Isomorphisms of types in the presence of higher-order references

Pierre Clairambault

University of Bath

LICS 2011

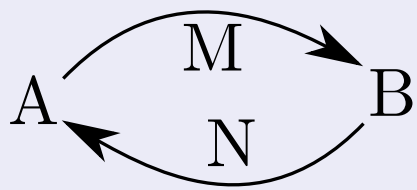
I. Short overview on isomorphisms of types

Isomorphisms of types

Let $\mathcal{L}$ be a typed programming language, its operational semantics defines an equational theory $=_{\mathcal{L}}$ on its terms.

Definition

An **isomorphism** between types A and B consists of two programs $\vdash M : A \to B$ and $\vdash N : B \to A$ such that $M \circ N =_{\mathcal{L}} id_B$ and $N \circ M =_{\mathcal{L}} id_A$.



$$\mathrm{M} \circ \mathrm{N} = \mathrm{id_B}$$

$$\mathrm{N} \circ \mathrm{M} = \mathrm{id_A}$$

Problem

Characterize the isomorphisms of types in $\mathcal{L}$.

Syntactic approach

Theorem (Dezani-Ciancaglini, 1976)

The normal forms of the untyped λ-calculus possessing an inverse are the **finite hereditary permutators**.

- Bruce and Longo (1985): finite hereditary permutators can be simply typed by $T \to T'$, when T and T' only differ with applications of **swap**:

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

- Later extended to richer types and constructors (Balat, Bruce, Di Cosmo, Fiore, Longo, ...).

Some results – typed call-by-name λ-calculi

$$
\begin{array}{rcl}
A \to (B \to C) & \simeq & B \to (A \to C) \\
A \times B & \simeq & B \times A \\
A \times (B \times C) & \simeq & (A \times B) \times C \\
(A \times B) \to C & \simeq & A \to (B \to C) \\
A \to (B \times C) & \simeq & (A \to B) \times (A \to C) \\
A \times 1 & \simeq & A \\
A \to 1 & \simeq & 1 \\
1 \to A & \simeq & A \\
\forall X.\forall Y.A & \simeq & \forall Y.\forall X.A \\
\forall X.A & \simeq & \forall Y.A[Y/X] \\
\forall X.(A \to B) & \simeq & A \to \forall X.B \\
\forall X.A \times B & \simeq & \forall X.A \times \forall X.B \\
\forall X.1 & \simeq & 1
\end{array}
$$

Semantic approach

An analysis of isomorphisms in game semantics yields:

Theorem (Laurent, 2005)

In innocent game semantics, two arenas are isomorphic if and only if they are identical, up to the renaming of moves.

- Using this, Laurent could find the isomorphisms of types for LLP and the CBN and CBV $\lambda\mu$-calculi.
- De Lataillade (2007) later extended these tools to characterize isomorphisms for classical System F and Curry-style System F.

Laurent's proof relies on **visibility**. He conjectures that it is not necessary.

Application: software modularity

Type isomorphisms are a useful tool for modularity.

- Rittri, 1991: Search functions in libraries using their type modulo isomorphism as a key,
- Extensions: matching and unification of types modulo isomorphism.

Isomorphisms can also be used **inside** the language:

- Wadler, 1987: Use isomorphisms as intermediate between an internal hidden implementation of a data type and an external one,
- Nipkow, 1990: Incorporate them directly in the type system,
- Di Cosmo, 1986: Automatically correct typing errors, when the coercion is unique.

There were several implementations of these ideas.

Isomorphisms in the presence of references

Motivation for isomorphisms of types in the presence of references:

- The mentionned works apply isomorphisms for **pure** languages (λ-calculus, System F...) to languages with effect (ML).
- The theory of isomorphisms without effects is still sound in the presence of references, but it may not be complete.

In this work:

- We prove Laurent's conjecture for finitary types,
- We characterize isomorphisms in a language with finite types and higher-order store,
- We refute Laurent's conjecture in the presence of natural numbers.

II. A language $\mathcal{L}_2$ with higher-order store

Types and terms of $\mathcal{L}_2$

Definition (Types)

$$A ::= \texttt{unit} \mid \texttt{bool} \mid A \times B \mid A \to B \mid \texttt{var}[A]$$

Definition (Terms)

$$\begin{array}{rcll} M & ::= & x \mid \lambda x.M \mid M\ M \mid \langle M, M \rangle \mid \texttt{fst}\ M \mid \texttt{snd}\ M & (\lambda\text{-calculus}) \\ & & \mid \texttt{true} \mid \texttt{false} \mid \texttt{if}\ M\ M\ M & (\text{booleans}) \\ & & \mid \texttt{new}_A \mid M := M \mid !M \mid \texttt{mkvar}\ M\ M \mid \texttt{skip} & (\text{references}) \end{array}$$

Typing rules

Usual rules for λ-calculus and booleans, plus:

$$\frac{}{\Gamma \vdash \mathtt{new}_A : \mathtt{var}[A]} \qquad \frac{\Gamma \vdash M : \mathtt{var}[A]}{\Gamma \vdash !M : A}$$

$$\frac{\Gamma \vdash M : \mathtt{var}[A] \quad \Gamma \vdash N : A}{\Gamma \vdash M := N : \mathtt{unit}}$$

$$\frac{\Gamma \vdash M : A \to \mathtt{unit} \quad \Gamma \vdash N : \mathtt{unit} \to A}{\Gamma \vdash \mathtt{mkvar}\ M\ N : \mathtt{var}[A]}$$

Note the presence of **bad variables**.

Equality on terms and isomorphisms

We equip $\mathcal{L}_2$ with its standard call-by-value operational semantics.

Definition (Observational equivalence)

If M and N are two terms, then $M \cong N$ if and only if for all context $C[-]$ such that $C[M]$ and $C[N]$ are closed,

$$C[M] \Downarrow \quad \Leftrightarrow \quad C[N] \Downarrow$$

Definition (Isomorphisms of types)

Two types A and B are **isomorphic** ($A \simeq_{\mathcal{L}_2} B$) if there are terms $x : A \vdash M : B$ and $y : B \vdash N : A$ such that

$$\begin{aligned} (\lambda y.N)\ M &\cong x \\ (\lambda x.M)\ N &\cong y \end{aligned}$$

Our main result

Theorem

Isomorphisms in $\mathcal{L}_2$ are given by the following equational theory:

$$\begin{aligned} A \times B &=_{\mathcal{E}} B \times A \\ A \times (B \times C) &=_{\mathcal{E}} (A \times B) \times C \\ A \times \mathtt{unit} &=_{\mathcal{E}} A \\ \mathtt{bool} \times A \to B &=_{\mathcal{E}} (A \to B) \times (A \to B) \\ \mathtt{var}[A] &=_{\mathcal{E}} (A \to \mathtt{unit}) \times (\mathtt{unit} \to A) \end{aligned}$$

Note that $A \to (B \to C) \simeq B \to (A \to C)$ is not satisfied, because of the CBV setting (the η-rule only applies on values).

III. Game semantics for $\mathcal{L}_2$

Interpretation of types: arenas

Following Abramsky, Honda & McCusker, 1998.

Definition

An **arena** is a tuple $A = \langle M_A, \lambda_A, I_A, \vdash_A \rangle$ where

- M_A is a set of **moves**,
- $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$ is a **labelling** function,
- $I_A \subseteq {\lambda_A}^{-1}(\{OQ\})$ is a set of **initial moves**,
- $\vdash_A \subseteq M_A^2$ is a relation called **enabling**.

For the purposes of this work, we additionally require A to be:

- **Complete**: for each question $q \in M_A$, there should be an answer $a \in M_A$ such that $q \vdash_A a$,
- **Finitely branching**: any $m \in M_A$ justifies finitely many moves.

For each type constructor, there is a corresponding arena construction.

Legal plays and strategies

Definition

A **legal play** on A is a **pointing string** on M_A abiding with $\vdash_A$ and I_A. Moreover, it is

- **Alternating**: O and P alternate,
- **Well-bracketed**: each answer points to the pending question.

If s only satisfies well-bracketing, it is a **pre-legal play**. Let $\mathcal{L}_A$ denote the set of legal plays on A, and $\mathcal{L}'_A$ the set of pre-legal plays on A.

Definition

A **strategy** $\sigma : A$ is a subset $\sigma \subseteq \mathcal{L}_A$ which satisfies:

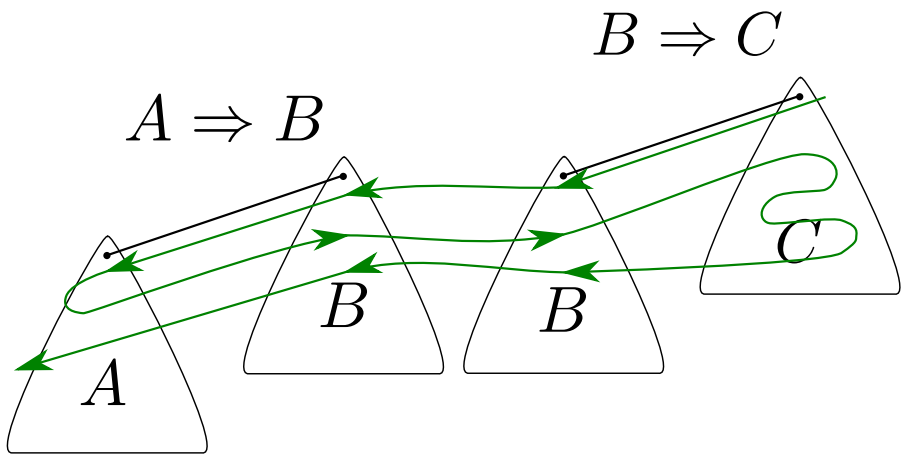
- All plays in σ are P-ending, and σ is closed under P-ending prefix,
- σ is **deterministic**: $\forall sab, sab' \in \sigma,\ b = b'$.

Composition and the category **Gam**

Definition

Let $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$, then we define:

$$\sigma;\tau = \left\{ u_{\restriction A,C} \;\middle|\; \begin{array}{l} u_{\restriction A,B} \in \sigma \\ u_{\restriction B,C} \in \tau \\ u_{\restriction A,C} \in \mathcal{L}_{A \Rightarrow C} \end{array} \right\} : A \Rightarrow C$$



Theorem

This operation defines a category **Gam**, *in which one can interpret* $\mathcal{L}_2$ *(not straightforward).*

IV. Isomorphisms in **Gam**

Paths and path isomorphisms

Definition

A **path** on A is a play $p \in \mathcal{L}_A$ such that each move points to the previous move. We write $p \in \mathcal{P}_A$. A **path morphism** from A to B is a function $\phi : \mathcal{P}_A \to \mathcal{P}_B$ which:

- Preserves sequentiality: $\phi(\epsilon) = \epsilon$ and $\phi(sa) = \phi(s)b$ for some b.
- Preserves labelling: if $\phi(sa) = \phi(s)b$, $\lambda_A(a) = \lambda_B(b)$.

Remark

If A and B are forests, they are path-isomorphic if and only if they are identical up to renaming of moves.

Definition

We have a category **Path** of arenas and path morphisms

Our goal is to relate isomorphisms in **Gam** with isomorphisms in **Path**.

Weaker notions of morphisms

Definition

A **sequential play morphism** from A to B is $\phi : \mathcal{L}'_A \to \mathcal{L}'_B$ such that:

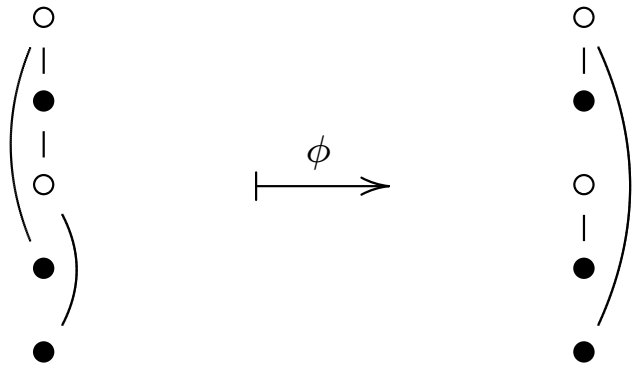
- It preserves sequentiality:

$$\begin{aligned} \phi(\epsilon) &= \epsilon \\ \phi(sa) &= \phi(s)b \text{ for some move } b \text{ in } B \end{aligned}$$

- It preserves labelling:

$$\text{If } \phi(sa) = \phi(s)b, \text{ then } \lambda_A(a) = \lambda_B(b)$$

There is a category **Seq** of arenas and sequential play morphisms,



$$\lambda(\circ) = O$$

$$\lambda(\bullet) = P$$

Relating isos in **Gam** with isos in **Seq**

Isomorphisms must be **zig-zag**:

Definition

A play $s \in \mathcal{L}_{A \Rightarrow B}$ is **zig-zag** if:

1. Each P-move following a O-move in A is in B,
2. Each P-move following a O-move in B is in A.

Definition

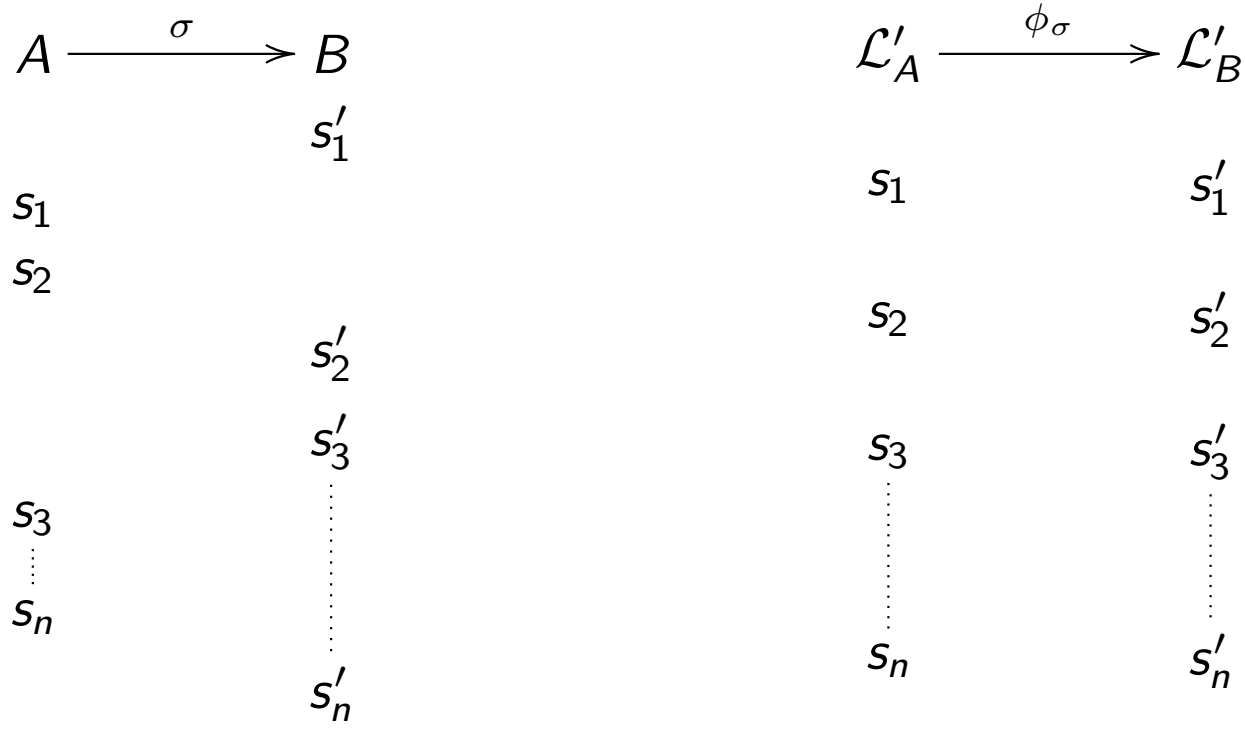
A strategy $\sigma : A \Rightarrow B$ is **zig-zag** if all its plays are.

Proposition

Isomorphisms in **Gam** *are zig-zag strategies.*

From zig-zag isomorphisms to play isomorphisms

Take the relation $\phi_\sigma = Rel(\sigma)$ (Hyland and Schalk's relational functor)



When σ is iso, ϕ_σ is in fact a function, and a sequential play morphism.

Proposition

$S(\sigma) = \phi_\sigma$ yields an isomorphism of groupoids between $\mathbf{Gam}_i$ *and* $\mathbf{Seq}_i$.

(where $\mathcal{C}_i$ denotes the groupoid of isomorphisms in a category $\mathcal{C}$)

Possible extensions of a play

How many ways are there to extend $s \in \mathcal{L}'_A$?

- If $a \in M_A$, its **arity** is $ar(a) = |\{b \in M_A \mid a \vdash_A b\}|$
- An extension of $s \in \mathcal{L}'_A$ corresponds to a move and the choice of its justifier:

$$ext(s) = \{(i, b) \mid b \in M_A \wedge s_i \vdash_A b\}$$

- Thus for all $s \in \mathcal{L}'_A$, there are

$$\begin{aligned} |ext(s)| &= |\{(i, b) \mid b \in M_A \wedge s_i \vdash_A b\}| \\ &= |\bigcup_{i \leq |s|} \{(i, b) \mid s_i \vdash_A b\}| \\ &= \sum_{i=1}^{|s|} ar(s_i) \end{aligned}$$

ways to extend s to sa.

Actions of sequential isomorphisms on extensions of plays

Suppose $\phi : A \to B$ is an iso in **Seq**. Then:

- For all $s \in \mathcal{L}'_A$, ϕ induces an isomorphism

$$\phi_s : ext(s) \to ext(\phi(s))$$

  Thus we always have $|ext(s)| = |ext(\phi(s))|$
- For all $\phi(sa) = \phi(s)b$,

$$\begin{aligned} |ext(sa)| &= |ext(\phi(s)b)| \\ |ext(s)| + ar(a) &= |ext(\phi(s))| + ar(b) \\ ar(a) &= ar(b) \end{aligned}$$

So sequential isomorphisms preserve the arity of moves.

Generalization: Sequential isomorphisms associate to each move a move with a k-isomorphic subtree, for all $k \in \mathbb{N}$.

Laurent's conjecture is true

Theorem

If A and B are isomorphic in **Gam**, *then they are isomorphic in* **Path**.

Proof.

We have built in fact a family of functions:

$$G_{A,B} : \mathbf{Gam}_i(A, B) \to \mathbf{Path}_i(A, B)$$

The extraction function $G_{A,B}$ can be made natural in A and B, but (we conjecture) **not** functorial. □

It is crucial that A and B are **finitely branching**.

Corollary: isomorphisms in $\mathcal{L}_2$

Theorem

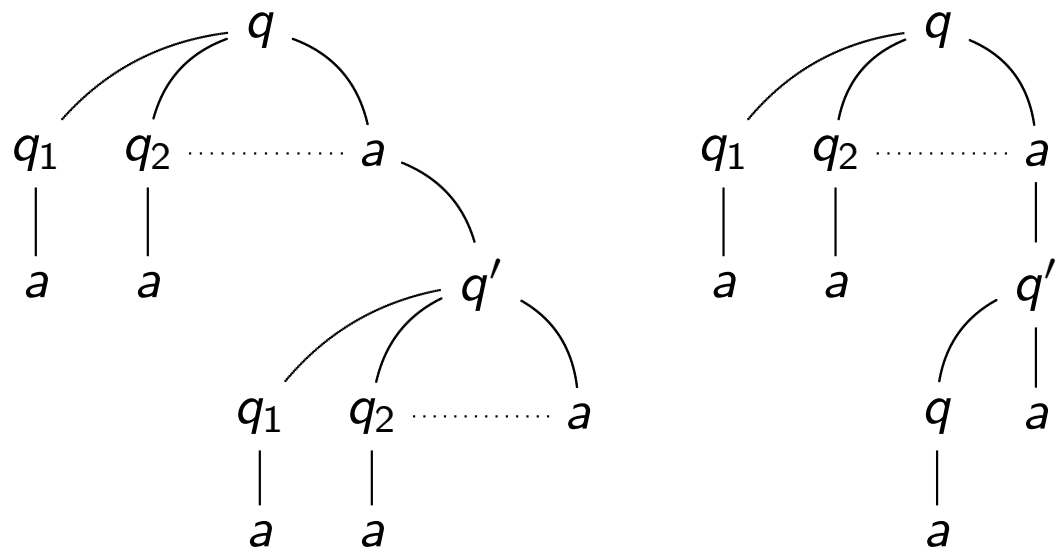
Isomorphisms in $\mathcal{L}_2$ are given by the following equational theory:

$$
\begin{aligned}
A \times B &=_{\mathcal{E}} B \times A \\
A \times (B \times C) &=_{\mathcal{E}} (A \times B) \times C \\
A \times \mathtt{unit} &=_{\mathcal{E}} A \\
\mathtt{bool} \times A \to B &=_{\mathcal{E}} (A \to B) \times (A \to B) \\
\mathtt{var}[A] &=_{\mathcal{E}} (A \to \mathtt{unit}) \times (\mathtt{unit} \to A)
\end{aligned}
$$

V. Isomorphisms with infinite data types

Laurent's conjecture is false

The following arenas are isomorphic in $\mathbf{Gam}_\infty$:



- On one side there are $2\mathbb{N}$ choices, on the other side $\mathbb{N} + 1$.
- In general, if q' has been played n times, we need a bijection between $(n+1)\mathbb{N}$ and $\mathbb{N} + n$.

So in the general case, Laurent's conjecture is **false**.

$$(\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{nat} \to \mathtt{unit}) \to \mathtt{unit}$$
$$\simeq_{\mathcal{L}} (\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{unit} \to \mathtt{unit}) \to \mathtt{unit}.$$

$$\begin{array}{l}
f : (\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{nat} \to \mathtt{unit}) \to \mathtt{unit} \vdash \\
\ \mathtt{new\ count} := 0,\ \mathtt{func} := \bot\ \mathtt{in} \\
\ \lambda g : \mathtt{nat} \to \mathtt{unit}. \\
\ \ \mathtt{let}\ x = f\ (\lambda n.\ g(n * (!\mathtt{count} + 1)))\ \mathtt{in} \\
\ \ \lambda h : \mathtt{unit} \to \mathtt{unit}. \\
\ \ \ \mathtt{count}\ := !\mathtt{count} + 1; \\
\ \ \ \mathtt{let}\ c\ = !\mathtt{count}\ \mathtt{in} \\
\ \ \ \mathtt{func}\ := \ \lambda n.\ \mathtt{if}\ n = !\mathtt{count} \\
\qquad\qquad \mathtt{then}\ h \\
\qquad\qquad \mathtt{else}\ !\mathtt{func}\ n; \\
\ \ \ x\ (\lambda n.\ \mathtt{if}\ n = 0\ \mathtt{then}\ !\mathtt{func}\ c\ ()\ \mathtt{else} \\
\qquad\qquad\qquad\qquad g((n-1) * (!\mathtt{count} + 1) + c))
\end{array}$$

$$\begin{array}{l}
f : (\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{unit} \to \mathtt{unit}) \to \mathtt{unit} \vdash \\
\ \mathtt{new\ count} := 0,\ \mathtt{func} := \bot\ \mathtt{in} \\
\ \lambda g : \mathtt{nat} \to \mathtt{unit}. \\
\ \ \mathtt{let}\ x = f\ (\lambda n. \\
\ \ \ \mathtt{let}\ (q, r)\ =\ \mathtt{div}\ n\ (!\mathtt{count} + 1)\ \mathtt{in} \\
\ \ \ \ \mathtt{if}\ r = 0\ \mathtt{then}\ g\ q\ \mathtt{else}\ !\mathtt{func}\ r\ (q + 1)) \\
\ \ \mathtt{in} \\
\ \ \lambda h : \mathtt{nat} \to \mathtt{unit}. \\
\ \ \ \mathtt{count} := !\mathtt{count} + 1; \\
\ \ \ \mathtt{func} := \lambda n.\ \mathtt{if}\ n = !\mathtt{count}\ \mathtt{then}\ h \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{else}\ !\mathtt{func}\ n; \\
\ \ \ \mathtt{let}\ c = !\mathtt{count}\ \mathtt{in}\ x\ (\lambda_.\ !\mathtt{func}\ c\ 0)
\end{array}$$

These two terms are inverses of one another!

More examples

Some more isomorphisms in $\mathcal{L}$:

$$\begin{array}{rcl}
\mathtt{nat} \times \mathtt{nat} & \simeq & \mathtt{nat} \\
(\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{nat} \to \mathtt{unit}) \to \mathtt{unit} & \simeq & (\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{unit} \to \mathtt{unit}) \to \mathtt{unit} \\
\mathtt{nat} \to (\mathtt{nat} \to \mathtt{unit}) & \simeq & \mathtt{nat} \to \mathtt{unit} \\
\mathtt{nat} \to (\mathtt{nat} \to \mathtt{unit}) & \simeq & (\mathtt{nat} \times \mathtt{nat}) \to \mathtt{unit}
\end{array}$$

Some non-isomorphisms:

$$\begin{array}{rcl}
(\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{nat} \to \mathtt{bool}) \to \mathtt{unit} & \not\simeq & (\mathtt{nat} \to \mathtt{unit}) \to (\mathtt{unit} \to \mathtt{bool}) \to \mathtt{unit} \\
\mathtt{nat} \to (\mathtt{nat} \to \mathtt{bool}) & \not\simeq & (\mathtt{nat} \times \mathtt{nat}) \to \mathtt{bool}
\end{array}$$

VI. Conclusion

Conclusion

The game-theoretic theorem also directly applies:

- To $\mathcal{L}_2$ extended with sums and empty type,
- To `var`-free isomorphisms in a language without bad variables.

It should extend as well:

- To $\mathcal{L}_2$ with `call/cc`,
- To call-by-name $\mathcal{L}$ **with** `nat` but **without** `call/cc`.

In CBN, building a non-trivial isomorphism requires `call/cc`.