, there is no $b \in \text{Av}(u' \upharpoonright AB) \sqcup \nu(m)$ such that $a$ be reachable from $b$ via $S$. But, by frugality of $u$, there must be some $b \in \text{Av}(u') \sqcup \nu(m)$ such that $a$ be reachable from $b$ via $S$. Since $b \notin \text{Av}(u' \upharpoonright AB)$, 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, $a \in \text{Av}(u'm^S \upharpoonright AB)$ and, consequently, $a \in \text{P}(u \upharpoonright_{\gamma} AB)$. 

5There 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.
5.1. Moves, arenas, plays, strategies

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 \( A \rightarrow C \) (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 \( u \in \text{Int}(ABC) \), \( u'm^Sn^S \subseteq u \) be such that \( (m, n) \notin M^2_A \cup M^2_B \cup M^2_C \). Then there is \( X \in \{AB, BC\} \) such that \( (m, n) \in M^2_X \), \( m \) is an O-move in \( X \) and \( n \) is a P-move in \( X \).

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

**Lemma 5.19** (Interaction visibility). Given \( u \in \text{Int}(ABC) \), define the view of \( u \) inductively by: \( \langle \epsilon \rangle = \epsilon \); \( \tau \text{um}^S \text{u'nm}^S \tau = \tau \text{um}^S \text{n}^S \tau \), if \( n \) a move in \( AC \) justified by \( m \); and \( \tau \text{um}^S \text{\tau} = \tau \text{um}^S \) otherwise. Then, for all \( u'm^S \subseteq u \), the justifier of \( m^S \) appears in \( \tau u^\gamma \).

We can now prove that composition is well defined.

**Proposition 5.20.** If \( u \in \text{Int}(ABC) \) then \( (u \upharpoonright_\gamma AC) \in P_{A \rightarrow C} \).

**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 \( \gamma \). \( \square \)

The above proposition gives us a way to compose plays: given \( s \in P_{A \rightarrow B} \) and \( t \in P_{B \rightarrow C} \), these can be composed to a play in \( P_{A \rightarrow C} \) if there is some \( u \in \text{Int}(ABC) \) 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 \( \sigma : A \rightarrow B \) and \( \tau : B \rightarrow C \), their **composition** \( \sigma; \tau \subseteq P_{A \rightarrow C} \) is given by:

\[
\sigma; \tau = \{ u \upharpoonright_\gamma AC \mid u \in \text{Int}(ABC) \land (u \upharpoonright_\gamma AB) \in \sigma \land (u \upharpoonright_\gamma BC) \in \tau \}.
\]
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 $\sigma = [\text{ref}(0)] : A \rightarrow B$ and $\tau = [x : \text{ref int} \vdash \lambda z. x := z ; x] : B \rightarrow C$. The composition conditions enforce that $i_0 = i_0' = 0$.

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

$$\text{gen'} \equiv \text{let } x = \text{ref}(0) \text{ in } \lambda z. x := z ; x : \text{int} \rightarrow \text{ref int}.$$  

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

$$[\text{gen'}] = 1 \xrightarrow{\text{[ref(0)]}} \text{A} \xrightarrow{\text{[x : \text{ref int} \vdash \lambda z. x := z ; x]}} \text{Z} \Rightarrow \text{A} \xrightarrow{\tau}$$

with the semantics of each component being:

$$[\text{ref}(0)] = \{ \star a^{(a,0)} \mid a \in \text{A} \} \ {\text{cl}}$$

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

$$\{ b^{(b,i_0)} \uparrow^{(b,i_0)} b^{(b,i_1)} \uparrow^{(b,i'_0)} b^{(b,i_1)} \ldots \uparrow^{(b,i'_{n-1})} b^{(b,i_n)} \mid b \in \text{A} \land n \geq 0 \} \ {\text{cl}}$$

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

$$\star b^{(b,i_0)} \uparrow^{(b,i_0)} b^{(b,i_1)} \uparrow^{(b,i'_0)} b^{(b,i_1)} \ldots \uparrow^{(b,i'_{n-1})} b^{(b,i_n)}$$
with \( i_0 = 0 \) (see also Figure 5.2). Note that the above projects as \( \ast b^{(b,0)} \) on \( 1\mathcal{A}_{\text{int}} \), which is in \( \llbracket \text{ref}(0) \rrbracket \) 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 \( 1(\mathbb{Z} \Rightarrow \mathcal{A}_{\text{int}}) \) and, if we project on \( 1(\mathbb{Z} \Rightarrow \mathcal{A}_{\text{int}}) \) and apply \( \gamma \) we see that \( b \) disappears, i.e. is not available. Since \( b \) is a P-name in \( 1\mathcal{A}_{\text{int}} \), 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 \( 1(\mathbb{Z} \Rightarrow \mathcal{A}_{\text{int}}) \), we obtain:

\[
\llbracket \text{gen}' \rrbracket = \{ \ast \uparrow i_1 b^{(b,i_1)} \cdots i_{n-1}^{(b,i_{n-1})} b^{(b,i_n)} \mid b \in \mathcal{A}_{\text{int}} \land n \geq 0 \} \}
\]

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:

\[
M_1 \equiv \text{let } s = \text{ref}(0) \text{ in } \lambda y. \text{ref int} \cdot s = y : \text{ref int} \rightarrow \text{int}
\]

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

\[
\llbracket M_1 \rrbracket = \frac{1}{\text{ref}(0)} \xrightarrow{\mathcal{A}_{\text{int}}} \text{ref int} \xrightarrow{\lambda y. \text{ref int} \cdot s = y} \mathcal{A}_{\text{int}} \Rightarrow \mathbb{Z}
\]

\[
\llbracket \text{ref}(0) \rrbracket = \{ \ast a^{(a,0)} \}
\]

\[
\llbracket \lambda y. s = y \rrbracket = \{ b^{S_0} \uparrow i_1^{S_1} \cdots a_n^{S_n} \mid b, a_1, \ldots, a_n \in \mathcal{A}_{\text{int}} \land n \geq 0 \}
\]

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 \( \text{dom}(S_j) = \{ b, a_1, \ldots, a_j \} \). For composition, we consider interaction sequences \( u \) over \( 1\mathcal{A}_{\text{int}}(\mathcal{A}_{\text{int}} \Rightarrow \mathbb{Z}) \):

\[
\ast b^{S_0} \uparrow i_1^{S_1} \cdots a_n^{S_n}
\]

with \( S_0 = \{(b,0)\} \). We now observe the following fact. The projection of \( \ast b^{S_0} \uparrow i_0 \) on \( 1(\mathcal{A}_{\text{int}} \Rightarrow \mathbb{Z}) \), after applying \( \gamma \), is \( \ast \uparrow \), and \( b \) is a P-name in \( 1\mathcal{A}_{\text{int}} \) while the next move, \( a_1^{S_1} \), is an O-move in \( 1(\mathcal{A}_{\text{int}} \Rightarrow \mathbb{Z}) \). This

\[\text{For example, } S_0 = \{(b,i)\} \text{ for any } i \in \mathbb{Z}. \] Note there may be repetitions in the sequence \( b, a_1, \ldots, a_j \).
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, and therefore $u$ must be of the form:

$$* b^{S_0} \downarrow S_0 a_1^{S_1} 0^{S_1} \cdots a_n^{S_n} 0^{S_n}$$

with $S_j(b) = 0$. Projecting onto $1(\chi_{\text{int}} \Rightarrow \mathbb{Z})$, we obtain that $[M_1] = \{ * \uparrow a_1^{S_1} 0^{S_1} \cdots a_n^{S_n} 0^{S_n} | a_1, \cdots, a_n \in \chi_{\text{int}} \land n \geq 0 \}$. Where, for each $j$, $\text{dom}(S'_j) = \{ a_1, \cdots, a_j \}$. This is precisely $[M_2]$, from Example 5.7.

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 $A \rightarrow C$. 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 $\sigma$ 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, $\pi \cdot (s_2 m_2^{S_2}) = s_1 m_1^{S_1} \in \sigma$, where $m_2^{S_2} = \pi \cdot m_2^{S_2}$. Hence, by determinacy, $s_1 m_1^{S_1} \sim s_2 m_2^{S_2}$ and so, using $s_2 m_2^{S_2} \sim s_1 m_1^{S_1}$, we get $s_1 m_1^{S_1} \sim s_2 m_2^{S_2}$. \qed

We shall use the following lemmas, the proofs of which are deferred to Section 5.3.

Lemma 5.25. Let $u_1 m_1^{S_1}, u_2 m_2^{S_2} \in \text{Int}(ABC)$ 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 $(u_1 m_1^{S_1} \uparrow_{\gamma} AB) \sim (u_2 m_2^{S_2} \uparrow_{\gamma} AB)$ (resp. $\uparrow_{\gamma} BC$, $\uparrow_{\gamma} AC$) then $u_1 m_1^{S_1} \sim u_2 m_2^{S_2}$.

Lemma 5.26. Let $u_1, u_2 \in \text{Int}(ABC)$ with $(u_1 \uparrow_{\gamma} AB) \in \sigma$ and $(u_1 \uparrow_{\gamma} BC) \in \tau$, for $i = 1, 2$, and suppose there is $u_1' \subseteq u_1$ such that $(u_1' \uparrow_{\gamma} AC) \subseteq (u_2 \uparrow_{\gamma} AC)$. Then, there is $u_2' \subseteq u_2$ 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 $\sigma$ and $\tau$ and the switching conditions of Lemma 5.17. Equivariance follows from the fact that all operations applied for composition are equivariant. Now let $sm^{S_1}_1, sm^{S_2}_2 \in \sigma;\tau$ with $sm^{S_1}_1 = u_i m^{S_i}_i \upharpoonright \gamma AC$. 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 $\sigma$, $(u_1 m^{S_1}_1 \upharpoonright \gamma AB) \sim (u_2 m^{S_2}_2 \upharpoonright \gamma AB)$ and thus, by penultimate lemma, $u_1 m^{S_1}_1 \sim u_2 m^{S_2}_2$. \qed

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:

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

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-arena $A$ to another, is called a copycat.

We can immediately verify the following.

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

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 $A \xleftarrow{\sigma} B \xrightarrow{\tau} C \xleftarrow{\rho} D$, 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)

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

we can straightforwardly verify the coproduct equations:

- $\sigma = A \xrightarrow{\text{in}_l} A + B \xrightarrow{[\sigma, \tau]} C$ and $\tau = B \xrightarrow{\text{in}_r} A + B \xrightarrow{[\sigma, \tau]} C$,

- for any $\rho : A + B \to C$, $\rho = [\text{in}_l; \rho, \text{in}_r; \rho]$.

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

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

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

which plays like $\text{id}_{A \otimes C}$ or $\text{id}_{B \otimes C}$ respectively depending on whether the initial move is of the form $(i_A, i_C)$ or $(i_B, i_C)$.

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 $G$, written $G_{it}$, obeying some further conditions which we shall discuss next.

Consider a value term $U, \Gamma \vdash V : \theta$ and its denotation $[V] : [U, \Gamma] \rightarrow [\theta]$. 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 $\sigma$ total if for all $i^S \in P_A$ there is $i^S m^S \in \sigma$, with $m$ an answer (to $i$).

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

$$\sigma : [\theta] \otimes A_{\text{int}} \rightarrow [\theta] \Rightarrow [\theta']$$

Suppose $O$ plays an initial move $(a, i_0)^S$ where, say, $S = \{(a, 5)\}$. As we argued above, $\sigma$ 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 $(a, i_0)^S$ to $\dagger$, $P$ is not going to make any changes to the store: as $\lambda x^\theta. M$ 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 $(a, i_0)^S$ 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 $(a, i)^S$ 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 $\sigma$ 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 $s = i^S m^S s'$
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 \( s = i^SmS' \) with \( m \) an answer, define \( \text{thrr}(s) \) inductively by

\[
\text{thrr}(i^SmS's_1^nT) = n
\]

\[
\text{thrr}(i^Sm's_1p^Sm's_2^nT) = \text{thrr}(i^Sm's_1^nT) = \text{thrr}(i^Sm'S1)
\]

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

\[
\lceil i^SmS's' \rceil = \gamma(i^SmS'_{m'}^n)
\]

where \( s'' \) is the subsequence of \( s' \) containing all of its moves whose threader move is \( \text{thrr}(i^SmS's') \). 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 \( i \in I_A \) there is a unique answer \( m_i \in M_A \) such that whenever \( i^SmS' \in \sigma \) then \( S' = S \) and \( m = m_i \);
- for all \( s_1m_1^nT_1, s_2 \in \sigma \) and \( s_2m_2^nT_2 \in P_A \) such that \( \lceil s_1m_1^nT_1 \rceil = \lceil s_2m_2^nT_2 \rceil \), there is \( s_2m_2^nT_2 \in \sigma \) such that \( \lceil s_1m_1^nT_1 \rceil \sim \lceil s_2m_2^nT_2 \rceil \);
- for all \( sm^nT \in \sigma \) and \( a \in \text{dom}(S) \setminus \nu([sm^n]) \) we have \( a \notin \nu([sm^n]) \) 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.\footnote{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.} We shall write \( G_{tti} \) for the wide subcategory of \( G \) of tti.
strategies. We can already see that the following strategies in $[U, \Gamma \vdash \theta]$ are tti.

$$[U, \Gamma \vdash \emptyset] = \{ i^S \circ \circ \}_{cl} \quad [U, \Gamma \vdash x_i : \theta_i] = \{ i^S \circ \circ \}_{cl}$$

$$[U, \Gamma \vdash i] = \{ i^S \circ \circ \}_{cl} \quad [U, \Gamma \vdash \ast : \theta_{\ast}] = \{ i^S \circ \circ \}_{cl}$$

Here $i$ ranges over initial moves of $[U] \otimes [\theta_1] \otimes \cdots \otimes [\theta_n]$ and is of the form $(a'_1, \cdots, a'_m, i_{\gamma_1}, \cdots, i_{\gamma_k})$, assuming $\Gamma = \{ x_1 : \theta_1, \cdots, x_k : \theta_k \}$ and $U = \{ a_1, \cdots, a_n \}$.

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

$$\pi_1 : A \otimes B \rightarrow A = \{ (i_A, i_B)^S \circ \circ i_A^S s \mid i_A^S i_A^S s \in \text{id}_A \}_{cl}$$

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

$$\text{sy}_{A,B} : A \otimes B \rightarrow B \otimes A$$

$$= \{ (i_A, i_B)^S (i_B, i_A)^S s \mid (i_A, i_B)^S (i_A, i_B)^S s \in \text{id}_{A \otimes B} \}_{cl}$$

as $A, B, C : A \otimes (B \otimes C) \rightarrow (A \otimes B) \otimes C$

$$= \{ (i_A, m)^S ((i_A, i_B), i_C)^S s \mid (i_A, m)^S (i_A, m)^S s \in \text{id}_{A \otimes (B \otimes C)} \}_{cl}$$

with $m = (i_B, i_C)$. These are in fact isomorphisms (e.g. $\text{sy}_{A,B} ; \text{sy}_{B,A} = \text{id}_{A \otimes B}$) and therefore we will usually leave them implicit as “$\cong$” in diagrams. Given a strategy $\sigma : A \rightarrow B$, we can build the strategy:

$$\langle \sigma, \text{id}_A \rangle : A \rightarrow B \otimes A = \{ \epsilon \} \cup \{ i_A^S s \in \sigma \mid \forall m^T \in s. m \in M_A \}$$

$$\cup \{ i_A^S \circ \circ (i_B, i_A)^S s' \mid i_A^S i_B^S (s' \circ \circ B) \in \sigma, \gamma (i_A^S i_A^S (s' \circ \circ A)) \in \text{id}_A \}$$

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

$$\langle \sigma, \tau \rangle = A \xrightarrow{\langle \sigma, \text{id} \rangle} B \otimes A \xrightarrow{\langle \pi_2, \text{id} \rangle} C \otimes (B \otimes A) \xrightarrow{\text{id} \otimes \text{id}} B \otimes C$$

\footnote{Recall from Remark \[5.7\] that, while $U = \{ a_1, \cdots, a_n \}$ contains specific choices of names for each of its components, its semantic interpretation $[U]$ 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 $\sigma \otimes \sigma' = A \otimes A' \xrightarrow{\{\pi_1, \pi_2, \sigma, \sigma'\}} B \otimes B'$.

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

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

For instance, for all $A, B$, we have an evaluation morphism $\text{ev}_{A, B} : (A \Rightarrow B) \otimes A \to B = \Lambda^{-1}(\text{id}_{A \Rightarrow B})$.

We can now complete the translation of values by setting:

$$\llbracket \langle V, V' \rangle \rrbracket = [U, \Gamma] \frac{[S[V, V']]}{[\theta] \otimes [\theta']}$$

$$\llbracket \lambda x. M \rrbracket = [U, \Gamma] \frac{[S[M]]}{[\theta] \Rightarrow [\theta']}$$

given $\llbracket M \rrbracket : [U, \Gamma, x : \theta] \to [\theta']$. 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 \(\lambda\)-calculus fragment of 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,

\[
\begin{align*}
\text{new}_{\zeta} : [\zeta] & \to A_\zeta = \{ v^S a^{S[a \to v]} | a \in A_\zeta \setminus \nu(v, S) \} \text{cl} \\
\text{get}_{\zeta} : A_\zeta & \to [\zeta] = \{ a^S v^S | v = S(a) \} \text{cl} \\
\text{set}_{\zeta} : A_\zeta \otimes [\zeta] & \to 1 = \{ (a, v)^S \star^{S[a \to v]} \} \text{cl}
\end{align*}
\]

for any ground type $\zeta$.

The full translation of GroundML into $G$ is given inductively as below. Suppose that $|U| = n$ and $\Gamma = \{ x_1 : \theta_1, \cdots, x_k : \theta_k \}$. We write $A$ for the arena $\llbracket U \rrbracket \otimes [\theta_1] \otimes \cdots \otimes [\theta_k]$. 
5.1. Moves, arenas, plays, strategies

- \([U, \Gamma \vdash () : \text{unit}] = A \xrightarrow{1} 1\), where \(t = \{ i_A S * S \mid i_A \in P_{A\rightarrow 1} \}\).
- \([U, \Gamma \vdash i : \text{int}] = A \xrightarrow{1} 1 \xrightarrow{i} Z\), where \(i = \{ * i \}\).
- \([U, \Gamma \vdash x_j : \theta_j] = A \xrightarrow{n+j} [\theta_j]\).
- \([U, \Gamma \vdash M_1 \oplus M_2 : \text{int}] = A \xrightarrow{[M_1], [M_2]} Z \xrightarrow{\sigma_\oplus} Z\), where \(\sigma_\oplus = \{ (i_1, i_2) | i_1, i_2 \in \mathbb{Z} \}\).
- \([U, \Gamma \vdash \text{if } M \text{ then } N_1 \text{ else } N_0 : \theta] = A \xrightarrow{[M], \text{id}} Z \xrightarrow{\theta} A \xrightarrow{\theta} Z\), where \(\text{if } = \{ 0 * r \}\) \(\cup \{ i * i | i \neq 0 \}\).
- \([U, \Gamma \vdash M_1 = M_2 : \text{int}] = A \xrightarrow{[M_1], [M_2]} A_\zeta \xrightarrow{\sigma_\pi} Z\), where \(\sigma_\pi = \{ (a, b) | a, b \in A_\zeta \} \cup \{ (a, b) | a, b \neq b \}\).
- \([U, \Gamma \vdash \text{ref}(M) : \text{ref}\zeta] = A \xrightarrow{[M]} [\zeta] \xrightarrow{\text{new}} A_\zeta\).
- \([U, \Gamma \vdash !M : \zeta] = A \xrightarrow{[M]} A_\zeta \xrightarrow{\text{get}} [\zeta]\).
- \([U, \Gamma \vdash M := N : \text{unit}] = A \xrightarrow{[M], [N]} A_\zeta \xrightarrow{[\zeta]} \xrightarrow{\text{set}} 1\).
- \([U, \Gamma \vdash MN : \theta'] = A \xrightarrow{[M], [N]} ([\theta] \xrightarrow{[\theta]} [\theta] \xrightarrow{\text{ev}} [\theta']\).
- \([U, \Gamma \vdash \lambda x.\theta. M : \theta \rightarrow \theta'] = \Lambda([M] : A \xrightarrow{[\theta]} [\theta']\).
- \([U, \Gamma \vdash \text{while}(M) : \text{unit}] = A \xrightarrow{\Lambda([M])} 1 \xrightarrow{\text{wh}} 1\), where \(\text{wh} = \{ \uparrow * i_1 * i_2 * \cdots * i_n | n \geq 0, i_1, \cdots, i_n > 0 \}\).

It is worth commenting briefly on the function of \(\text{wh} : 1 \rightarrow \mathbb{Z} \rightarrow 1\). The strategy replies to the initial \(\uparrow\) move by a question \(*\), thus querying the 1 component of \(1 \rightarrow \mathbb{Z}\). If \(O\) answers 0 then \(P\) replies with the answer \(*'\), which is played in the right-hand-side 1 and answers the initial question \(\uparrow\). On the other hand, if \(O\) answers some \(i_1 > 0\) then \(P\) asks another \(*\), 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 \texttt{gen}. Omitting the defined let-in notation, \texttt{gen} can be written as:

\[
\texttt{gen} \equiv \lambda z^\text{int} \cdot (\lambda x^\text{ref} \cdot x := z; x)(\text{ref}(0))
\]

We construct its denotation inductively as follows.

\begin{itemize}
  \item Let $\sigma_0 = \llbracket \text{ref}(0) \rrbracket = 1 \xrightarrow{0} Z \xrightarrow{\text{new}} A_{\text{int}}$.
  \item $\sigma_1 = \llbracket x := z; x \rrbracket = Z \otimes A_{\text{int}} \xrightarrow{(\text{set}, \pi_2)} 1 \otimes A_{\text{int}} \xrightarrow{\approx} A_{\text{int}}$.
  \item $\sigma_2 = \llbracket \lambda x. x := z; x \rrbracket = Z \xrightarrow{\Lambda(\sigma_1)} A_{\text{int}} \Rightarrow A_{\text{int}}$.
  \item $\sigma_3 = \llbracket (\lambda x. x := z; x)(\text{ref}(0)) \rrbracket = Z \xrightarrow{(\sigma_2, t, \sigma_0)} (A_{\text{int}} \Rightarrow A_{\text{int}}) \otimes A_{\text{int}} \xrightarrow{\text{ev}} A_{\text{int}} = Z \xrightarrow{(\text{id}, t, \sigma_0)} Z \otimes A_{\text{int}} \xrightarrow{\sigma_1} A_{\text{int}}$, using $\sigma_2 = \Lambda(\sigma_1)$ in (*).
  \item $\llbracket \text{gen} \rrbracket = 1 \xrightarrow{\Lambda(\pi_2; \sigma_3)} Z \Rightarrow A_{\text{int}}$.
\end{itemize}

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 \texttt{GroundML} terms are equivalent if, and only if, their strategy denotations contain the same complete plays:

\[
M \cong N \iff \text{comp}(\llbracket M \rrbracket) = \text{comp}(\llbracket N \rrbracket)
\]

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

#### 5.2.1 Correctness, adequacy and soundness

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

\begin{itemize}
  \item Correctness: the model is coherent with the operational semantics — every term has the same denotation as its reducts.
\end{itemize}
5.2. Full abstraction

• 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 $\nu(M) \subseteq \text{dom}(S)$, we define the terms:

$$S \triangleright M \equiv a_1 := S(a_1); \cdots ; a_n := S(a_n); M$$

$$\text{new } S \text{ in } M \equiv \text{let } x_1 = \text{ref}(S(a_1)) \text{ in } \cdots \text{ let } x_n = \text{ref}(S(a_n)[\bar{x}/\bar{a}]_1^{n-1}) \text{ in } M[\bar{x}/\bar{a}]_1^n$$

where $a_1, \cdots, a_n$ some ordering of $\text{dom}(S)$ that is type-increasing (i.e. all $a \in \mathcal{A}_\text{int}$ come first, followed by $a \in \mathcal{A}_\text{refint}$, etc.), and $M[\bar{x}/\bar{a}]_1^n$ 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.

$$(!a, S) \rightarrow (S(a), S) \quad \text{(DRF)}$$
$$(a := V, S) \rightarrow (((), S[a \mapsto V]) \quad \text{(ASN)}$$
$$(\text{ref}(V), S) \rightarrow (a', S[a' \mapsto V]) \quad a' \notin \text{dom}(S) \quad \text{(NEW)}$$

$$\frac{(M, S) \rightarrow (M', S')}{(E[M], S) \rightarrow (E[M'], S')} \quad \text{(CTX)}$$

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

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

1. If $\kappa \notin \{\text{DRF, ASN, NEW}\}$ then $[M] = [M']$.
2. If $\kappa \in \{\text{DRF, ASN}\}$ then $[S \triangleright M] = [S' \triangleright M']$.
3. If $\kappa = \text{NEW}$ then $[M] = [\text{new } \{(a', V)\} \text{ in } M']$. 
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 \(G\), e.g. for \(\beta\)-reduction rules we use Proposition 5.33 and the only interesting case is the one for \(\text{while}(M)\), which is shown using the equality:

\[
\text{wh} = 1 \Rightarrow Z \xrightarrow{(\Sigma; \text{ev}; \text{id})} (1 + 1) \otimes (1\Rightarrow Z) \xrightarrow{\text{ev}} (1\Rightarrow Z) \otimes (1\Rightarrow Z) \xrightarrow{[\text{wh}, t]} 1
\]

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

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 \([\text{new} \{(a', V)\} \text{in} E[M]] = [E[\text{new} \{(a', V)\} \text{in} M]]\).

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

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 \([\text{new} S\text{in} M] = [\text{new} S'\text{in} M']\) and, for case 3, that the order of assignments in the \(\text{new} S\text{in} M\) construct does not matter semantically.

We next look into adequacy. We shall argue as follows. Given a diverging term \(\vdash M : \text{unit}\), 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 \(\vdash M : \text{unit}\) if \(M \not\llbracket\) then \([M] = \{\epsilon\}\).
5.2. Full abstraction

\textbf{Proof.} Suppose, for the sake of contradiction, that $M \notin \mathcal{N}$ and $[[\vdash M]] = \{\epsilon, \ast \ast\}$. For any term $U, \Gamma \vdash N : \theta$ and $a \in A_{\text{int}} \setminus U$ construct $U, a, \Gamma \vdash N_a$ by recursively replacing each subterm of $N$ of the shape $\text{while}(N')$ with $\text{while}(a := (!a + 1); N')$. Observe that each $s \in [U, \Gamma \vdash N]$ induces some $s' \in [U, a, \Gamma \vdash N_a]$ 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 $\text{let } x = \text{ref}(i) \text{ in } N_a[x/a]; !x$, for some fresh variable $x$. Because $\ast \ast \in [[\vdash M]]$, we shall have $\ast j \in [[\vdash M_0]]$ 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 $s_{i \ast}^\odot$ 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 $(\emptyset, M) \rightarrow (S, M')$ in $j + 1$ such unfoldings. Then, we obtain $(\emptyset, M_0) \rightarrow (S_a, (M')_a; !a)$ with $S_a = \{(a, j + 1)\} \cup S$. By Proposition 5.36 and $\ast j \in [[\vdash M_0]]$, we have $\ast j \in [[\vdash \text{new } S_a \text{ in } (M')_a; !a]] = [[\vdash \text{new } S \text{ in } (M')_{j + 1}]]$, a contradiction.

We can therefore prove the main result of this section.

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

\textbf{Proof.} We show the contrapositive. Suppose $C$ is some context such that $C[M] \not\Downarrow$ but $C[N] \Downarrow$. Then, by Propositions 5.36 and 5.37 respectively, $[[C[M]]] = \{\epsilon, \ast \ast\}$ and $[[C[N]]] = \{\epsilon\}$. Now observe that $[[C[M]]] = [[(\lambda f.C[f(x_1, \ldots, x_n)])(\lambda x.\theta).\langle \pi_1 x, \ldots, \pi_n x \rangle]] = \Lambda([M]); [C[f(\bar{x})]]$, assuming $\Gamma = \{x_1 : \theta_1, \ldots, x_n : \theta_n\}$, so in particular there is some interaction sequence $u$ in $1(\bar{\theta} \Rightarrow [\theta])1$ such that $s = (u \downarrow \gamma_1 1(\bar{\theta} \Rightarrow [\theta])) \in \Lambda([M])$, $(u \downarrow \gamma_1 (\bar{\theta} \Rightarrow [\theta])1) \in [C[f(\bar{x})]]$, and $(u \downarrow 11) = \ast \ast$. By bracketing, $u$ is complete and, hence, so is $s$. This means $s \in \text{comp}(\Lambda([M])) \setminus \text{comp}(\Lambda([N]))$, therefore $\text{comp}([[M]]) \neq \text{comp}([[N]])$. \qed
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 $\sigma : A \to B$ we let its set of maximal orbits be:

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

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

$$\text{epref}(s) = \bigcup\{ [t] \mid t \in P_A \land t \sqsubseteq \text{even } s \}$$

which is clearly a strategy in $A$. We call $\sigma$ 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 $\theta_i$).

1. If $\mathcal{MO}(\sigma) = \{ [s_1], \ldots, [s_n] \}$ then $\sigma = \bigcup_{i=1}^{n} \text{epref}(s_i)$.

2. If $\sigma = \text{epref}(i^S m^S s)$ with $m$ a question in $A_i = C_1 \to C_2$ then:

$$\sigma = A \xrightarrow{(\pi_i, \sigma')} (C_1 \Rightarrow C_2) \otimes C_1 \otimes (C_2 \Rightarrow B) \xrightarrow{\text{ev} \circ \text{id}} C_2 \otimes (C_2 \Rightarrow B) \xrightarrow{\pi_i \circ \text{ev}} B$$

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

- \( m \) 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 \( \dagger \) in \( (m, \dagger) \), and \( n' \) answers \( n \).

3. If \( \sigma = \text{epref}(i^S m^{S'} s) \) with \( m \) an answer and \( a_1, \ldots, a_n \) is an enumeration of \( P(i^S m^{S'} s) \) then

\[
\sigma = A \xrightarrow{(\text{id}, \tau)} A \otimes \mathbb{A}_{\xi_1} \otimes \cdots \otimes \mathbb{A}_{\xi_n} \xrightarrow{\sigma'} B
\]

where \( \tau = \text{epref}(i^S (a_1, \ldots, a_n)^{S\otimes S'}) \) with \( S' \) some minimal (fresh) store with \( \{a_1, \ldots, a_n\} \subseteq \text{dom}(S') \), and \( \sigma' = \text{epref}((i, a_1, \ldots, a_n)^{S\otimes S'} S'[s]) \) where \( S'[s] \) is \( s \) with each move \( n^T \) replaced by \( n^{S'[s]} \).

4. If \( \sigma = \text{epref}(i^S m^{S'} s) \) with \( m \) an answer, \( P(i^S m^{S'} s) = \emptyset \) and \( a_1, \ldots, a_n \) is an enumeration of \( O(i^S m^{S'} s) \setminus \nu(i^S) \) with \( a_i \in \mathbb{A}_{\xi_i} \) for each \( i \), then

\[
\sigma = A \xrightarrow{(\text{id}, \tau)} A \otimes \mathbb{A}_{\text{ref}_{\xi_1}} \otimes \cdots \otimes \mathbb{A}_{\text{ref}_{\xi_n}} \xrightarrow{\sigma'} B
\]

where \( \tau = \text{epref}(i^S (a'_1, \ldots, a'_n)^{S\otimes S'}) \), \( a'_i \) some fresh name in \( \mathbb{A}_{\text{ref}_{\xi_i}} \), \( S' \) is some minimal (fresh) store with \( \{a'_1, \ldots, a'_n\} \subseteq \text{dom}(S') \), and \( \sigma' = \text{epref}((i, a'_1, \ldots, a'_n)^{S\otimes S'} S'[s[a'^{\rightarrow} \mapsto a]] \) where \( s[a'^{\rightarrow} \mapsto a] \) is \( s \) with each move \( n^T \) replaced by \( n^{T\otimes\{a'_1, a'_2\} \mid a_i \in \text{dom}(T) \} \).

5. If \( \sigma = \text{epref}(i^S m^{S'} s) \) with \( m \) an answer, \( P(i^S m^{S'} s) = \emptyset, a_1, \ldots, a_n \) an enumeration of \( \text{dom}(S) \) and \( \forall s' n^T \subseteq_{\text{odd}} i^S m^{S'} s \). \( \gamma(i^S n^T) = i^S n^T \), then

\[
\sigma = A \xrightarrow{(\text{id}, \tau)} A \otimes [\xi_1] \otimes \cdots \otimes [\xi_n] \otimes \mathbb{A}_{\text{int}} \xrightarrow{\sigma'} B
\]

where \( \tau = \text{epref}(i^S (S(a_1), \ldots, S(a_n), a)^{S\otimes ((a,0))}) \) with \( a \in \mathbb{A}_{\text{int}} \) some fresh name, and

\[
\sigma' = \text{epref}((i, S(a_1), \ldots, S(a_n), a)^{S\otimes ((a,0))} m^{S\otimes ((a,0))} s')
\]

\[\text{If } m \text{ or } i \text{ are not answered in } s \text{ then } s' \text{ is the respective prefix of the above } s'.\]
is thread-independent, where \( s' \) is obtained from \( s \) by replacing each move \( n^T \) of its \( i \)-th thread by \( n^{T' \in \{(a,i)\}} \), except if \( n^T \) is the \( i \)-th threader move in which case it is replaced by \( n^{T' \in \{(a,i-1)\}} \).

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

7. If \( \sigma = \operatorname{epref}(i^S \downarrow^S s) \) is thread-independent then \( \sigma = \Lambda(\Lambda^{-1}(\pi)) \uparrow [i^S] \), where \( \pi = \sigma \cup \{ i^{S'} m^{S'} \in P_{A \rightarrow B} \mid m = \hat{\tau} \} \).

**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 \( \sigma = \langle \pi_1, \sigma'; \ev \otimes \id; \equiv \rangle \). We need also check that \( \sigma' \) is well-defined and for the latter it suffices to ensure that \( t' = i^S(m, \hat{\tau})^S_{s_1 n^T s_2 n^{T'}} s_3 \) is a valid play. Given that \( t = i^S m^{S'} s_1 n^T s_2 n^{T'} s_3 \) 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 \( \ominus i^S m^{S'} s_1 n^T s_2 \), 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 \( \tau \), so \( \sigma' \) does not need to create any names. In 4, \( \tau \) 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 \( \sigma' \) of 3 satisfies \( P(t) = \emptyset \) for all \( t \in \sigma' \), whereas for the one from 4 we have \( \forall t' n^{T'} \sqsubseteq \text{odd} i^T t \in \sigma' \). \( \gamma(i^{T'} n^{T'}) = i^{T'} n^{T'} \). Finally, for 5, we observe that the composition indeed yields \( \sigma \). 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 \( \sigma \) ensure that, for all \( t \in \sigma' \), \( \nu(t) = \nu([t]) \); hence, there is no interference of private names between different threads, simply because there are no thread-private names.

**Proposition 5.40 (Definability).** Let \( \sigma : [U, \Gamma \vdash \theta] \) be a finitary strategy. There exists \( U, \Gamma \vdash M_\sigma : \theta \) such that \( \operatorname{comp}([U, \Gamma \vdash M_\sigma : \theta]) = \)
5.2. Full abstraction

Proof. We construct $M_\sigma$ by induction on $\|\sigma\| = (|\sigma|, 8 - \phi(\sigma))$, ordered lexicographically, where $\phi(\sigma)$ the greatest index $i$ such that $\sigma$ satisfies the conditions of case $i$ of the previous lemma, unless $\sigma = \epref(i^S \nu^S)$ with $\nu(v) \subseteq \nu(i)$ and $v \in \{\star\} \cup A \cup Z$ in which case we set $\phi(\sigma) = 8$.

Let us write $[U, \Gamma \vdash \theta]$ as $A \rightarrow B$. 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 $\sigma_i = \epref(i^S_i s_i)$. We use the induction hypothesis on the $\sigma_i$'s to obtain terms $M_{\sigma_i}$. For each $i^S_i$, we can construct a characteristic term $U, \Gamma \vdash M_i : \text{int}$ such that $[U, \Gamma \vdash M_i : \text{int}] = \epref(i^S_i^1) \cup \{i^S_0 | i^S \not\sim i^S_i^1\} - M_i$ simply checks that the elements of $\Gamma$ and $U$ satisfy the specifications dictated by $i_i$ and $S_i$. We can therefore take:

$$M_\sigma \equiv \begin{cases} M_1 & \text{if } M_1 \text{ then } M_{\sigma_1} \\ M_2 & \text{if } M_2 \text{ then } M_{\sigma_2} \\ \cdots & \text{if } M_n \text{ then } M_{\sigma_n} \\ \text{div}_\theta & \text{else} \end{cases}$$

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

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

$$\sigma = A \xrightarrow{(\pi, \sigma')} (C_1 \Rightarrow C_2) \otimes C_1 \otimes (C_2 \Rightarrow B) \xrightarrow{\text{ev} \otimes \text{id}} C_2 \otimes (C_2 \Rightarrow B) \xrightarrow{\text{ev}} B$$

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

For case 3, taking $M'$ to be some term creating the fresh names $a_1, \ldots, a_n$ with default values and $U, \Gamma, y_1 : \text{ref}_{\xi_1}, \ldots, y_n : \text{ref}_{\xi_n} \vdash M_{\sigma} : \theta$ the one given by the IH for $\sigma'$ (noting $\|\sigma'\| < \|\sigma\|$ as there are no P-names in $\sigma'$), we set:

$$M_\sigma \equiv \text{let } x = M' \text{ in } M_{\sigma'}[\pi_1 x/y_1, \ldots, \pi_n x/y_n]$$. Case 4 is similar.

For case 5, we take:

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

assuming $U, \Gamma, y_1 : \xi_1, \ldots, y_n : \xi_n, y : \text{ref int} \vdash M_{\sigma'} : \theta$ given by the IH.

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

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

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

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.

\[
M_1 \equiv \text{let } x = \text{ref}(0) \text{ in } \lambda y^\text{refint}. x = y
\]
\[
M_2 \equiv \lambda y^\text{refint}. 0
\]
\[
M_3 \equiv \text{let } x = \text{ref}(0) \text{ in let } c = \text{ref}(0) \text{ in } f(\lambda \_. \text{ if !}c = 0 \text{ then div else } x); c := 1; \lambda y^\text{refint}. x = y
\]
\[
M_4 \equiv f(\lambda \_. \text{ div}); \lambda y^\text{refint}. 0
\]
\[
M_5 \equiv \text{let } x = \text{ref}(\text{ref}(0)) \text{ in } \lambda y^\text{refint}. \text{let } z = !x \text{ in if } y = z \text{ then div else } (x := \text{ref}(0); z)
\]
\[
M_6 \equiv \lambda y^\text{refint}. \text{ref}(0)
\]
5.2. Full abstraction

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

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

$$M'_3 \equiv f(\lambda \_ . \text{if } \lnot c = 0 \text{ then div else } x) ; c := 1 ; \lambda y^{\text{ref int}} . x = y$$

Its denotation consists of even-length prefixes of plays of the form\(^\text{10}\)

\[
(\dagger, a_s, a_c)^{S_0} \triangleright^{S_0} S_1 a_s^{S_1} \cdots a_s^{S_n} a_s^{\tau T_0} \triangleright^{\tau T_0} \downarrow T_0 T_1 \uparrow T_1 \cdots a_m^{T_m} i_m^{T_m} \cdots
\]

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

$$[M_3] = A \xrightarrow{(\text{id} \cdot c \cdot [\text{ref(0)}], [\text{ref(0)}])} A \otimes \text{int} \otimes \text{int} \xrightarrow{[M'_3]} \text{int} \Rightarrow \text{Z}$$

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

\[
\uparrow(\dagger, a_s, a_c)^{S_0} \triangleright^{S_0} S_1 a_s^{S_1} \cdots a_s^{S_n} a_s^{\tau T_0} \triangleright^{\tau T_0} \downarrow T_0 T_1 \uparrow T_1 \cdots a_m^{T_m} i_m^{T_m} \cdots
\]

The initial values of $S_0$ are thus determined by $[\text{ref(0)}]$ and, therefore, $S_0(a_s) = S_0(a_c) = 0$. Moreover, $\ast S_1$ is an O-move in $A(\text{int} \Rightarrow \text{Z})$ 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 $\triangleright^{\tau T_0}$ is played, the move $\triangleright^{S_0}$ is hidden from the view of O for the remaining play. That is, for each even-length play $t = (\dagger, a_s, a_c)^{S_0} \cdots \triangleright^{\tau T_0} \cdots \dagger T = (\dagger, a_s, a_c)^{S_0} \triangleright^{\tau T_0} \cdots$. Hence, O cannot ask a question $\ast^{S_0}$ under $\triangleright^{S_0}$ 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 $a_S^{S_1}$ but, rather, immediately answers $\dagger S_0$ with $a^{T_0}$. 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:

$$\dagger (\dagger, a_s, a_c) S_0 \dagger S_0 a_1 T_1 \ldots a_m T_m \ldots$$

and projects on $A(A \Rightarrow Z)$ as:

$$\dagger \dagger \dagger \dagger a_1 T_1 \ldots a_m T_m \ldots$$

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 $nT$, must be a P-move in $BC$. Using again the projection condition on $BC$ and Lemma 5.10, $nT$ must in particular be in $C$, contradicting the IH as $nT$ 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: $u' = u_1q_2q'u_3$. 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: \( u' = u_1 q u_2 q' q_1 \cdots m_1 q z_2 \cdots m_k \), for some \( k \), with each \( m_i \) answering \( q_i \). Let \( i \) be the greatest index such that \( m_1, \ldots, m_i \) 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. \qed

\[ \]

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 5.28**. 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 \( u_1 S_i \Gamma AB = u_1' m_i'^i \) 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 \( (u_1', m_{1i}^{S_i}) \sim (u_2', m_{2i}^{S_i}) \) by hypothesis. Moreover, \( \nu(u_i) \cap \nu(m_{1i}^{S_i}) \subseteq \nu(u'_i) \), for \( i = 1, 2 \). Because if \( a \in \nu(m_{1i}^{S_i}) \setminus \nu(u'_i) \) then \( a \) is introduced by \( m_{1i}^{S_i} \) in \( u'_i m_{1i}^{S_i} \), and therefore introduced by \( m_{1i}^{S_i} \) in \( u_1 m_{1i}^{S_i} \), by Lemma 5.16, so \( a \notin \nu(u_i) \). Thus, by strong support lemma, \( (u_1', u_1, m_{1i}^{S_i}) \sim (u_2', u_2, m_{2i}^{S_i}) \). Finally, for all \( a \in \text{dom}(S_i) \setminus \text{dom}(S_i') \) we have \( a \notin \nu(u'_i m_{1i}^{S_i}) \) and therefore \( S_i(a) = S_i''(a) \), where \( S_i'' \) the store of the last move of \( u_i \). We thus get \( u_1 m_{1i}^{S_i} \sim u_2 m_{2i}^{S_i} \). \qed

**Proof of Lemma 5.29**. By induction on \(|u'_1|\). Suppose \( u'_1 = u''_1 m^{S_i} \). By IH there is \( u''_2 \sqsubseteq u_2 \) with \( u'_1 \sim u''_2 \). Suppose \( m \) is a P-move in \( AB \). Then
$u''_1$ 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 $\sigma$, $(u''_1 m^S \upharpoonright_{\gamma AB}) \sim (u''_2 n^T \upharpoonright_{\gamma AB})$ 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, $u''_1 m^S \sim u''_2 n^T$. \qed
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 $\dagger$ 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:

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

and

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

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.

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,

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

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 \(mth : \text{int} \to \text{int}\), let us assume that each store \(S\) containing \(a\) satisfies \(\langle a.mth, \dagger \rangle \in S\). 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 \(a.mth\) is played:

\[
\ldots m_1^{(a.mth, \dagger)} \ m_2^{(a.mth, \dagger)} \ m_3^{(a.mth, \dagger)} 1^{(a.mth, \dagger)}\]

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

\[
\ldots m_1^{(a.mth, \dagger)} \ m_2^{(a.mth, \dagger)} \ m_3^{(a.mth, \dagger)} 1^{(a.mth, \dagger)} 1^{(a.mth, \dagger)} 1^{(a.mth, \dagger)}\]

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:

\[
\ldots m_1^{(a.mth,\dagger)} \xrightarrow{} m_2^{(a.mth,\dagger)} \xrightarrow{} m_3^{(a.mth,\dagger)} 1^{(a.mth,\dagger)}
\]

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

\[
\text{unit} \rightarrow \text{unit} \rightarrow \text{unit} \quad \text{and} \quad ((\text{unit} \rightarrow \text{unit}) \rightarrow \text{unit}) \rightarrow \text{unit}
\]

with unit \rightarrow unit and (unit \rightarrow unit) \rightarrow 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


References


