Type Isomorphisms and Programs Isomorphisms

Carlos C. Martínez

Department of Mathematics and Computer Science,

Wesleyan University

cmartinez@wesleyan.edu

January, 2006

Overview:

- Motivation and Definitions.
- Matching and Program Transformation.
- Matching Modulo Type Isomorphisms.
- Final Remarks.

Example 1:

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

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

f1: $\mathtt{int}*\mathtt{int} \rightarrow \mathtt{int}$ = ⟨**fun**⟩

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

f2: $\mathtt{int}*\mathtt{int} \rightarrow \mathtt{int}$ = ⟨**fun**⟩

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

f3: $\mathtt{int} \rightarrow \mathtt{int} \rightarrow \mathtt{int}$ = ⟨**fun**⟩

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

f4: $\mathtt{int} \rightarrow \mathtt{int} \rightarrow \mathtt{int}$ = ⟨**fun**⟩

Example 1:

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

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

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

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

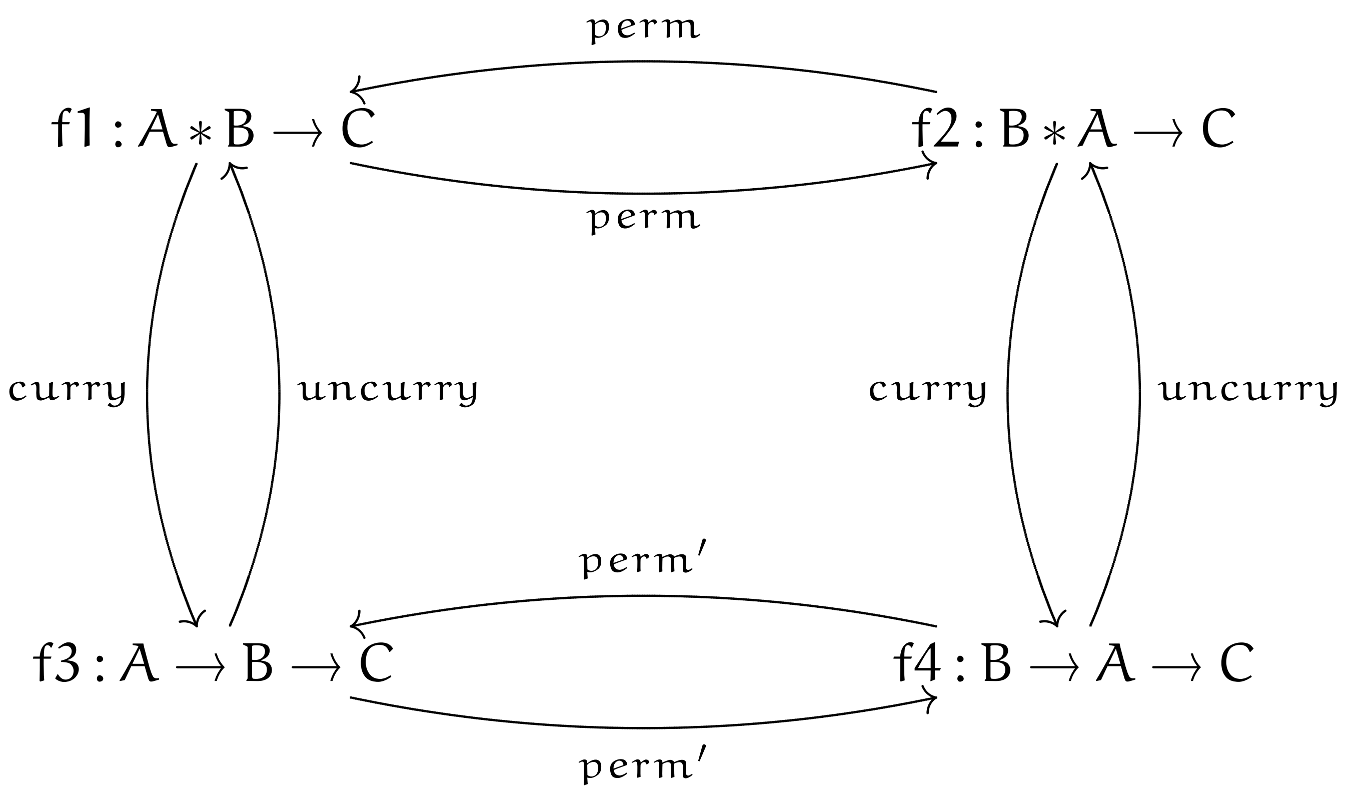
#**Let** perm f $xy = \text{fun}\,yx$;;

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

#**Let** perm′ f $(x,y) = \text{fun}(y,x)$;;

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

Example 1:


perm
f1 : A ∗ B → C
f2 : B ∗ A → C
perm
curry
uncurry
curry
uncurry
perm′
f3 : A → B → C
f4 : B → A → C
perm′

Definable isomorphisms:

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

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

Example 2:

#**Let rec** fact $n =$ **if** $(n = 0)$ **then** 1
else $n*$ fact$(n-1)$;;
fact: $\mathtt{int} \to \mathtt{int}$ = $\langle$**fun**$\rangle$

#**Let rec** rev $l =$ **if** $(l = [])$ **then** []
else $\mathtt{append}'([\mathtt{head}(l)],$ rev$(\mathtt{tail}(l)))$;;
rev: $\mathtt{int}^* \to \mathtt{int}^*$ = $\langle$**fun**$\rangle$

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

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

iter: $\underline{(A \times A \to A)} \to A \to (A \to A)$= $\langle$**fun**$\rangle$

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

Remark: $\mathtt{append}'$ is an isomorphic version of the usual $\mathtt{append}$

Example 2:

Thus:

σ_1(iter $*n$) = fact n $\qquad$ σ_2(iter $\mathtt{append}'\ l$) = rev l

$$\sigma_1 \begin{cases} a \leftarrow \lambda x.(x == 0); \\ b \leftarrow \lambda x.1; \\ c \leftarrow \lambda x.x; \\ d \leftarrow \lambda x.(x - 1); \end{cases} \qquad \sigma_2 \begin{cases} a \leftarrow \lambda x.(x == []); \\ b \leftarrow \lambda x.0; \\ c \leftarrow \lambda x.[(\mathtt{head}\ x)]; \\ d \leftarrow \lambda x.(\mathtt{tail}\ x); \end{cases}$$

Matching Problem

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 $\sigma(M) = N$*

Definition ***(Matching Programs)*** *Let P and Q be programs, we say that P is matchable to Q if and only if there exists inputs $in_1, ..., in_n$ such that $Pin_1 ... in_n = Q$*

Example 2:

Optimization: Consider the following "Loop" function

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

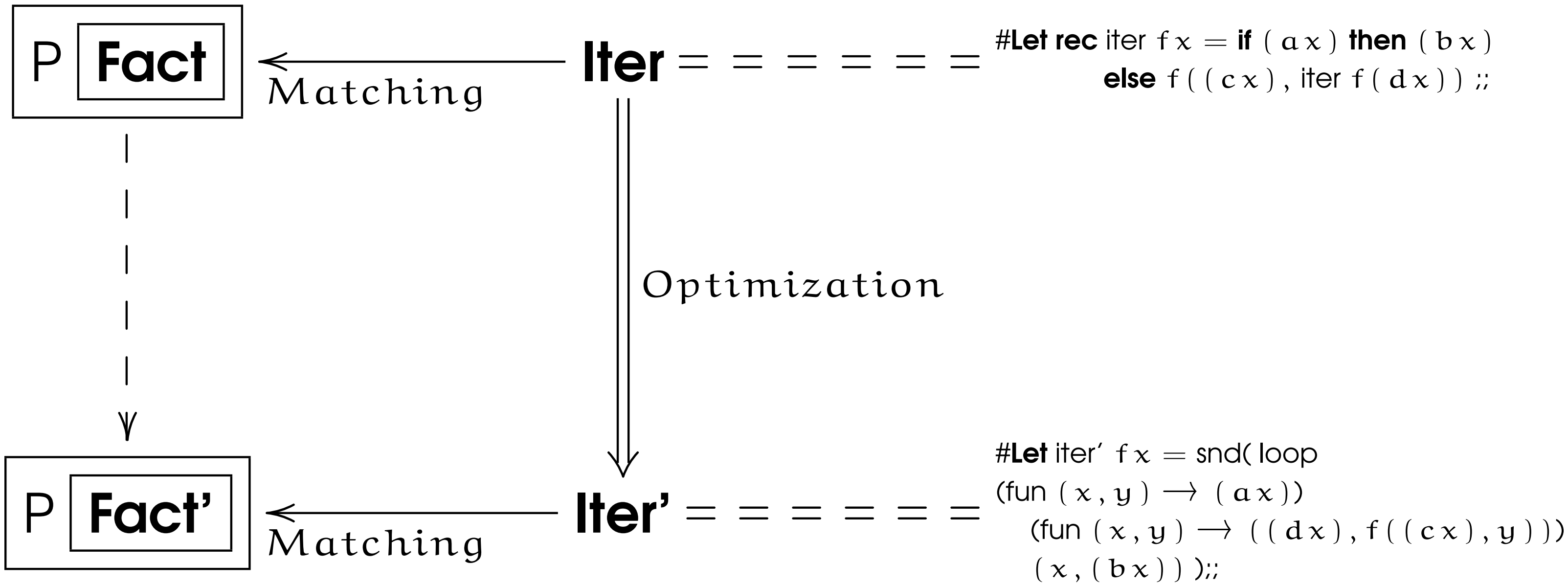
loop: $(A \to \mathrm{Bool}) \to (A \to A) \to (A \to A)$= ⟨**fun**⟩

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

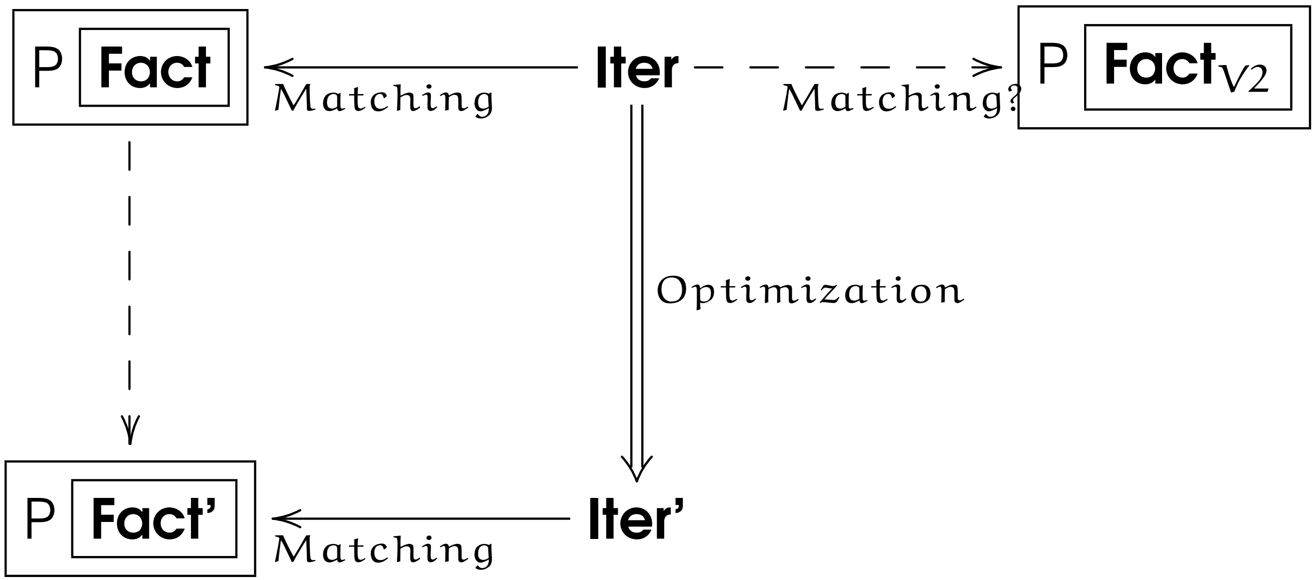
#**Let** fact′ $n =$
snd(loop (fun $(m, y) \to m = 0$)
(fun $(m, y) \to (m - 1, m * y)$)
$(n, 1)$);;
fact′: $int \to int$ = ⟨**fun**⟩

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

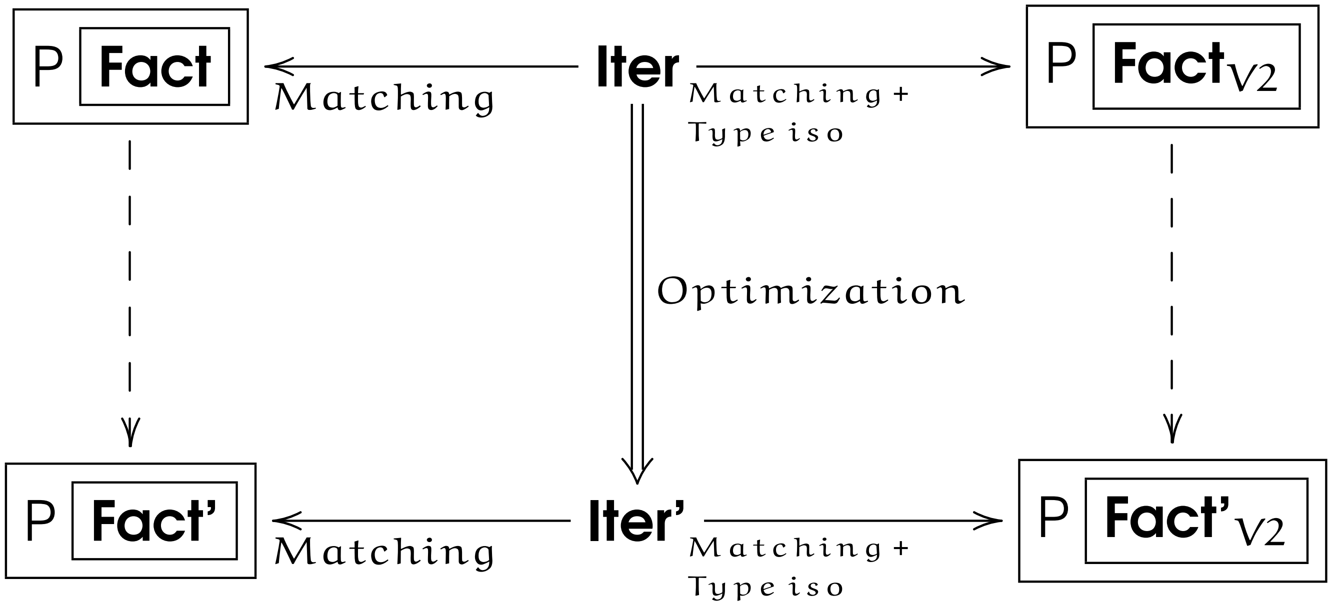
Example 2:


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)));;

Example 3:


P Fact
Matching
Iter
Matching?
P $\mathbf{Fact}_{V2}$
Optimization
P Fact'
Matching
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* $\boxed{\sigma(M) \simeq N}$

Definition ***(Matching Programs)*** *Let* P *and* Q *be programs, we say that* P *is matchable to* Q *if and only if there exists inputs* $in_1, ..., in_n$ *such that* $\boxed{P in_1 ... in_n \simeq Q}$

Final Remarks:

- Decidability.
- Efficiency.
- Extensions.

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.