On the expressive power of types

Sam Lindley

The University of Edinburgh

TYPES 2022

What do we mean by expressive power?

What do we mean by expressive power?

Some possible answers:

- Computability

What do we mean by expressive power?

Some possible answers:

- Computability
- Algorithmic complexity

What do we mean by expressive power?

Some possible answers:

- Computability
- Algorithmic complexity
- **Macro expressiveness**

Matthias Felleisen

[Felleisen, 1990, "On the expressive power of programming languages"]

Quiz

Quiz

Can function types encode product types?

Quiz

Can function types encode product types?

Can positive iso-recursive types encode positive equi-recursive types?

Quiz

Can function types encode product types?

Can positive iso-recursive types encode positive equi-recursive types?

Can existential types encode universal types?

Quiz

Can function types encode product types?

Can positive iso-recursive types encode positive equi-recursive types?

Can existential types encode universal types?

Can simple-typed lambda calculus encode System F?

Quiz

Can function types encode product types?

Can positive iso-recursive types encode positive equi-recursive types?

Can existential types encode universal types?

Can simple-typed lambda calculus encode System F?

Can row polymorphism encode row subtyping?

Part I

Products

Motivation

Alonzo Church

The standard Church encoding for a pair can only be ascribed a type in simply-typed lambda calculus if both components of the pair have the same type.

Motivation

Alonzo Church

The standard Church encoding for a pair can only be ascribed a type in simply-typed lambda calculus if both components of the pair have the same type.

Do alternative simply-typed encodings exist for heterogeneous pairs?

Call-by-name CPS

$$
\begin{aligned}
\mathcal{N}[\![A]\!] &= (A^\star \to R) \to R \\
X^\star &= X \\
(A \times B)^\star &= \mathcal{N}[\![A]\!] \times \mathcal{N}[\![B]\!] \\
(A \to B)^\star &= \mathcal{N}[\![A]\!] \to \mathcal{N}[\![B]\!]
\end{aligned}
$$

Call-by-name CPS

$$\begin{aligned}
\mathcal{N}[\![X]\!] &= (X \to R) \to R \\
\mathcal{N}[\![A \to B]\!] &= ((\mathcal{N}[\![A]\!] \to \mathcal{N}[\![B]\!]) \to R) \to R \\
\mathcal{N}[\![A \times B]\!] &= ((\mathcal{N}[\![A]\!] \times \mathcal{N}[\![B]\!]) \to R) \to R
\end{aligned}$$

Call-by-name CPS

$$
\begin{array}{rcl}
\mathcal{N}[\![X]\!] & = & (X \to R) \to R \\
\mathcal{N}[\![A \to B]\!] & = & ((\mathcal{N}[\![A]\!] \to \mathcal{N}[\![B]\!]) \to R) \to R \\
\mathcal{N}[\![A \times B]\!] & = & ((\mathcal{N}[\![A]\!] \times \mathcal{N}[\![B]\!]) \to R) \to R
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{N}[\![x]\!] & = & \lambda k.x\ k \\
\mathcal{N}[\![\lambda x.M]\!] & = & \lambda k.k\ (\lambda x.\mathcal{N}[\![M]\!]) \\
\mathcal{N}[\![M\ N]\!] & = & \lambda k.\mathcal{N}[\![M]\!]\ (\lambda f.f\ \mathcal{N}[\![N]\!]\ k) \\
\mathcal{N}[\![\mathsf{pair}\ M\ N]\!] & = & \lambda k.k\ (\mathsf{pair}\ \mathcal{N}[\![M]\!]\ \mathcal{N}[\![N]\!]) \\
\mathcal{N}[\![\mathsf{fst}\ M]\!] & = & \lambda k.\mathcal{N}[\![M]\!]\ (\lambda p.(\mathsf{fst}\ p)\ k) \\
\mathcal{N}[\![\mathsf{snd}\ M]\!] & = & \lambda k.\mathcal{N}[\![M]\!]\ (\lambda p.(\mathsf{snd}\ p)\ k)
\end{array}
$$

Call-by-name CPS

Products are encodable via a curried **global** CPS translation

$$
\begin{array}{rcl}
\mathcal{C}[\![X]\!] & = & (X \to R) \to R \\
\mathcal{C}[\![A \to B]\!] & = & ((\mathcal{C}[\![A]\!] \to \mathcal{C}[\![B]\!]) \to R) \to R \\
\mathcal{C}[\![A \times B]\!] & = & (\mathcal{C}[\![A]\!] \to \mathcal{C}[\![B]\!] \to R) \to R
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{C}[\![x]\!] & = & \lambda k.x\ k \\
\mathcal{C}[\![\lambda x.M]\!] & = & \lambda k.k\ (\lambda x.\mathcal{C}[\![M]\!]) \\
\mathcal{C}[\![M\ N]\!] & = & \lambda k.\mathcal{C}[\![M]\!]\ (\lambda f.f\ \mathcal{C}[\![N]\!]\ k) \\
\mathcal{C}[\![\mathsf{pair}\ M\ N]\!] & = & \lambda k.k\ \mathcal{C}[\![M]\!]\ \mathcal{C}[\![N]\!] \\
\mathcal{C}[\![\mathsf{fst}\ M]\!] & = & \lambda k.\mathcal{C}[\![M]\!]\ (\lambda x.\lambda y.x\ k) \\
\mathcal{C}[\![\mathsf{snd}\ M]\!] & = & \lambda k.\mathcal{C}[\![M]\!]\ (\lambda x.\lambda y.y\ k)
\end{array}
$$

Call-by-name CPS

Products are encodable via a curried **global** CPS translation

$$
\begin{array}{rcl}
\mathcal{C}[\![X]\!] & = & (X \to R) \to R \\
\mathcal{C}[\![A \to B]\!] & = & ((\mathcal{C}[\![A]\!] \to \mathcal{C}[\![B]\!]) \to R) \to R \\
\mathcal{C}[\![A \times B]\!] & = & (\mathcal{C}[\![A]\!] \to \mathcal{C}[\![B]\!] \to R) \to R
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{C}[\![x]\!] & = & \lambda k.x\ k \\
\mathcal{C}[\![\lambda x.M]\!] & = & \lambda k.k\ (\lambda x.\mathcal{C}[\![M]\!]) \\
\mathcal{C}[\![M\ N]\!] & = & \lambda k.\mathcal{C}[\![M]\!]\ (\lambda f.f\ \mathcal{C}[\![N]\!]\ k) \\
\mathcal{C}[\![\mathsf{pair}\ M\ N]\!] & = & \lambda k.k\ \mathcal{C}[\![M]\!]\ \mathcal{C}[\![N]\!] \\
\mathcal{C}[\![\mathsf{fst}\ M]\!] & = & \lambda k.\mathcal{C}[\![M]\!]\ (\lambda x.\lambda y.x\ k) \\
\mathcal{C}[\![\mathsf{snd}\ M]\!] & = & \lambda k.\mathcal{C}[\![M]\!]\ (\lambda x.\lambda y.y\ k)
\end{array}
$$

What about a **local** encoding?

Localising CPS

Untyped

$$
\begin{aligned}
\mathcal{U}[\![\mathsf{pair}\ M\ N]\!] &= \lambda s.s\,\mathcal{U}[\![M]\!]\,\mathcal{U}[\![N]\!] \\
\mathcal{U}[\![\mathsf{fst}\ M]\!] &= \mathcal{U}[\![M]\!]\ (\lambda x.\lambda y.x) \\
\mathcal{U}[\![\mathsf{snd}\ M]\!] &= \mathcal{U}[\![M]\!]\ (\lambda x.\lambda y.y)
\end{aligned}
$$

Localising CPS

Untyped

$$
\begin{aligned}
\mathcal{U}[\![\mathsf{pair}\ M\ N]\!] &= \lambda s.s\,\mathcal{U}[\![M]\!]\,\mathcal{U}[\![N]\!] \\
\mathcal{U}[\![\mathsf{fst}\ M]\!] &= \mathcal{U}[\![M]\!]\ (\lambda x.\lambda y.x) \\
\mathcal{U}[\![\mathsf{snd}\ M]\!] &= \mathcal{U}[\![M]\!]\ (\lambda x.\lambda y.y)
\end{aligned}
$$

Simply typed — homogeneous products

$$
\begin{aligned}
\mathcal{H}[\![A \times A]\!] &= (\mathcal{H}[\![A]\!] \to \mathcal{H}[\![A]\!] \to \mathcal{H}[\![A]\!]) \to \mathcal{H}[\![A]\!] \\
\mathcal{H}[\![\mathsf{pair}\ M^A\ N^A]\!] &= \lambda s^{\mathcal{H}[\![A]\!] \to \mathcal{H}[\![A]\!] \to \mathcal{H}[\![A]\!]}.s\ \mathcal{H}[\![M]\!]\ \mathcal{H}[\![N]\!] \\
\mathcal{H}[\![\mathsf{fst}\ M^{A \times A}]\!] &= \mathcal{H}[\![M]\!]\ (\lambda x^{\mathcal{H}[\![A]\!]}.\lambda y^{\mathcal{H}[\![A]\!]}.x) \\
\mathcal{H}[\![\mathsf{snd}\ M^{A \times A}]\!] &= \mathcal{H}[\![M]\!]\ (\lambda x^{\mathcal{H}[\![A]\!]}.\lambda y^{\mathcal{H}[\![A]\!]}.y)
\end{aligned}
$$

Localising CPS

Untyped

$$\begin{array}{rcl}\mathcal{U}[\![\mathsf{pair}\ M\ N]\!] & = & \lambda s.s\,\mathcal{U}[\![M]\!]\,\mathcal{U}[\![N]\!]\\ \mathcal{U}[\![\mathsf{fst}\ M]\!] & = & \mathcal{U}[\![M]\!]\ (\lambda x.\lambda y.x)\\ \mathcal{U}[\![\mathsf{snd}\ M]\!] & = & \mathcal{U}[\![M]\!]\ (\lambda x.\lambda y.y)\end{array}$$

Simply typed — homogeneous products

$$\begin{array}{rcl}\mathcal{H}[\![A\times A]\!] & = & (\mathcal{H}[\![A]\!]\to\mathcal{H}[\![A]\!]\to\mathcal{H}[\![A]\!])\to\mathcal{H}[\![A]\!]\\ \mathcal{H}[\![\mathsf{pair}\ M^A\ N^A]\!] & = & \lambda s^{\mathcal{H}[\![A]\!]\to\mathcal{H}[\![A]\!]\to\mathcal{H}[\![A]\!]}.s\ \mathcal{H}[\![M]\!]\ \mathcal{H}[\![N]\!]\\ \mathcal{H}[\![\mathsf{fst}\ M^{A\times A}]\!] & = & \mathcal{H}[\![M]\!]\ (\lambda x^{\mathcal{H}[\![A]\!]}.\lambda y^{\mathcal{H}[\![A]\!]}.x)\\ \mathcal{H}[\![\mathsf{snd}\ M^{A\times A}]\!] & = & \mathcal{H}[\![M]\!]\ (\lambda x^{\mathcal{H}[\![A]\!]}.\lambda y^{\mathcal{H}[\![A]\!]}.y)\end{array}$$

Polymorphic

$$\begin{array}{rcl}\mathcal{F}[\![A\times B]\!] & = & \forall Z.(\mathcal{F}[\![A]\!]\to\mathcal{F}[\![B]\!]\to Z)\to Z\\ \mathcal{F}[\![\mathsf{pair}_{A,B}\ M\ N]\!] & = & \Lambda Z.\lambda s^{\mathcal{F}[\![A]\!]\to\mathcal{F}[\![B]\!]\to Z}.s\ \mathcal{F}[\![M]\!]\ \mathcal{F}[\![N]\!]\\ \mathcal{F}[\![\mathsf{fst}_{A,B}\ M]\!] & = & \mathcal{F}[\![M]\!]\ \mathcal{F}[\![A]\!]\ (\lambda x^{\mathcal{F}[\![A]\!]}.\lambda y^{\mathcal{F}[\![B]\!]}.x)\\ \mathcal{F}[\![\mathsf{snd}_{A,B}\ M]\!] & = & \mathcal{F}[\![M]\!]\ \mathcal{F}[\![B]\!]\ (\lambda x^{\mathcal{F}[\![A]\!]}.\lambda y^{\mathcal{F}[\![B]\!]}.y)\end{array}$$

No local encoding of $X \times Y$

We seek β-normal forms $\mathsf{fst}_{X,Y}$ and $\mathsf{snd}_{X,Y}$ such that:

$$
\begin{aligned}
[\![p : X \times Y \vdash \mathsf{fst}\ p : X]\!] &= p : [\![X \times Y]\!] \vdash \mathsf{fst}_{X,Y} : X \\
[\![p : X \times Y \vdash \mathsf{snd}\ p : Y]\!] &= p : [\![X \times Y]\!] \vdash \mathsf{snd}_{X,Y} : Y
\end{aligned}
$$

No local encoding of $X \times Y$

We seek β-normal forms $\mathsf{fst}_{X,Y}$ and $\mathsf{snd}_{X,Y}$ such that:

$$\begin{array}{rcl} [\![p : X \times Y \vdash \mathsf{fst}\ p : X]\!] & = & p : [\![X \times Y]\!] \vdash \mathsf{fst}_{X,Y} : X \\ [\![p : X \times Y \vdash \mathsf{snd}\ p : Y]\!] & = & p : [\![X \times Y]\!] \vdash \mathsf{snd}_{X,Y} : Y \end{array}$$

So we must have $m, n, M_1, ..., M_m, N_1, ..., N_n$ such that:

$$\begin{array}{rcl} \mathsf{fst}_{X,Y} & = & p\ M_1\ \ldots\ M_m \\ \mathsf{snd}_{X,Y} & = & p\ N_1\ \ldots\ N_n \end{array}$$

No local encoding of $X \times Y$

We seek β-normal forms $\mathsf{fst}_{X,Y}$ and $\mathsf{snd}_{X,Y}$ such that:

$$
\begin{aligned}
[\![p : X \times Y \vdash \mathsf{fst}\ p : X]\!] &= p : [\![X \times Y]\!] \vdash \mathsf{fst}_{X,Y} : X \\
[\![p : X \times Y \vdash \mathsf{snd}\ p : Y]\!] &= p : [\![X \times Y]\!] \vdash \mathsf{snd}_{X,Y} : Y
\end{aligned}
$$

So we must have $m, n, M_1, \ldots, M_m, N_1, \ldots, N_n$ such that:

$$
\begin{aligned}
\mathsf{fst}_{X,Y} &= p\ M_1\ \ldots\ M_m \\
\mathsf{snd}_{X,Y} &= p\ N_1\ \ldots\ N_n
\end{aligned}
$$

The typing rule for application means that we also have

$$
A_1 \to \cdots \to A_m \to X \quad = \quad [\![X \times Y]\!] \quad = \quad B_1 \to \cdots \to B_n \to Y
$$

where

$$
\begin{aligned}
&(p : [\![X \times Y]\!] \vdash M_i : A_i)_{1 \le i \le m} \\
&(p : [\![X \times Y]\!] \vdash N_j : B_i)_{1 \le j \le n} j
\end{aligned}
$$

No local encoding of $X \times Y$

We seek β-normal forms $\mathsf{fst}_{X,Y}$ and $\mathsf{snd}_{X,Y}$ such that:

$$\begin{array}{rcl} [\![p : X \times Y \vdash \mathsf{fst}\ p : X]\!] & = & p : [\![X \times Y]\!] \vdash \mathsf{fst}_{X,Y} : X \\ [\![p : X \times Y \vdash \mathsf{snd}\ p : Y]\!] & = & p : [\![X \times Y]\!] \vdash \mathsf{snd}_{X,Y} : Y \end{array}$$

So we must have $m, n, M_1, ..., M_m, N_1, ..., N_n$ such that:

$$\begin{array}{rcl} \mathsf{fst}_{X,Y} & = & p\ M_1\ \ldots\ M_m \\ \mathsf{snd}_{X,Y} & = & p\ N_1\ \ldots\ N_n \end{array}$$

The typing rule for application means that we also have

$$A_1 \to \cdots \to A_m \to X \quad = \quad [\![X \times Y]\!] \quad = \quad B_1 \to \cdots \to B_n \to Y$$

where

$$\begin{array}{c} (p : [\![X \times Y]\!] \vdash M_i : A_i)_{1 \le i \le m} \\ (p : [\![X \times Y]\!] \vdash N_j : B_i)_{1 \le j \le n} j \end{array}$$

But these equations could only hold if X and Y were the same type!

Hang on a minute!

John Longley

Dag Normann

Oleg Kiselyov

Type-indexed local encodings of products are well-known in PCF and System T. Examples:

- [Longley and Normann, 2015]
- [Kiselyov, 2021]
  `http://okmij.org/ftp/Computation/simple-encodings.html#product`

How do we reconcile the existence of such encodings with the non-existence result?

No local encoding of $X \times (X \to X)$

Consider $X \times (X \to X)$. We seek β-normal forms $\mathsf{fst}_{X,X\to X}$ and $\mathsf{snd}_{X,X\to X}$ such that:

$$
\begin{aligned}
[\![p : X \times (X \to X) \vdash \mathsf{fst}\ p : X]\!] &= p : [\![X \times (X \to X)]\!] \vdash \mathsf{fst}_{X,X\to X} : X \\
[\![p : X \times (X \to X) \vdash \mathsf{snd}\ p : X \to X]\!] &= p : [\![X \times (X \to X)]\!] \vdash \mathsf{snd}_{X,X\to X} : X \to X
\end{aligned}
$$

No local encoding of $X \times (X \to X)$

Consider $X \times (X \to X)$. We seek β-normal forms $\mathsf{fst}_{X,X\to X}$ and $\mathsf{snd}_{X,X\to X}$ such that:

$$
\begin{array}{rcl}
[\![p : X \times (X \to X) \vdash \mathsf{fst}\ p : X]\!] & = & p : [\![X \times (X \to X)]\!] \vdash \mathsf{fst}_{X,X\to X} : X \\
[\![p : X \times (X \to X) \vdash \mathsf{snd}\ p : X \to X]\!] & = & p : [\![X \times (X \to X)]\!] \vdash \mathsf{snd}_{X,X\to X} : X \to X
\end{array}
$$

As before, $\mathsf{fst}_{X,X\to X}$ must be of the form

$$p\ M_1\ \ldots\ M_m$$

and hence:

$$[\![X \times (X \to X)]\!] = A_1 \to \cdots \to A_m \to X$$

No local encoding of $X \times (X \to X)$ (continued)

Two choices for $\mathsf{snd}_{X, X \to X}$:

No local encoding of $X \times (X \to X)$ (continued)

Two choices for $\mathsf{snd}_{X, X \to X}$:

1. $p\ N_1\ \ldots\ N_{m-1}$

No local encoding of $X \times (X \to X)$ (continued)

Two choices for $\mathsf{snd}_{X, X \to X}$:

1. $p\ N_1\ \ldots\ N_{m-1}$

$$\begin{aligned} \Longrightarrow\ A_m = X \text{ and } M_m &= p\ M'_1\ \ldots\ M'_m \\ M'_m &= p\ M''_1\ \ldots\ M''_m \\ &\ldots \end{aligned}$$

No local encoding of $X \times (X \to X)$ (continued)

Two choices for $\mathsf{snd}_{X,X \to X}$:

1. $p\ N_1\ \ldots\ N_{m-1}$

$$\begin{aligned} \Longrightarrow\ A_m = X \text{ and } M_m &= p\ M'_1\ \ldots\ M'_m \\ M'_m &= p\ M''_1\ \ldots\ M''_m \\ &\ldots \end{aligned}$$

No finite such snd can exist.

No local encoding of $X \times (X \to X)$ (continued)

Two choices for $\mathsf{snd}_{X,X\to X}$:

1. $p\ N_1\ \ldots\ N_{m-1}$
   $$\begin{aligned} \Longrightarrow\ A_m = X \text{ and } & M_m = p\ M'_1\ \ldots\ M'_m \\ & M'_m = p\ M''_1\ \ldots\ M''_m \\ & \ldots \end{aligned}$$
   No finite such snd can exist.
2. $\lambda z.N'$

No local encoding of $X \times (X \to X)$ (continued)

Two choices for $\mathsf{snd}_{X,X \to X}$:

1. $p\ N_1\ \ldots\ N_{m-1}$

$$\Longrightarrow\ A_m = X \text{ and } \begin{aligned} M_m &= p\ M'_1\ \ldots\ M'_m \\ M'_m &= p\ M''_1\ \ldots\ M''_m \\ &\ldots \end{aligned}$$

No finite such snd can exist.

2. $\lambda z.N'$

$$\Longrightarrow\ [\![\mathsf{snd}\ (\mathsf{pair}\ x\ y)]\!] = \lambda z.N'[[\![\mathsf{pair}\ x\ y]\!]/p]$$

No local encoding of $X \times (X \to X)$ (continued)

Two choices for $\mathsf{snd}_{X, X \to X}$:

1. $p\ N_1\ \ldots\ N_{m-1}$

$$\begin{aligned} \implies\ A_m = X \text{ and } & M_m = p\ M'_1\ \ldots\ M'_m \\ & M'_m = p\ M''_1\ \ldots\ M''_m \\ & \ldots \end{aligned}$$

   No finite such snd can exist.

2. $\lambda z.N'$

$$\implies [\![\mathsf{snd}\ (\mathsf{pair}\ x\ y)]\!] = \lambda z.N'[[\![\mathsf{pair}\ x\ y]\!]/p]$$

   No lambda abstraction can be β-converted to y.

Local encoding of $X \times (X \to X)$ with η

$$
\begin{array}{rcl}
\mathcal{E}[\![X \times (X \to X)]\!] & = & (X \to (X \to X) \to X) \to X \\
\mathcal{E}[\![\mathsf{pair}\ M^{X}\ N^{X \to X}]\!] & = & \lambda f.f\ \mathcal{E}[\![M]\!]\ \mathcal{E}[\![N]\!] \\
\mathcal{E}[\![\mathsf{fst}\ M^{X \times (X \to X)}]\!] & = & \mathcal{E}[\![M]\!]\ (\lambda x\ y.x) \\
\mathcal{E}[\![\mathsf{snd}\ M^{X \times (X \to X)}]\!] & = & \lambda z.\mathcal{E}[\![M]\!]\ (\lambda x\ y.y\ z)
\end{array}
$$

Local encoding of $X \times (X \to X)$ with η

$$
\begin{array}{rcl}
\mathcal{E}[\![X \times (X \to X)]\!] & = & (X \to (X \to X) \to X) \to X \\
\mathcal{E}[\![\mathsf{pair}\ M^X\ N^{X \to X}]\!] & = & \lambda f.f\ \mathcal{E}[\![M]\!]\ \mathcal{E}[\![N]\!] \\
\mathcal{E}[\![\mathsf{fst}\ M^{X \times (X \to X)}]\!] & = & \mathcal{E}[\![M]\!]\ (\lambda x\ y.x) \\
\mathcal{E}[\![\mathsf{snd}\ M^{X \times (X \to X)}]\!] & = & \lambda z.\mathcal{E}[\![M]\!]\ (\lambda x\ y.y\ z)
\end{array}
$$

Now we have

$$
\begin{array}{l}
\mathcal{E}[\![\mathsf{fst}\ (\mathcal{E}[\![\mathsf{pair}\ x\ y]\!])]\!] \sim_\beta x \\
\mathcal{E}[\![\mathsf{snd}\ (\mathcal{E}[\![\mathsf{pair}\ x\ y]\!])]\!] \sim_\beta \lambda z.y\ z \sim_\eta y
\end{array}
$$

Local encoding of $A \times B$ with η and a single base type X

$$
\begin{array}{rcl}
\mathcal{E}[\![A \times B]\!] & = & (\mathcal{E}[\![A]\!] \to \mathcal{E}[\![B]\!] \to X) \to X \\
\mathcal{E}[\![\mathsf{pair}\ M\ N]\!] & = & \lambda f.f\ \mathcal{E}[\![M]\!]\ \mathcal{E}[\![N]\!] \\
\mathcal{E}[\![\mathsf{fst}\ M^{A_1 \to \ldots A_n \to X, B}]\!] & = & \lambda z_1\ \ldots\ z_n.\mathcal{E}[\![M]\!]\ (\lambda x\ y.x\ z_1\ \ldots\ z_n) \\
\mathcal{E}[\![\mathsf{snd}\ M^{A, B_1 \to \ldots B_n \to X}]\!] & = & \lambda z_1\ \ldots\ z_n.\mathcal{E}[\![M]\!]\ (\lambda x\ y.y\ z_1\ \ldots\ z_n)
\end{array}
$$

Local encoding of $A \times B$ with η and a single base type X

$$
\begin{array}{rcl}
\mathcal{E}[\![A \times B]\!] & = & (\mathcal{E}[\![A]\!] \to \mathcal{E}[\![B]\!] \to X) \to X \\
\mathcal{E}[\![\mathsf{pair}\ M\ N]\!] & = & \lambda f.f\ \mathcal{E}[\![M]\!]\ \mathcal{E}[\![N]\!] \\
\mathcal{E}[\![\mathsf{fst}\ M^{A_1 \to \ldots A_n \to X, B}]\!] & = & \lambda z_1\ \ldots\ z_n.\mathcal{E}[\![M]\!]\ (\lambda x\ y.x\ z_1\ \ldots\ z_n) \\
\mathcal{E}[\![\mathsf{snd}\ M^{A, B_1 \to \ldots B_n \to X}]\!] & = & \lambda z_1\ \ldots\ z_n.\mathcal{E}[\![M]\!]\ (\lambda x\ y.y\ z_1\ \ldots\ z_n)
\end{array}
$$

This is a **type-indexed** local encoding.

Can function types encode product types?

Can function types encode product types?

It depends...

Can function types encode product types?

It depends...

- Parametric global CPS encoding
- Parametric local Church encodings
  - untyped
  - simple types, but only homogeneous products
  - polymorphic
- Multiple base types — no local encoding
- Single base type without η — no local encoding
- Single base type with η — type-indexed local encoding

Can function types encode product types?

It depends...

- Parametric global CPS encoding
- Parametric local Church encodings
  - untyped
  - simple types, but only homogeneous products
  - polymorphic
- Multiple base types — no local encoding
- Single base type without η — no local encoding
- Single base type with η — type-indexed local encoding

Incidentally, none of these encodings preserves the η-rule for products.

Part II

Recursive types

Motivation

Stephen Dolan

Alan Mycroft

[Dolan and Mycroft, 2017, "Polymorphism, subtyping, and type inference in MLsub"]

Motivation

Stephen Dolan

Alan Mycroft

[Dolan and Mycroft, 2017, “Polymorphism, subtyping, and type inference in MLsub”]

MLsub relies on simulating general equi-recursive types by positive equi-recursive types.

Motivation

Stephen Dolan

Alan Mycroft

[Dolan and Mycroft, 2017, “Polymorphism, subtyping, and type inference in MLsub”]

MLsub relies on simulating general equi-recursive types by positive equi-recursive types.

General equi-recursive types can type non-terminating programs.

Motivation

Stephen Dolan

Alan Mycroft

[Dolan and Mycroft, 2017, "Polymorphism, subtyping, and type inference in MLsub"]

MLsub relies on simulating general equi-recursive types by positive equi-recursive types.

General equi-recursive types can type non-terminating programs.

Positive iso-recursive types can be encoded in System F.

Motivation

Stephen Dolan

Alan Mycroft

[Dolan and Mycroft, 2017, “Polymorphism, subtyping, and type inference in MLsub”]

MLsub relies on simulating general equi-recursive types by positive equi-recursive types.

General equi-recursive types can type non-terminating programs.

Positive iso-recursive types can be encoded in System F.

Why the discrepancy between equi and iso?

Recursive types

Algebraic datatypes

$$\begin{aligned}
\text{data Nat} &= \text{Z} \mid \text{S Nat} \\
\text{data List} &= \text{Nil} \mid \text{Cons Int List} \\
\text{data Tree} &= \text{Leaf} \mid \text{Node Tree Int Tree}
\end{aligned}$$

Inline recursive types

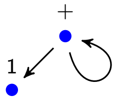
$$\begin{aligned}
\text{Nat} &= \mu X.1 + X \\
\text{List} &= \mu X.1 + \text{Int} \times X \\
\text{Tree} &= \mu X.1 + X \times \text{Int} \times X
\end{aligned}$$

Recursive type equations

$$\begin{aligned}
\text{Nat} &= 1 + \text{Nat} \\
\text{List} &= 1 + \text{Int} \times \text{List} \\
\text{Tree} &= 1 + \text{Tree} \times \text{Int} \times \text{Tree}
\end{aligned}$$

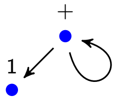
Recursive types as regular trees

Nat $\quad\quad \mu X.1 + X$

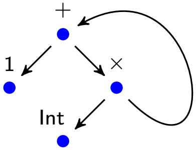


Recursive types as regular trees

Nat $\mu X.1 + X$

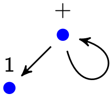


List $\mu X.1 + \mathsf{Int} \times X$

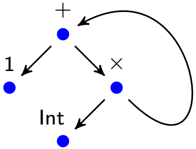


Recursive types as regular trees

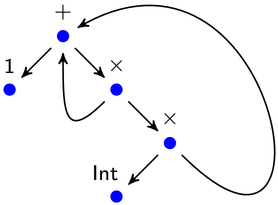
Nat $\mu X.1 + X$



List $\mu X.1 + \mathsf{Int} \times X$



Tree $\mu X.1 + X \times \mathsf{Int} \times X$



Equi-recursive types

$$\frac{\Gamma \vdash M : A \qquad \vdash A \approx B}{\Gamma \vdash M : B}$$

$\vdash A \approx B$ means

A and B are equivalent up to infinite unrolling

Equi-recursive types

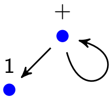
$$\frac{\Gamma \vdash M : A \qquad \vdash A \approx B}{\Gamma \vdash M : B}$$

$\vdash A \approx B$ means

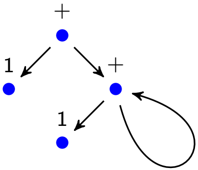
A and B are equivalent up to infinite unrolling

Example

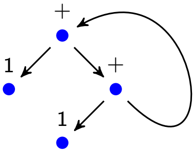
$$\mu X.1 + X \quad \approx \quad 1 + \mu X.1 + X \quad \approx \quad \mu X.1 + (1 + X)$$



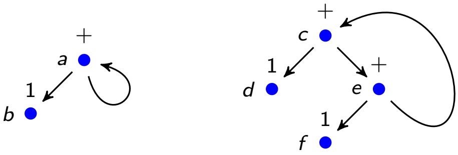
$\approx$



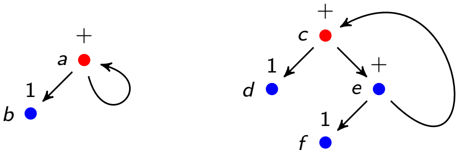
$\approx$



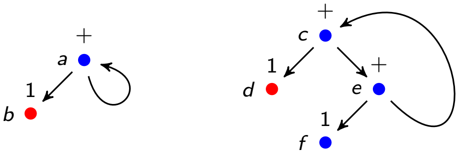
Deciding equality by unrolling



Deciding equality by unrolling

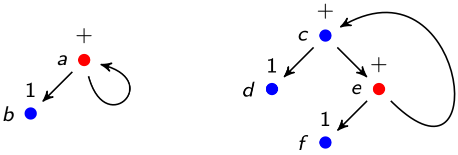


Deciding equality by unrolling



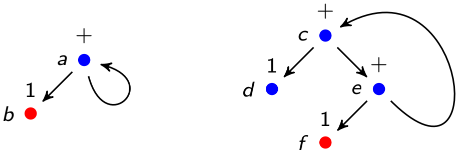
$$a \approx c$$

Deciding equality by unrolling



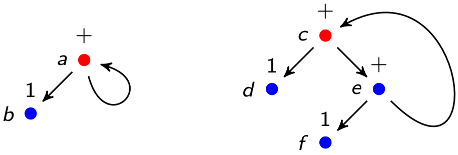
$$a \approx c$$

Deciding equality by unrolling



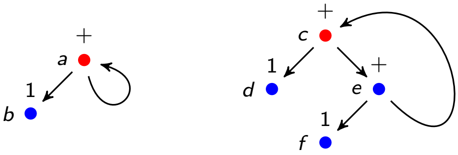
$$a \approx c, a \approx e$$

Deciding equality by unrolling



$a \approx c$, $a \approx e$

Deciding equality by unrolling



$a \approx c, a \approx e$

$\boxed{\Phi \vdash a \approx b}$

$$\textsc{Repeat}\quad \frac{}{\Phi, a \approx b \vdash a \approx b}$$

$$\textsc{Rec-Cons}\quad \frac{label(a) = label(b) \qquad \Phi, a \approx b \vdash children(a) \approx children(b)}{\Phi \vdash a \approx b}$$

Φ a set — comma is disjoint extension

Iso-recursive types

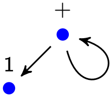
$$\textsc{Roll}\ \frac{\Gamma \vdash M : A[\mu X.A/X]}{\Gamma \vdash \mathsf{roll}\ M : \mu X.A} \qquad \textsc{Unroll}\ \frac{\Gamma \vdash M : \mu X.A}{\Gamma \vdash \mathsf{unroll}\ M : A[\mu X.A/X]}$$

Iso-recursive types

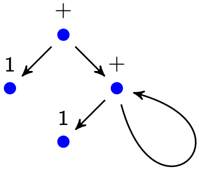
$$\textsc{Roll}\quad \frac{\Gamma \vdash M : A[\mu X.A/X]}{\Gamma \vdash \mathsf{roll}\ M : \mu X.A} \qquad\qquad \textsc{Unroll}\quad \frac{\Gamma \vdash M : \mu X.A}{\Gamma \vdash \mathsf{unroll}\ M : A[\mu X.A/X]}$$

Example

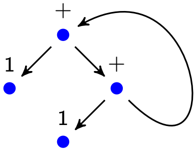
$$\mu X.1 + X \quad \neq \quad 1 + \mu X.1 + X \quad \neq \quad \mu X.1 + (1 + X)$$



$\neq$



$\neq$



Equi-recursive versus iso-recursive types

STLC	Simply-Typed Lambda Calculus
FPC	Fixed Point Calculus
FPC	STLC + iso-recursive types
$\mathrm{FPC}_{=}$	STLC + equi-recursive types

Equi-recursive versus iso-recursive types

STLC	Simply-Typed Lambda Calculus
FPC	Fixed Point Calculus
FPC	STLC + iso-recursive types
$\mathrm{FPC}_=$	STLC + equi-recursive types

Interdefinability

- $\mathrm{FPC}_=$ can simulate FPC
  — just erase roll and unroll
- FPC can simulate $\mathrm{FPC}_=$ up to observational equivalence
  — coercion functions witness type equivalence
  [Abadi and Fiore, 1996; Brandt and Henglein, 1998]

Positive recursive types

Positive type variable: occurs on left of even number of arrows

Negative type variable: occurs on left of odd number of arrows

Strictly positive type variable: occurs on right of arrows

Examples — X occurs:

- strictly positively in X, $1 + \mathsf{Int} \times X$, and $\mathsf{Int} \to X$
- positively in $(X \to \mathsf{Int}) \to \mathsf{Int}$
- negatively in $X \to \mathsf{Int}$
- both positively and negatively in $X \to X$

A recursive type is positive if all occurrences of the bound variable are positive, e.g:

$$\mu X.1 + \mathsf{Int} \times X \qquad\qquad \mu X.(X \to \mathsf{Int}) \to \mathsf{Int}$$

Positive equi- versus positive iso-

FPC^{+}	STLC $+$ positive iso-recursive types
$\mathsf{FPC}^{+}_{=}$	STLC $+$ positive equi-recursive types
FPC^{++}	STLC $+$ strictly positive iso-recursive types
$\mathsf{FPC}^{++}_{=}$	STLC $+$ strictly positive equi-recursive types

Positive equi- versus positive iso-

FPC^{+}	STLC + positive iso-recursive types
$\mathrm{FPC}^{+}_{=}$	STLC + positive equi-recursive types
FPC^{++}	STLC + strictly positive iso-recursive types
$\mathrm{FPC}^{++}_{=}$	STLC + strictly positive equi-recursive types

Interdefinability

- $\mathrm{FPC}^{+}_{=}$ can simulate FPC^{+} — just erase roll and unroll
- $\mathrm{FPC}^{++}_{=}$ can simulate FPC^{++} — just erase roll and unroll
- System F can simulate FPC^{+} (strong normalisation)
- $\mathrm{FPC}^{+}_{=}$ can simulate $\mathrm{FPC}_{=}$ (non-termination)
- $\mathrm{FPC}^{+}_{=}$ is stricty more expressive than FPC^{+}
- FPC^{+} + general recursion can simulate FPC up to observational equivalence
- FPC^{++} + fold can simulate $\mathrm{FPC}^{++}_{=}$ up to observational equivalence

Yeah, yeah

Universal type with a **negative** occurrence

$$U = U \to U \quad = \quad \mu X.X \to X$$

All untyped lambda terms can be typed with U

$$\omega = \lambda x.x\ x \qquad\qquad \Omega = \omega\ \omega \qquad\qquad \Omega \leadsto_\beta \Omega$$

Yeah, yeah

Universal type with a **negative** occurrence

$$U = U \to U \quad = \quad \mu X.X \to X$$

All untyped lambda terms can be typed with U

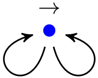
$$\omega = \lambda x.x\ x \qquad\qquad \Omega = \omega\ \omega \qquad\qquad \Omega \rightsquigarrow_\beta \Omega$$

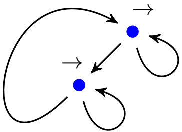
Universal type as mutually recursive **positive** type

$$\begin{aligned} P &= Q \to P &&= \quad \mu X.(\mu Y.X \to Y) \to X \\ Q &= P \to Q &&= \quad \mu X.(\mu Y.X \to Y) \to X \end{aligned}$$

U, P, Q all represent infinite binary tree of $\to$ nodes

$$\vdash U \approx P \approx Q$$





Yeah, yeah

Universal type with a **negative** occurrence

$$U = U \to U \quad = \quad \mu X.X \to X$$

All untyped lambda terms can be typed with U

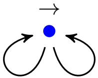
$$\omega = \lambda x.x\ x \qquad\qquad \Omega = \omega\ \omega \qquad\qquad \Omega \rightsquigarrow_\beta \Omega$$

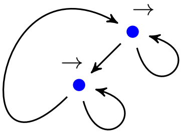
Universal type as mutually recursive **positive** type

$$\begin{aligned} P &= Q \to P &&= \quad \mu X.(\mu Y.X \to Y) \to X \\ Q &= P \to Q &&= \quad \mu X.(\mu Y.X \to Y) \to X \end{aligned}$$

U, P, Q all represent infinite binary tree of $\to$ nodes

$$\vdash U \approx P \approx Q$$





All untyped lambda terms can be typed with P!

$\mathrm{FPC}_{=}$ in $\mathrm{FPC}^{\pm}_{=}$

Idea: split the positive and negative occurrences

$$\mu X.F[X, X] \approx \mu X.F[\mu Y.F[X, Y], X]$$

[Bekić, 1984]

Example: universal data type

$$F[X^-, X^+] \quad = \quad X^- \to X^+$$

$$\begin{array}{rcl} U & = & \mu X.F[X, X] = \mu X.X \to X \\ P & = & \mu X.F[\mu Y.F[X, Y], X] = \mu X.(\mu Y.X \to Y) \to X \end{array}$$

FPC in FPC$^+$ + general recursion

Coercions for simulating FPC in FPC$_=$ use general recursion

FPC$^+$ + general recursion $\longrightarrow$ FPC$^\pm_=$ $\longrightarrow$ FPC$_=$ $\longrightarrow$ FPC

Example:

$$
\begin{array}{rcl}
\omega^{U\to U} & = & \lambda x^{U}.(\mathrm{unroll}\ x)\ x \\
\Omega^{U} & = & \omega^{U\to U}\ (\mathrm{roll}\ \omega^{U\to U}) \\
\\
\omega^{Q\to P} & = & \lambda x^{Q}.(\mathrm{unroll}\ (V_{QP}\ x))\ (V_{QP}\ x) \\
\Omega^{P} & = & \omega^{Q\to P}\ (V_{PQ}\ (\mathrm{roll}\ \omega^{Q\to P})) \\
 & & \text{where} \\
 & & \quad V_{QP} = \mathrm{rec}\ f^{Q\to P}\ x^{Q}.\mathrm{roll}\ (f \circ (\mathrm{unroll}\ x) \circ f) \\
 & & \quad V_{PQ} = \mathrm{rec}\ f^{P\to Q}\ x^{P}.\mathrm{roll}\ (f \circ (\mathrm{unroll}\ x) \circ f)
\end{array}
$$

Summary

$$\mathsf{FPC} \simeq \mathsf{FPC}_{=} \simeq \mathsf{FPC}^{+}_{=} \gtrsim \mathsf{FPC}^{+} \lesssim \text{System F}$$

$$\mathsf{FPC}^{++} \simeq \mathsf{FPC}^{++}_{=}$$

Can positive iso-recursive types encode positive equi-recursive types?

Can positive iso-recursive types encode positive equi-recursive types?

Not without general recursion

Can positive iso-recursive types encode positive equi-recursive types?

Not without general recursion

Intuitively the encoding requires the insertion of an infinite number of **roll**s and **unroll**s

Can positive iso-recursive types encode positive equi-recursive types?

Not without general recursion

Intuitively the encoding requires the insertion of an infinite number of **roll**s and **unroll**s

Morally the notion of a “positive” equi-recursive type is rather misleading

Part III

Existential types

Motivation

Well-known how universal types can encode existential types

Motivation

Well-known how universal types can encode existential types

What if we already have existential types, and want to encode universal types?

(I ran into this situation when trying to define a minimal effect handler calculus where parametric algebraic operations provide existentials, but there are no universals.)

Existentials as universals

De Morgan dual

$$\exists X.A \;\equiv\; \neg\forall X.\neg A \;=\; (\forall X.(A \to \bot)) \to \bot$$

Existentials as universals

De Morgan dual

$$\exists X.A \;\equiv\; \neg\forall X.\neg A \;=\; (\forall X.(A \to \bot)) \to \bot$$

System F encoding generalises the de Morgan dual

$$\exists X.A \;\equiv\; \forall Z.(\forall X.(A \to Z)) \to Z$$

Existentials as universals

De Morgan dual

$$\exists X.A \;\equiv\; \neg\forall X.\neg A \;=\; (\forall X.(A \to \bot)) \to \bot$$

System F encoding generalises the de Morgan dual

$$\exists X.A \;\equiv\; \forall Z.(\forall X.(A \to Z)) \to Z$$

What about the other way round?

$$\forall X.A \;\equiv\; \neg\exists X.\neg A \;=\; (\exists X.(A \to \bot)) \to \bot$$

Existentials as universals

De Morgan dual

$$\exists X.A \;\equiv\; \neg\forall X.\neg A \;=\; (\forall X.(A \to \bot)) \to \bot$$

System F encoding generalises the de Morgan dual

$$\exists X.A \;\equiv\; \forall Z.(\forall X.(A \to Z)) \to Z$$

What about the other way round?

$$\forall X.A \;\equiv\; \neg\exists X.\neg A \;=\; (\exists X.(A \to \bot)) \to \bot$$

The generalisation trick is no good as it depends on another universal quantifier

$$\forall X.A \;\equiv\; \forall Z.(\exists X.A \to Z) \to Z$$

Minimal existential logic

Types

$$A, B ::= \bot \mid \neg A \mid A \times B \mid \exists X.A \mid X$$

Terms

$$\begin{aligned} M, N ::= {} & x \\ \mid \quad & \lambda x^A.M \mid M\ N \\ \mid \quad & (M, N) \mid \text{let } (x, y) = M \text{ in } N \\ \mid \quad & (A, M) \mid \text{let } (X, y) = M \text{ in } N \end{aligned}$$

Universals as existentials [Fujita, 2010]

Judgements

$$(\Gamma \vdash M : A)^* \quad = \quad \neg\Gamma^* \vdash M^* : \neg A^*$$

Types

$$\begin{array}{rcl} X^* & = & X \\ (A \to B)^* & = & \neg A^* \times B^* \\ (\forall X.A)^* & = & \exists X.A^* \end{array}$$

Terms

$$\begin{array}{rcl} (x : A)^* & = & \lambda k^{A^*}.x\ k \\ (\lambda x.M : A)^* & = & \lambda k^{A^*}.\mathsf{let}\ (x, k) = k\ \mathsf{in}\ M^*\ k \\ (M\ N : A)^* & = & \lambda k^{A^*}.M^*\ (N^*, k) \\ (\Lambda X.M : A)^* & = & \lambda k^{A^*}.\mathsf{let}\ (X, k) = k\ \mathsf{in}\ M^*\ k \\ (M\ B : A)^* & = & \lambda k^{A^*}.M^*\ (B^*, k) \end{array}$$

Can existentials encode universals?

Can existentials encode universals?

Yes, with a global CPS translation [Fujita, 2010]

Can existentials encode universals?

Yes, with a global CPS translation [Fujita, 2010]

Open question: can we prove that there is no local encoding?

Part IV

Effectful anecdotes

Idioms are oblivious, arrows are meticulous, monads are meticulous

Philip Wadler

Jeremy Yallop

[Lindley, Wadler, and Yallop, 2008]

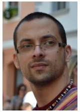
Semantically: (monads $<$ arrows) and (idioms $<$ arrows)

As programs: idioms $<$ arrows $<$ monads

On the expressive power of user-defined effects

Yannick Forster

Ohad Kammar



Matija Pretnar

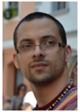
[Forster, Kammar, Lindley, and Pretnar, 2019]

Eff = effect handlers   Mon = monadic reflection   Del = delimited continuations

On the expressive power of user-defined effects

Yannick Forster

Ohad Kammar



Matija Pretnar

[Forster, Kammar, Lindley, and Pretnar, 2019]

Eff = effect handlers Mon = monadic reflection Del = delimited continuations

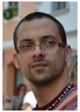
Untyped Eff $\longleftrightarrow$ Mon $\longleftrightarrow$ Del $\longleftrightarrow$ Eff

- Translations Mon $\longrightarrow$ Eff and Del $\longrightarrow$ Eff simulate reduction on the nose
- Others translations don't

On the expressive power of user-defined effects

Yannick Forster

Ohad Kammar



Matija Pretnar

[Forster, Kammar, Lindley, and Pretnar, 2019]

Eff = effect handlers Mon = monadic reflection Del = delimited continuations

Untyped Eff $\longleftrightarrow$ Mon $\longleftrightarrow$ Del $\longleftrightarrow$ Eff

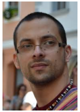
- Translations Mon $\longrightarrow$ Eff and Del $\longrightarrow$ Eff simulate reduction on the nose
- Others translations don't

Simply-typed Eff $\not\longleftrightarrow$ Del

On the expressive power of user-defined effects


Yannick Forster


Ohad Kammar


Matija Pretnar

[Forster, Kammar, Lindley, and Pretnar, 2019]

Eff = effect handlers   Mon = monadic reflection   Del = delimited continuations

Untyped   Eff $\longleftrightarrow$ Mon $\longleftrightarrow$ Del $\longleftrightarrow$ Eff

- Translations Mon $\longrightarrow$ Eff and Del $\longrightarrow$ Eff simulate reduction on the nose
- Others translations don't

Simply-typed   Eff $\not\longleftrightarrow$ Del

Polymorphic   Eff $\longleftrightarrow$ Del

[Piróg, Polesiuk, Sieczkowski, 2019]   Novel form of answer-type polymorphism

Asymptotic improvement with control operators

Daniel Hillerström

John Longley

[Hillerström, Lindley, and Longley, 2020]

Generic search algorithm is:

- $\Omega(n2^n)$ in PCF
- $O(2^n)$ in PCF + effect handlers

Key constraint: no change of types

Higher-order computability [Longley and Normann, 2015]

Part V

Wrapping up

Closing remarks

The range of notions of expressiveness is broad

Expressiveness results are fragile

Types enable richer notions of expressiveness