Type Isomorphisms and Program Isomorphisms.

Carlos C. Martínez

(Join work with Dan Dougherty)

Department of Mathematics and Computer Science,

Wesleyan University

cmartinez@wesleyan.edu

February, 2005

Overview

- PART I
  - **Type Isomorphisms and Program Isomorphisms.**
  - Program Transformation and Matching.
  - Merging ideas.
- PART II
  - λ-Calculus setting.
  - Deciding Type Isomorphisms.
  - References.

Example 1:

There are four Caml functions abstracted from $2*x+3*y$:

#Let f1 $=$ fun$(x,y) \rightarrow 2*x+3*y$;;

f1: $int*int \rightarrow int$ = $\langle$**fun**$\rangle$

#Let f2 $=$ fun$(y,x) \rightarrow 2*x+3*y$;;

f1: $int*int \rightarrow int$ = $\langle$**fun**$\rangle$

#Let f3 $=$ fun$xy \rightarrow 2*x+3*y$;;

f3: $int \rightarrow int \rightarrow int$ = $\langle$**fun**$\rangle$

#Let f4 $=$ fun$yx \rightarrow 2*x+3*y$;;

f4: $int \rightarrow int \rightarrow int$ = $\langle$**fun**$\rangle$

Example 1:

#Let f1 $= \boxed{\text{fun}(x,y)} \rightarrow 2*x+3*y;;$

f1: $int*int \rightarrow int$ = $\langle$**fun**$\rangle$

#Let f2 $= \boxed{\text{fun}(y,x)} \rightarrow 2*x+3*y;;$

f2: $int*int \rightarrow int$ = $\langle$**fun**$\rangle$

#Let f3 $= \boxed{\text{fun}xy} \rightarrow 2*x+3*y;;$

f3: $int \rightarrow int \rightarrow int$ = $\langle$**fun**$\rangle$

#Let f4 $= \boxed{\text{fun}yx} \rightarrow 2*x+3*y;;$

f4: $int \rightarrow int \rightarrow int$ = $\langle$**fun**$\rangle$

Example 1:

#Let curry f $xy = \mathrm{fun}(x,y)$;;

curry: $((A * B) \to C) \to (A \to (B \to C))$ = $\langle$**fun**$\rangle$

#Let uncurry f $(x,y) = \mathrm{fun}xy$;;

uncurry: $(A \to (B \to C)) \to ((A * B) \to C)$ = $\langle$**fun**$\rangle$

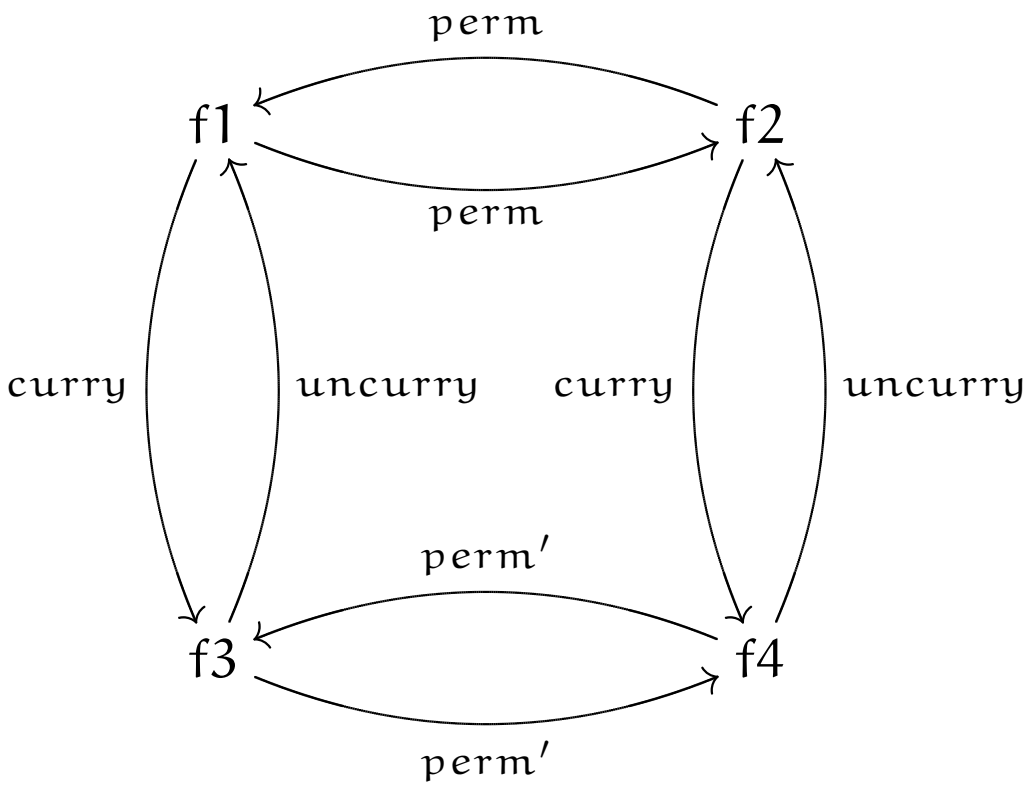
#Let perm f $xy = \mathrm{fun}yx$;;

perm : $(A \to (B \to C)) \to (B \to (A \to C))$ = $\langle$**fun**$\rangle$

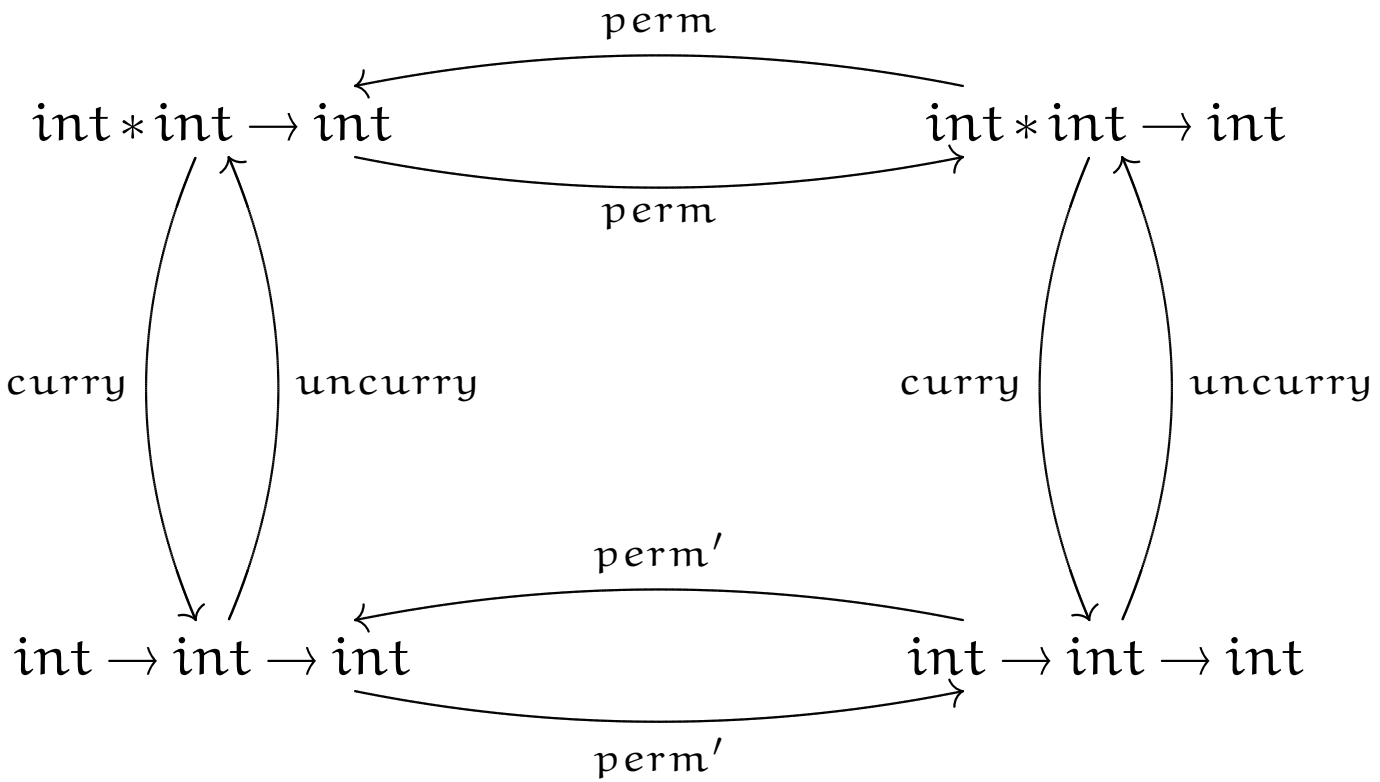
#Let perm' f $(x,y) = \mathrm{fun}(y,x)$;;

perm' : $((A * B) \to C)) \to ((B * A) \to C))$= $\langle$**fun**$\rangle$

Example 1:

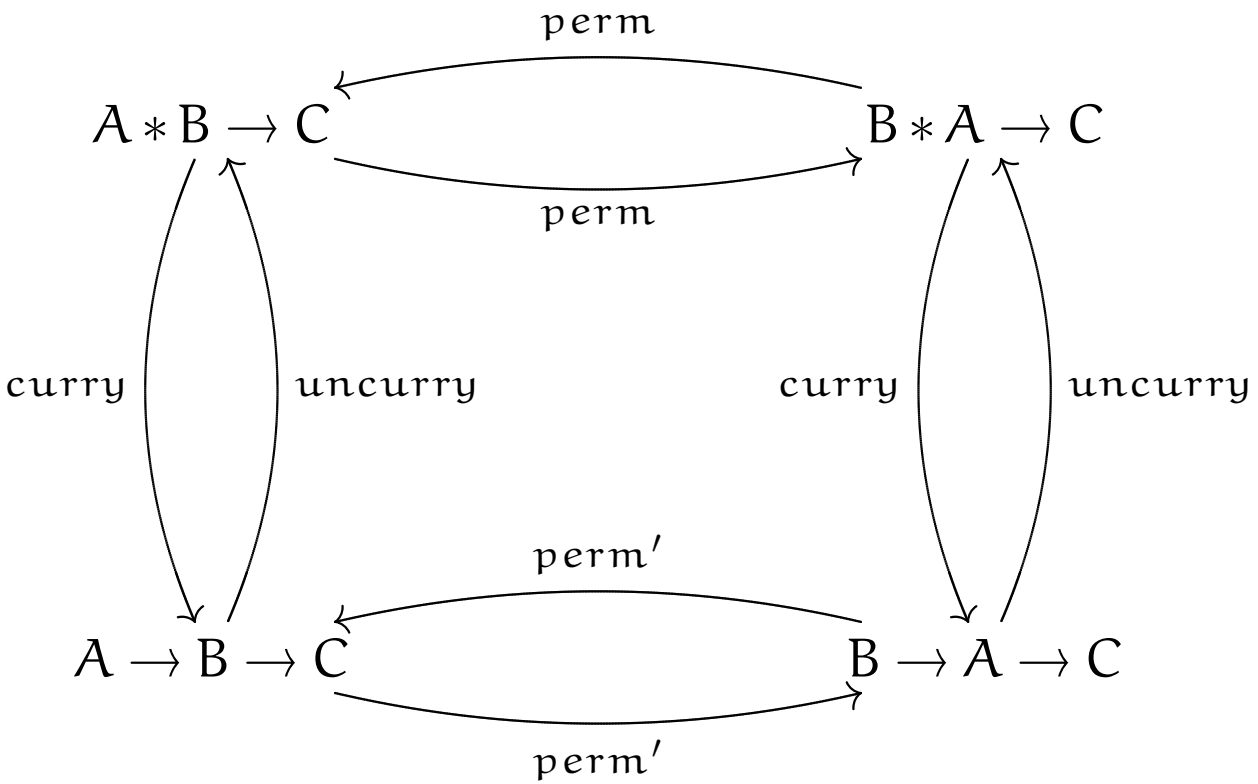


Example 1:


perm
int * int → int
int * int → int
perm
curry
uncurry
curry
uncurry
perm′
int → int → int
int → int → int
perm′

Example 1:

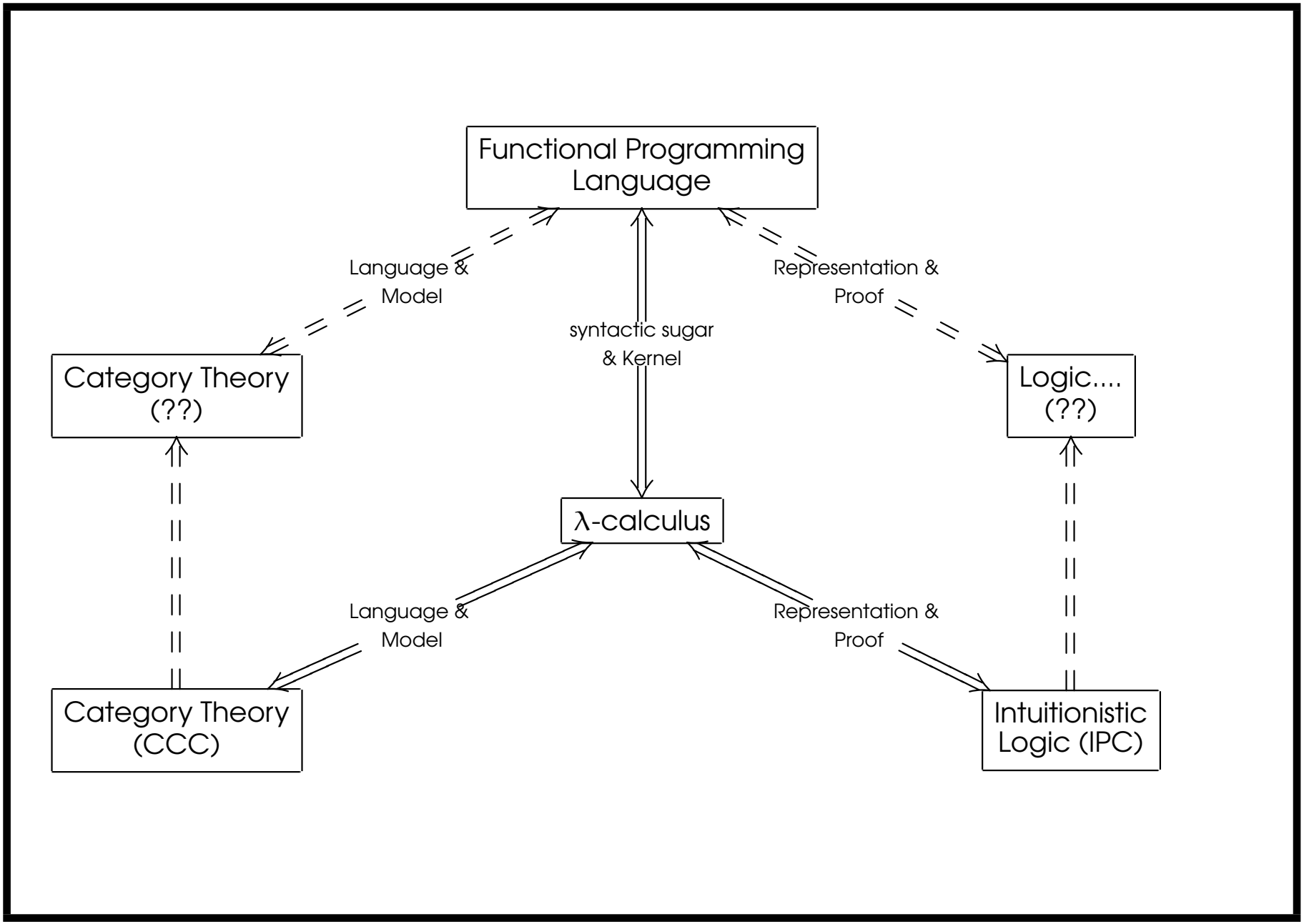
As we might guess, their function space are isomorphic:



Definable isomorphisms:

Definition ***(Type Isomorphisms)*** *Two type* A *and* B *are definably isomorphic* $(\mathrm{A} \cong_{\mathrm{d}} \mathrm{B})$ *if and only if there exists programs* $\mathrm{M} : \mathrm{A} \to \mathrm{B}$ *and* $\mathrm{N} : \mathrm{B} \to \mathrm{A}$ *such that* $\mathrm{M} \circ \mathrm{N} = \mathrm{Id}$ *and* $\mathrm{N} \circ \mathrm{M} = \mathrm{Id}$*, the identities of type* A *and* B*.*

Definition ***(Program Isomorphisms)*** *Two programs* $\mathrm{P} : \mathrm{A}$ *and* $\mathrm{Q} : \mathrm{B}$ *are definably isomorphic* $(\mathrm{P} \cong_{\mathrm{d}} \mathrm{Q})$ *if and only if* $(\mathrm{A} \cong_{\mathrm{d}} \mathrm{B})$ *by an invertible program* $\mathrm{F} : \mathrm{A} \to \mathrm{B}$ *such that* $\mathrm{FP} = \mathrm{Q}$*.*


Functional Programming Language
Language & Model
syntactic sugar & Kernel
Representation & Proof
Category Theory (??)
Logic.... (??)
λ-calculus
Language & Model
Representation & Proof
Category Theory (CCC)
Intuitionistic Logic (IPC)

Overview

- PART I
  - **Type Isomorphisms and Program Isomorphisms.**
  - **Program Transformation and Matching.**
  - Merging ideas.
- PART II
  - λ-Calculus setting.
  - Deciding Type Isomorphisms.
  - References.

Example 2:

#Let rec fact n = **if** $(n = 0)$ **then** 1
else $n*$ fact$(n-1)$;;
fact: $int \to int$ = ⟨**fun**⟩

#Let rec rev l = **if** $(l = [])$ **then** []
else $append'([head(l)], rev(tail(l)))$;;
rev: $int^* \to int^*$ = ⟨**fun**⟩

By "abstracting", we can see that they are instances of a "iterator"

#Let rec iter fx = **if** (ax) **then** (bx) **else** $f((cx), \text{iter } f(dx))$;;

iter: $\underline{(A \times A \to A)} \to A \to (A \to A)$= ⟨**fun**⟩

Where $a : A \to$Bool, b, c and $d : A \to A$

Remark: $append'$ is an isomorphic version of the usual $append$

Example 2:

Thus:

σ_1(iter $*n$) = fact n

$$\sigma_1 \begin{cases} a \leftarrow \lambda x.(x == 0); \\ b \leftarrow \lambda x.1; \\ c \leftarrow \lambda x.x; \\ d \leftarrow \lambda x.(x - 1); \end{cases}$$

σ_2(iter $append'\ l$) = rev l

$$\sigma_2 \begin{cases} a \leftarrow \lambda x.(x == []); \\ b \leftarrow \lambda x.0; \\ c \leftarrow \lambda x.[(head\ x)]; \\ d \leftarrow \lambda x.(tail\ x); \end{cases}$$

Matching

Definition ***(Matching terms)*** *Let* M *be term, and* N *be a close term, we say that* M *is matchable to* N *if and only if there exists a substitution* σ *such that* $\sigma(\mathrm{M}) = \mathrm{N}$

Definition ***(Matching programs)*** *Let* P *and* Q *be programs, we say that* P *is matchable to* Q *if and only if there exists inputs* $\mathrm{in}_1, ..., \mathrm{in}_n$ *such that* $\mathrm{P in}_1 ... \mathrm{in}_n = \mathrm{Q}$

Example 2:

Optimization: Consider the following "Loop" function

#**Let rec** loop pfx = **if** (px) **then** x **else** loop $\mathrm{pf}(\mathrm{fx})$;;

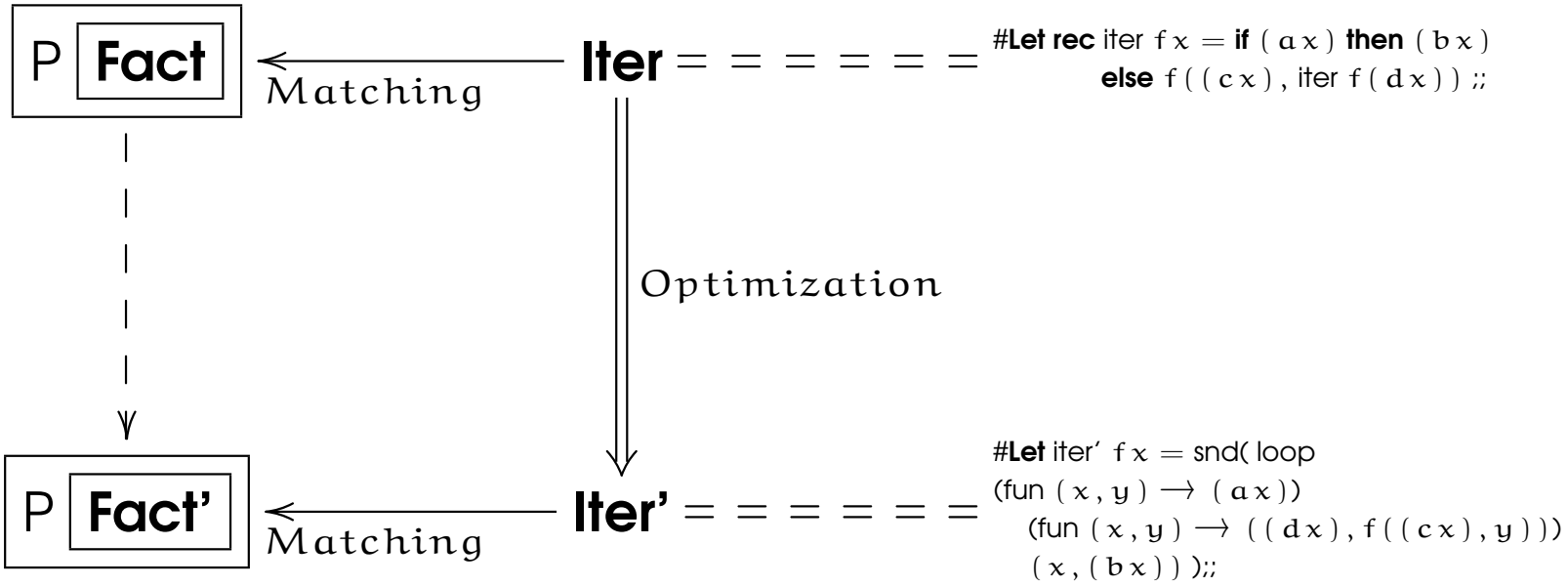
loop: $(A \to \mathrm{Bool}) \to (A \to A) \to (A \to A)$= ⟨**fun**⟩

We gain a more efficient stack usage, and...

#**Let** fact' $\mathrm{n} =$
snd(loop (fun $(\mathrm{m}, \mathrm{y}) \to \mathrm{m} = 0$)
(fun $(\mathrm{m}, \mathrm{y}) \to (\mathrm{m} - 1, \mathrm{m} * \mathrm{y})$)
$(\mathrm{n}, 1)$);;
fact': $\mathrm{int} \to \mathrm{int}$ = ⟨**fun**⟩

#**Let** rev' $\mathrm{l} =$
snd(loop (fun $(\mathrm{l}, \mathrm{y}) \to \mathrm{l} = []$)
(fun $(\mathrm{l}, \mathrm{y}) \to (\mathrm{tail}(\mathrm{l}), \mathrm{append}'([\mathrm{head}(\mathrm{l})], \mathrm{y}))$)
$(\mathrm{l}, [])$);;
rev': $\mathrm{int}^* \to \mathrm{int}^*$ = ⟨**fun**⟩

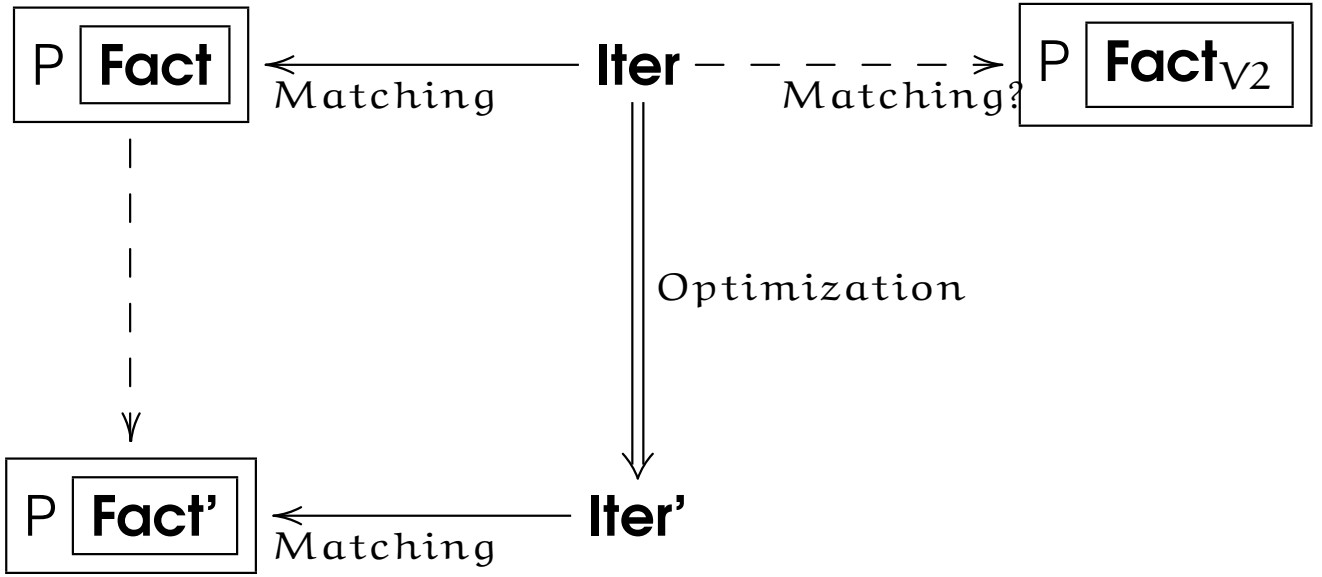
Example 2:


P Fact
Matching
Iter = = = = = =
#Let rec iter f x = if (a x) then (b x)
else f ((c x) , iter f (d x)) ;;
Optimization
P Fact'
Matching
Iter' = = = = = =
#Let iter' f x = snd(loop
(fun (x , y) → (a x))
(fun (x , y) → ((d x) , f ((c x) , y)))
(x , (b x)));;

Overview

- PART I
  - **Type Isomorphisms and Program Isomorphisms.**
  - **Program Transformation and Matching.**
  - **Merging ideas.**
- PART II
  - λ-Calculus setting.
  - Deciding Type Isomorphisms.
  - References.

Example 3:


P Fact
Matching
Iter
Matching?
P Fact$_{V2}$
Optimization
P Fact'
Matching
Iter'

Example 3:

#**Let rec** fact $n =$ **if** $(n = 0)$ **then** 1
else $n *$ fact$(n-1)$;;

fact: $int \rightarrow int$ = $\langle$**fun**$\rangle$

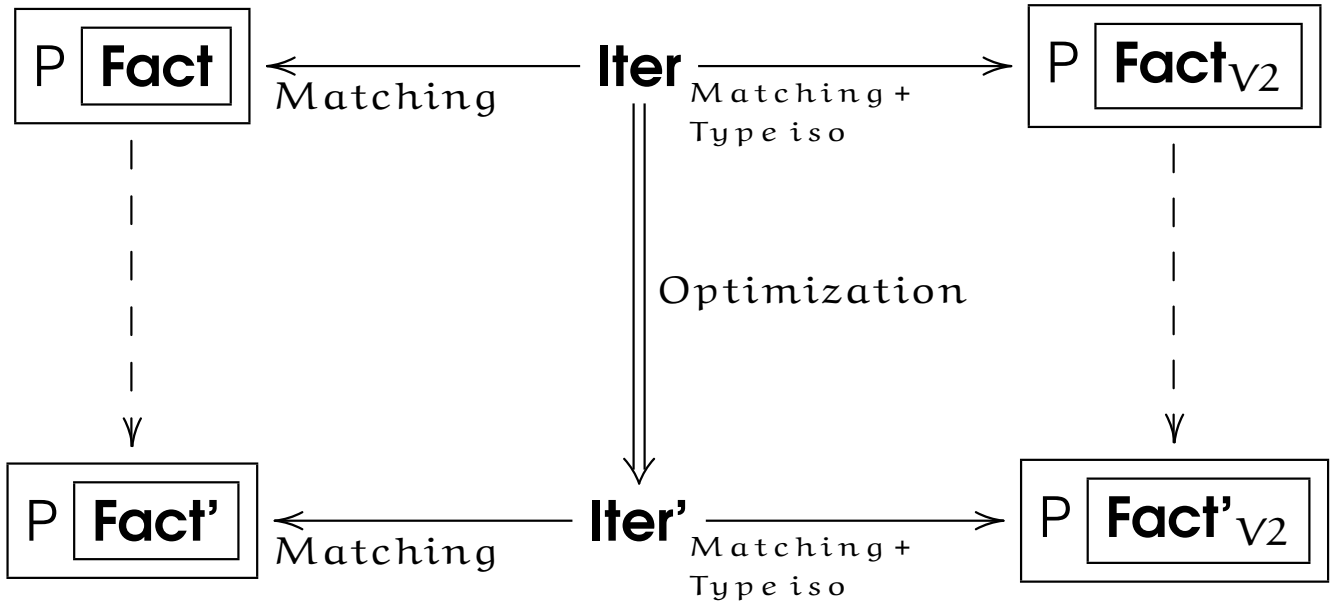
#**Let rec** fact$_{V2}$ $n =$ **if** $(n = 0)$ **then** 1
else $\boxed{*' n \text{ fact}_{V2}\,(n-1)}$;;

fact$_{V2}$: $int \rightarrow int$ = $\langle$**fun**$\rangle$

Problem: fact$_{V2}$ is not an instance of our **iter** function...

Solution: Actually fact$_{V2}$ is an instance of an isomorphic version of **iter**...

Example 3:


P Fact
Matching
Iter
Matching +
Type iso
P Fact$_{V2}$
Optimization
P Fact'
Matching
Iter'
Matching +
Type iso
P Fact'$_{V2}$

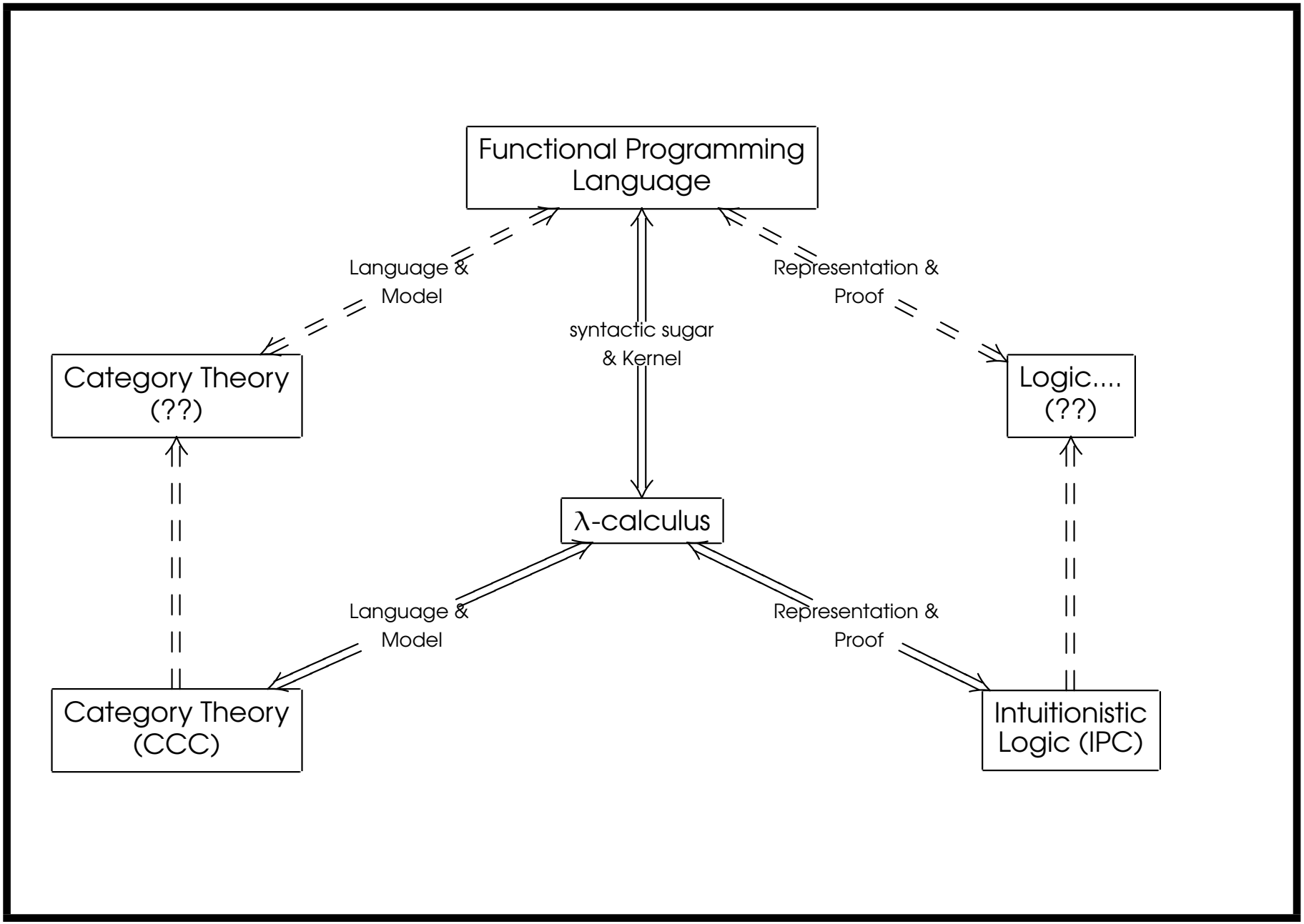
Matching modulo Type isomorphisms

Definition ***(Matching terms)*** *Let* M *be term, and* N *be a close term, we say that* M *is matchable to* N *if and only if there exists a substitution* σ *such that* $\boxed{\sigma(\mathrm{M}) \cong_{\mathrm{d}} \mathrm{N}}$

Definition ***(Matching programs)*** *Let* P *and* Q *be programs, we say that* P *is matchable to* Q *if and only if there exists inputs* $\mathrm{in}_1, ..., \mathrm{in}_n$ *such that* $\boxed{\mathrm{P}\mathrm{in}_1 ... \mathrm{in}_n \cong_{\mathrm{d}} \mathrm{Q}}$

Overview

- PART I
  - **Type Isomorphisms and Program Isomorphisms.**
  - **Program Transformation and Matching.**
  - **Merging ideas.**
- PART II
  - **λ-Calculus setting.**
  - Deciding Type Isomorphisms.
  - References.


Functional Programming Language
Language & Model
syntactic sugar & Kernel
Representation & Proof
Category Theory (??)
Logic.... (??)
λ-calculus
Language & Model
Representation & Proof
Category Theory (CCC)
Intuitionistic Logic (IPC)

λ-Calculus

Types: τ

- Let ι be a *basic type*

$$\tau ::= \iota \mid (\tau \rightarrow \tau) \mid \ldots$$

Terms: Λ

- Let x be a *variable*

$$\Lambda ::= x \mid (\lambda x.\Lambda) \mid (\Lambda\Lambda) \mid \ldots$$

λ-Calculus

Statements, Declaration, and Basis: $t : A$, $x : A$, Γ

- A *statement* is of the form $t : A$, where $t \in \Lambda$, $A \in \tau$.
- A *Declaration* is of the form $x : A$, where $x \in Var$, $A \in \tau$.
- A *Basis* Γ is a finite set of declarations.

λ-Calculus

Type system:

$$\Gamma, x : A \vdash x : A \qquad (Axiom)$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash (\lambda x.t) : (A \to B)} \qquad (\to \text{Elimination})$$

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash (tu) : B} \qquad (\to \text{Introduction})$$

Typable Terms: $\Lambda^{\to}$

$$\Lambda^{\to} := \{t \in \Lambda : \exists \Gamma, A \vdash t : A\}$$

λ-Calculus

Computations: $\triangleright$

- $(\lambda x.t)u \triangleright_\beta t[x := u]$ *(beta-reduction)*.
- $(\lambda x.(tx)) \triangleright_\eta t$, if x is not free in t *(eta-reduction)*.
- ...

Definable isomorphisms: λ-Calculus

Definition *Two type* A *and* B *are definably isomorphic* $(A \cong_d B)$ *iff there exists* λ*-terms* $M : A \to B$ *and* $N : B \to A$ *such that* $M \circ N = \lambda x^B.x$ *and* $N \circ M = \lambda x^A.x$*, the identities of type* A *and* B.

Note: $f \circ g = \lambda x.\lambda f.\lambda g.(f(gx))$

Semantic isomorphisms: Models

Definition *Two type* A *and* B *are isomorphic in a model* $\mathcal{M}$ *if their interpretations are isomorphic en* $\mathcal{M}$ *in a usual sense (i.e. there are in the model invertible functions* f *and* g *between them), we write* $\mathcal{M} \models \mathrm{A} \cong \mathrm{B}$. *Two types are semantically isomorphic* $\mathrm{A} \cong \mathrm{B}$, *if* $\mathcal{M} \models \mathrm{A} \cong \mathrm{B}$ *holds for every model* $\mathcal{M}$ *of the calculus.*

Definable and Semantic Isomorphisms

Theorem *Let* A *and* B *be types, then* $\mathrm{A} \cong \mathrm{B} \Leftrightarrow \mathrm{A} \cong_{\mathrm{d}} \mathrm{B}$.

Proof *(*$\Rightarrow$*) Take the term model. (*$\Leftarrow$*) Straightforward.*

Remark if we have $\mathcal{M} \models \mathrm{A} \cong \mathrm{B}$ for some particular model, it might be the case that $\mathrm{A} \ncong \mathrm{B}$

Isomorphisms and Invertibility

Definition ***(Finite hereditary permutation, f.h.p.)*** *An untyped λ-term M is a f.h.p if and only if*

- $M = \lambda z.z$,
- $M = \lambda z.\lambda x_1 \ldots \lambda x_n.z(Q_1 x_{\pi(1)}) \ldots (Q_n x_{\pi(n)})$ *where* $\pi : n \to n$ *is a permutation and* Q_i *is a f.h.p. for all* $1 \leq i \leq n$

Theorem ***(Dezani-Ciancaglini 1976)*** *Let M be an untyped term possessing $\beta\eta$-normal form then M is invertible if and only if M is a finite hereditary permutation.*

Soundness and Completeness

Definition *We say that an equational theory Th is a sound theory of isomorphism for a calculus if*

$$\forall \mathrm{A}, \mathrm{B} \ \epsilon \ \mathcal{T} : \mathit{Th} \vdash \mathrm{A} = \mathrm{B} \Rightarrow \mathrm{A} \cong \mathrm{B}$$

Respectively, an equational theory Th is a complete theory of isomorphism for a calculus

$$\forall \mathrm{A}, \mathrm{B} \ \epsilon \ \mathcal{T} : \mathrm{A} \cong \mathrm{B} \Rightarrow \mathit{Th} \vdash \mathrm{A} = \mathrm{B}$$

Where $\mathcal{T}$ is an arbitrary type system.

Example: Swap

Theorem ***(Soundness)***

$$\forall A, B \in \mathcal{T} : \mathit{Swap} \vdash A = B \Rightarrow A \cong B$$

Proof Recall $A \cong B \Leftrightarrow A \cong_d B$, then it is enough to show $\mathit{Swap} \vdash A = B \Rightarrow A \cong_d B$, observe $\lambda x^{A \to (B \to C)}.\lambda y^{B \to C}.\lambda z^{A}.xzy$ is invertible and witness the equation $A \to (B \to C) = B \to (A \to C)$.

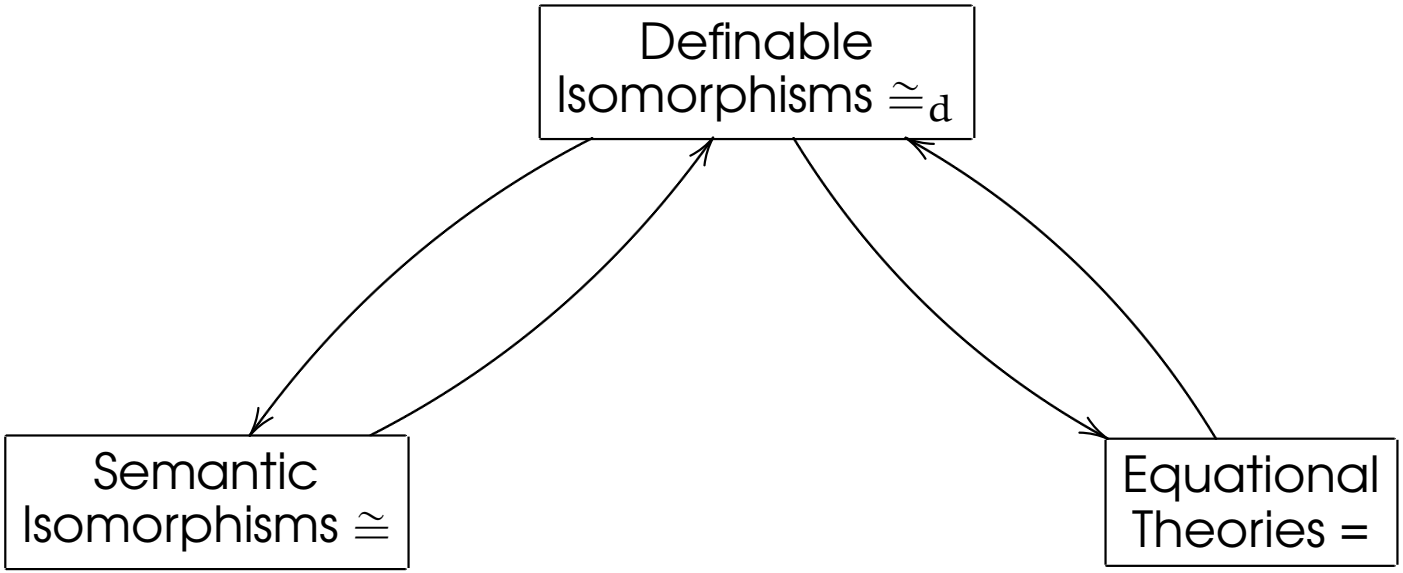
Example: Swap

Theorem ***(Completeness)***

$$\forall A, B \in \mathcal{T} : A \cong B \Rightarrow \mathit{Swap} \vdash A = B$$

Proof It is enough to show $A \cong_d B \Rightarrow \mathit{Swap} \vdash A = B$, suppose that $M : A \cong_d B : N$ then proceed by structural induction on M, where M is a f.h.p.

We have...for Swap



Overview

- PART I
  - **Type Isomorphisms and Program Isomorphisms.**
  - **Program Transformation and Matching.**
  - **Merging ideas.**
- PART II
  - **λ-Calculus setting.**
  - **Deciding Type Isomorphisms.**
  - References.

Type Isomorphism:

Equational Theories.

(A0) $A \to (B \to C) = B \to (A \to C)$ (Swap)

(A1) $A \times B = B \times A$

(A2) $A \times (B \times C) = (A \times B) \times C$

(A3) $(A \times B) \to C = A \to (B \to C)$

(A4) $C \to (A \times B) = (C \to A) \times (C \to B)$

(A5) $A \times T = A$

(A6) $A \to T = T$

(A7) $T \to A = A$

Type Systems:

Type System	Axioms
Swap	A0
Product Types	A1, A2 & (*)
Linear Types	A1, A2, A3 & (*)
First Order Types	A1, A2, A3, A4 & (*)

Where (*) are A5 - A7.

Decidability

Via an appropriate Rewriting System; **confluent**, **strong normalizing** up to commutative product + sorting of primitive types.

(R2) $A(BC) > (AB)C$

(R3) $(C^B)^A > C^{AB}$

(R4) $(AB)^C > (A^C)(B^C)$

(R5) $A1 > A$

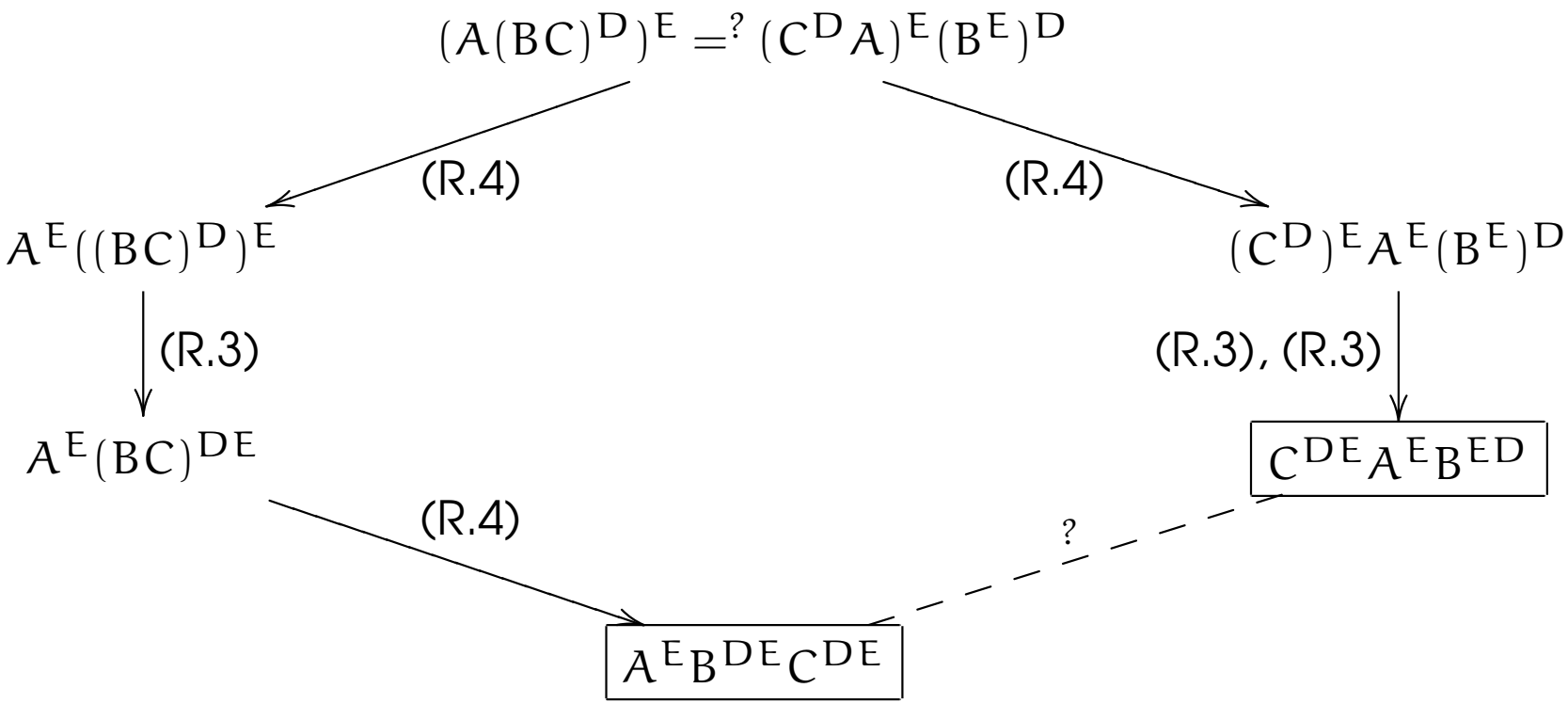
(R5)' $1A > A$

(R6) $A^1 > A$

(R7) $1^A > 1$

Observe: (R5) - (R7) eliminates unit types.

Example:



Sorting products: $C^{DE}A^E B^{ED} \rightarrow A^E B^{DE} C^{DE}$

Complexity:

Theorem *(Zibin, Gil, Considine'03)* *Type isomorphism can be decided in* $O(n \log^2(n))$ *time and* $O(n)$ *space, where* n *is the size of the input.*

Overview

- **Type isomorphisms and Program isomorphisms.**
- **Program transformation and Matching.**
- **Merging ideas.**
- **References.**

References:

Bruce K., Di Cosmo R., Longo G.
Provable Isomorphism of Types, *Mathematical Structures in Computer Science*, **2**, 231 - 247, 1991.

Dezani-Ciancaglini M.
Characterization of Normal Forms Possessing Inverse in the $\lambda_{\beta\eta}$-Calculus, *Theoretical Computer Science*, **2**, 323 - 337, 1976.

Di Cosmo R.
Isomorphism of Types: from λ-calculus to information retrieval and language design, *Birkhäuser*, 1995.

References:

Rittri M.
Retrieving Library Functions by Unifying Types Modulo Linear Type Isomorphism, *Theoretical Informatics and Applications*, **27**, 71 -89, 1993.

Soloviev S.
The Category of Finite Sets and Cartesian Closed Categories, *Journal of Soviet Mathematics*, **22(3)**, 1387 - 1400, 1983.

Zibin Y., Gil J., Considine J.
Efficient Algorithm for Isomorphism of Simple Types, *Proceedings POPL'03 in ACM SIGPLAN*, **38**, 160 - 171, 2003.

References:

Huet G. and Lang B.,
Proving and Applying Program Transformations Expressed with Second Order Patterns, *Acta Informatica*, 11:31–55, 1978.