Isomorphisms of types in the presence of higher-order references
Pierre Clairambault

To cite this version:

Pierre Clairambault. Isomorphisms of types in the presence of higher-order references. Twenty-Sixth Annual IEEE Symposium on Logic In Computer Science (LICS 2011), Jun 2011, Toronto, Canada. pp.152 - 161, 2011, <10.1109/LICS.2011.32>. <hal-00651816>

HAL Id: hal-00651816
https://hal.archives-ouvertes.fr/hal-00651816
Submitted on 14 Dec 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Isomorphisms of types in the presence of higher-order references

Pierre Clairambault
University of Bath
p.clairambault@bath.ac.uk

Abstract—We investigate the problem of type isomorphisms in a programming language with higher-order references. We first recall the game-theoretic model of higher-order references by 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 a finitary language \( \mathcal{L}_2 \) with higher order references. We show however that Laurent’s conjecture does not hold on infinitely branching arenas, yielding a non-trivial type isomorphism in the extension of \( \mathcal{L}_2 \) with natural numbers.

I. 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 \( D \cong D^2 \)) 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 that the only isomorphisms of types present in simply typed \( \lambda \)-calculus with respect to \( \beta \eta \) equality are those induced by the equation \( A \rightarrow (B \rightarrow C) \cong B \rightarrow (A \rightarrow C) \). Later this was extended to handle such things as products [7], higher order [8], possibly with unit types [10], or sums [11].

The interest in type isomorphisms grew significantly when their practical impact was realized. In [23], 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 unification of types modulo isomorphisms [24]. 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 Youillon for CamlLight.

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: the dynamics of terms are no longer defined by reduction rules but by operational semantics, the natural equality between terms is no longer convertibility but observational equivalence, so new methods are required.

In [18], 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 [17], 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 \( \text{Inn} \) [13] 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 [15], ground type [2] or higher-order references [1] or even concurrency [16] in one single framework. Moreover, Laurent pointed out in [18] that the main part of his result, namely the fact that each \( \text{Inn} \)-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 contribution of this paper is threefold: (1) We give a new and synthetic reformulation of Laurent’s tools to approach
game-theoretically the problem of type isomorphisms, (2) 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 $L_2$ with higher-order references by the theory $E$ presented in Figure 1, (3) We show however a counter-example to the conjecture when dealing with infinitely branching arenas, and the counter-example yields a non-trivial type isomorphism between the types $(\text{n} \to \text{unit}) \to (\text{n} \to \text{unit}) \to \text{unit}$ and $(\text{n} \to \text{unit}) \to (\text{unit} \to \text{unit}) \to \text{unit}$ in the extension of $L_2$ with natural numbers. So Laurent’s conjecture, in the general case, is false.

In Section II we introduce the finitary language $L_2$ strongly inspired by Abramsky, Honda and McCusker’s language $L$ [1], along with its standard game semantics. Then we turn to the problem of isomorphisms of types. In Section III we first give an analysis of isomorphisms in several subcategories of the games model, reproving and extending Laurent’s theorem, then we use it to characterize isomorphisms in $L_2$. We show how this characterization fails in the presence of natural numbers, and we give a non-trivial type isomorphism in $L$.

II. The Language $L_2$ and its Game Semantics

A. Definition of $L_2$

a) Basic definitions: We introduce here a finitary variant $L_2$ of the programming language $L$ with higher-order references modeled by Abramsky, Honda and McCusker in [1]; it only differs from $L$ in the fact that the type of natural numbers has been replaced with a type for booleans, along with all the associated combinators. The terms and types of $L_2$ are given by the following grammars.

$$A ::= \text{unit} | \text{bool} | A \times A | A \to A | \text{var}[A]$$

$$M ::= x | \lambda x . M | M M | \langle M, M \rangle | \text{fst } M \text{ and } M | \text{skip } | \text{true } | \text{false } | \text{if } M M M | \text{new}_A M : M = M | \text{mkvar } M M$$

The typing rules for $\lambda$-calculus, pairs and booleans are standard. The rules for references follow.

$$\Gamma \vdash \text{new}_A : \text{var}[A]$$

$$\Gamma \vdash M : \text{var}[A]$$

$$\Gamma \vdash \text{mkvar } M N : \text{var}[A]$$

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:

$$V ::= \text{skip } | \text{true } | \text{false } | \lambda x . M | \langle V, V \rangle | l | \text{mkvar } V V$$

The operational semantics of $L_2$ are then given as an inductively generated relation $(L, s)M \Downarrow (L', s')V$, where $s$ is a partial map from locations in $L$ to values. The rules for $\lambda$-calculus, products and booleans are standard (they do not affect the store) and we give in Figure 2 the rules for references. Note that as usual, some store annotations are omitted to aid readability; the rules can be disambiguated as explained in [1]. For a closed term $M$ without free locations, we write $M \Downarrow$ to indicate that $(\emptyset, s)M \Downarrow (L, s)V$ for some $L, s$ and $V$. The observational preorder $M \preceq N$ 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, if $C[M] \Downarrow$ then $C[N] \Downarrow$. The corresponding equivalence relation is denoted by $\equiv$.

b) 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 unit, sequential composition $C_1; C_2$ is given by:

$$(\lambda d : \text{unit}. \; C_2) \; C_1$$

This works only because the evaluation of $L_2$ is call-by-value. Likewise, a variable declaration new $x : A$ in $N$ (where $M$ has type $A$) can be obtained by

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

and its initialized variant new $x = M$ in $N$ as expected. As usual with general references one can define a fixed point combinator $Y$ by

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

This can be easily applied to implement a while loop. We can also use it to build an inhabitant $\bot$ to any type $A$.

c) Bad variables and isomorphisms: 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 include bad variable in $L_2$ for two reasons. First, because games models that allow bad variables are notably simpler than those which do not [22], for which it is not clear whether our methods apply. Second, because we expect the problem of isomorphisms without bad variables to be far more subtle than what we consider here, because of the observation by O’Hearn that

$$A \times B \simeq (B \times C) \simeq (A \times C)$$

mentioned in the introduction may seem strange, but is standard in call-by-value [18] due to the restriction of the $\eta$-rule on values.
without bad variables, not only $\text{var}[X]$ is not functorial, but it does not even preserve isomorphisms. However, note that $\text{var}$-free isomorphisms are the same with or without bad variables.

d) Isomorphisms of types: We are now ready to define the notion of isomorphism of types in $\mathcal{L}_2$.

**Definition 1.** If $A$ and $B$ are two types of $\mathcal{L}_2$, we say that $A$ and $B$ are isomorphic, denoted by $A \cong B$, if and only if there are two terms $x : A \vdash M : B$ and $y : B \vdash N : A$ such that:

$\begin{align*}
(x : A \vdash (\lambda y. N)M) & \equiv id_A \\
(y : B \vdash (\lambda x. M)N) & \equiv id_B
\end{align*}$

where $id_A = x : A \vdash x : A$.

**B. The games model**

We now describe the fully abstract games model of $\mathcal{L}_2$. Note that except a few details there is nothing new here, as this is precisely the model described in [1]. We however include the definitions (but no proofs) for the sake of self-completeness.

e) Arenas, plays: Our games have two participants: Player (P) and Opponent (O). Valid plays between $O$ and $P$ are generated by directed graphs called arenas, which are the abstract representation of types. An arena $A$ 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

$\langle O, P \rangle \times \{Q, A\} = \{OQ, OA, PQ, PA\}$

$\lambda_A = \langle \lambda_A^{OP}, \lambda_A^{AQ} \rangle$

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

- $I_A \subseteq M_A$ is a set of initial moves
- $\vdash_A : M_A$ 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 $\lambda_A^{AQ}(b) = A$ then $\lambda_A^{AQ}(a) = Q$.

We require two additional conditions on arenas: they should be complete (for each question $m \in M_A$, there should be an answer $n \in M_A$ such that $m \vdash_A n$) and finitely branching (for all $a \in M_A$, the set $\{m \in M_A \mid a \vdash_A m\}$ is finite). We consider the usual arrow construction $A \Rightarrow B$ on arenas, as well as products $\Pi_{i \in I} A_i$ and lifted sums $\Sigma_{i \in I} A_i$ of finite families of arenas. Their definitions can be found, for example, in [1]. It is obvious that they preserve the fact of being complete and finitely branching.

The 0-ary product (the empty arena) is denoted by $1$, and will be a terminal object in our category.

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.

f) Notations: The relation $\subseteq$ will denote the prefix ordering on justified sequences. By $s \subseteq^P t$, we mean that $s$ is a $P$-ending prefix of $t$. If $s$ is a sequence, then $|s|$ will denote its length. We also define the prefix functions $ip$ and $jp$ by $ip(e) = e$ and $ip(sa) = s$, and $jp(si) = e$ if $i$ is initial, $jp(s_1as_2b) = s_1a$ if $b$ is justified by $a$.

A justified sequence $s$ over $A$ is a legal play if it is:

- Alternating: If $s'ab \subseteq s$, 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 $\mathcal{L}_A^{\prime}$ of well-bracketed but not necessarily alternating plays on $A$, called pre-legal plays.

g) 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 $u_{1,A,B} \in \mathcal{L}_A \Rightarrow B$, $u_{1,B,C} \in \mathcal{L}_B \Rightarrow C$ and $u_{1,A,C} \in \mathcal{L}_A \Rightarrow C$, where $u_{1,A,B}$ 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 $\sigma[\tau] = \{u \in I(A, B, C) \mid u_{1,A,B} \in \sigma \land u_{1,B,C} \in \tau\}$, and the composition of $\sigma$ and $\tau$ is obtained by the hiding operation, i.e. $\sigma; \tau = \{u_{1,A,C} \mid u \in \sigma[\tau]\}$. It is known (e.g. [19]) that composition is associative. It admits copycat strategies as identities: $id_A = \{s \in \mathcal{L}_A \Rightarrow A_2 \mid \forall s' \subseteq^P s, s'|_{A_1} = s'|_{A_2}\}$.

If $s \in \mathcal{L}_A$, the current thread of $s$, denoted $[s]$, 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 \( s a b \in \sigma \), \( b \) points in \([s a]\).

For all \( s a b, t \in \sigma \) such that \( t a \in L_A \) and \([s a] = [t a] \), we have \( t a b \in \sigma \).

It is straightforward to prove that single-threaded strategies are stable under composition and that \( id_A \) is single-threaded. Hence, there is a category \( \text{Gam} \) of arenas and single-threaded strategies. The category \( \text{Gam} \) 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 \( A \times - \Rightarrow A \Rightarrow - \), hence \( \text{Gam} \) is cartesian closed and is a model of simply typed \( \lambda \)-calculus. It also has \textit{weak coproducts}, given by the lifted sum [1].

\( h) \) 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 \textit{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. Their definition requires the notion of \( P \)-view, defined as usual by induction on \( P \)-plays as follows.

\[
\begin{align*}
\tau s^i & = i & \text{if } i \in I_A \\
\tau s a^i & = \tau a^i & \text{if } \lambda_A^{P}(a) = P \\
\tau s_1 a s_2 b & = \tau s_1 a b & \text{if } \lambda_A^{P}(b) = O \text{ and } a \text{ justifies } b
\end{align*}
\]

A strategy \( \sigma : A \) is then said to be \textit{visible} if it always points inside its \( P \)-view, that is, for all \( s a b \in \sigma \) the justifier of \( b \) appears in \( \tau s a^i \). The strategy \( \sigma \) is \textit{innocent} if it is visible, and if its behaviour only depends on the information contained in its \( P \)-view. More formally, whenever \( s a, t \in \sigma \) such that \( t a \in L_A \) and \( \tau s a^i = \tau t a^i \), we must also have \( t a b \in \sigma \).

Both visibility and innocence are stable under composition [13], [2], thus let us denote by \textit{Vis} the category of arenas and visible single-threaded strategies and by \textit{Inn} the category of arenas and innocent strategies. Both categories inherit the cartesian closed structure of \( \text{Gam} \), but strategies in \textit{Inn} are actually nothing but abstract representation of (\( \eta \)-long \( \beta \)-normal) \( \lambda \)-terms and form a fully complete model of simply-typed \( \lambda \)-calculus. Strategies in \textit{Vis} have more freedom, they correspond in fact to programs with first-order store [2].

\( C. \) Modeling \( \text{L}_2 \) in \( \text{Fam}_I(\text{Gam}) \)

\( i) \) Interpretation: The three categories \( \text{Gam} \), \textit{Vis} and \textit{Inn} are categories of \textit{negative games} (in which Opponent always plays first), and these are known to model call-by-name computation whereas \( \text{L}_2 \) is call-by-value. We could have modeled it using positive games, following the lines of [12]. Instead, we follow [1] and model \( \text{L}_2 \) in the free completion \( \text{Fam}(\text{Gam}) \) of \( \text{Gam} \) with respect to coproducts. This will allow us to first characterize isomorphisms in \( \text{Gam} \) (result which could be applied to a call-by-name language with state) then deduce from it the isomorphisms in \( \text{Fam}(\text{Gam}) \). In fact we will consider the completion \( \text{Fam}_I(\text{Gam}) \) of \( \text{Gam} \) with respect to finite coproducts, since \( \text{L}_2 \) has only finite types.

The objects of \( \text{Fam}_I(\text{Gam}) \) are finite families \( \{A_i \mid i \in I\} \) of arenas. A map from \( \{A_i \mid i \in I\} \) to \( \{B_j \mid j \in J\} \) is the data of a function \( f : I \to J \) together with a family of strategies \( \{\sigma_i : A_i \to B_{f(i)} \mid i \in I\} \). As shown in [3], the cartesian closed structure of \( \text{Gam} \) extends to \( \text{Fam}_I(\text{Gam}) \). Moreover, the weak coproducts in \( \text{Gam} \) give rise to a \textit{strong monad} \( T \) on \( \text{Fam}_I(\text{Gam}) \). Given families \( A = \{A_i \mid i \in I\} \) and \( B = \{B_j \mid j \in J\} \), we define

\[
\begin{align*}
A \times B & = \{A_i \times B_j \mid (i, j) \in I \times J\} \\
A \Rightarrow B & = \{\Pi_{i \in I}(A_i \Rightarrow B_{f(i)}) \mid f : I \to J\} \\
T A & = \{\Sigma_{i \in I} A_i\}
\end{align*}
\]

The singleton family \( \{1\} \) is the terminal object of \( \text{Fam}_I(\text{Gam}) \). By abuse of notation, we will still denote it by \( 1 \). The other components of the cartesian closed structure of \( \text{Fam}_I(\text{Gam}) \) and of the strong monad structure of \( T \) follow naturally from these definitions. We skip the details, as all of this is already covered in [1]. It is known that given a cartesian closed category with a strong monad, one can interpret call-by-value languages in the Kleisli category of the monad [21], and the interpretation of \( \text{L}_2 \) in \( \text{Fam}_I(\text{Gam}) \) follows these lines.

We interpret \textit{unit} and \textit{bool} as the families with respectively one and two elements whose components are all empty arenas. Of course we also need to give an interpretation for \textit{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 \( \text{var} \( A \) \mid \{A\} = (\{A\} \times T\{A\}) \times T\{A\} \). The interpretation relies on the definition of a morphism \( 1 \to \text{var} \( A \) \), that is, if \( \{A\} = \{A_i \mid i \in I\} \), a strategy \( \text{cell} : (\Pi_{i \in I} A_i \Rightarrow 1_i) \times \Sigma_{i \in I} A_i \) where \( A_i = T\{A\} \) 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 [20]. As proved in [1], this gives a sound interpretation of \( \text{L}_2 \) in \( \text{Fam}_I(\text{Gam}) \).

\( j) \) Complete plays and full abstraction: A fully abstract model of \( \text{L}_2 \) is obtained by quotienting \( \text{Fam}_I(\text{Gam})_T \) by the usual observational preorder. However, as is often the case with fully abstract game semantics of languages with store, it is \textit{effectively presentable}: the observational preorder can be characterized directly. Say a play \( s \in L_A \) is \textit{complete} if it has as many questions and answers, \textit{i.e.} all questions have been answered. If \( \sigma \) is a strategy on an arena \( A \), then let us denote by \( \text{comp}(\sigma) \) the set of complete plays in \( \sigma \). Take \( \sigma, \tau : A \rightharpoonup B \) two morphisms in \( \text{Fam}_I(\text{Gam})_T \), with \( A = \{A_i \mid i \in I\} \) and \( B = \{B_j \mid j \in J\} \). Then, \( \sigma \) and \( \tau \) consist of families \( \{\sigma_i : A_i \rightharpoonup \Sigma_{j \in J} B_j \mid i \in I\} \) and \( \{\tau_i : A_i \rightharpoonup \Sigma_{j \in J} B_j \mid i \in I\} \). We then say that \( \sigma \simeq \tau \) if and only if for all \( i \in I \), \( \text{comp}(\sigma_i) \subseteq \text{comp}(\tau_i) \).

Take now two terms \( M \) and \( N \) of type \( A \), and suppose \( \{A\} = \{A_i \mid i \in I\} \). Then \( [M] \) and \( [N] \) are morphisms from \( 1 \) to \( T\{A\} \) in \( \text{Fam}_I(\text{Gam}) \), \textit{i.e.} strategies on \( \Sigma_{i \in I} A_i \). Given the full abstraction result of [1], it is then straightforward to prove the following equivalence:

\[
M \leq N \iff [M] \preceq [N]
\]

This concrete representation of the observational preorder will be central to our characterization of isomorphisms in \( \text{L}_2 \).
III. ISOMORPHISMS IN Gam

We are now going to extend Laurent’s tools [18] to characterize isomorphisms of types for $L_2$. We will first recall Laurent’s work in the visible and innocent cases, then extend it to characterize isomorphisms in $\text{Gam}$. From there, we will switch to call-by-value and study the isomorphisms in $\text{Gam}_1(\text{Gam})$, and in its Kleisli category over $T$.

A. Isomorphisms and zig-zag strategies

We start by giving the definition of (a subtle adaptation of) Laurent’s zig-zag plays.

**Definition 2.** Let $s \in L_{A \Rightarrow B}$ 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 $s_{1A}$ and $s_{1B}$ have the same points.

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 $\text{Vis}$ 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 $\text{Gam}$ are pre-zig-zag. Here, being mainly interested in $\text{Gam}$, we make this explicit. We need first the following lemma.

**Lemma 1** (Dual pre-zig-zag play). Let $s \in L_{A \Rightarrow B}$ be a pre-zig-zag play, then there exists an unique pre-zig-zag $\pi \in L_{B \Rightarrow A}$ such that $s_{1A} \pi s_{1B} = s_{1B} \pi s_{1A}$. 

Proof: We define $\pi$ by induction on $s$; $\pi = \epsilon$, and $s\pi b = \pi ba$. 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 $\pi$. There is no other possible $\pi$, 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 $\pi$.

**Lemma 2.** If $\sigma : A \Rightarrow B$, $\tau : B \Rightarrow A$ form an isomorphism in $\text{Gam}$, then they are pre-zig-zag and for all $s, s, s \in \sigma \Leftrightarrow \pi \in \tau$.

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

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 $s_{1B} = \pi_{1B}$, by a straightforward zig-zag argument we can build an interaction $u \in I(A_1, B, A_2)$ such that $u_{1A}, B = s$ and $u_{1B}, A = \pi$, moreover since $\sigma, \tau$ form an isomorphism we must have $u_{1A}, A_2 \in i_d A$. Now, we necessarily have $b \in M_B$, otherwise $u$ could be extended to $uab \in \tau | \pi$ with $uab_{1A}, B = u_{1A}, A, ab$ 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 $\pi \in \tau$, and by the same zigging argument as above we have an unique $u \in I(B_1, A, B_2)$ such that $u_{1B}, , A = \pi$ and $u_{1A}, B_2 = s$. Since $\sigma, \tau$ form an isomorphism we also have $u_{1B}, B_2 \in id_B$. Let us now extend $u$ to $u' = uba_{1B}$ in the unique way such that $u_{1A}, B_2 = sba$ and $u_{1B}, A = \pi$. Note that we are sure that $b_1$ is a move on $B_1$ since $sba_{1B}$ 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 also we have $u_{1A}, B_2 \in id_B$, hence $b_2$ points in $\pi$ as $b_1$ points in $s$. This means that we have $\pi ab \in \tau$, such that $a$ is initial and $b$ points in $\pi$, 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 $\sigma$ is defined. By induction hypothesis $\pi \in \tau$ and the same reasoning as above shows that it extends to $sba \in \tau$. The argument is symmetric, hence $\{ \pi : | s \in \sigma \wedge | s | \leq k + 2 \} = \{ s \in \tau : | s | \leq k + 2 \}$.

For the sake of completeness, let us include Laurent’s argument which proves that isomorphisms in $\text{Vis}$ are zig-zag.

**Lemma 3.** If $\sigma : A \Rightarrow B$, $\tau : B \Rightarrow A$ form an isomorphism in $\text{Vis}$, 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 $s_{1A}$ and $s_{1B}$ 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 $(s_{1A})$, then $b$ points to $(s_{1B})$. Indeed, it cannot point to $(s_{1B})$, with $j > i$ since that would break visibility for $\sigma$. But if it points to $(s_{1B})$, with $j < i$ we use the same reasoning on the dual pre-zig-zag play $sab$ and get a contradiction with the fact that $\tau$ is visible.

Let us denote by $\text{Gam}_1$, $\text{Vis}_1$, and $\text{Inn}_1$, the groupoids of arenas and isomorphisms on the respective categories. In the next sections, we use these facts to give more combinatorial representations of $\text{Gam}_1$, $\text{Vis}_1$, and $\text{Inn}_1$.

B. Notions of game morphisms

Laurent’s isomorphism theorem works by relating isomorphisms in $\text{Gam}$ 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.

**Definition 3.** Let $A$ be an arena. A path on $A$ is a play $s \in L_A$ such that except for the initial move, every move in $s$ points to the previous move. Formally, for all $s \emptyset s \leq s$, $a$ 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 $ip \circ \phi = \phi \circ ip$ and which preserves $Q/A$ labeling: for all

$2$Note that in [18] arenas are forests, which is not the case here.
so ∈ P with φ(sa) = φ(s)b, we have \( \lambda_{QA}^A(a) = \lambda_{QB}^B(b) \). There is a category \( \text{Path} \) of arenas and path morphisms.

This category \( \text{Path} \) 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 \( T_A \). Likewise, let us call a pre-legal play with one initial move a pre-legal thread and let us denote these by \( T_A' \).

**Definition 4.** Let \( A, B \) be arenas, and let \( \phi : T_A' \to T_B \). We say that \( \phi \) is a sequential morphism from \( A \) to \( B \) if \( ip \circ \phi = \phi(s)b \) and if it preserves \( QA \) labeling, i.e., for all \( \phi(sa) = \phi(s)b \) we have \( \lambda_{QA}^A(a) = \lambda_{QB}^B(b) \). We say that it is a justified morphism if, additionally, \( \phi \circ ip = \phi \circ jp \). There are two categories \( \text{Seq} \) of arenas and sequential and \( \text{Jus} \) of arenas and justified morphisms.

As above, we will denote by \( \text{Seq}_i, \text{Jus}_i \) and \( \text{Path}_i \) the groupoids of invertible maps in \( \text{Seq}_i, \text{Jus}_i \) and \( \text{Path} \). These groupoids will soon appear to be identical to \( \text{Gam}_i, \text{Vis}_i \) and \( \text{Inn}_i \). To prove this, we need the following lemma.

**Lemma 4.** Let \( s \in T_A' \), and \( A \to B \) an isomorphism in \( \text{Gam} \). There is then an unique play \( s' \in \sigma \) such that \( s'_A = s \).

**Proof:** Remark first that if \( \sigma : A \to B \) and \( \tau : B \to A \) are inverses then they are both total, i.e., for all \( s \in \sigma \) and \( sa \in L_{A \to B} \) there must be \( b \) such that \( s s b \in \sigma \), 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 \( sa \in T_A' \) and we have by induction hypothesis \( s' \in \sigma \) such that \( s'_A = s \). If \( a \) is a \( P \)-move in \( A \) (hence an \( O \)-move in \( A \to B \)), there is an unique \( b \) such that \( s' b b \in \sigma \), and we do have \( s' a b_A = sa \). If \( a \) is an \( O \)-move in \( A \) (hence a \( P \)-move in \( A \to B \)), then let \( \tau : B \to A \) be the inverse of \( \sigma \), since \( s' \in \sigma \) we have \( s' \in \tau \). Being part of an isomorphism \( \tau \) is total, hence there is \( b \) such that \( s' \in \tau \). We deduce from this that \( s' b a \in \sigma \), and we have \( s' b a A = sa \) as needed.

This choice is unique: if there is another play \( t \in \sigma \) such that \( t_A = sa \), 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 \( s' b a b \in \tau \), so \( b = b' \) by determinism of \( \tau \).

**Proposition 1.** If \( C \simeq D \) means that two groupoids \( C \) and \( D \) are isomorphic, then we have:

\[
\begin{align*}
\text{Gam}_i & \simeq \text{Seq}_i \\
\text{Vis}_i & \simeq \text{Jus}_i
\end{align*}
\]

**Proof:** Let us first define a functor \( F : \text{Gam}_i \to \text{Seq}_i \). It is defined as the identity on arenas. Let \( \sigma : A \to B \) be an isomorphism, and let \( s \in T_A' \) then we define \( \phi_\sigma(s) = s'_B \), where \( s' \) is the unique play on \( A \to B \) which existence is ensured by the lemma above. The function \( \phi_\sigma \) commutes with \( ip \) 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.

Reciprocally, suppose \( \phi : A \to B \) is a sequential isomorphism. We mimic the usual definition of the identity by setting \( G(\phi) = \{ s \in L_{A \to B} \mid \forall s' \in \phi(s') \cdot \phi(s'_A) = s'_B \} \) (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 \( \text{Gam}_i \simeq \text{Seq}_i \), which restricts naturally to \( \text{Vis}_i \) and \( \text{Jus}_i \). Indeed if \( \sigma : A \to B \) is a visible isomorphism, it is a zig-zag strategy therefore \( s \in T_A \) and \( \phi_\sigma(s) \) have the same pointers, which means that \( jp \circ \phi_\sigma = \phi_\sigma \circ jp \). Reciprocally if \( \phi_\sigma \) is a justified morphism, all \( s \in \sigma \) must be such that \( s_A \) and \( s_B \) have the same pointers, therefore \( \sigma \), being pre-zig-zag, always points in its \( P \)-view.

**C. 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 \( \text{Vis}_i \) correspond to isomorphisms in \( \text{Jus}_i \), which we are now going to compare with isomorphisms in \( \text{Path}_i \).

**Lemma 5.** There is a full functor \( H : \text{Vis}_i \to \text{Path}_i \).

**Proof:** We have built in the above section a full and faithful functor (actually an isomorphism) \( F : \text{Vis}_i \to \text{Jus}_i \). From a visible isomorphism \( \sigma : A \to B \) we set \( H(\sigma) : \text{PA} \to \text{PB} \). To see why \( H \) is full, suppose we have a path morphism \( \phi : \text{PA} \to \text{PB} \). Then \( \phi \) admits a canonical extension \( \phi^* : T_A' \to T_B \). 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 \( H(\phi^*) = \phi \).

This ensures that arenas \( A \) and \( B \) are isomorphic in \( \text{Vis}_i \) if and only if they are isomorphic in \( \text{Path}_i \), 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 \( \text{H}^* : \text{Inn}_i \to \text{Path}_i \).

1) Faithfulness of \( H \): Note however that \( H \) itself is not faithful, because 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 \( A = q_1 \xrightarrow{\lambda} q_2 \xrightarrow{\epsilon} a \) which is the interpretation of \( \text{bool} \to \text{unit} \to \text{unit} \) in call-by-value and of \( \text{unit} \times \text{unit} \to \text{unit} \) in call-by-name.

2In fact, this construction can be seen as a particular case of Hyland and Schalk’s faithful functor from games to relations [14], where the relation happens to be functional.
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 example play of $i$ is given in Figure 3. Although it is not the identity, $i$ is its own inverse. Its image by $H$ only takes into account the first behaviour or $i$, thus is the same as for $id_A$: the identity path morphism on $A$. From this strategy we can extract the following term $f : B \vdash M : B$ of $\mathcal{L}_2$, where $B = (\text{bool} \rightarrow \text{unit}) \rightarrow \text{unit}$.

\[
\begin{align*}
\text{new } r := \text{true} & \quad \lambda g.(\lambda b. \text{if } r \text{ then } b \text{ else } g) (b \text{ else } g, \text{not } b) \\
\text{new } r := \text{false} & \quad \lambda g.(\lambda b. \text{if } r \text{ then } b \text{ else } g) (b \text{ else } g, \text{not } b)
\end{align*}
\]

Although $M$ is not the identity it is an involution on $B$, i.e. we have $(\lambda f. M)(M x) \cong x$. Such non-trivial involutions cannot be defined using only purely functional behaviour.

We give in Figure 4 a summary of all the groupoids of isomorphisms encountered for the moment, along with their relations. Following it, the question of finding the isomorphisms in $\text{Gam}$ boils down to the definition of an arrow from $Seq_i$ to $Path_i$ in this diagram, which is what we will attempt in the next two subsections.

### D. Non-visible isomorphisms by counting

We have seen above that we can build a full functor $\text{Vis}_i \rightarrow \text{Path}_i$, which allows to characterize isomorphic arenas in $\text{Vis}$. 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 $\text{Gam}$ if and only if they are isomorphic in $\text{Path}$. 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 \textbf{arity} the quantity $ar(a) = |\{m \in M_A \mid a \vdash_A m\}|$. On pre-threads $s \in T_A^*$ we define:

\[
Q(s) = \sum_{i=1}^{\vert s \vert} ar(s_i)
\]

If $s \in T_A^*$, $Q(s)$ is also the number of ways $s$ can be extended to some $sa$ (let us recall here that as a member of $T_A^*$, $s$ 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 $Q(s_{1A}) = Q(s_{1B})$, because $\sigma$ being an isomorphism, it must associate each possible extension of $s_{1A}$ to an unique extension of $s_{1B}$. But this also means that if $sab \in \sigma$ we have $Q(s_{1A}) + ar(a) = Q(s_{1A}a) = Q(s_{1B}b) = Q(s_{1B}) + ar(b)$, 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 $\iota_{a,b}$. 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 form $m$ 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.

\textbf{Lemma 6} (Slicing of isomorphisms). Suppose $E' \subseteq E$ and $F' \subseteq F$ are finite sets, and that $f : E \rightarrow F$ and $g : E' \rightarrow F'$ are isomorphisms. Then there is an isomorphism $f \setminus g : E \setminus E' \rightarrow F \setminus F'$.

The obvious proof of this lemma is 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 $\text{Gam}$ we will extract an isomorphism in $\text{Path}$, 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
E. 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 6 by something else than counting. As formalized in the following proof, the idea is to remark that given the data of Lemma 6, starting from a path iso from a game iso, we must replace the proof of a path iso from a game iso, we must replace the proof of a path iso from a game iso, we must replace the proof of a path iso from a game iso. This means that we cannot deduce from the proof above an algorithm to extract path isomorphisms.

**LEMMA 6.** Two finitely branching arenas $A, B$ are $\text{Gam}$-isomorphic if and only if they are $\text{Path}$-isomorphic.

**PROOF:** Consider an isomorphism $\sigma : A \rightarrow B$ in $\text{Gam}$. 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 $h_{i,k} : i \rightarrow f(i)$. Additionally, all these $k$-isomorphisms are compatible with each other, so they converge to an $\omega$-isomorphism $h_{i,\omega} : i \rightarrow f(i)$. The iso $f$ together with $h_{i,\omega}$ for all $i$ define a path isomorphism from $A$ to $B$.

**Example:** For each pair of arenas $A, B$, we have built a function $K_{A,B} : \text{Gam}(A, B) \rightarrow \text{Path}_1(A, B)$. The natural question is then whether, like in the other cases, this function defines a full functor. Unfortunately the answer is no, in fact $K_{A,B}$ is not even functorial. Indeed, the construction is based on the more explicit proof of Lemma 6 illustrated in Figure 5, which is not functorial; one can find sets $E' \subseteq E$, $F' \subseteq F$ and $G' \subseteq G$ and isomorphisms $E \rightarrow F \rightarrow G$ and $E' \rightarrow F' \rightarrow G'$ such that $(f \setminus f') \not\equiv (f; g) \setminus (f'; g')$. From this it is not hard to find a counterexample to the functoriality of $K_{A,B}$. It is a bit lengthy to describe it properly though, so we do not include it.

Although not being a functor, $K$ does satisfy some canonical property: its result is invariant under renaming of moves in $A$ and $B$. In other terms, $K_{A,B} : \text{Gam}(A, B) \rightarrow \text{Path}_1(A, B)$ is natural in $A$ and $B$, if both $\text{Gam}_{i}(\_ \rightarrow \_)$ and $\text{Path}_1(\_ \rightarrow \_)$ are seen as bifunctors from $\text{Path}_1^{op} \times \text{Path}_1$ to $\text{Set}$ (using implicitly the faithful functor from $\text{Path}_1$ to $\text{Gam}$, of Figure 4).

**F. Application to $L_2$**

Our isomorphism theorem most naturally applies to $\text{Gam}$ (so to call-by-name languages), but $L_2$ is modeled in $\text{Fami}_1(\text{Gam})$, and more precisely in the Kleisli category of the strong monad $T$, so we have to check how our result extends to this. Let us first relate isomorphisms in $\text{Fami}_1(\text{Gam})_T$ and isomorphisms in $\text{Gam}$ using the following lemma.
**Lemma 7.** Let \( A = (A_i)_{i \in I} \) and \( B = (B_j)_{j \in J} \), then isomorphisms between \( A \) and \( B \) in \( \text{Fam}_I(\text{Gam}) \) are one-to-one correspondence with pairs \( (f, (\sigma_i)_{i \in I}) \) where \( f : I \to J \) is a bijection and each \( \sigma_i : A_i \Rightarrow B_{f(i)} \) is an iso in \( \text{Gam} \).

**Proof:** Imagine \( \sigma : A \to T(B) \) and \( \tau : B \to T(A) \) are isomorphisms in \( \text{Fam}_I(\text{Gam}) \) that form an isomorphism in \( \text{Fam}_I(\text{Gam})_T \). Here \( A \) and \( B \) are families of arenas, so \( A = (A_i)_{i \in I} \), and \( \sigma = (\sigma_i)_{i \in I} \) with \( \sigma_i : A_i \Rightarrow B_{f(i)} \). Similarly, we have \( B = (B_j)_{j \in J} \) and \( \tau_j : B_j \Rightarrow T(A) \). Then, first note that \( \sigma_i \) and \( \tau_j \) necessarily first give an answer to the initial Opponent move in \( T \), i.e. the initial question of the lifted sum in \( T(B) \) and \( T(A) \). Indeed take \( i \in I \), and consider \( \sigma_i; \tau^* : A_i \to T(A) \), where \( \tau^* : T(B) \to T(A) \) is the lifting of \( \tau : B \to T(A) \). This morphism must be a component of the identity on \( A \) in \( \text{Fam}_I(\text{Gam})_T \) since \( \sigma, \tau \) form an iso. In particular, it does directly answer the initial move in \( T \). However, by definition of \( \tau^* : T(B) \to T(A) \) it is strict, i.e. it directly interrogates the left occurrence of \( T \), so \( \sigma_i \) must necessarily first answer the initial move in \( T \) otherwise we would immediately get a contradiction.

This means that each \( \sigma_i \) must first choose a component \( j \in J \) (thus inducing a function \( f : I \to J \)), then play as \( \sigma'_i : A_i \Rightarrow B_j \). The same analysis on \( \tau \) provides a function \( g : J \to I \) and strategies \( \tau'_j : B_j \Rightarrow A_{g(j)} \), and it is then obvious that since \( \sigma, \tau \) form an isomorphism \( g \) must be inverse of \( f \) and each \( \tau'_j \) inverse of \( \sigma'_{g(j)} \).

**Proposition 8.** Let us prove now that the equation theory \( E \) given in Figure 1 characterizes the types \( A \) and \( B \) such that \([A]\) and \([B]\) are isomorphic in \( \text{Fam}_I(\text{Gam})_T \).

**Lemma 8 (Type normal form).** Any type \( A \) has an unique representative (up to \( \simeq_E \)) generated by the non-terminal \( S \) in:

\[
S ::= \text{bool}^n \times T \\
T ::= \text{unit} \mid \Pi_{i \in I} U \\
U ::= T \to S
\]

**Proof:** Straightforward.

**Proposition 3.** If \([A]\) and \([B]\) are isomorphic in \( \text{Fam}_I(\text{Gam})_T \), then \( A \simeq_E B \).

**Proof:** By induction on their normal forms. For types generated by \( S \) take \( A \simeq_E \text{bool}^n \times A' \) and \( B \simeq_E \text{bool}^n \times B' \). By Lemma 7 we have \( n = p \) (since \([A']\) and \([B']\), generated by \( T \), must be singletons) and we still have \([A'] \simeq_{\text{Fam}_I(\text{Gam})} [B']\). The case of types generated by \( T \) and \( U \) is direct.

**Theorem 2.** For any types \( A, B \) of \( \mathcal{L}_2 \) whose interpretation give families \((A_i)_{i \in I}\) and \((B_j)_{j \in J}\) the following propositions are equivalent:

1. \( A \simeq_E B \)
2. \( (A_i)_{i \in I} \simeq_{\text{Fam}_I(\text{Gam})} (B_j)_{j \in J} \)
3. \( (A_i)_{i \in I} \simeq_{\text{Fam}_I(\text{Gam})} (B_j)_{j \in J} \)
4. \( A \simeq_E B \)

**Proof:** (1) \( \Rightarrow \) (2) by soundness and by definition of type isomorphisms in \( \mathcal{L}_2 \), (2) \( \Rightarrow \) (3) because the arenas are complete, hence every play of the identity is a prefix of a complete play of the identity, so any isomorphism \( \sigma : A \Rightarrow A \) such that \( \text{comp}(\sigma) = \text{comp}(id_A) \) must satisfy \( id_A \subseteq \sigma \). But as isomorphisms both are total strategies, so \( \sigma = id_A \), (3) \( \Rightarrow \) (4) by Proposition 3. Finally, (4) \( \Rightarrow \) (1) because equations in \( E \) can be implemented in the syntax of \( \mathcal{L}_2 \).

**G. Isomorphisms in the presence of \text{nat}**

As suggested by the importance of counting in the proof, the presence of \text{nat} makes it possible to build a non-trivial isomorphism by playing Hilbert’s hotel. Consider the programming language \( \mathcal{L} \) from \cite{1}, obtained from \( \mathcal{L}_2 \) by replacing \text{bool} with \text{nat}. This language has a fully abstract interpretation in \( \text{Fam}(\text{Gam}_\infty)_T \), where \( \text{Gam}_\infty \) is the category of not necessarily finitely branching arenas, and single-threaded strategies.

**Proposition 4.** There is an isomorphism in \( \text{Fam}(\text{Gam}_\infty)_T \) between \( [(\text{nat} \to \text{unit}) \to (\text{nat} \to \text{unit}) \to \text{unit}] \) and \( [(\text{nat} \to \text{unit}) \to (\text{unit} \to \text{unit}) \to \text{unit}] \).

**Proof:** By definition of the interpretation of types, this boils down to an isomorphism in \( \text{Gam}_\infty \) between the two arenas represented in Figure 6. Informally, the isomorphism from left to right can be described as follows. As long as \( q' \) has not been played, it behaves as the identity. Whenever Opponent plays \( q' \), it copies it to the other side. Then if \( q' \) has only appeared once, there are two available copies of \text{nat} \to \text{unit} on the left side, one \text{nat} \to \text{unit} and one \text{unit} \to \text{unit} on the right side, so Player picks a bijection between \( \mathbb{N} + \mathbb{N} \) and \( \mathbb{N} + 1 \) and plays accordingly. More generally, if \( q' \) has appeared \( n \) times, there are exactly \( n + 1 \) available copies of \text{nat} \to \text{unit} on the left hand side, one copy of \text{nat} \to \text{unit} and \( n \) copies of \text{unit} \to \text{unit} on the right hand side, so Player has to follow a bijection between \( (n + 1)\mathbb{N} \) and \( \mathbb{N} + n \). So any choice of a bijection between \( (n + 1)\mathbb{N} \) and \( \mathbb{N} + n \) for all \( n \in \mathbb{N} \) will provide an isomorphism.

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 7, where \( * \) denotes the product operation on natural numbers, and \( \text{div} M N \) outputs the result of the division algorithm on \( M : \text{nat} \) and \( N : \text{nat} \). 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.
We can define one using call-by-value allowed to define new non-trivial type isomorphisms. For example, we could try to eliminate bad variables. Murawski and Tzevelekos’ games model of Reduced ML may be a good setting to try that, however it is not clear whether our core results can be reproven in their nominal setting.

Acknowledgments. We would like to thank Guy McCusker for interesting discussions on the games models for state and for his help to get a term from the strategy described in Proposition 4, and the anonymous referees for their useful comments and suggestions. We also would like to acknowledge the support of (UK) EPSRC grant RC-CM1025.

REFERENCES


Figure 7. Type isomorphism in $\mathcal{L}$ between (nat → unit) → (nat → unit) → unit and (nat → unit) → (unit → unit) → unit.