

## ISOMORPHISMS OF TYPES IN THE PRESENCE OF HIGHER-ORDER REFERENCES \*

PIERRE CLAIRAMBAULT

Computer Laboratory, University of Cambridge, United Kingdom  
*e-mail address:* pierre.clairambault@cl.cam.ac.uk

**ABSTRACT.** We investigate the problem of type isomorphisms in the presence of higher-order references. We first introduce a finitary programming language with sum types and higher-order references, for which we build a fully abstract games model following the work of Abramsky, Honda and McCusker. Solving an open problem by Laurent, we show that two finitely branching arenas are isomorphic if and only if they are geometrically the same, up to renaming of moves (Laurent’s forest isomorphism). We deduce from this an equational theory characterizing isomorphisms of types in our language. We show however that Laurent’s conjecture does not hold on infinitely branching arenas, yielding new non-trivial type isomorphisms in a variant of our language with natural numbers.

### 1. INTRODUCTION

During the development of denotational semantics of programming languages, there was a crucial interest in defining models of computation satisfying particular type equations. For instance, a model of the untyped  $\lambda$ -calculus can be obtained by isolating a *reflexive* object (that is, an object  $D$  such that  $D \simeq D^D$ ) in a cartesian closed category. In the 80s, some people started to consider the dual problem of finding these equations that must hold in *every* model of a given language: they were coined *type isomorphisms* by Bruce and Longo. In [8], they exploited a theorem by Dezani [9] giving a syntactic characterization of invertible terms in the untyped  $\lambda$ -calculus to prove that the only isomorphisms of types present in simply typed  $\lambda$ -calculus with respect to  $\beta\eta$  equality are those induced by the equation  $A \rightarrow (B \rightarrow C) \simeq B \rightarrow (A \rightarrow C)$ . Later this was extended to handle such things as products [7], polymorphism [8], possibly with unit types [10], or sums [12].

The interest in type isomorphisms grew significantly when their practical impact was realized. In [26], Rittri proposed to search functions in software libraries using their type modulo isomorphism as a key. He also considered the possibilities offered by matching and

---

*1998 ACM Subject Classification:* F.3.2.

*Key words and phrases:* Isomorphisms of types, general references, game semantics.

\* A short version of this work has appeared in the Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS), 2011.

The author acknowledges the support of the (UK) EPSRC grant EP/HO23097 and of the Advanced Grant ECSYM of the ERC.

unification of types modulo isomorphisms [27]. A whole line of research has also been dedicated to the study of type isomorphisms and their use for search tools in richer type systems (such as dependent types [5]), along with studies about the automatic generation of the corresponding coercions [4]. Such tools were implemented for several programming languages, let us mention the command line tool `camlsearch` written by Vouillon for CamlLight. The interested reader may refer to the nice survey by Di Cosmo [11].

It is worth noting that even though these tools are written for powerful programming languages featuring complex computational effects such as higher-order references or exceptions, they rely on the theory of isomorphisms in weaker (purely functional) languages, such as the second-order  $\lambda$ -calculus with pairs and unit types for `camlsearch`. Clearly, all type isomorphisms in  $\lambda$ -calculus are still valid in the presence of computational effects (indeed, the operational semantics are compatible with  $\beta\eta$ ). What is less clear is whether those effects allow the definition of new isomorphisms. However, it seems that syntactic methods deriving from Dezani’s theorem on invertible terms in  $\lambda$ -calculus cannot be extended to complex computational effects. The base setting itself is completely different: there is no longer a canonical notion of normal form, the natural equality between terms is no longer convertibility but observational equivalence, so new methods are required.

In [20], Laurent introduced the idea of applying game semantics to the study of type isomorphisms (although one should mention the precursor characterization of isomorphisms by Berry and Curien [6] in the category of concrete data structures and sequential algorithms). Exploiting his earlier work on game semantics for polarized linear logic [19], he found the theory of isomorphisms for LLP from which he deduced (by translations) the isomorphisms for the call-by-name and call-by-value  $\lambda\mu$ -calculus. The core of his analysis is the observation that isomorphisms between arenas  $A$  and  $B$  in the category **Inn** [14] of arenas and innocent strategies are in one-to-one correspondence with *forest isomorphisms* between  $A$  and  $B$ , so in particular two arenas are isomorphic if and only if their representations as forests are identical up to the renaming of vertices.

From the point of view of computational effects this looks promising, since game semantics are known to accommodate several computational effects such as control operators [17], ground type [2] or higher-order references [1] or even concurrency [18] in one single framework. Moreover, Laurent pointed out in [20] that the main part of his result, namely the fact that each **Inn**-isomorphism induces a forest isomorphism, does not really depend on the innocence hypothesis but only on the weaker *visibility* condition. As a consequence, his method for characterizing isomorphisms still applies to programming languages such as Idealized Algol whose terms can be interpreted as visible strategies [2]. Laurent raised the question whether his result could be proved without the visibility condition, therefore yielding a characterization of isomorphisms in a programming language whose terms have access to higher-order references and hence get interpreted as non-visible strategies [1].

The contributions of this paper are the following: (1) We extend the full abstraction result in [1] in order to deal with sum types and the empty type, (2) We give a new and synthetic reformulation of Laurent’s tools to approach game-theoretically the problem of type isomorphisms, (3) We prove Laurent’s conjecture in the case of finitely branching arenas, allowing us to characterize all type isomorphisms in a finitary (integers-free) programming language  $\mathcal{L}_+$  with higher-order references by the theory  $\mathcal{E}$  presented<sup>1</sup> in Figure 1, (4) We show however a counter-example to the conjecture when dealing with infinitely branching

---

<sup>1</sup>The absence of the equation  $A \rightarrow (B \rightarrow C) \simeq B \rightarrow (A \rightarrow C)$  mentioned in the introduction may seem strange, but is standard in call-by-value [20] due to the restriction of the  $\eta$ -rule on values. Because

$$\begin{aligned}
A \times B &\simeq_{\mathcal{E}} B \times A \\
A \times (B \times C) &\simeq_{\mathcal{E}} (A \times B) \times C \\
1 \times A &\simeq_{\mathcal{E}} A \\
A + B &\simeq_{\mathcal{E}} B + A \\
A + (B + C) &\simeq_{\mathcal{E}} (A + B) + C \\
0 + A &\simeq_{\mathcal{E}} A \\
A \times (B + C) &\simeq_{\mathcal{E}} A \times B + A \times C \\
(A + B) \rightarrow C &\simeq_{\mathcal{E}} (A \rightarrow C) \times (B \rightarrow C) \\
0 \rightarrow A &\simeq_{\mathcal{E}} 1 \\
A \rightarrow 0 &\simeq_{\mathcal{E}} 1 \\
\mathbf{var}[A] &\simeq_{\mathcal{E}} (A \rightarrow 1) \times (1 \rightarrow A)
\end{aligned}$$

Figure 1: Isomorphisms in  $\mathcal{L}_+$ 

arenas, and the counter-example yields a non-trivial type isomorphism in a variant of  $\mathcal{L}_+$  with natural numbers. So Laurent’s conjecture, in the general case, is false.

In Section 2 we introduce the finitary language  $\mathcal{L}_+$  with sums, unit types and higher-order references, on which we define isomorphisms of types. In Section 3, we build a fully abstract games model for  $\mathcal{L}_+$ , drawing inspiration from [1]. Then we turn to the problem of isomorphisms of types. In Section 4 we first give an analysis of isomorphisms in several subcategories of the games model, reproving and extending Laurent’s theorem. Finally, we apply all of this in Section 5 to give a characterization of isomorphisms of types in  $\mathcal{L}_+$  and to obtain new non-trivial isomorphisms in a variant of  $\mathcal{L}_+$  with natural numbers.

## 2. ISOMORPHISMS OF TYPES IN $\mathcal{L}_+$

### 2.1. The language $\mathcal{L}_+$ .

2.1.1. *Syntax.* We introduce here a finitary variant  $\mathcal{L}_+$  of the programming language  $\mathcal{L}$  with higher-order references modeled by Abramsky, Honda and McCusker in [1]: it essentially differs from  $\mathcal{L}$  in the fact that the type for natural numbers has been removed. On the other hand a sum type has been added, allowing to define all polynomial data types. The terms and types of  $\mathcal{L}_+$  are defined as follows.

$$A ::= 0 \mid 1 \mid A + A \mid A \times A \mid A \rightarrow A \mid \mathbf{var}[A]$$

$$\begin{aligned}
M ::= & x \mid \lambda x.M \mid M M \mid \langle M, M \rangle \mid \pi_1 M \mid \pi_2 M \mid () \\
& \mid \iota_1 M \mid \iota_2 M \mid \delta(M, x_1 \cdot N_1, x_2 \cdot N_2) \\
& \mid \mathbf{new}_A \mid M := M \mid !M \mid \mathbf{mkvar} M M
\end{aligned}$$

---

of call-by-value, we also have that 1 is *not* terminal, so we don’t have  $A \rightarrow 1 \simeq 1$ ; instead we have the isomorphism  $A \rightarrow 0 \simeq 1$  up to observational equivalence.

$\frac{}{\Gamma \vdash () : 1}$	$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B}$	$\frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1 M : A}$
$\frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2 M : B}$	$\frac{\Gamma \vdash M : A}{\Gamma \vdash \iota_1 M : A + B}$	$\frac{\Gamma \vdash M : B}{\Gamma \vdash \iota_2 M : A + B}$
$\frac{\Gamma \vdash M : A + B \quad \Gamma, x_1 : A \vdash N_1 : C \quad \Gamma, x_2 : B \vdash N_2 : C}{\Gamma \vdash \delta(M, x_1 \cdot N_1, x_2 \cdot N_2) : C}$		
$\frac{}{\Gamma, x : A \vdash x : A}$	$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$	$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x. M : A \rightarrow B}$
$\frac{}{\Gamma \vdash \mathbf{new}_A : \mathbf{var}[A]}$		$\frac{\Gamma \vdash M : \mathbf{var}[A]}{\Gamma \vdash !M : A}$
$\frac{\Gamma \vdash M : \mathbf{var}[A] \quad \Gamma \vdash N : A}{\Gamma \vdash M := N : 1}$	$\frac{\Gamma \vdash M : A \rightarrow 1 \quad \Gamma \vdash N : 1 \rightarrow A}{\Gamma \vdash \mathbf{mkvar} M N : \mathbf{var}[A]}$	

Figure 2: Typing rules for  $\mathcal{L}_+$ 

The type annotation on **new** will often be omitted, whenever it is irrelevant or obvious from the context. The typing rules for  $\mathcal{L}_+$  are standard, and summarized in Figure 2. Note that in the presence of the empty type, a term constructor is generally included as an elimination rule for 0, along with its typing rule. We skip it here because it is *definable*: as we will see, higher-order references can be used to build an inhabitant  $\perp_A : A$  for all types  $A$ .

**2.1.2. Operational semantics.** This language is equipped with a standard big-step call-by-value operational semantics. To define it, we temporarily extend the syntax of terms with identifiers for **locations**, denoted by  $l$ . Then, **values** are formed as follows:

$$V ::= () \mid \lambda x. M \mid \pi_i V \mid \langle V, V \rangle \mid l \mid \iota_i V \mid \mathbf{mkvar} V V$$

The operational semantics of  $\mathcal{L}_+$  are then given as an inductively generated relation  $(L, s) M \Downarrow (L', s') V$ , where  $L$  is a (functional) set of location-type pairs, and  $s$  is a partial map from locations in  $L$  to values of the corresponding type, with free locations in  $L$ . By abuse of notation, we will write  $l \in L$  if  $(l, A) \in L$  for some type  $A$ . The rules are given in Figure 3. Note that as usual, some store annotations are omitted when the rule considered does not affect the store. For example,

$$\frac{M \Downarrow V \quad M' \Downarrow V'}{M'' \Downarrow V''}$$

is an abbreviation for:

$$\frac{(L, s) M \Downarrow (L', s') V \quad (L', s') M' \Downarrow (L'', s'') V'}{(L, s) M'' \Downarrow (L'', s'') V''}$$

$\frac{}{V \Downarrow V}$	$\frac{M \Downarrow \langle V_1, V_2 \rangle}{\pi_1 M \Downarrow V_1}$	$\frac{M \Downarrow \langle V_1, V_2 \rangle}{\pi_2 M \Downarrow V_2}$	$\frac{M_1 \Downarrow V_1 \quad M_2 \Downarrow V_2}{\langle M_1, M_2 \rangle \Downarrow \langle V_1, V_2 \rangle}$
$\frac{M \Downarrow \iota_1 V_1 \quad M_1[V_1/x_1] \Downarrow V_2}{\delta(M, x_1 \cdot M_1, x_2 \cdot M_2) \Downarrow V_2}$	$\frac{M \Downarrow \iota_2 V_1 \quad M_2[V_1/x_2] \Downarrow V_2}{\delta(M, x_1 \cdot M_1, x_2 \cdot M_2) \Downarrow V_2}$	$\frac{M \Downarrow V}{\iota_1 M \Downarrow \iota_1 V}$	
$\frac{M \Downarrow V}{\iota_2 M \Downarrow \iota_2 V}$	$\frac{M \Downarrow \lambda x. M' \quad N \Downarrow V_1 \quad M'[V_1/x] \Downarrow V_2}{M N \Downarrow V_2}$		
$\frac{M_1 \Downarrow V_1 \quad M_2 \Downarrow V_2}{\text{mkvar } M_1 M_2 \Downarrow \text{mkvar } V_1 V_2}$	$\frac{}{(L, s) \text{ new}_A \Downarrow (L \cup \{l : A\}, s) l} \quad (l \notin L)$		
$\frac{(L, s) M \Downarrow (L', s') l \quad (L', s') N \Downarrow (L'', s'') V}{(L, s) M := N \Downarrow (L'', s'' \cup \{l \mapsto V\}) ()}$		$\frac{(L, s) M \Downarrow (L', s') l \quad s'(l) = V}{(L, s) !M \Downarrow (L', s') V}$	
$\frac{M \Downarrow \text{mkvar } V_1 V_2 \quad N \Downarrow V \quad V_1 V \Downarrow ()}{M := N \Downarrow ()}$		$\frac{M \Downarrow \text{mkvar } V_1 V_2 \quad V_2 () \Downarrow V}{!M \Downarrow V}$	

Figure 3: Big-step operational semantics of  $\mathcal{L}_+$ .

For a closed term  $M$  without free locations, we write  $M \Downarrow$  to indicate that  $(\emptyset, \emptyset) M \Downarrow (L, s) V$  for some  $L, s$  and  $V$  (and  $M \Uparrow$  to indicate that there are no such  $L, s$  and  $V$ ). Observational equivalence  $M \cong N$  between terms  $M$  and  $N$  is then defined as usual, by requiring that for all contexts  $C[-]$  such that  $C[M]$  and  $C[N]$  are closed and contain no free location,  $C[M] \Downarrow$  iff  $C[N] \Downarrow$ . The corresponding equivalence relation is written  $\cong$ .

2.1.3. *Syntactic extensions.* In this core language, one can define all the constructs of a basic imperative programming language. For instance if  $C_1$  has type 1, sequential composition  $C_1; C_2$  is given by:

$$(\lambda x : 1. C_2) C_1$$

This works only because the evaluation of  $\mathcal{L}_+$  is call-by-value. Likewise, a variable declaration  $\text{new } x : A \text{ in } N$  (where  $M$  has type  $A$ ) can be obtained by

$$(\lambda x : \text{var}[A]. N) \text{new}_A$$

and its initialized variant  $\text{new } x = M \text{ in } N$  as expected. As usual with general references one can define a fixed point combinator  $Y_{A \rightarrow B}$  by

$$\begin{aligned} \lambda f : (A \rightarrow B) \rightarrow (A \rightarrow B). \\ \text{new } y : A \rightarrow B \text{ in} \\ y := \lambda a : A. f !y a; \\ !y \end{aligned}$$

This can be easily applied to implement a `while` loop. We can also use it to build an inhabitant  $\perp_A : A$  for any type  $A$ , for example by  $\perp_A = Y_{1 \rightarrow A}(\lambda x. x)()$ .

Sum types can also be used to define datatypes. For instance, we define  $\text{bool} = 1 + 1$ . It is easy to check that the usual combinators for `bool` can be defined using injections and elimination of sums and that they behave in the same way *w.r.t.* the operational semantics.

**2.2. Isomorphisms of types.** We are now ready to define the notion of isomorphism of types in  $\mathcal{L}_+$ .

**Definition 2.1.** If  $A$  and  $B$  are two types of  $\mathcal{L}_+$ , we say that  $A$  and  $B$  are *isomorphic*, denoted by  $A \simeq_{\mathcal{L}_+} B$ , if and only if there are two terms  $x : A \vdash M : B$  and  $y : B \vdash N : A$  such that:

$$\begin{aligned} (x : A \vdash (\lambda y. N)M) &\cong \text{id}_A \\ (y : B \vdash (\lambda x. M)N) &\cong \text{id}_B \end{aligned}$$

where  $\text{id}_A = x : A \vdash x : A$ .

This notion of isomorphism relies on the following notion of composition: if  $x : A \vdash M : B$  and  $y : B \vdash N : C$ , we define  $N \circ M = x : A \vdash (\lambda y. N) M : C$ . Although we do not need it formally, let us note in passing that this composition is associative and behaves well with respect to identities (up to observational equivalence). This can be proved directly, although reasoning on call-by-value  $\beta\eta$ -reductions does not suffice — one needs a more powerful tool such as logical relations. That this composition induces a category will also follow directly from full abstraction since this composition coincides with composition in the games model.

**2.2.1. Isomorphisms and bad variables.** The `mkvar` construct allows to combine arbitrary “write” and “read” methods, forming terms of type `var[A]` not behaving as reference cells: those are called *bad variables*. We chose to include bad variables in the language we consider for two reasons. Firstly, the games models that allow bad variables are notably simpler than those which do not [24], for which our methods do not directly apply. Secondly, the impact of allowing bad variables on our result will be reduced by the following proposition:

**Proposition 2.2.** *Let  $\mathcal{L}'_+$  denote the variant of  $\mathcal{L}_+$  without `mkvar`. Then, if  $A$  and  $B$  are `var`-free types, we have  $A \simeq_{\mathcal{L}_+} B$  if and only if  $A \simeq_{\mathcal{L}'_+} B$ .*

*Proof.* Clearly, if  $A \simeq_{\mathcal{L}'_+} B$  we must have  $A \simeq_{\mathcal{L}_+} B$  as well. Conversely if  $A \simeq_{\mathcal{L}_+} B$ , there are terms  $x : A \vdash M : B$  and  $y : B \vdash N : A$  possibly making use of bad variables, such that  $M \circ N \cong \text{id}_B$  and  $N \circ M \cong \text{id}_A$ . Then, the use of bad variables in  $M$  and  $N$  can be eliminated.

To see how, we consider an extension  $\mathcal{L}''_+$  of  $\mathcal{L}_+$  where we add a type constructor `gvar` for *good variables*, so that  $\mathcal{L}''_+$  has both types `var` for bad variables and `gvar` for good variables. The term constructors for `gvar` are written `newg`, `!gM`, `M :=g N` and obey the same rules as the corresponding constructors for `var`; there is no `mkvarg`. Then, there is translation  $(-)^t$  from  $\mathcal{L}''_+$  to itself, eliminating all uses of `mkvar`. Let us write only the non-trivial cases:

$$\begin{aligned} (\text{var}[A])^t &= \text{gvar}[A] + (A \rightarrow 1) \times (1 \rightarrow A) \\ \text{new}^t &= \iota_1 \text{new}_g \\ l^t &= \iota_1 l \\ (!M)^t &= \delta(M^t, x_1 \cdot !_g x_1, x_2 \cdot \pi_2 x_2 ()) \\ (M := N)^t &= \delta(M^t, x_1 \cdot x_1 :=_g N^t, x_2 \cdot \pi_1 x_2 N^t) \\ (\text{mkvar } M \ N)^t &= \iota_2 \langle M^t, N^t \rangle \end{aligned}$$

In all the other cases, the translation simply goes through the term without changing it. Likewise if  $(L, s)$  is a store,  $(L, s)^t$  is obtained by pointwise application of  $(-)^t$ . It is straightforward to prove by induction that if  $(L, s) M \Downarrow (L', s') V$ , then  $(L, s)^t M^t \Downarrow (L', s')^t V^t$ . The converse is also true and easily provable by induction, with the slightly stronger induction hypothesis that if  $(L, s)^t M^t \Downarrow (L_1, s_1) V_1$  then there exists a store  $(L_0, s_0)$  and a value  $V_0$  in  $\mathcal{L}''_+$  such that  $(L_1, s_1) = (L_0, s_0)^t$ ,  $V_1 = V_0^t$  and  $(L, s) M \Downarrow (L_0, s_0) V_0$ . In particular if  $M$  is closed,  $M \Downarrow$  iff  $M^t \Downarrow$ . This translation is extended to contexts in the straightforward way, with  $[]^t = []$ , such that we always have  $(C[M])^t = C^t[M^t]$ . Note that since  $(-)^t$  does not affect **gvar** it is idempotent, *i.e.*  $t \circ t = t$ .

Since  $\mathcal{L}_+$  is a sublanguage of  $\mathcal{L}''_+$ , there is an obvious translation  $i$  of the former to the latter. Likewise, there is a translation  $j$  from  $\mathcal{L}''_+$  to  $\mathcal{L}_+$  merging the two types for references. Overall,  $j \circ t \circ i$  is a translation from  $\mathcal{L}_+$  to itself whose effect is to eliminate uses of **mkvar**, of course modifying types as a consequence. Note as well that  $j \circ i$  is the identity translation. Putting all of these together, if  $M$  is a closed term of  $\mathcal{L}_+$ , we have:

$$\begin{aligned} C[M] \Downarrow &\Leftrightarrow (C[M])^{toi} \Downarrow \\ &\Leftrightarrow C^{toi}[M^{toi}] \Downarrow \\ &\Leftrightarrow C^{toi}[M^{ttoi}] \Downarrow \\ &\Leftrightarrow C^i[M^{toi}] \Downarrow \\ &\Leftrightarrow C[M^{jtoi}] \Downarrow \end{aligned}$$

Therefore, if  $M \cong N$ , we have  $M^{jtoi} \cong N^{jtoi}$ . But if  $M$  and  $N$  are composable, it is straightforward to check that  $(N \circ M)^{jtoi} = N^{jtoi} \circ M^{jtoi}$ . Similarly, we have  $x^{jtoi} = x$  for any variable  $x$ . From this it follows that if  $M, N$  give a type isomorphism between  $A$  and  $B$ , then  $M^{jtoi}, N^{jtoi}$  give a **mkvar**-free type isomorphism between  $A^{jtoi}$  and  $B^{jtoi}$ . But if  $A$  and  $B$  are **var**-free, we have  $A^{jtoi} = A$  and  $B^{jtoi} = B$ , so we have a **mkvar**-free isomorphism between  $A$  and  $B$ , thus an isomorphism in  $\mathcal{L}'_+$ .  $\square$

2.2.2. *On isomorphisms without bad variables.* In  $\mathcal{L}'_+$ , when are  $\mathbf{var}[A]$  and  $\mathbf{var}[B]$  isomorphic? Without bad variables, there is in general no canonical way to transform a variable of type  $A$  into a variable of type  $B$ . It is easy to see that dereferencing  $M : \mathbf{var}[A]$ , applying the isomorphism between  $A$  and  $B$  and storing the result in a new reference of type  $B$  will not yield an isomorphism because even if the language does not come with a variable equality test, it can be defined on non-trivial types. The handling of good variables with *names* in [25] suggests that to get back the original name when going back and forth between  $\mathbf{var}[A]$  and  $\mathbf{var}[B]$  one has no choice but to simply forward it, which is only possible when  $A$  and  $B$  are syntactically equal. Therefore we expect that a general treatment of isomorphisms with good general references would have to treat variable types as *atoms*, that you can move around but never look inside. We leave that open for future work.

### 3. THE GAMES MODEL

We now describe the fully abstract games model of  $\mathcal{L}_+$ , which closely follows [1] and extends it with sums and the empty type.

3.1. **The basic category.** Our games have two players: Player (P) and Opponent (O).

3.1.1. *Arenas.* Valid plays between  $O$  and  $P$  are generated by directed graphs called *arenas*, which are abstract representations of types. An **arena** is a tuple  $A = \langle M_A, \lambda_A, I_A, \vdash_A \rangle$  where

- $M_A$  is a set of **moves**,
- $\lambda_A : M_A \rightarrow \{O, P\} \times \{Q, A\}$  is a **labeling** function which indicates whether a move is by Opponent or Player, and whether it is a Question or Answer. We write

$$\begin{aligned} \{O, P\} \times \{Q, A\} &= \{OQ, OA, PQ, PA\} \\ \lambda_A &= \langle \lambda_A^{OP}, \lambda_A^{AQ} \rangle \end{aligned}$$

The function  $\overline{\lambda_A}$  denotes  $\lambda_A$  with the  $O/P$  part reversed. A move  $a \in M_A$  is a  $O$ -move (resp.  $P$ -move) if  $\lambda_A^{OP}(a) = O$  (resp.  $\lambda_A^{OP}(a) = P$ ).

- $I_A \subseteq \lambda_A^{-1}(\{OQ\})$  is a set of **initial moves**
- $\vdash_A \subseteq M_A^2$  is a relation called **enabling**, which satisfies that if  $a \vdash_A b$ , then  $\lambda_A^{OP}(a) \neq \lambda_A^{OP}(b)$ , and if  $\lambda_A^{QA}(b) = A$  then  $\lambda_A^{QA}(a) = Q$ .

Additionally, all the arenas we consider will be **finitely branching** (for all  $a \in M_A$ , the set  $\{m \in M_A \mid a \vdash_A m\}$  is finite). This is crucial, since our main result relies on a counting argument.

3.1.2. *Constructions on arenas.* In what follows, if  $S_1$  and  $S_2$  are two sets,  $S_1 + S_2$  will denote their *disjoint union* defined as  $\{(1, x) \mid x \in S_1\} \cup \{(2, x) \mid x \in S_2\}$ . The  $n$ -ary variant of this operation will be written  $\coprod_{i \in I} S_i$ . Whenever convenient, if  $f : S_1 \rightarrow T$  and  $g : S_2 \rightarrow T$  are functions, we will write  $[f, g] : S_1 + S_2 \rightarrow T$  for their co-pairing, *i.e.* the function applying  $f$  on elements of  $S_1$  and  $g$  on elements of  $S_2$ .

We define the **arrow arena**  $A \Rightarrow B$  and the **binary product**  $A \times B$ :

$$\begin{aligned} M_{A \Rightarrow B} &= M_A + M_B \\ \lambda_{A \Rightarrow B} &= [\overline{\lambda_A}, \lambda_B] \\ I_{A \Rightarrow B} &= \{(2, i) \mid i \in I_B\} \\ \vdash_{A \Rightarrow B} &= \{((1, m), (1, n)) \mid m \vdash_A n\} \cup \{((2, m), (2, n)) \mid m \vdash_B n\} \\ &\quad \cup \{((2, i_1), (1, i_2)) \mid (i_1, i_2) \in I_B \times I_A\} \\ \\ M_{A \times B} &= M_A + M_B \\ \lambda_{A \times B} &= [\lambda_A, \lambda_B] \\ I_{A \times B} &= I_A + I_B \\ \vdash_{A \times B} &= \{((1, m), (1, n)) \mid m \vdash_A n\} \cup \{((2, m), (2, n)) \mid m \vdash_B n\} \end{aligned}$$

Another construction of central importance in the model is the **lifted sum**, giving rise to a weak coproduct in **Gam**. If  $(A_i)_{i \in I}$  is a finite family of arenas, we define:



$$\begin{aligned}
M_{\Sigma_{i \in I} A_i} &= \{q\} + \{a_i \mid i \in I\} + \prod_{i \in I} M_{A_i} \\
\lambda_{\Sigma_{i \in I} A_i} &= (1, q) \mapsto OQ \\
&\quad (2, a_i) \mapsto PA \\
&\quad (3, (i, m)) \mapsto \lambda_{A_i}(m) \\
I_{\Sigma_{i \in I} A_i} &= \{(1, q)\} \\
\vdash_{\Sigma_{i \in I} A_i} &= \{((1, q), (2, a_i)) \mid i \in I\} \cup \\
&\quad \{((2, a_i), (3, (i, m))) \mid i \in I \ \& \ m \in I_{A_i}\} \cup \\
&\quad \{((3, (i, m)), (3, (i, n))) \mid m \vdash_{A_i} n\}
\end{aligned}$$

It is obvious that these constructions preserve the fact of being finitely branching. The 0-ary product (the empty arena) is denoted by 1, and will be terminal in our category.

3.1.3. *Plays.* If  $A$  is an arena, a **justified sequence** over  $A$  is a sequence of moves in  $M_A$  together with **justification pointers**: for each non-initial move  $b$ , there is a pointer to an earlier move  $a$  such that  $a \vdash_A b$ . In this case, we say that  $a$  **justifies**  $b$ . The transitive closure of the justification relation is called **hereditary justification**. The relation  $\sqsubseteq$  will denote the prefix ordering on justified sequences. By  $s \sqsubseteq^P t$ , we mean that  $s$  is a  $P$ -ending prefix of  $t$ . If  $s$  is a sequence, then  $|s|$  will denote its length. Moreover if  $i \leq |s|$ ,  $s_i$  will denote the  $i$ -th move in  $s$ . A justified sequence  $s$  over  $A$  is a **legal play** if it is:

- **Alternating:** If  $s'ab \sqsubseteq s$ , then  $\lambda_A^{OP}(a) \neq \lambda_A^{OP}(b)$ .
- **Well-bracketed:** a question  $q$  is **answered** by a later answer  $a$  if  $q$  justifies  $a$ . A justified sequence  $s$  is well-bracketed if each answer is justified by the last unanswered question, that is, the **pending** question.

The set of all legal plays on  $A$  is denoted by  $\mathcal{L}_A$ . We will also be interested in the set  $\mathcal{L}'_A$  of well-bracketed but not necessarily alternating justified sequences on  $A$ , called **pre-legal plays**.

3.1.4. *Strategies, composition.* A **strategy**  $\sigma$  on an arena  $A$  (denoted  $\sigma : A$ ) is a non-empty set of  $P$ -ending legal plays on  $A$  satisfying **prefix-closure**, *i.e.* that for all  $sab \in \sigma$ , we have  $s \in \sigma$  and **determinism**, *i.e.* that if  $sab, sac \in \sigma$ , then  $b = c$ . As usual, strategies form a category which has arenas as objects, and strategies  $\sigma : A \Rightarrow B$  as morphisms from  $A$  to  $B$ . If  $\sigma : A \Rightarrow B$  and  $\tau : B \Rightarrow C$  are strategies, their composition  $\sigma; \tau : A \Rightarrow C$  is defined as usual by first defining the set of **interactions**  $u \in I(A, B, C)$  of plays  $u \in \mathcal{L}_{(A \Rightarrow B) \Rightarrow C}$  such that  $u \upharpoonright A, B \in \mathcal{L}_{A \Rightarrow B}$ ,  $u \upharpoonright B, C \in \mathcal{L}_{B \Rightarrow C}$  and  $u \upharpoonright A, C \in \mathcal{L}_{A \Rightarrow C}$  (where  $s \upharpoonright A, B$  is the usual restriction operation essentially taking the subsequence of  $s$  in  $M_A$  and  $M_B$ , along with the possible natural reassignment of justification pointers). The **parallel interaction** of  $\sigma$  and  $\tau$  is then the set  $\sigma \parallel \tau = \{u \in I(A, B, C) \mid u \upharpoonright A, B \in \sigma \wedge u \upharpoonright B, C \in \tau\}$ , and the composition of  $\sigma$  and  $\tau$  is obtained by the **hiding** operation, *i.e.*  $\sigma; \tau = \{u \upharpoonright A, C \mid u \in \sigma \parallel \tau\}$ . It is known (e.g. [21]) that composition is associative. It admits *copycat strategies* as identities:  $\text{id}_A = \{s \in \mathcal{L}_{A_1 \Rightarrow A_2} \mid \forall s' \sqsubseteq^P s, s' \upharpoonright A_1 = s' \upharpoonright A_2\}$ .

If  $s \in \mathcal{L}_A$ , the **current thread** of  $s$ , denoted  $\lceil s \rceil$ , is the subsequence of  $s$  consisting of all moves hereditarily justified by the same initial move as the last move of  $s$ . All strategies

we are interested in will be *single-threaded*, *i.e.* they only depend on the current thread. Formally,  $\sigma : A$  is **single-threaded** if

- For all  $sab \in \sigma$ ,  $b$  points in  $\lceil sa \rceil$ ,
- For all  $sab, t \in \sigma$  such that  $ta \in \mathcal{L}_A$  and  $\lceil sa \rceil = \lceil ta \rceil$ , we have  $tab \in \sigma$ .

It is straightforward to prove that single-threaded strategies are stable under composition and that  $\text{id}_A$  is single-threaded. Hence, there is a category **Gam** of arenas and single-threaded strategies. The category **Gam** will be the base setting for our analysis. Given arenas  $A$  and  $B$ , the arena  $A \times B$  defines a cartesian product of  $A$  and  $B$  and the construction  $A \Rightarrow B$  extends to a right adjoint  $A \times - \dashv A \Rightarrow -$ , hence **Gam** is cartesian closed and is a model of simply typed  $\lambda$ -calculus.

3.1.5. *Views, classes of strategies.* In this paper, we are mainly interested in the properties of single-threaded strategies. However, to give a complete account of the context it seems necessary to mention several classes of strategies of interest in this setting. The most important one is certainly the class of *innocent* strategies, both for historical reasons and because it is at the core of the frequent definability results – and thus of the full abstraction results – in game semantics. Its definition relies on the notion of  $P$ -view, defined as usual by induction on plays as follows.

$$\begin{aligned} \lceil si \rceil &= i && \text{if } i \in I_A \\ \lceil sa \rceil &= \lceil s \rceil a && \text{if } \lambda_A^{OP}(a) = P \\ \lceil s_1 a s_2 b \rceil &= \lceil s_1 \rceil ab && \text{if } \lambda_A^{OP}(b) = O \text{ and } a \text{ justifies } b \end{aligned}$$

A strategy  $\sigma : A$  is then said to be **visible** if it always points inside its  $P$ -view, that is, for all  $sab \in \sigma$  the justifier of  $b$  appears in  $\lceil sa \rceil$ . The strategy  $\sigma$  is **innocent** if it is visible, and if its behaviour only depends on the information contained in its  $P$ -view. More formally, whenever  $sab, t \in \sigma$  such that  $ta \in \mathcal{L}_A$  and  $\lceil sa \rceil = \lceil ta \rceil$ , we must also have  $tab \in \sigma$ . Both visibility and innocence are stable under composition [14, 2], thus let us denote by **Vis** the category of arenas and visible single-threaded strategies and by **Inn** the category of arenas and innocent strategies. Both categories inherit the cartesian closed structure of **Gam**, but strategies in **Inn** are actually nothing but an abstract representation of ( $\eta$ -long  $\beta$ -normal)  $\lambda$ -terms and form a fully complete model of simply-typed  $\lambda$ -calculus. Strategies in **Vis** have more freedom, they correspond in fact to programs with first-order store [2].

3.2. **The model of  $\mathcal{L}_+$ .** We now show how to turn **Gam** into a model of  $\mathcal{L}_+$ .

3.2.1. *Call-by-value and the  $\text{Fam}_f$  construction.* The three categories **Gam**, **Vis** and **Inn** are categories of *negative games* (in which Opponent always plays first), and these are known to model call-by-name computation whereas  $\mathcal{L}_+$  is call-by-value. We could have modeled it using positive games, following the lines of [13]. Instead, we follow [1] and model  $\mathcal{L}_+$  in the free completion  $\text{Fam}(\mathbf{Gam})$  of **Gam** with respect to coproducts. This will allow us to first characterize isomorphisms in **Gam** (result which could be applied to a call-by-name language with state) then deduce from it the isomorphisms in  $\text{Fam}(\mathbf{Gam})$ . In fact we will consider the completion  $\text{Fam}_f(\mathbf{Gam})$  of **Gam** with respect to *finite* coproducts, since  $\mathcal{L}_+$  has only finite types.

The objects of  $\text{Fam}_f(\mathbf{Gam})$  are finite families  $(A_i)_{i \in I}$  of arenas. A map from  $(A_i)_{i \in I}$  to  $(B_j)_{j \in J}$  is given by a function  $f : I \rightarrow J$  together with a family of strategies  $(\sigma_i)_{i \in I}$  where

for all  $i \in I$ ,  $\sigma_i : A_i \rightarrow B_{f(i)}$ . When  $I$  is a singleton, we will write the family  $(A)_{i \in I}$  simply as  $\{A\}$ . Given families  $A = (A_i)_{i \in I}$  and  $B = (B_j)_{j \in J}$ , their *disjoint* sum is the family  $A + B = (X_i)_{i \in I+J}$  where  $X_{(1,i)} = A_i$ , and  $X_{(2,j)} = B_j$ . Likewise, we define:

$$\begin{aligned} A \times B &= (A_i \times B_j)_{(i,j) \in I \times J} \\ A \Rightarrow B &= (\Pi_{i \in I} (A_i \Rightarrow B_{f(i)}))_{f \in J^I} \end{aligned}$$

With these definitions,  $\text{Fam}_f(\mathbf{Gam})$  inherits a cartesian closed structure from  $\mathbf{Gam}$ . It also has coproducts given by disjoint sum of families, let us write  $\iota_1 : A \rightarrow A + B$  and  $\iota_2 : B \rightarrow A + B$  the injections. As in any bicartesian closed category the product distributes over the sum, let us write  $d_{\Gamma, A, B} : \Gamma \times (A + B) \rightarrow \Gamma \times A + \Gamma \times B$  for the distributivity law. By an abuse of notation we keep using  $1$  for the terminal object of  $\text{Fam}_f(\mathbf{Gam})$ , that is the singleton family containing the empty arena, we write  $!_A : A \rightarrow 1$  for the terminal projection. The category  $\text{Fam}_f(\mathbf{Gam})$  also has an initial object given by the empty family, we denote it by  $0$ .

3.2.2. *Strong monad.* Moreover, the weak coproducts in  $\mathbf{Gam}$  give rise to a *strong monad*  $T$  on  $\text{Fam}_f(\mathbf{Gam})$ . Its image of a family  $(A_i)_{i \in I}$  is given by:

$$TA = \{\Sigma_{i \in I} A_i\}$$

The unit  $\eta_A$  of the monad is the family of strategies  $\text{in}_i : A_i \rightarrow \Sigma_{i \in I} A_i$  (the injections for the weak coproduct structure of  $\Sigma_{i \in I} A_i$ ) which responds to the initial Opponent move by playing  $a_i$  (unless  $i = 0$ ), then plays as copycat. The lifting  $f^* : \Gamma \times TA \rightarrow TB$  of a morphism  $f : \Gamma \times A \rightarrow TB$  is given by the copairing operation of the weak coproduct, and the distributivity law of the product over it. Using this lifting operation, there are two natural ways to define a double strength  $\text{dst}, \text{dst}' : TA \times TB \rightarrow T(A \times B)$ :  $\text{dst}$  interrogates first  $TA$ , whereas  $\text{dst}'$  interrogates first  $TB$ . The fact that  $\text{dst}$  and  $\text{dst}'$  are distinct means that  $T$  is not commutative, and the choice of preferring one or the other parallels the design choice between left-to-right and right-to-left evaluation of a pair in a call-by-value language. Since in  $\mathcal{L}_+$  we have adopted left-to-right evaluation, we prefer  $\text{dst}$  over  $\text{dst}'$ .

Most of the structure of  $\mathcal{L}_+$  (with the exception of memory cells) can be interpreted in  $\text{Fam}_f(\mathbf{Gam})$  following the standard interpretation of a call-by-value language in a cartesian category with a strong monad and Kleisli exponentials [23]: A term  $x_1 : A_1, \dots, x_n : A_n \vdash M : B$  is interpreted as a morphism  $\llbracket M \rrbracket : \Pi_{i \leq n} \llbracket A_i \rrbracket \rightarrow T\llbracket B \rrbracket$ , (where the  $n$ -ary product and its projections  $\pi_i$  is obtained trivially by iteration of the binary product). Details are displayed in Figure 4.

3.2.3. *Interpretation of memory cells.* Of course we also need to give an interpretation for  $\text{var}[A]$ , along with morphisms for the read and write operations of the reference cell. Once again, we follow the lines of [1] and consider the type  $\text{var}[A]$  as the product of its read and write methods, hence we set  $\llbracket \text{var}[A] \rrbracket = (\llbracket A \rrbracket \Rightarrow T1) \times T\llbracket A \rrbracket$ . The interpretation relies on the definition of a morphism  $1 \rightarrow \llbracket \text{var}[A] \rrbracket$ , that is, if  $\llbracket A \rrbracket = \{A_i \mid i \in I\}$ , a strategy  $\text{cell} : (\Pi_{i \in I} (A_i \Rightarrow 1_\perp) \times \Sigma_{i \in I} A_i)_\perp$ , where  $A_\perp = T\{A\}$  is the lift operation. Apart from the initial protocol due to the lift, the strategy  $\text{cell}$  works by associating each read request with the latest write request and playing copycat between them. A more detailed description is given in [1], and an algebraic definition is obtained in [22]. Using  $\text{cell}$  we can complete the interpretation of  $\mathcal{L}_+$  in  $\text{Fam}_f(\mathbf{Gam})_T$ , as displayed in Figure 5. The fact that  $\text{cell}$  behaves correctly is expressed by the following lemma:

$$\begin{aligned}
\llbracket \Gamma \vdash x_i : A_i \rrbracket &= \pi_i; \eta : \prod_{i \leq n} \llbracket A_i \rrbracket \rightarrow T \llbracket A_i \rrbracket \\
\llbracket \Gamma \vdash \lambda x. M : A \rightarrow B \rrbracket &= \Lambda(\llbracket \Gamma, x : A \vdash M : B \rrbracket); \eta : \llbracket \Gamma \rrbracket \rightarrow T(\llbracket A \rrbracket \rightarrow T \llbracket B \rrbracket) \\
\llbracket \Gamma \vdash M N \rrbracket &= \langle \llbracket \Gamma \vdash M : A \rightarrow B \rrbracket, \llbracket N : A \rrbracket \rangle; \text{dst}; \text{ev}^* : \llbracket \Gamma \rrbracket \rightarrow T \llbracket B \rrbracket \\
\llbracket \Gamma \vdash () : 1 \rrbracket &= !_{\llbracket \Gamma \rrbracket}; \eta : \llbracket \Gamma \rrbracket \rightarrow T1 \\
\llbracket \Gamma \vdash \langle M, N \rangle : A \times B \rrbracket &= \langle \llbracket \Gamma \vdash M : A \rrbracket, \llbracket \Gamma \vdash N : B \rrbracket \rangle; \text{dst} \\
\llbracket \Gamma \vdash \pi_i M : A \rrbracket &= \llbracket \Gamma \vdash M : A \times B \rrbracket; T\pi_i \\
\llbracket \Gamma \vdash \iota_i M : A + B \rrbracket &= \llbracket \Gamma \vdash M : A \rrbracket; T\iota_i \\
\llbracket \Gamma \vdash \delta(M, x_1 \cdot M_1, x_2 \cdot M_2) \rrbracket &= \langle \text{id}, \llbracket M \rrbracket \rangle; (\text{d}; (\llbracket M_1 \rrbracket, \llbracket M_2 \rrbracket))^*
\end{aligned}$$

Figure 4: Interpretation of the pure fragment of  $\mathcal{L}_+$ 

$$\begin{aligned}
\llbracket \Gamma \vdash M := N : 1 \rrbracket &= \langle \llbracket \Gamma \vdash M : \text{var}[A] \rrbracket, \llbracket \Gamma \vdash N : A \rrbracket \rangle; \text{dst}; (\pi_1 \times \llbracket A \rrbracket; \text{ev})^* : \llbracket \Gamma \rrbracket \rightarrow T \llbracket 1 \rrbracket \\
\llbracket \Gamma \vdash !M : A \rrbracket &= \llbracket \Gamma \vdash M : \text{var}[A] \rrbracket; \pi_2^* : \llbracket \Gamma \rrbracket \rightarrow T \llbracket A \rrbracket \\
\llbracket \Gamma \vdash \text{mkvar } M N : \text{var}[A] \rrbracket &= \langle \llbracket \Gamma \vdash M : A \rightarrow 1 \rrbracket, \llbracket \Gamma \vdash N : 1 \rightarrow A \rrbracket \rangle; \text{dst} : \llbracket \Gamma \rrbracket \rightarrow T \llbracket \text{var}[A] \rrbracket
\end{aligned}$$

Figure 5: Interpretation of variables

$$\begin{aligned}
\llbracket \Gamma \vdash \text{new } x : A, y : B \text{ in } M \rrbracket &= \llbracket \Gamma \vdash \text{new } y : B, x : A \text{ in } M \rrbracket \\
\llbracket \Gamma, x : \text{var}[A] \vdash \text{new } y : B \text{ in } x := V; M \rrbracket &= \llbracket \Gamma, x : \text{var}[A] \vdash x := V; \text{new } y : B \text{ in } M \rrbracket \\
\llbracket \Gamma \vdash \text{new } x : A, y : B \text{ in } x := V_1; y := V_2; M \rrbracket &= \llbracket \Gamma \vdash \text{new } x : A, y : B \text{ in } y := V_2; x := V_1; M \rrbracket \\
\llbracket \Gamma \vdash \text{new } x : A \text{ in } x := V_1; x := V_2; M \rrbracket &= \llbracket \Gamma \vdash \text{new } x : A \text{ in } x := V_2; M \rrbracket \\
\llbracket \Gamma \vdash \text{new } x : A \text{ in } x := V; !x \rrbracket &= \llbracket \Gamma \vdash \text{new } x : A \text{ in } x := V; V \rrbracket
\end{aligned}$$

Figure 6: Equations concerning assignments and allocations

**Lemma 3.1.** *The equations in Figure 6 hold whenever the terms concerned are well-typed.*

*Proof.* As in [1], the presence of sums does not affect the proof.  $\square$

### 3.3. Full abstraction for $\mathcal{L}_+$ .

**3.3.1. Soundness and adequacy.** If we have a store  $(L, s)$  and a term  $l_1 : \text{var}[A_1], \dots, l_n : \text{var}[A_n] \vdash M : A$  where the  $l_i$ s appear in  $L$  with type  $A_i$ , we write  $\text{new } L, s \text{ in } M$  as a shortcut for  $\text{new } l_1 : A_1, \dots, l_n : A_n \text{ in } l_1 := s(l_1); \dots, l_n := s(l_n); M$ . Note that the order in which variables are introduced and assigned values does not matter, because of Lemma 3.1.

**Proposition 3.2** (Soundness). *If we have  $(L, s)M \Downarrow (L', s')V$ , then for any suitably typed term  $N$  we have  $\llbracket \text{new } L, s \text{ in } (\lambda x. N) M \rrbracket = \llbracket \text{new } L', s' \text{ in } (\lambda x. N) V \rrbracket$ .*

*Proof.* This is proved by induction on the derivation of  $(L, s)M \Downarrow (L', s')V$ , using standard facts about bicartesian closed categories and strong monads, along with the equations of Lemma 3.1.  $\square$

The next step is to extend the adequacy result of [1] with sums, *i.e.* that for any closed term  $M$ , if  $\llbracket M \rrbracket \neq \perp$  then  $M$  converges. We do that by exploiting the retraction  $A + B \triangleleft \mathbf{bool} \times A \times B$ . Consider the language  $\mathcal{L}$  of [1]. We define a translation of  $\mathcal{L}_+$  into  $\mathcal{L}$  by defining  $(A + B)^t = \mathbf{bool} \times A^t \times B^t$ ,  $0^t = 1$ , and  $(-)^t$  preserves all the other constructors. To extend the translation to terms, one must first note that in  $\mathcal{L}$  every type has a value, let us fix a value  $V_A$  for every type  $A$ . Let us define the translation on terms, for the only non-trivial cases:

$$\begin{aligned} (\Gamma \vdash \iota_1 M : A + B)^t &= \Gamma^t \vdash \langle \mathbf{true}, M^t, V_{B^t} \rangle : (A + B)^t \\ (\Gamma \vdash \iota_2 M : A + B)^t &= \Gamma^t \vdash \langle \mathbf{false}, V_{A^t}, M^t \rangle : (A + B)^t \\ (\Gamma \vdash \delta(M, x_1 \cdot N_1, x_2 \cdot N_2) : C)^t &= \Gamma^t \vdash \mathbf{if} \pi_1 M^t \mathbf{then} (\lambda x_1. N_1^t) (\pi_2 M^t) \\ &\quad \mathbf{else} (\lambda x_2. N_2^t) (\pi_3 M^t) : C^t \end{aligned}$$

The translation extends immediately to stores. It is then a straightforward induction to prove that if  $(L, s)^t M^t \Downarrow (L_1, s_1) V_1$ , then there exists a value  $V_0$  and a store  $(L_0, s_0)$  in  $\mathcal{L}_+$  such that  $V_1 = V_0^t$ ,  $(L_1, s_1) = (L_0, s_0)^t$  and  $(L, s) M \Downarrow (L_0, s_0) V_0$ . Therefore if  $M^t \Downarrow$ ,  $M \Downarrow$  as well. Let us define an **embedding** of arena  $\phi : A \hookrightarrow B$  as an injective function  $\phi : M_A \rightarrow M_B$  preserving and reflecting initial moves, enabling and labelling. Likewise, there is an embedding from a family  $(A_i)_{i \in I}$  to  $(B_j)_{j \in J}$  if there is an injective  $f : I \rightarrow J$  and for all  $i \in I$  an embedding  $\phi_i : A_i \hookrightarrow B_{f(i)}$ .

For every type  $A$  and sequent  $\Gamma \vdash A$  we build an embedding  $\phi_A : \llbracket A \rrbracket \hookrightarrow \llbracket A^t \rrbracket$ . We detail the only non-trivial case, *i.e.* the definition of  $\phi_{A+B}$ . Note that if  $A$  is a type in  $\mathcal{L}$  and  $\llbracket A \rrbracket = (A_i)_{i \in I}$ , then the choice of a value  $\vdash V_A : A$  fixes a particular  $i_0 \in I$  (such that  $\llbracket \vdash V_A : A \rrbracket$  responds  $a_{i_0}$  to the Opponent initial move). Then,  $\phi_{A+B}$  is defined from the function:

$$\begin{aligned} f : I + J &\rightarrow (I \times J) + (I \times J) \\ (1, i) &\mapsto (1, (i, j_0)) \\ (2, j) &\mapsto (2, (i_0, j)) \end{aligned}$$

along with the canonical embeddings of  $A_i$  into  $A_i \times B_{j_0}$  and of  $B_j$  into  $A_{i_0} \times B_j$ . This embedding  $\phi : A \hookrightarrow B$  can also be applied move-by-move to plays, hence to strategies. Then, we can prove by induction that for any term  $\Gamma \vdash M : A$ , we have  $\phi_{\Gamma \vdash A}(\llbracket M \rrbracket) \subseteq \llbracket M^t \rrbracket$ . It follows that if  $\llbracket M \rrbracket \neq \perp$ , we have  $\llbracket M^t \rrbracket = \phi_{\Gamma \vdash A}(\llbracket M \rrbracket) \neq \perp$  as well. Thus by the adequacy result in [1],  $M^t \Downarrow$ . Therefore,  $M \Downarrow$ . We have proved:

**Lemma 3.3** (Adequacy). *For any well typed term  $M$ , if  $\llbracket M \rrbracket \neq \perp$  then  $M \Downarrow$ .*

3.3.2. *Definability and full abstraction.* In order to get full abstraction, the main missing ingredient is definability for compact (finite) strategies. In turn, this relies on the following factorization result:

**Proposition 3.4.** *For any arena  $A$  and any finite (as a set of plays) thread-independent strategy  $\sigma : 1 \Rightarrow TA$ , there exist natural numbers  $k_1, k_2$  and an innocent strategy with finite view function  $\tau : (\mathbf{var}[T1])^{k_1} \times \mathbf{var}[\mathbf{bool}^{k_2}] \Rightarrow TA$  such that:*

$$\langle \mathbf{cell}_{T1}, \dots, \mathbf{cell}_{T1}, \mathbf{cell}_{\mathbf{bool}^{k_2}} \rangle; \tau = \sigma$$

Where  $A^k$  is an iterated binary product and  $\langle \sigma_1, \dots, \sigma_{k+3} \rangle = \langle \langle \sigma_1, \dots, \sigma_{k+2} \rangle, \sigma_{k+3} \rangle$ .

*Proof.* The main factorization result of [1] gives a natural number  $k_1$  and a thread-independent finite visible strategy  $\tau_1 : (\mathbf{var}[T1])^{k_1} \Rightarrow TA$  such that  $\langle \mathbf{cell}_{T1}, \dots, \mathbf{cell}_{T1} \rangle; \tau_1 = \sigma$ . The factorization theorem of [2] then allows to factorize  $\tau_1$  as  $\mathbf{cell}_{\mathbf{nat}}; \tau_2$ , where  $\tau_2$  is an innocent strategy with finite view functions. But we have no interpretation for  $\mathbf{nat}$  in our model, since all arenas are supposed finitely branching! Fortunately this is not a problem: the proof works by exploiting an injective function  $code : \tau_1 \rightarrow \mathbb{N}$ , encoding plays in  $\tau_1$  as natural numbers and storing them in the reference cell. However  $\tau_1$  is finite, so for some  $k_2 \in \mathbb{N}$  there is an encoding of  $\tau_1$  in  $\mathbb{B}^{k_2}$ , where  $\mathbb{B} = \{\mathbf{t}, \mathbf{f}\}$ . Exploiting this encoding as in [2] yields the required factorization.  $\square$

**Proposition 3.5** (Definability). *Let  $A$  be a type of  $\mathcal{L}_+$ , and  $\sigma : 1 \Rightarrow T[[A]]$  a finite strategy. Then there is a well-typed term  $\vdash M : A$  of  $\mathcal{L}_+$  such that  $[[M]] = \sigma$ .*

*Proof.* By the above factorization result, we have two natural numbers  $k_1, k_2$  and an innocent strategy  $\tau : (\mathbf{var}[T1])^{k_1} \times \mathbf{var}[\mathbf{bool}^{k_2}] \Rightarrow TA$  with finite view function such that  $\sigma = \langle \mathbf{cell}_{T1}, \dots, \mathbf{cell}_{T1}, \mathbf{cell}_{\mathbf{bool}^{k_2}} \rangle; \tau$ . However, recall that  $\mathbf{var}[A]$  is just a shortcut for  $(A \Rightarrow T1) \times TA$ . Therefore we can apply the definability result for innocent strategies of [3] (the generalization of this result in the presence of the empty type is straightforward), which gives a term:

$$x_1, \dots, x_{k_1} : ((1 \rightarrow 1) \rightarrow 1) \times (1 \rightarrow 1 \rightarrow 1), y : ((\mathbf{bool}^{k_2} \rightarrow 1) \times (1 \rightarrow \mathbf{bool}^{k_2})) \vdash N : A$$

With the use of bad variables, this gives  $x_1, \dots, x_{k_1} : \mathbf{var}[1 \rightarrow 1], y : \mathbf{var}[\mathbf{bool}^{k_2}] \vdash N' : A$  such that  $[[N']] = \tau_2$ . Putting this together, we get  $M = \mathbf{new} x_1, \dots, x_{k_1} : 1 \rightarrow 1, \mathbf{new} y : \mathbf{bool}^{k_2} \in N'$  with  $\vdash M : A$ , such that  $[[M]] = \sigma$ .  $\square$

Given this we can now build the fully abstract model in a standard way, as follows. If  $A$  is an arena, then the **complete** plays on  $A$  are the plays  $s \in \mathcal{L}_A$  such that all questions in  $s$  have been answered. We write  $\sigma \cong \tau$  the fact that  $\sigma, \tau : A$  have the same complete plays. This equivalence extends to  $\mathbf{Fam}_f(\mathbf{Gam})_T$ : if  $A$  and  $B$  are families and  $\sigma, \tau : A \rightarrow TB$  are morphisms  $\mathbf{Fam}_f(\mathbf{Gam})_T$ , we write  $\sigma \cong \tau$  iff for every component  $i$  of  $A$ ,  $\sigma_i \cong \tau_i$ . It is straightforward to check that all the morphism constructions in  $\mathbf{Fam}_f(\mathbf{Gam})$  and  $\mathbf{Fam}_f(\mathbf{Gam})_T$  preserve  $\cong$ , so  $\mathbf{Fam}_f(\mathbf{Gam})_T / \cong$  is also a model of  $\mathcal{L}_+$ . This does not change the interpretation, so we still have soundness and adequacy. Putting all of this together:

**Theorem 3.6** (Full abstraction). *The model is fully abstract, i.e. for all  $M$  and  $N$  of the same type, we have  $M \cong N \iff [[M]] \cong [[N]]$ .*

*Proof.*  $\Rightarrow$ . Suppose  $M \cong N$ . We can assume without loss of generality that  $M$  and  $N$  are closed, since  $\cong$  is a congruence (hence stable under  $\lambda$ -abstraction), so  $\vdash M, N : A$ . Suppose  $[[M]]$  and  $[[N]]$  do not have the same complete plays, e.g.  $s \in [[M]]$  but  $s \notin [[N]]$ . Then,  $qsa \in \mathcal{L}_{[[A]] \Rightarrow T1}$  where  $q$  and  $a$  are respectively the question and answer in  $T1$ . Viewing  $\alpha = qsa$  as a strategy, we have by definability a term  $x : A \vdash M_\alpha : 1$ , such that  $[[M_\alpha]] = \alpha$ . By construction, we have  $[[M]]; [[M_\alpha]] \neq \perp$  and  $[[N]]; [[M_\alpha]] = \perp$ , but  $[[M]]; [[M_\alpha]] = [[(\lambda x.M_\alpha)M]]$  (and similarly for  $N$ ), thus by adequacy  $(\lambda x.M_\alpha)M \Downarrow$  and  $(\lambda x.M_\alpha)N \Uparrow$ , which is absurd. Therefore  $[[M]] \cong [[N]]$ .

$\Leftarrow$ . Suppose  $[[M]] \cong [[N]]$ , and take a context  $C$  such that  $C[M]$  is closed and  $C[M] \Downarrow$ . By soundness,  $[[C[M]]] \neq \perp$ . Since  $[[M]]$  and  $[[N]]$  have the same complete plays, by immediate induction on  $C$  we have  $[[C[N]]] \neq \perp$  as well. By adequacy, we have  $C[N] \Downarrow$  and  $M \cong N$ .  $\square$

4. ISOMORPHISMS IN **Gam**

We are now going to extend Laurent's tools [20] to characterize isomorphisms of types for  $\mathcal{L}_+$ . We will first reformulate Laurent's work in the visible and innocent cases, then extend it to characterize isomorphisms in **Gam**.

**4.1. Isomorphisms and zig-zag strategies.** We first recall Laurent's notion of *zig-zag* play.

**Definition 4.1.** Let  $s \in \mathcal{L}_{A \Rightarrow B}$  be a legal play. It is **zig-zag** if

- (1) Each  $P$ -move following an  $O$ -move in  $A$  (resp. in  $B$ ) is in  $B$  (resp. in  $A$ ),
- (2) A  $P$ -move in  $A$  immediately follows an initial  $O$ -move in  $B$  if and only if it is justified by it,
- (3) The (not necessarily legal) sequences  $s \upharpoonright A$  and  $s \upharpoonright B$  have the same pointers, *i.e.* for all indices  $i, j$  with  $(s \upharpoonright A)_i$  and  $(s \upharpoonright A)_j$  defined,  $(s \upharpoonright A)_i$  points to  $(s \upharpoonright A)_j$  iff  $(s \upharpoonright B)_i$  points to  $(s \upharpoonright B)_j$ .

If  $s$  only satisfies the first two conditions, then it is **pre-zig-zag**.

By extension, we will say that a strategy  $\sigma$  is **pre-zig-zag** (resp. **zig-zag**) if all its plays are so. The core of Laurent's theorem is then that all isomorphisms in **Vis** are zig-zag strategies. His proof does rely on visibility, however it only gets involved to prove that the condition 3 of zig-zag plays is satisfied. The first half of his argument does not use visibility and actually proves that all isomorphisms in **Gam** are pre-zig-zag. Here, being mainly interested in **Gam**, we make this explicit. We need first the following lemma.

**Lemma 4.2** (Dual pre-zig-zag play). *Let  $s \in \mathcal{L}_{A \Rightarrow B}$  be a pre-zig-zag play, then there exists an unique pre-zig-zag  $\bar{s} \in \mathcal{L}_{B \Rightarrow A}$  such that  $\bar{s} \upharpoonright A = s \upharpoonright A$  and  $\bar{s} \upharpoonright B = s \upharpoonright B$ .*

*Proof.* We define  $\bar{s}$  by induction on  $s$ ;  $\bar{\epsilon} = \epsilon$ , and  $\overline{sab} = \bar{s}ba$ . We keep the same pointers, except for the case where a move  $a$  in  $A$  was justified by an initial move  $b$  in  $B$ . Then because of the pre-zig-zag condition on  $s$ ,  $a$  is necessarily an initial move in  $A$  and is set as the new justifier of  $b$  in  $\bar{s}$ . There is no other possible  $\bar{s}$ , since the restrictions on  $A$  and  $B$  are constrained by the hypotheses and their interleaving is forced by the alternation and the pre-zig-zag conditions on  $\bar{s}$ .  $\square$

**Lemma 4.3.** *If  $\sigma : A \Rightarrow B$ ,  $\tau : B \Rightarrow A$  form an isomorphism in **Gam**, then they are pre-zig-zag and for all  $s$ ,  $s \in \sigma \Leftrightarrow \bar{s} \in \tau$ .*

*Proof.* Consider an isomorphism  $\sigma : A \Rightarrow B$ ,  $\tau : B \Rightarrow A$  in **Gam**. We will prove by induction on even  $k \in \mathbb{N}$  that all plays of  $\sigma, \tau$  whose length is less than  $k$  are pre-zig-zag, and that moreover  $\{\bar{s} \mid s \in \sigma \wedge |s| \leq k\} = \{s \in \tau \mid |s| \leq k\}$ .

If  $k = 0$ , this is trivial. Otherwise, suppose this is true up to  $k \in \mathbb{N}$ , and consider  $sab \in \sigma$  of length  $k + 2$ ; let us first prove condition (1). Without loss of generality, suppose  $a \in M_A$ . Since  $s \upharpoonright B = \bar{s} \upharpoonright B$ , by a straightforward zipping argument we can build an interaction  $u \in I(A_1, B, A_2)$  such that  $u \upharpoonright A_1, B = s$  and  $u \upharpoonright B, A_2 = \bar{s}$ , moreover since  $\sigma, \tau$  form an isomorphism we must have  $u \upharpoonright A_1, A_2 \in \text{id}_A$ . Now, we necessarily have  $b \in M_B$ , otherwise  $u$  could be extended to  $uab \in \sigma \parallel \tau$  with  $uab \upharpoonright A_1, A_2 = (u \upharpoonright A_1, A_2)ab$  which is not a play of the identity, contradiction. Hence  $sab$  satisfies condition 1 of pre-zig-zag plays.

To see why it satisfies condition 2, take  $sba \in \sigma$  with  $b$  in  $B$  and  $a$  in  $A$ . If  $b$  is initial in  $B$ , then  $a$  necessarily points to it since  $\sigma$  is single-threaded. Reciprocally, suppose

$a$  points to an initial move in  $B$  earlier than  $b$ . Then we have  $\bar{s} \in \tau$ , and by the same zipping argument as above we have an unique  $u \in I(B_1, A, B_2)$  such that  $u \upharpoonright B_1, A = \bar{s}$  and  $u \upharpoonright A, B_2 = s$ . Since  $\sigma, \tau$  form an isomorphism we also have  $u \upharpoonright B_1, B_2 \in \text{id}_B$ . Let us now extend  $u$  to  $u' = ub_2ab_1$  in the unique way such that  $u' \upharpoonright A, B_2 = sba$  and  $u' \upharpoonright B_1, A \in \tau$ . Note that we are sure that  $b_1$  is a move on  $B_1$  since  $\bar{s}ab_1$  is a play of  $\tau$  of length  $k+2$  and we already know that these satisfy the condition 1 of pre-zig-zag plays. But we also have  $u' \upharpoonright B_1, B_2 \in \text{id}_B$ , hence  $b_2$  points in  $\bar{s}$  as  $b_1$  points in  $s$ . This means that we have  $\bar{s}ab \in \tau$ , such that  $a$  is initial and  $b$  points in  $\bar{s}$ , impossible since  $\tau$  is single-threaded. Hence  $sba$  satisfies condition 2 of pre-zig-zag plays.

We have proved that  $sab$  is pre-zig-zag, so  $\overline{sab}$  is defined. By induction hypothesis  $\bar{s} \in \tau$  and the same reasoning as above shows that it extends to  $\overline{sab} \in \tau$ . The argument is symmetric, hence  $\{\bar{s} \mid s \in \sigma \wedge |s| \leq k+2\} = \{s \in \tau \mid |s| \leq k+2\}$ .  $\square$

For the sake of completeness, let us include Laurent's argument which proves that isomorphisms in **Vis** are zig-zag.

**Lemma 4.4.** *If  $\sigma : A \Rightarrow B$ ,  $\tau : B \Rightarrow A$  form an isomorphism in **Vis**, then  $\sigma$  and  $\tau$  are zig-zag strategies.*

*Proof.* We already know that  $\sigma$  and  $\tau$  are pre-zig-zag strategies. We show by induction on  $n \in \mathbb{N}$  that for all  $s \in \sigma$ , if  $|s| \leq n$  then  $s \upharpoonright A$  and  $s \upharpoonright B$  have the same pointers. Take now  $s \in \sigma$ , and  $sab \in \sigma$ , suppose w.l.o.g. that  $a \in M_A$ . Suppose  $a$  points to  $(s \upharpoonright A)_i$ , then  $b$  points to  $(s \upharpoonright B)_i$ . Indeed, it cannot point to  $(s \upharpoonright B)_j$  with  $j > i$  since that would break visibility for  $\sigma$ . But if it points to  $(s \upharpoonright B)_j$  with  $j < i$  we use the same reasoning on the dual pre-zig-zag play  $\overline{sab}$  and get a contradiction with the fact that  $\tau$  is visible.  $\square$

Let us denote by **Gam** $_i$ , **Vis** $_i$  and **Inn** $_i$  the groupoids of arenas and isomorphisms on the respective categories. In the next sections, we use these facts to give more combinatorial representations of **Gam** $_i$ , **Vis** $_i$  and **Inn** $_i$ .

**4.2. Notions of game morphisms.** Laurent's isomorphism theorem works by relating isomorphisms in **Gam** with isomorphisms in a simpler category which has arenas as objects and *forest morphisms*, i.e. maps on moves that preserve initiality and enabling. Relaxing the visibility conditions requires us to also consider relaxed notions of game morphisms, that we present here.

In what follows we will make use of the **prefix functions**  $\text{ip}$  and  $\text{jp}$  on justified sequences, defined by  $\text{ip}(\epsilon) = \epsilon$  and  $\text{ip}(sa) = s$ , and  $\text{jp}(si) = \epsilon$  if  $i$  does not have a pointer,  $\text{jp}(s_1as_2b) = s_1a$  if  $b$  points to  $a$ .

**Definition 4.5.** Let  $A$  be an arena. A **path** on  $A$  is a play  $s \in \mathcal{L}_A$  such that except for the initial move, every move in  $s$  points to the previous move. Formally, for all  $s'ab \sqsubseteq s$ ,  $a$  justifies  $b$  in  $s$ . Let  $\mathcal{P}_A$  denote the set of paths on  $A$ . A **path morphism** from  $A$  to  $B$  is a function  $\phi : \mathcal{P}_A \rightarrow \mathcal{P}_B$  such that  $\text{ip} \circ \phi = \phi \circ \text{ip}$  and which preserves  $Q/A$  labeling: for all  $sa \in \mathcal{P}_A$  with  $\phi(sa) = \phi(s)b$ , we have  $\lambda_A^{QA}(a) = \lambda_B^{QA}(b)$ . There is a category **Path** of arenas and path morphisms.

This category **Path** comes with its own notion of isomorphisms of arenas. Note that whenever  $A$  is a forest, this is exactly Laurent's notion of forest isomorphism. We now introduce two weaker notions of morphisms for arenas. In what follows, let us call a legal play on  $A$  with only one initial move a **thread** on  $A$ , and denote the set of threads on  $A$



by  $\mathcal{T}_A$ . Likewise, let us call a pre-legal play with one initial move a pre-legal thread and let us denote these by  $\mathcal{T}'_A$ .

**Definition 4.6.** Let  $A, B$  be arenas, and let  $\phi : \mathcal{T}'_A \rightarrow \mathcal{T}'_B$ . We say that  $\phi$  is a **sequential morphism** from  $A$  to  $B$  if  $\text{ip} \circ \phi = \phi \circ \text{ip}$ , and if it preserves  $Q/A$  labeling, *i.e.* for all  $\phi(sa) = \phi(s)b$  we have  $\lambda_A^{QA}(a) = \lambda_B^{QA}(b)$ . We say that it is a **justified morphism** if, additionally,  $\text{jp} \circ \phi = \phi \circ \text{jp}$ . There are two categories **Seq** of arenas and sequential morphisms and **Jus** of arenas and justified morphisms.

The condition on sequential morphisms amounts to the fact that they preserve play extension, *i.e.* for all pre-legal threads  $sa \in \mathcal{T}'_A$ ,  $\phi(sa)$  must be an immediate extension of  $\phi(s)$ . In other words, a sequential morphism preserves the forest structure of the set of pre-legal threads given by the prefix ordering. However it does not have to preserve pointers : it could for instance send a play  $\circ - \bullet - \circ - \bullet$  to  $\circ - \overbrace{\bullet - \circ} - \bullet$ , where occurrences of  $\circ$  and  $\bullet$  are respectively Opponent and Player moves. These weak forms of morphisms will play an important role in the subsequent development as they have a close relationship to isomorphisms in **Gam**. Justified morphisms are those sequential morphisms which additionally preserve pointers: those will appear to be in relationship with isomorphisms in **Vis**.

As above, we will denote by **Seq<sub>i</sub>**, **Jus<sub>i</sub>** and **Path<sub>i</sub>** the groupoids of invertible maps in **Seq**, **Jus** and **Path**. These groupoids will soon appear to be identical to **Gam<sub>i</sub>**, **Vis<sub>i</sub>** and **Inn<sub>i</sub>**. To prove this, we need the following lemma.

**Lemma 4.7.** *Let  $s \in \mathcal{T}'_A$ , and  $\sigma : A \Rightarrow B$  an isomorphism in **Gam**. There is then an unique play  $s' \in \sigma$  such that  $s' \upharpoonright A = s$ .*

*Proof.* Remark first that if  $\sigma : A \Rightarrow B$  and  $\tau : B \Rightarrow A$  are inverses then they are both total, *i.e.* for all  $s \in \sigma$  and  $sa \in \mathcal{L}_{A \Rightarrow B}$  there must be  $b$  such that  $sab \in \sigma$ , assuming it is not the case easily leads to a contradiction. We now prove the lemma by induction on  $s$ . If  $s = \epsilon$ , this is trivial. Otherwise, suppose  $sa \in \mathcal{T}'_A$  and we have by induction hypothesis  $s' \in \sigma$  such that  $s' \upharpoonright A = s$ . If  $a$  is a  $P$ -move in  $A$  (hence an  $O$ -move in  $A \Rightarrow B$ ), there is an unique  $b$  such that  $s'ab \in \sigma$ , and we do have  $s'ab \upharpoonright A = sa$ . If  $a$  is an  $O$ -move in  $A$  (hence a  $P$ -move in  $A \Rightarrow B$ ), then let  $\tau : B \Rightarrow A$  be the inverse of  $\sigma$ , since  $s' \in \sigma$  we have  $\overline{s'} \in \tau$ . Being part of an isomorphism  $\tau$  is total, hence there is  $b$  such that  $\overline{s'}ab \in \tau$ . We deduce from this that  $s'ba \in \sigma$ , and we have  $s'ba \upharpoonright A = sa$  as needed. This choice is unique: if there is another play  $t \in \sigma$  such that  $t \upharpoonright A = sa$ , then  $t = t'b'a$  (since  $t$  is zig-zag). By induction hypothesis we have  $t' = s'$ , thus  $s'b'a \in \sigma$ . From this we deduce that  $\overline{s'}ab' \in \tau$ , so  $b = b'$  by determinism of  $\tau$ .  $\square$

**Proposition 4.8.** *If  $\mathcal{C} \simeq \mathcal{D}$  means that two groupoids  $\mathcal{C}$  and  $\mathcal{D}$  are isomorphic, then we have:*

$$\begin{aligned} \mathbf{Gam}_i &\simeq \mathbf{Seq}_i \\ \mathbf{Vis}_i &\simeq \mathbf{Jus}_i \end{aligned}$$

*Proof.* Let us first define a functor  $F : \mathbf{Gam}_i \rightarrow \mathbf{Seq}_i$ . It is defined as the identity on arenas. Let  $\sigma : A \Rightarrow B$  be an isomorphism, and let  $s \in \mathcal{T}'_A$  then we define  $\phi_\sigma(s) = s' \upharpoonright B$ , where  $s'$  is the unique play on  $A \Rightarrow B$  which existence is ensured by the lemma above. The function  $\phi_\sigma$  commutes with  $\text{ip}$  since  $\sigma$  is a pre-zig-zag strategy. To any question it cannot associate an answer, as that would immediately break well-bracketing on  $\sigma$ . But to any

answer it cannot associate a question, as that would immediately break well-bracketing on  $\sigma^{-1}$ . Then we define  $F(\sigma) = \phi_\sigma$ . It is obvious that  $F$  preserves identities and composition<sup>2</sup>.

Reciprocally, suppose  $\phi : A \rightarrow B$  is a sequential isomorphism. We mimic the usual definition of the identity by setting  $G(\phi) = \{s \in \mathcal{L}_{A \Rightarrow B} \mid \forall s' \sqsubseteq^P s, \phi(s' \upharpoonright A) = s' \upharpoonright B\}$  (We apply  $\phi$  on plays whereas it is normally only defined on *threads*, however it can be canonically extended to plays, so this is not ambiguous). It is obvious that this construction is functorial, and that it is inverse to  $F$ .

We have now an isomorphism  $\mathbf{Gam}_i \simeq \mathbf{Seq}_i$  which restricts naturally to  $\mathbf{Vis}_i$  and  $\mathbf{Jus}_i$ . Indeed if  $\sigma : A \Rightarrow B$  is a visible isomorphism, it is a zig-zag strategy therefore  $s \in \mathcal{T}'_A$  and  $\phi_\sigma(s)$  have the same pointers, which means that  $\text{jp} \circ \phi_\sigma = \phi_\sigma \circ \text{jp}$ . Reciprocally if  $\phi_\sigma$  is a justified morphism, all  $s \in \sigma$  must be such that  $s \upharpoonright A$  and  $s \upharpoonright B$  have the same pointers, therefore  $\sigma$ , being pre-zig-zag, always points in its  $P$ -view.  $\square$

**4.3. Innocent and visible case.** In this section, we use the framework described above to recall Laurent's results. We have proved above that isomorphisms in  $\mathbf{Vis}$  correspond to isomorphisms in  $\mathbf{Jus}$ , which we are now going to compare with isomorphisms in  $\mathbf{Path}$ .

**Lemma 4.9.** *There is a full functor  $H : \mathbf{Vis}_i \rightarrow \mathbf{Path}_i$ .*

*Proof.* We have built in the above section a full and faithful functor (actually an isomorphism)  $F : \mathbf{Vis}_i \rightarrow \mathbf{Jus}_i$ . From a visible isomorphism  $\sigma : A \Rightarrow B$  we set  $H(\sigma) = F(\sigma) \upharpoonright \mathcal{P}_A$ , where  $f \upharpoonright E'$  restricts a function  $f : E \rightarrow F$  to a subset  $E' \subseteq E$  of its domain. The image of a path by  $F(\sigma)$  is always a path since it is a justified morphism, hence  $H(\sigma) : \mathcal{P}_A \rightarrow \mathcal{P}_B$ .

To see why  $H$  is full, suppose we have a path morphism  $\phi : \mathcal{P}_A \rightarrow \mathcal{P}_B$ . Then  $\phi$  admits a canonical extension  $\phi^* : \mathcal{T}'_A \rightarrow \mathcal{T}'_B$ . To define  $\phi^*(s)$  we reason by induction on  $s$ , and set  $\phi^*(\epsilon) = \epsilon$  and  $\phi^*(sa) = \phi^*(s)a'$ , where  $a'$  is the last move of  $\phi(p_a)$ ,  $p_a$  being the path of  $a$  in  $s$ . The move  $a'$  keeps the same pointer as  $a$ . It is clear that this defines as needed a justified morphism  $\phi^*$  such that  $H(\phi^*) = \phi$ .  $\square$

This ensures that arenas  $A$  and  $B$  are isomorphic in  $\mathbf{Vis}$  if and only if they are isomorphic in  $\mathbf{Path}$ , *i.e.* they are geometrically the same. Let us mention that as Laurent proved, this correspondence is one-to-one in the innocent case: one can prove that there is only one innocent zig-zag strategy corresponding to a particular path isomorphism, hence  $H$  restricts to an isomorphism of groupoids  $H' : \mathbf{Inn}_i \rightarrow \mathbf{Path}_i$ .

**Example 4.10.** Note that  $H$  itself is *not* faithful: we can exploit non-innocence to build non-uniform isomorphisms, *i.e.* isomorphisms which change their underlying path isomorphism as the interaction progresses. For an example, consider the arena

$$A = \begin{array}{ccc} & & q \\ & \curvearrowright & \vdots \\ q_1 & & a \\ \vdots & & \\ a & & \end{array} \quad \begin{array}{c} q_2 \\ \vdots \\ a \end{array}$$

which is the interpretation of  $(\text{bool} \rightarrow 1) \rightarrow 1$  in call-by-value and of  $1 \times 1 \rightarrow 1$  in call-by-name. Consider now the strategy  $i : A \Rightarrow A$  which behaves as follows. It starts by playing as the identity on  $A$ . The first time Opponent plays  $q_1$  or  $q_2$  on the left hand side, it simply copies it. Starting from the second time Opponent plays  $q_1$  or  $q_2$  though, it swaps them. An

<sup>2</sup>In fact, this construction can be seen as a particular case of Hyland and Schalk's faithful functor from games to relations [15], where the relation happens to be functional.

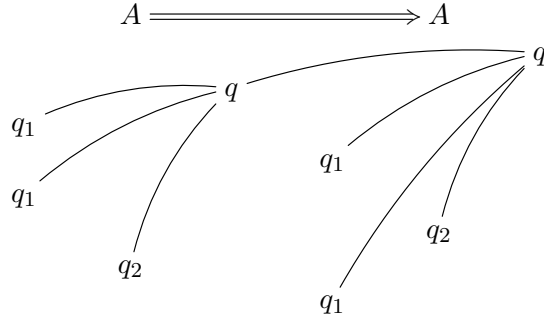
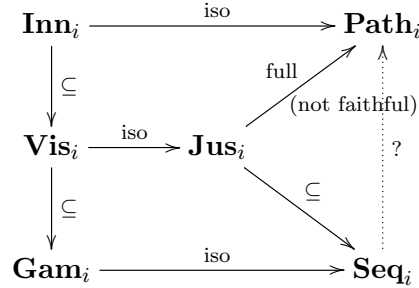
Figure 7: A play of the non-trivial involution  $i$  on  $A$ 

Figure 8: Relations between all groupoids of isomorphisms

example play of  $i$  is given in Figure 7. Although it is not the identity,  $i$  is its own inverse. Its image by  $H$  only takes into account the first behaviour of  $i$ , thus is the same as for  $\text{id}_A$ : the identity path morphism on  $A$ . From this strategy we can extract the following term  $f : B \vdash M : B$  of  $\mathcal{L}_+$ , where  $B = (\text{bool} \rightarrow 1) \rightarrow 1$ .

```

new r := true in
λg.f(λb.if !r then r := false; g b else g (not b))

```

Although  $M$  is not the identity it is an involution on  $B$ , *i.e.* we have  $(\lambda f.M)(Mx) \cong_{\mathcal{L}_+} x$ . Such non-trivial involutions cannot be defined using only purely functional behaviour.

We give in Figure 8 a summary of all the groupoids of isomorphisms encountered so far, along with their relations. Following it, the question of finding the isomorphisms in **Gam** boils down to the definition of an arrow from **Seq<sub>i</sub>** to **Path<sub>i</sub>** in this diagram, which is what we will attempt in the next two subsections.

**4.4. Non-visible isomorphisms by counting.** We have seen above that we can build a full functor  $\mathbf{Vis}_i \rightarrow \mathbf{Path}_i$ , which allows to characterize isomorphic arenas in **Vis**. However, this construction relies heavily on visibility. We now investigate how to get rid of it and prove that two arenas  $A$  and  $B$  are isomorphic in **Gam** if and only if they are isomorphic in **Path**. In this subsection, we will describe for pedagogical reasons an intuitive approach to the proof, which relies on counting. However this approach suffers from some defects, hence the full proof (described in the next subsection) will follow slightly different lines.

If  $a \in M_A$ , let us call its **arity** the quantity  $ar(a) = |\{m \in M_A \mid a \vdash_A m\}|$ . On pre-legal threads  $s \in \mathcal{T}'_A$  we define:

$$Q(s) = \sum_{i=1}^{|s|} ar(s_i)$$

If  $s \in \mathcal{T}'_A$ ,  $Q(s)$  is also the number of ways  $s$  can be extended to some  $sa$  (let us recall here that as a member of  $\mathcal{T}'_A$ ,  $s$  need not be alternating): the choice of a justifier  $s_i$  plus a move enabled by  $s_i$ . These definitions allow to express the following observation. If  $\sigma : A \Rightarrow B$  is an isomorphism (thus a pre-zig-zag strategy) and  $s \in \sigma$ , then  $Q(s \upharpoonright A) = Q(s \upharpoonright B)$ , because  $\sigma$  being an isomorphism, it must associate each possible extension of  $s \upharpoonright A$  to a unique extension of  $s \upharpoonright B$ . But this also means that if  $sab \in \sigma$  we have  $Q(s \upharpoonright A) + ar(a) = Q((s \upharpoonright A)a) = Q((s \upharpoonright B)b) = Q(s \upharpoonright B) + ar(b)$ , hence  $ar(a) = ar(b)$ . Thus to each move  $a$ ,  $\sigma$  must associate a move with the same arity. This is a step in the right direction, but we would like a deeper connection between  $a$  and  $b$ .

If  $a \in M_A$ , we will use the notation  $J_a = \{m \in M_A \mid a \vdash_A m\}$ . Let us define by induction on  $k$  the notion of a  $k$ -isomorphism between  $a \in M_A$  and  $b \in M_B$ . For any  $a \in M_A$  and  $b \in M_B$  there is automatically a 0-isomorphism  $i_{a,b}$ . A  $(k+1)$ -isomorphism from  $a$  to  $b$  is the data of an isomorphism  $f : J_a \rightarrow J_b$  along with, for all  $m \in J_a$ , a  $k$ -isomorphism  $f_m : m \rightarrow f(m)$ . We use the notation  $m \simeq_k n$  to denote the fact that there is a  $k$ -isomorphism from  $m$  to  $n$ . In other words, we have  $m \simeq_k n$  if the tree of paths of length at most  $k$  starting from  $m$  is tree-isomorphic to the tree of paths of length at most  $k$  starting from  $n$ . If  $k_1 \leq k_2$ ,  $f_1$  is a  $k_1$ -isomorphism and  $f_2$  is a  $k_2$ -isomorphism, we say that  $f_1$  is a prefix of  $f_2$  if they agree up to depth  $k_1$ . Note that in particular we have  $m \simeq_1 n$  if and only if  $ar(m) = ar(n)$ , so  $m \simeq_k n$  is indeed a generalization of  $ar(m) = ar(n)$ . By induction on  $k$ , one can then prove that  $\sigma$  must always associate to each move  $m$  a move  $n$  such that  $m \simeq_k n$ : to prove it for  $k+1$ , just apply the counting argument above on  $\simeq_k$ -equivalence classes. From all these  $k$ -isomorphisms, one can then deduce the existence of a path isomorphism between  $A$  and  $B$ .

This counting argument has several unsatisfying aspects, which are caused by the implicit use of the following lemma.

**Lemma 4.11** (Slicing of bijections). *Suppose  $E = E_1 + E_2$  and  $F = F_1 + F_2$  are finite sets, and that  $f : E \rightarrow F$  and  $g : E_1 \rightarrow F_1$  are bijections. Then there is a bijection  $f \setminus g : E_2 \rightarrow F_2$ .*

This lemma is obviously true by cardinality reasons. However this proof is, computationally speaking, “almost non-effective”, in the sense that the isomorphism it produces implicitly depends on the choice of a total ordering for  $E$  and  $F$ . A consequence of that is that from any isomorphism in **Gam** we will extract an isomorphism in **Path**, but we cannot hope its choice to be canonical, for any reasonable meaning of “canonical”. Even worse, the witness isomorphisms given by this proof for  $\simeq_k$  and  $\simeq_{k+1}$  need not agree together. This implies that for infinitely deep arenas, one requires König’s lemma to actually build a path isomorphism from a game isomorphism. This means that we cannot deduce from the proof above an algorithm to extract path isomorphisms.

**4.5. Extraction of a path isomorphism.** To obtain a more computationally meaningful extraction of a path iso from a game iso, we must replace the proof of Lemma 4.11 by

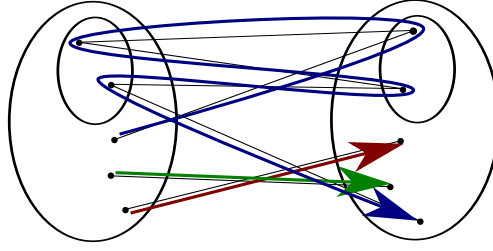


Figure 9: Slicing of isomorphisms.

something else than counting. As formalized in the following proof, the idea is to remark that given the data of Lemma 4.11, starting from  $x \in E_2$ , the sequence

$$\begin{aligned} x_0 &= f(x) \\ x_{n+1} &= f \circ g^{-1}(x_n) \end{aligned}$$

must eventually reach  $F_2$ , as illustrated in Figure 9, yielding a bijection between  $E_2$  and  $F_2$  (this corresponds to the construction of a *trace* [16] on the category of finite sets and permutations).

**Proposition 4.12.** *If  $\phi : A \rightarrow B$  is a sequential play isomorphism, then for all  $sa \in \mathcal{T}'_A$  with  $\phi(sa) = \phi(s)b$ , there is a family  $(h_{s,sa}^k)_{k \in \mathbb{N}}$  such that for all  $k$ ,  $h_{s,sa}^k$  is a  $k$ -isomorphism from  $a$  to  $b$ . This family is coherent, in the following sense: if  $k_1 \leq k_2$ ,  $h_{s,sa}^{k_1}$  is a prefix of  $h_{s,sa}^{k_2}$ .*

*Proof.* We will use the following notations. If  $s \in \mathcal{T}'_A$ ,  $E_s$  will be the set of atomic extensions of  $s$ , that is of plays  $sa \in \mathcal{T}'_A$ , and  $F_s$  will be the set of atomic extensions of  $\phi(s)$ . For all plays  $sa \in \mathcal{T}'_A$ , although strictly speaking  $E_s$  is *not* a subset of  $E_{sa}$ , we have the following decomposition:

$$E_{sa} = E_s + J_a$$

Indeed, a move extending  $sa$  can either point to some  $s_i$  or to  $a$ . Note also that for any  $s$ ,  $\phi : sa \mapsto \phi(s)b$  induces an isomorphism  $f_s : a \mapsto b$  from  $E_s$  to  $F_s$ .

For all  $s \in \mathcal{T}'_A$  and  $sa \in E_s$ , we follow the reasoning illustrated in Figure 9 and consider a bipartite directed graph  $G_{s,sa}$  defined as follows: its set of vertices is  $V = E_{sa} + F_{sa}$  and its set of edges is  $E = \{(x, f_{sa}(x)) \mid x \in E_{sa}\} + \{(y, f_s^{-1}(y)) \mid y \in F_s\}$ . This graph is “deterministic”, in the sense that the outwards degree of each vertex is at most one, moreover the only vertices whose outwards degree is 0 are those of  $J_b$  (where  $b = f_s(a)$ , so  $F_{sa} = F_s + J_b$ ). Moreover  $G_{s,sa}$  must be acyclic, since  $f_s$  and  $f_{sa}$  are isomorphisms. Thus from any vertex in  $J_a$ , there is a unique path in  $G$  leading to a vertex in  $J_b$ ; this induces an isomorphism  $g_{s,sa} : J_a \rightarrow J_b$ . For each pair  $(m, g_{s,sa}(m))$  we also keep track of the corresponding path  $p_{s,sa}^m = (m, f_{sa}(m), f_s^{-1}(f_{sa}(m)), \dots, g_{s,sa}(m))$ .

It is now time to build the  $k$ -isomorphisms, by induction on  $k$ . For  $k = 0$  this is obvious. For fixed  $k + 1 \geq 1$ , by induction hypothesis there is for each  $sa \in \mathcal{T}'_A$  with  $\phi(sa) = \phi(s)b$  a  $k$ -isomorphism  $h_{s,sa}^k$  from  $a$  to  $b$ . In particular, for fixed  $sa \in \mathcal{T}'_A$ , consider the graph  $G_{s,sa}$ . Each of its edges of the form  $(x, f_{sa}(x))$  are now labeled by the  $k$ -isomorphism  $h_{sa,x}^k$  and all its edges of the form  $(y, f_s^{-1}(y))$  are labeled by  $(h_{s,f_s^{-1}(y)}^k)^{-1}$ . For each pair  $(m, g_{s,sa}(m))$  we can now compose the labels along the path  $p_{s,sa}^m$  and get a  $k$ -isomorphism  $i_m : m \rightarrow g_{s,sa}(m)$ .

We then define  $h_{s,sa}^{k+1} = (g_{s,sa}, (i_m)_{m \in J_a})$  which is as needed a  $(k+1)$ -isomorphism from  $a$  to  $b$ .

Note finally that if  $k_1 \leq k_2$ ,  $h_{s,sa}^{k_1}$  is a prefix of  $h_{s,sa}^{k_2}$ . This is proved by simultaneous induction on  $k_1$  and  $k_2$ . If  $k_1 = 0$  this is obvious. Otherwise, it relies on the fact that the graph  $G_{s,sa}$  does not depend on  $k$ . Hence  $h_{s,sa}^{k_1+1} = (g_{s,sa}, (i_m)_{m \in J_a})$  and  $h_{s,sa}^{k_2+1} = (g_{s,sa}, (j_m)_{m \in J_a})$ , and each  $i_m$  has been obtained from  $k_1$ -isomorphisms in the same way as  $j_m$  has been obtained from  $k_2$ -isomorphisms, so it immediately boils down to the induction hypothesis.  $\square$

**Theorem 4.13.** *Two finitely branching arenas  $A$  and  $B$  are **Gam**-isomorphic if and only if they are **Path**-isomorphic.*

*Proof.* Consider an isomorphism  $\sigma : A \Rightarrow B$  in **Gam**. Restricted on plays with only two moves, it gives an isomorphism  $f : I_A \rightarrow I_B$ . By the previous proposition, there is for each  $i \in I_A$  and for each  $k \in \mathbb{N}$  a  $k$ -isomorphism  $h_{\epsilon,i}^k : i \rightarrow f(i)$ . Additionally, all these  $k$ -isomorphisms are compatible with each other, so they converge to an  $\omega$ -isomorphism  $h_{\epsilon,i} : i \rightarrow f(i)$ . The iso  $f$  together with  $h_{\epsilon,i}$  for all  $i$  define a path isomorphism from  $A$  to  $B$ .  $\square$

For each pair of arenas  $A, B$ , we have a function  $K_{A,B} : \mathbf{Gam}_i(A, B) \rightarrow \mathbf{Path}_i(A, B)$ . Unfortunately, this function fails to be a functor. Indeed, the construction is based on the more explicit proof of Lemma 4.11 illustrated in Figure 9, which is not functorial; one can easily find sets  $E = E_1 + E_2$ ,  $F = F_1 + F_2$ ,  $G = G_1 + G_2$  along with bijections  $f_1 : E \rightarrow F$ ,  $f_2 : E_1 \rightarrow F_1$ ,  $g_1 : F \rightarrow G$  and  $g_2 : F_1 \rightarrow G_1$  such that  $(f \setminus f'); (g \setminus g') \neq (f; g) \setminus (f'; g')$ , and extract from this a counter-example for the functoriality of  $K_{A,B}$ . However,  $K$  is a natural transformation:

**Proposition 4.14.** *The family  $K_{A,B} : \mathbf{Gam}_i(A, B) \rightarrow \mathbf{Path}_i(A, B)$  is natural in  $A$  and  $B$ , where both  $\mathbf{Gam}_i(-, -)$  and  $\mathbf{Path}_i(-, -)$  are seen as bifunctors from  $\mathbf{Path}_i^{op} \times \mathbf{Path}_i$  to **Set** (using implicitly the faithful functor from  $\mathbf{Path}_i$  to  $\mathbf{Gam}_i$  of Figure 8).*

*Proof.* The naturality conditions expresses invariance of  $K_{A,B}$  under renaming of moves in  $A$  and  $B$ , as composing with **Path**-isomorphisms or **Gam**-isomorphisms generated from **Path**-isomorphisms only rename moves. The proof proceeds by showing that all  $k$ -isomorphisms  $h_{s,sa}^k$  on which the definition of  $K$  relies are invariant under renaming of moves, by induction on  $k$ , then on  $s$ .  $\square$

## 5. SYNTACTIC ISOMORPHISMS

**5.1. Application to  $\mathcal{L}_+$ .** Our isomorphism theorem most naturally applies to **Gam** (so to call-by-name languages), but  $\mathcal{L}_+$  is modeled in  $\mathbf{Fam}_f(\mathbf{Gam})_T$ , so we have to check how our result extends to this. Let us first relate isomorphisms in  $\mathbf{Fam}_f(\mathbf{Gam})_T$  and isomorphisms in **Gam**. We start by recalling some terminology: an arena  $A$  is **pointed** if it has only one initial move. A strategy  $\sigma : A \rightarrow B$  where  $A$  and  $B$  are pointed is **strict** if it responds to the initial move in  $B$  with the initial move in  $A$ , which it never plays again. Pointed arenas and strict maps form a subcategory  $\mathbf{Gam}_\perp$  of **Gam**. As such, our characterisation of the isomorphisms in **Gam** will apply just as well on  $\mathbf{Gam}_\perp$ .

**Lemma 5.1.** *If  $A$  and  $B$  are isomorphic in  $\mathbf{Fam}_f(\mathbf{Gam})_T / \cong$ , then  $TA$  and  $TB$  are isomorphic in  $\mathbf{Gam} / \cong$ .*

*Proof.* It is well-known that there is a full and faithful functor from  $\mathbf{Fam}_f(\mathbf{Gam})_T$  to  $\mathbf{Gam}_\perp$ , mapping  $A$  to  $TA$  and  $f : A \rightarrow TB$  to  $f^* : TA \rightarrow TB$  (assimilating the singleton family  $TA$  with the arena it contains). This functor preserves and reflects  $\cong$ , so isomorphisms in  $\mathbf{Fam}_f(\mathbf{Gam})_T / \cong$  correspond to isomorphisms in  $\mathbf{Gam}_\perp / \cong$ . They are then transferred to  $\mathbf{Gam} / \cong$  since it contains  $\mathbf{Gam}_\perp / \cong$  as a subcategory.  $\square$

Because of the presence of the empty type, isomorphisms in  $\mathbf{Gam}$  do not exactly correspond to isomorphisms in  $\mathbf{Gam}_\perp$ : unanswerable moves (as in  $\llbracket 1 \rightarrow 0 \rrbracket$ ) do not appear in complete plays, so  $\sigma \cong \text{id}_A$  can do anything as soon as one of those has been played. If  $A$  is an arena such that all questions in  $A$  are answerable (*i.e.* for all  $q \in M_A$  such that  $\lambda^{Q_A}(q) = Q$ , there is  $a \in M_A$  such that  $q \vdash_A a$  and  $\lambda^{Q_A}(a) = A$ ), we say that  $A$  is **complete**. If  $A$  is any arena,  $\text{trim}(A)$  is the **trimmed** version of  $A$ , where we have removed the unanswerable moves along with all the moves hereditarily justified by them. Note that for all arena  $A$ ,  $\text{trim}(A)$  is always complete. This operation can also be applied to strategies by setting  $\text{trim}(\sigma)$  as the set of plays in  $\sigma$  which do not contain unanswerable moves.

We handle the mismatch between isomorphisms in  $\mathbf{Gam}$  and  $\mathbf{Gam} / \cong$  as follows:

**Lemma 5.2.** *For any arenas  $A$  and  $B$ ,  $\sigma : A \rightarrow B$  and  $\tau : B \rightarrow A$  form a  $\mathbf{Gam} / \cong$ -isomorphism iff  $\text{trim}(\sigma) : \text{trim}(A) \rightarrow \text{trim}(B)$  and  $\text{trim}(\tau) : \text{trim}(B) \rightarrow \text{trim}(A)$  form a **Gam**-isomorphism.*

*Proof.* Let us first note that if  $\sigma : A \rightarrow B$  and  $\tau : B \rightarrow C$  and  $s \in \sigma; \tau$  is complete, then the witness  $u \in \sigma \parallel \tau$  must be complete as well, otherwise that would break well-bracketing. As a consequence,  $u$  contains no unanswerable move. Hence if  $\sigma$  and  $\tau$  form a  $\mathbf{Gam} / \cong$ -isomorphism, we still have  $\text{trim}(\sigma); \text{trim}(\tau) \cong \text{id}_{\text{trim}(A)}$  and  $\text{trim}(\tau); \text{trim}(\sigma) \cong \text{id}_{\text{trim}(B)}$ , since no unanswerable moves can arise in an interaction between  $\sigma$  and  $\tau$  giving rise to a complete play. We turn now to the proof of the equivalence.

$\Rightarrow$ . Take  $s \in \text{id}_{\text{trim}(A)}$ . It is straightforward to see that  $s$  can be completed, *i.e.* there is  $s' \in \text{id}_{\text{trim}(A)}$  such that  $s \sqsubseteq s'$  and  $s'$  is complete (Opponent only plays answers, he always can because  $\text{trim}(A)$  is complete, the number of unanswered questions decreases strictly). Therefore,  $s' \in \text{trim}(\sigma); \text{trim}(\tau)$ , hence  $s \in \text{trim}(\sigma); \text{trim}(\tau)$  as well, so  $\text{id}_{\text{trim}(A)} \subseteq \text{trim}(\sigma); \text{trim}(\tau)$ . But  $\text{id}_{\text{trim}(A)}$  is total and both strategies are deterministic, therefore this inclusion must be an equality. The same reasoning show that  $\text{trim}(\tau); \text{trim}(\sigma) = \text{id}_{\text{trim}(B)}$  as well, so  $\text{trim}(\sigma)$  and  $\text{trim}(\tau)$  form a **Gam**-isomorphism.

$\Leftarrow$ . If  $\text{trim}(\sigma)$  and  $\text{trim}(\tau)$  form a **Gam**-isomorphism, take a complete  $s \in \sigma; \tau$ . As we have proved above, the witness  $u$  for  $s$  does not contain any unanswerable move, hence  $s \in \text{trim}(\sigma); \text{trim}(\tau) = \text{id}_{\text{trim}(A)} \subseteq \text{id}_A$ . Conversely if  $s \in \text{id}_A$  is complete, then necessarily  $s \in \text{id}_{\text{trim}(A)}$  as well. Thus,  $s \in \text{trim}(\sigma); \text{trim}(\tau)$ . But we have seen above that by necessity the witness  $u \in \text{trim}(\sigma) \parallel \text{trim}(\tau)$  is complete as well and as such cannot contain any unanswerable move, so  $s \in \sigma; \tau$  and  $\sigma; \tau \cong \text{id}_A$ .  $\square$

The results above allow to prove that isomorphisms in  $\mathcal{L}_+$  yield **Gam**-isomorphisms, hence **Path**-isomorphism by an application of Theorem 4.13. It remains to show that types that give rise to **Path**-isomorphic arenas are characterized by the equational theory  $\mathcal{E}$ . For this purpose, it will be convenient to start by putting types in *canonical form*, as described below.

**Lemma 5.3** (Canonical form). *Any type of  $\mathcal{L}_+$  has a representative (up to  $\mathcal{E}$ ) generated by  $T$  in, with  $|I| \geq 2$ .*

$$\begin{aligned} T &::= 0 \mid 1 \mid S \mid P \mid A \\ S &::= \Sigma_{i \in I} L \\ P &::= \Pi_{i \in I} A \\ A &::= L \Rightarrow R \\ L &::= A \mid P \mid 1 \\ R &::= A \mid P \mid S \mid 1 \end{aligned}$$

*Proof.* First eliminate all occurrences of **var** using the last equation of  $\mathcal{E}$ . We make the rest of  $\mathcal{E}$  into a rewriting system by directing the equations from left to right, removing those for commutativity, adding an expansion  $(A + B) \times C \rightsquigarrow A \times C + B \times C$ , and right cancellation of units. It is then straightforward to prove that the following measure strictly decreases with each reduction:  $|0| = |1| = 1$ ,  $|A + B| = |A| + 2|B|$ ,  $|A \times B| = (|A| + 1)|B|$  and  $|A \rightarrow B| = (|B| + 1)^{|A|}$ . It is then a simple induction to find a derivation tree from  $T$  for types that are normal forms for this reduction.  $\square$

**Lemma 5.4.** *Let us extend  $\text{trim}$  to families by setting  $\text{trim}((A_i)_{i \in I}) = (\text{trim}(A_i))_{i \in I}$ . For any type  $B$  in canonical form, we have  $\text{trim}(\llbracket B \rrbracket) = \llbracket B \rrbracket$ . Moreover, we have the following equivalences:*

- (1)  $B = 0$  iff  $\llbracket B \rrbracket = 0$ ,
- (2)  $B = 1$  iff  $\llbracket B \rrbracket = 1$ ,
- (3)  $B$  is generated by  $S$  iff  $\llbracket B \rrbracket$  has at least two members,
- (4)  $B$  is generated by  $P$  iff  $\llbracket B \rrbracket = \{B'\}$  where  $B'$  has at least two initial moves,
- (5)  $B$  is generated by  $A$  iff  $\llbracket B \rrbracket = \{B'\}$  where  $B'$  has exactly one initial move.

*Proof.* Straightforward.  $\square$

**Proposition 5.5.** *If  $\text{trim}(T\llbracket A \rrbracket)$  and  $\text{trim}(T\llbracket B \rrbracket)$  are **Path**-isomorphic, then  $A \simeq_{\mathcal{E}} B$ .*

*Proof.* First, note that  $\text{trim}(T\llbracket A \rrbracket) = 1$  if  $\llbracket A \rrbracket$  is the empty family and  $T(\text{trim}(A_i))$  otherwise. We reason by simultaneous induction on  $A$  and  $B$ , that we both suppose in canonical form. By Lemma 5.4 and the remark above, this means that we get rid of  $\text{trim}$  and suppose  $T\llbracket A \rrbracket$  and  $T\llbracket B \rrbracket$  to be **Path**-isomorphic. Clearly,  $\llbracket A \rrbracket$  and  $\llbracket B \rrbracket$  must be in the same case of Lemma 5.4 (otherwise it is easily checked that they cannot be isomorphic). If it is case (1) (resp. (2)), then both  $A$  and  $B$  have 0 (resp. 1) as canonical form and  $A \simeq_{\mathcal{E}} B$ .

If it is case (3), then  $A = \Sigma_{i \in I} A_i$  and  $B = \Sigma_{j \in J} B_j$ , with  $\llbracket A \rrbracket = (\llbracket A_i \rrbracket)_{i \in I}$  and  $\llbracket B \rrbracket = (\llbracket B_j \rrbracket)_{j \in J}$ . The **Path**-isomorphism between  $T\llbracket A \rrbracket$  and  $T\llbracket B \rrbracket$  yields a bijection  $f : I \rightarrow J$  and for all  $i \in I$  a **Path**-isomorphism  $\phi_i : \llbracket A_i \rrbracket \rightarrow \llbracket B_{f(i)} \rrbracket$ . By induction hypothesis, this means that for all  $i \in I$  we have  $A_i \simeq_{\mathcal{E}} B_{f(i)}$ . By repeated uses of commutativity and associativity of  $+$ , we conclude that  $A \simeq_{\mathcal{E}} B$ .

If it is case (4), then  $A = \Pi_{i \in I} A_i$  and  $B = \Pi_{j \in J} B_j$ . Then  $\llbracket A \rrbracket = \{\Pi_{i \in I} A_i\}$  and  $\llbracket B \rrbracket = \{\Pi_{j \in J} B_j\}$ . Then, the **Path**-isomorphism between  $T\llbracket A \rrbracket$  and  $T\llbracket B \rrbracket$  yields a **Path**-isomorphism between  $\Pi_{i \in I} A_i$  and  $\Pi_{j \in J} B_j$ . In turn, this yields a bijection  $f : I \rightarrow J$ , and for each  $i \in I$  a **Path**-isomorphism between  $A_i$  and  $B_{f(i)}$ . By induction hypothesis, this means that for all  $i \in I$  we have  $A_i \simeq_{\mathcal{E}} B_{f(i)}$ , therefore  $A \simeq_{\mathcal{E}} B$  by repeated uses of associativity and commutativity for  $\times$ .



If it is case (5), then  $A = A_1 \rightarrow A_2$  and  $B = B_1 \rightarrow B_2$ . Both  $A_1$  and  $B_1$  are generated by  $L$ , so they must consist of singleton families  $\{A'_1\}$  and  $\{B'_1\}$ . Then,  $\llbracket A \rrbracket = \{A'_1 \Rightarrow T\llbracket A_2 \rrbracket\}$  and  $\llbracket B \rrbracket = \{B'_1 \Rightarrow T\llbracket B_2 \rrbracket\}$ , and the **Path**-isomorphism between  $T\llbracket A \rrbracket$  and  $T\llbracket B \rrbracket$  yields a **Path**-isomorphism  $\phi$  between  $A'_1 \Rightarrow T\llbracket A_2 \rrbracket$  and  $B'_1 \Rightarrow T\llbracket B_2 \rrbracket$ . Since  $\phi$  preserves Q/A labelling, it decomposes into **Path**-isomorphisms  $\phi_1 : A'_1 \rightarrow B'_1$  and  $\phi_2 : T\llbracket A_2 \rrbracket \rightarrow T\llbracket B_2 \rrbracket$ . By induction hypothesis this implies that  $A_1 \simeq_{\mathcal{E}} B_1$  and  $A_2 \simeq_{\mathcal{E}} B_2$ , thus  $A \simeq_{\mathcal{E}} B$ .  $\square$

Putting all of these together:

**Theorem 5.6.** *For any types  $A, B$  of  $\mathcal{L}_+$ , we have the following equivalence:*

$$A \simeq_{\mathcal{L}_+} B \Leftrightarrow A \simeq_{\mathcal{E}} B$$

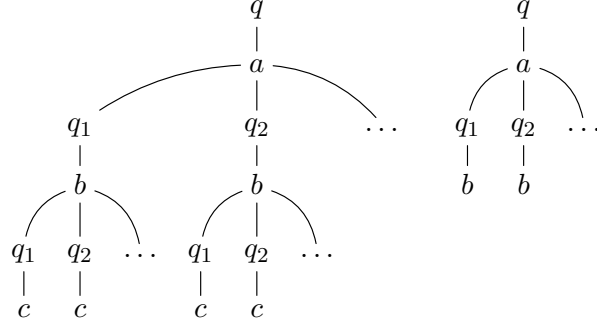
*Proof.* Suppose we have a (syntactic) isomorphism  $x : A \vdash M : B$  and  $y : B \vdash N : A$ . It then easy to check that  $\llbracket N \circ M \rrbracket = \llbracket M \rrbracket; \llbracket N \rrbracket$ , when the former composition is syntactic composition and the latter composition in  $\text{Fam}_f(\mathbf{Gam})_T$ . Likewise, we have  $\llbracket x : A \vdash x : A \rrbracket = \text{id}_{\llbracket A \rrbracket}$  (identity in  $\text{Fam}_f(\mathbf{Gam})_T$ ). By full abstraction, we have  $\llbracket M \rrbracket; \llbracket N \rrbracket \cong \text{id}_{\llbracket A \rrbracket}$  and  $\llbracket N \rrbracket; \llbracket M \rrbracket \cong \text{id}_{\llbracket B \rrbracket}$ , so we have a  $\text{Fam}_f(\mathbf{Gam})_T / \cong$ -isomorphism between  $\llbracket A \rrbracket$  and  $\llbracket B \rrbracket$ . By Lemma 5.1, this means that  $T\llbracket A \rrbracket$  and  $T\llbracket B \rrbracket$  are **Gam**/ $\cong$ -isomorphic. By Lemma 5.2,  $\text{trim}(T\llbracket A \rrbracket)$  and  $\text{trim}(T\llbracket B \rrbracket)$  are **Gam**-isomorphic. By Theorem 4.13, they are **Path**-isomorphic. By Proposition 5.5, this implies that  $A \simeq_{\mathcal{E}} B$ .

Conversely, it is straightforward to check that all equations in  $\mathcal{E}$  between  $A$  and  $B$  give rise to **Path**-isomorphisms between  $\text{trim}(T\llbracket A \rrbracket)$  and  $\text{trim}(T\llbracket B \rrbracket)$ . By Laurent's theorem (the isomorphism of groupoids  $H' : \mathbf{Inn}_i \rightarrow \mathbf{Path}_i$ , see Section 4.3), there is an innocent isomorphism  $\sigma : \text{trim}(T\llbracket A \rrbracket) \rightarrow \text{trim}(T\llbracket B \rrbracket)$ ,  $\tau : \text{trim}(T\llbracket B \rrbracket) \rightarrow \text{trim}(T\llbracket A \rrbracket)$ , note that  $\sigma$  and  $\tau$  have finite view functions. We also have  $\sigma : T\llbracket A \rrbracket \rightarrow T\llbracket B \rrbracket$  and  $\tau : T\llbracket B \rrbracket \rightarrow T\llbracket A \rrbracket$ , although they might not form an isomorphism anymore. However, they do form a **Gam**/ $\cong$ -isomorphism by Lemma 5.2. By construction they are strict, so they come from morphisms  $\sigma' : \llbracket A \rrbracket \rightarrow T\llbracket B \rrbracket$  and  $\tau' : \llbracket B \rrbracket \rightarrow T\llbracket A \rrbracket$  forming an isomorphism in  $\text{Fam}_f(\mathbf{Gam})_T / \cong$ . By innocent definability, there are  $x : A \vdash M : B$  and  $y : B \vdash N : A$  such that  $\llbracket M \rrbracket = \sigma'$  and  $\llbracket N \rrbracket = \tau'$ . By full abstraction,  $M$  and  $N$  must form a syntactic isomorphism of types.  $\square$

**5.2. Isomorphisms in the presence of nat.** Consider the programming language  $\mathcal{L}$  from [1], obtained from  $\mathcal{L}_+$  by replacing sums by **bool** and **nat**, along with the associated combinators. As proved in [1], this language has a fully abstract interpretation in  $\text{Fam}(\mathbf{Gam}_{\infty})_T$ , where  $\mathbf{Gam}_{\infty}$  is the category of not necessarily finitely branching arenas, and single-threaded strategies.

As suggested by the importance of counting in the proof, the presence of **nat** makes it possible to build new isomorphisms by playing Hilbert's hotel. Of course there are obvious new isomorphisms, such as  $\mathbf{nat} \simeq \mathbf{nat} + \mathbf{nat}$  or  $\mathbf{nat} \simeq \mathbf{nat} \times \mathbf{nat}$ , which are realizable by purely functional terms. What is less obvious is that in the presence of higher-order state, one can define new isomorphisms which did not exist in the purely functional fragment of  $\mathcal{L}$ . In this section, we will detail as much as possible one of those new isomorphisms, then mention a few others.

Our main example will be an isomorphism between  $\mathbf{nat} \rightarrow \mathbf{nat} \rightarrow 1$  and  $\mathbf{nat} \rightarrow 1$ . Although this seems to follow from  $\mathbf{nat} \times \mathbf{nat} \simeq \mathbf{nat}$ , this is not the case since curryfication is in general not a valid isomorphism in a call-by-value language. As a consequence of Laurent's theorem, no purely functional isomorphism can exist between these two types because their corresponding arenas are not tree-isomorphic.

Figure 10: Non-trivially isomorphic arenas in  $\mathbf{Gam}_\infty$ 

**Proposition 5.7.** *There is an isomorphism in  $\mathbf{Fam}(\mathbf{Gam}_\infty)_T$  between  $\llbracket \mathbf{nat} \rightarrow \mathbf{nat} \rightarrow 1 \rrbracket$  and  $\llbracket \mathbf{nat} \rightarrow 1 \rrbracket$ .*

*Proof.* By definition of the interpretation of types, this boils down to an isomorphism in  $\mathbf{Gam}_\infty$  between the two arenas  $T\llbracket \mathbf{nat} \rightarrow \mathbf{nat} \rightarrow 1 \rrbracket$  and  $T\llbracket \mathbf{nat} \rightarrow 1 \rrbracket$  represented in Figure 10. Informally, the left-to-right isomorphism can be described as follows:

As long as no  $b$  has been played, it behaves as the identity. The first time a  $b$  is played, Player copies it on the right side. One can then check that the play has  $\mathbb{N} + \mathbb{N}$  possible extensions on the left hand side, whereas it only has  $\mathbb{N}$  extensions on the right hand side. Therefore, Player has to fix a bijection  $\phi : \mathbb{N} + \mathbb{N} \rightarrow \mathbb{N}$  and play accordingly. In general if  $n$  occurrences of  $b$  have been played, there are  $n\mathbb{N}$   $q_i$ s available on the left hand side and still  $\mathbb{N}$  on the right hand side, therefore Player has to follow a bijection  $\phi_n : n\mathbb{N} \rightarrow \mathbb{N}$ .

Thus there is in fact an infinity of different isomorphisms between  $T\llbracket \mathbf{nat} \rightarrow \mathbf{nat} \rightarrow 1 \rrbracket$  and  $T\llbracket \mathbf{nat} \rightarrow 1 \rrbracket$ , one for each family  $(\phi_n)_{n \in \mathbb{N}}$  of bijections between  $n\mathbb{N}$  and  $\mathbb{N}$ .  $\square$

We note that the strategy from  $\mathbf{nat} \rightarrow 1$  to  $\mathbf{nat} \rightarrow \mathbf{nat} \rightarrow 1$  is visible, so this also gives an example of a morphism in  $\mathbf{Vis}$  which is not invertible in  $\mathbf{Vis}$  but becomes invertible in  $\mathbf{Gam}$ . These strategies are not compact so the definability theorem does not apply, however we can nonetheless manually extract corresponding programs from them. We display them in Figure 11, where we suppose that a family of bijections  $\phi_n : n\mathbb{N} \rightarrow \mathbb{N}$  has already been defined. Unfortunately, these terms are too complex to hope for a reasonably-sized direct proof that their interpretations give the strategies described above or even that they form an isomorphism. This kind of difficulty emphasizes the need for new algebraic methods to manipulate and prove properties of imperative higher-order programs.

It seems difficult to characterize exactly the new isomorphisms that natural numbers allow to define. One can prove that the types  $(\mathbf{nat} \rightarrow 1) \rightarrow (\mathbf{nat} \rightarrow 1) \rightarrow 1$  and  $(\mathbf{nat} \rightarrow 1) \rightarrow (1 \rightarrow 1) \rightarrow 1$  are isomorphic, showing that isomorphisms are non-local. Even worse, replacing any occurrence of  $1$  by  $\mathbf{bool}$  in the types above yields non-isomorphic types. Likewise,  $\mathbf{nat} \rightarrow \mathbf{nat} \rightarrow \mathbf{bool}$  and  $\mathbf{nat} \rightarrow \mathbf{bool}$  are not isomorphic.

It is also interesting to note that composing  $\mathbf{nat} \rightarrow \mathbf{nat} \rightarrow 1 \simeq \mathbf{nat} \rightarrow 1$  with  $\mathbf{nat} \times \mathbf{nat} \simeq \mathbf{nat}$  provides an isomorphism  $\mathbf{nat} \rightarrow \mathbf{nat} \rightarrow 1 \simeq \mathbf{nat} \times \mathbf{nat} \rightarrow 1$ , even though curryfication is not a valid isomorphism in general. However one should keep in mind that the terms realizing this isomorphism have nothing in common with curryfication, as they have to use higher-order references in a non-trivial way. In particular, it seems unlikely that they can

<pre> f : nat → nat → 1 ⊢ new count := 0, func := ⊥ in λn. let (p, q) = φ<sub>!count+1</sub><sup>-1</sup>(n) in   if p = 0 then     let x = f q in     count := !count + 1;     let c = !count in     func := (let g = !func in       (λn. if n = c then x         else g n))   else !func p q </pre>	<pre> f : nat → 1 ⊢ new count := 0 in λn. f (φ<sub>!count+1</sub>(0, n)); count := !count + 1; let c = !count in λp. f (φ<sub>!count+1</sub>(c, p)) </pre>
---	--

Figure 11: Type isomorphism in  $\mathcal{L}$  between  $\text{nat} \rightarrow \text{nat} \rightarrow 1$  and  $\text{nat} \rightarrow 1$ .

be used for modularity purposes, putting some limits to the idea that isomorphisms of types always provide the good notion of equivalence on which programmers should rely.

## 6. CONCLUSION

We solved Laurent’s conjecture and characterized the isomorphisms of types in  $\mathcal{L}_+$ . Surprisingly, we realized that the combination of higher-order references, natural numbers and call-by-value allowed to define new non-trivial type isomorphisms. Note however that if well-bracketing is satisfied, the proof of our core game-theoretic theorem adapts directly to arenas where all moves only enable a finite number of questions, but an arbitrary numbers of answers. As a consequence, there are no non-trivial isomorphisms (*i.e.* not already present in the  $\lambda$ -calculus) in the call-by-name variant of  $\mathcal{L}$ , although we can define one using `call/cc`.

Note that despite the seemingly restricted power of  $\mathcal{L}_+$ , our theorem does apply to all real-life programming languages that have a bounded type of integer, such as `bool`<sup>32</sup> or `bool`<sup>64</sup>: in this setting, no non-trivial isomorphism can exist. However unbounded natural numbers can be defined using recursive types, so the isomorphism above can be implemented in a call-by-value programming language with recursive types and general references, such as OCAML.

*Acknowledgments.* We would like to thank Guy McCusker and Nikos Tzevelekos for stimulating discussions about the new non-trivial isomorphisms.

## REFERENCES

- [1] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *13th IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1998.
- [2] S. Abramsky and G. McCusker. Linearity, Sharing and State: a Fully Abstract Game Semantics for Idealized Algol with active expressions, 1997.
- [3] S. Abramsky and G. McCusker. Call-by-value games. In Mogens Nielsen and Wolfgang Thomas, editors, *6th Annual Conference of the European Association for Computer Science Logic*, volume 1414 of *Lecture Notes in Computer Science*. Springer, 1998.
- [4] F. Atanassow and J. Jeuring. Inferring type isomorphisms generically. In Dexter Kozen and Carron Shankland, editors, *MPC*, volume 3125 of *Lecture Notes in Computer Science*, pages 32–53. Springer, 2004.

- [5] G. Barthe and O. Pons. Type isomorphisms and proof reuse in dependent type theory. In Furio Honsell and Marino Miculan, editors, *FoSSaCS*, volume 2030 of *Lecture Notes in Computer Science*, pages 57–71. Springer, 2001.
- [6] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20:265–321, 1982.
- [7] K.B. Bruce, R. Di Cosmo, and G. Longo. Provable isomorphisms of types. *Mathematical Structures in Computer Science*, 2(2):231–247, 1992.
- [8] K.B. Bruce and G. Longo. Provable isomorphisms and domain equations in models of typed languages (preliminary version). In *STOC*, pages 263–272. ACM, 1985.
- [9] M. Dezani-Ciancaglini. Characterization of normal forms possessing inverse in the  $\lambda$ - $\beta$ - $\eta$ -calculus. *Theoretical Computer Science*, 2(3):323–337, 1976.
- [10] R. Di Cosmo. Invertibility of terms and valid isomorphisms, a proof theoretic study on second order lambda-calculus with surjective pairing and terminal object. Technical report, Technical Report TR 10-91, LIENS Ecole Normale Supérieure, Paris, 1991.
- [11] R. Di Cosmo. A short survey of isomorphisms of types. *Mathematical Structures in Computer Science*, 15(5):825–838, 2005.
- [12] M. Fiore, R. Di Cosmo, and V. Balat. Remarks on isomorphisms in typed lambda calculi with empty and sum types. *Ann. Pure Appl. Logic*, 141(1-2):35–50, 2006.
- [13] K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation. *Theor. Comput. Sci.*, 221(1-2):393–456, 1999.
- [14] J.M.E. Hyland and C.H.L. Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163(2):285–408, December 2000.
- [15] J.M.E. Hyland and A. Schalk. Games on graphs and sequentially realizable functionals. In *Logic in Computer Science 02*, pages 257–264, Kopenhagen, July 2002. IEEE Computer Society Press.
- [16] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(447-468):184, 1996.
- [17] J. Laird. Full abstraction for functional languages with control. In *12th IEEE Symposium on Logic in Computer Science*, pages 58–67, 1997.
- [18] J. Laird. A game semantics of the asynchronous  $\pi$ -calculus. In *CONCUR*, pages 51–65, 2005.
- [19] O. Laurent. Polarized games. *Annals of Pure and Applied Logic*, 130(1-3):79–123, 2004.
- [20] O. Laurent. Classical isomorphisms of types. *Mathematical Structures in Computer Science*, 15(5):969–1004, 2005.
- [21] G. McCusker. Games and full abstraction for a functional metalanguage with recursive types. PhD thesis, Imperial College, University of London, 1996. Published in Springer-Verlag’s Distinguished Dissertations in Computer Science series, 1998.
- [22] P.-A. Mellies and N. Tabareau. An algebraic account of references in game semantics. *Electr. Notes Theor. Comput. Sci.*, 249:377–405, 2009.
- [23] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.
- [24] A. Murawski and N. Tzevelekos. Full Abstraction for Reduced ML. In Luca de Alfaro, editor, *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 32–47. Springer, 2009.
- [25] A. Murawski and N. Tzevelekos. Game semantics for good general references. In *LICS*, pages 75–84. IEEE Computer Society, 2011.
- [26] M. Rittri. Using types as search keys in function libraries. *J. Funct. Program.*, 1(1):71–89, 1991.
- [27] M. Rittri. Retrieving library functions by unifying types modulo linear isomorphism. *ITA*, 27(6):523–540, 1993.