Magnolia

Implementing System F with Anonymous Sums and Products

Andreas Halkjær From

DTU Compute
Department of Applied Mathematics and Computer Science

Introduction

What foundation should we build a functional language on in 2018?
Bidirectional typing seems a promising start: Used by Scala, PureScript.
Known for its easy scalability to advanced features such as rank-n, higher-kinded and sized types.

Starting point:

> *Complete and Easy Bidirectional Type Checking for Higher-Rank Polymorphism by Joshua Dunfield & Neelakantan R. Krishnaswami*

We will add sums and products manually, based on a common notion of rows of types.

Finally we will look briefly at Bob Harper's Abstract Binding Trees that aid in the implementation.

Magnolia

OCaml + Jane Street Core. 3100 lines + tests.

- OCamllex
- Menhir
- Elaborator
- Type checker:
  - Complete and Easy Bidirectional Type Checking for Higher-Rank Polymorphism by Joshua Dunfield & Neelakantan R. Krishnaswami
  - Sums & products
  - Strictly-positive recursive types & catamorphisms
  - Elaborate to explicit type abstraction/instantiation
- `prettiest`, based on Jean-Philippe Bernardy's A Pretty But Not Greedy Printer (Functional Pearl). ICFP 2017
- Interpreter

Example Code

```
alias listF A R = [nil: {} | cons: {head: A, tail: R}]
alias list A = mu R. listF A R

let nil : forall A. list A
  = fold (.nil {})

let cons : forall A. A -> list A -> list A
  | head tail = fold (.cons {head, tail})

let my-list : list int
  = cons 1 (cons 2 (cons 3 (cons 4 nil)))

let main : int
  = cata [ nil -> 0
         | cons {tail: n} -> n + 1 ]
         my-list
```

==> 4

Contents

Part I

Complete and Easy

Lambda Calculus

$$e ::= () \mid x \mid \lambda x.\ e \mid e\ e$$

$$\sigma, \tau ::= \mathbb{1} \mid \sigma \to \tau$$

Judgment $\Gamma \vdash e : \tau$

$$\frac{}{\Gamma \vdash () : \mathbb{1}} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x.e : \sigma \to \tau}$$

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\ e_2 : \tau}$$

Bidirectional Typing

Insight: Context matters. Two judgments!

Synthesis $\Gamma \vdash e \Rightarrow \tau$

Checking $\Gamma \vdash e \Leftarrow \tau$

```
syn : ctx * term -> typ
chk : ctx * term * typ -> bool
```

Well-formedness $\Gamma \vdash \tau$

$$\frac{}{\Gamma \vdash () \Rightarrow \mathbb{1}} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x \Rightarrow \tau}$$

Switching:

$$\frac{\Gamma \vdash \tau \quad \Gamma \vdash e \Leftarrow \tau}{\Gamma \vdash (e : \tau) \Rightarrow \tau} \qquad \frac{\Gamma \vdash e \Rightarrow \sigma \quad \sigma \equiv \tau}{\Gamma \vdash e \Leftarrow \tau}$$

We need to know σ and τ (but only once!):

$$\frac{\Gamma, x : \sigma \vdash e \Leftarrow \tau}{\Gamma \vdash \lambda x.e \Leftarrow \sigma \to \tau}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow \sigma \to \tau \quad \Gamma \vdash e_2 \Leftarrow \sigma}{\Gamma \vdash e_1\ e_2 \Rightarrow \tau}$$

Existential Type Variables I

Synthesis for λ-expressions? Existential variables + ordered output context.

$$\tau ::= \ldots \mid \hat{\alpha}, \hat{\beta}$$

Synthesis $\Gamma \vdash e \Rightarrow \tau \dashv \Delta$
Checking $\Gamma \vdash e \Leftarrow \tau \dashv \Delta$

Δ might solve more existentials than Γ.

$$\frac{}{\Gamma \vdash () \Rightarrow \mathbb{1} \dashv \Gamma} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x \Rightarrow \tau \dashv \Gamma}$$

$$\frac{\Gamma \vdash \tau \quad \Gamma \vdash e \Leftarrow \tau \dashv \Delta}{\Gamma \vdash (e : \tau) \Rightarrow \tau \dashv \Delta} \qquad \frac{\Gamma \vdash e \Rightarrow \tau \dashv \Delta \qquad \boxed{\sigma \equiv \tau}}{\Gamma \vdash e \Leftarrow \tau \dashv \Delta}$$

Remember to enforce scope:

$$\frac{\Gamma, x : \sigma \vdash e \Leftarrow \tau \dashv \Delta, x : \sigma, \Theta}{\Gamma \vdash \lambda x.e \Leftarrow \sigma \to \tau \dashv \Delta} \qquad \boxed{\frac{\Gamma, \hat{\alpha}, \hat{\beta}, x : \hat{\alpha} \vdash e \Leftarrow \hat{\beta} \dashv \Delta, x : \hat{\alpha}, \Theta}{\Gamma \vdash \lambda x.e \Rightarrow \hat{\alpha} \to \hat{\beta} \dashv \Delta}}$$

Existential Type Variables II

Application of an existential type variable? Extra judgment!

Application $\Gamma \vdash \sigma \bullet e \rightrightarrows \tau \dashv \Delta$

$$\frac{\Gamma \vdash e_1 \Rightarrow \sigma \dashv \Theta \quad \Theta \vdash [\Theta]\sigma \bullet e_2 \rightrightarrows \tau \dashv \Delta}{\Gamma \vdash e_1\ e_2 \Rightarrow \tau \dashv \Delta}$$

Context substitution $[\Theta]\tau$ substitutes existentials in τ for solutions from Θ.

$$\frac{\Gamma \vdash e \Leftarrow \sigma \dashv \Delta}{\Gamma \vdash \sigma \to \tau \bullet e \rightrightarrows \tau \dashv \Delta} \qquad \frac{\Gamma[\hat{\alpha}_2, \hat{\alpha}_1, \hat{\alpha} = \hat{\alpha}_1 \to \hat{\alpha}_2] \vdash e \Leftarrow \hat{\alpha}_1 \dashv \Delta}{\Gamma[\hat{\alpha}] \vdash \hat{\alpha} \bullet e \rightrightarrows \hat{\alpha}_2 \dashv \Delta}$$

The holed context $\Gamma[\hat{\alpha}]$ is short for $\Gamma_l, \hat{\alpha}, \Gamma_r$.
$\Gamma[\hat{\alpha}_2, \hat{\alpha}_1, \hat{\alpha} = \hat{\alpha}_1 \to \hat{\alpha}_2]$ plugs the hole with something else.

Subtyping I

Checking against an existential type variable? Subtyping!

Subtyping $\Gamma \vdash \sigma <: \tau \dashv \Delta$

$$\frac{\Gamma \vdash e \Rightarrow \sigma \dashv \Theta \qquad \boxed{\Theta \vdash [\Theta]\sigma <: [\Theta]\tau \dashv \Delta}}{\Gamma \vdash e \Leftarrow \tau \dashv \Delta}$$

Common-sense rules (omitted) + instantiation:

Instantiation $\Gamma \vdash \hat{\alpha} :\stackrel{\leq}{=} \tau \dashv \Delta$

$\Gamma \vdash \tau :\stackrel{\leq}{=} \hat{\alpha} \dashv \Delta$

$$\frac{\hat{\alpha} \notin \mathrm{FV}(\tau) \quad \Gamma[\hat{\alpha}] \vdash \hat{\alpha} :\stackrel{\leq}{=} \tau \dashv \Delta}{\Gamma[\hat{\alpha}] \vdash \hat{\alpha} <: \tau \dashv \Delta} \qquad \frac{\hat{\alpha} \notin \mathrm{FV}(\tau) \quad \Gamma[\hat{\alpha}] \vdash \tau :\stackrel{\leq}{=} \hat{\alpha} \dashv \Delta}{\Gamma[\hat{\alpha}] \vdash \tau <: \hat{\alpha} \dashv \Delta}$$

Instantiation I

$$\frac{\Gamma \vdash \tau}{\Gamma, \hat{\alpha}, \Gamma' \vdash \hat{\alpha} :\stackrel{\le}{=} \tau \dashv \Gamma, \hat{\alpha} = \tau, \Gamma'} \qquad \frac{}{\Gamma[\hat{\alpha}][\hat{\beta}] \vdash \hat{\alpha} :\stackrel{\le}{=} \hat{\beta} \dashv \Gamma[\hat{\alpha}][\hat{\beta} = \hat{\alpha}]}$$

Function arrow is contravariant:

$$\frac{\Gamma[\hat{\alpha}_2, \hat{\alpha}_1, \hat{\alpha} = \hat{\alpha}_1 \to \hat{\alpha}_2] \vdash \sigma :\stackrel{\le}{=} \hat{\alpha}_1 \dashv \Theta \quad \Theta \vdash \hat{\alpha}_2 :\stackrel{\le}{=} [\Theta]\tau \dashv \Delta}{\Gamma[\hat{\alpha}] \vdash \hat{\alpha} :\stackrel{\le}{=} \sigma \to \tau \dashv \Delta}$$

Symmetric rules for right instantiation.

Universal Quantifiers

$$\text{Monotypes } \sigma, \tau ::= \mathbb{1} \mid \alpha \mid \hat{\alpha} \mid \sigma \to \tau$$
$$\text{Types } A, B, C ::= \tau \mid A \to B \mid \forall \alpha . A$$

Existentials only stand in for monotypes.
Checking a polymorphic type:

$$\frac{\Gamma, \alpha \vdash e \Leftarrow A \dashv \Delta, \alpha, \Theta}{\Gamma \vdash e \Leftarrow \forall \alpha . A \dashv \Delta}$$

No synthesis rule.
Applying a polymorphic type instantiates it:

$$\frac{\Gamma, \hat{\alpha} \vdash [\hat{\alpha}/\alpha]A \bullet e \Rrightarrow C \dashv \Delta}{\Gamma \vdash \forall \alpha . A \bullet e \Rrightarrow C \dashv \Delta}$$

Subtyping II

How to answer $\forall\alpha.A <: B$?
Can we instantiate quantifier suitably:

$$\frac{\Gamma, \blacktriangleright_{\hat{\alpha}}, \hat{\alpha} \vdash [\hat{\alpha}/\alpha]A <: B \dashv \Delta, \blacktriangleright_{\hat{\alpha}}, \Theta}{\Gamma \vdash \forall\alpha.A <: B \dashv \Delta}$$

We may solve existential so use a marker, $\blacktriangleright_{\hat{\alpha}}$.

What about $A <: \forall\beta.B$?
Is A a subtype of B for arbitrary β:

$$\frac{\Gamma, \beta \vdash A <: B \dashv \Delta, \beta, \Theta}{\Gamma \vdash A <: \forall\beta.B \dashv \Delta}$$

Instantiation II

$\forall\beta.B :\overset{\leq}{=} \hat{\alpha}$?

Instantiate quantifier with fresh existential.

$$\frac{\Gamma[\hat{\alpha}], \blacktriangleright_{\hat{\beta}}, \hat{\beta} \vdash [\hat{\beta}/\beta]B :\overset{\leq}{=} \hat{\alpha} \dashv \Delta, \blacktriangleright_{\hat{\beta}}, \Theta}{\Gamma[\hat{\alpha}] \vdash \forall\beta.B :\overset{\leq}{=} \hat{\alpha} \dashv \Delta}$$

$\hat{\alpha} :\overset{\leq}{=} \forall\beta.B$?

Make $\hat{\alpha}$ a subtype of B for arbitrary β:

$$\frac{\Gamma[\hat{\alpha}], \beta \vdash \hat{\alpha} :\overset{\leq}{=} B \dashv \Delta, \beta, \Theta}{\Gamma[\hat{\alpha}] \vdash \hat{\alpha} :\overset{\leq}{=} \forall\beta.B \dashv \Delta}$$

Let

$$e ::= \dots \mid \text{let } x = e \text{ in } e$$

$$\frac{\Gamma \vdash e_1 \Rightarrow \sigma \dashv \Theta \quad \Theta, x : \sigma \vdash e_2 \Rightarrow A \dashv \Delta, x : \sigma, \Delta'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 \Rightarrow A \dashv \Delta}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow \sigma \dashv \Theta \quad \Theta, x : \sigma \vdash e_2 \Leftarrow A \dashv \Delta, x : \sigma, \Delta'}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 \Leftarrow A \dashv \Delta}$$

No generalisation to preserve cut-elimination property.
E.g. in $\text{let } id = \lambda x.x \text{ in } id\ ()$, id will have type $() \to ()$.

Part II

Data Types

Rows

Only monotypes in rows.

Component-wise well-formedness:

$$\frac{\ldots \quad \Gamma \vdash \tau_i \quad \ldots}{\Gamma \vdash \#\{\ldots, c_i : \tau_i, \ldots\} \text{ row}}$$

Component-wise subtyping:

$$\frac{\ldots \quad \Gamma_{i-1} \vdash \tau_i <: \sigma_i \dashv \Gamma_i \quad c_i = d_i \quad \ldots}{\Gamma_0 \vdash \#\{\ldots, c_i : \tau_i, \ldots\} <: \#\{\ldots, d_i : \sigma_i, \ldots\} \dashv \Gamma_n}$$

Component-wise instantiation:

$$\frac{\begin{array}{ll}\Gamma_0[\hat{\alpha}_1, \ldots, \hat{\alpha} = \#\{c_1 : \hat{\alpha}_1, \ldots\}] \vdash \hat{\alpha}_1 :\overset{\leq}{=} \tau_1 \dashv \Gamma_1 & \ldots \\ \Gamma_{i-1} \vdash \hat{\alpha}_i :\overset{\leq}{=} \tau_i \dashv \Gamma_i & \ldots\end{array}}{\Gamma_0[\hat{\alpha}] \vdash \hat{\alpha} :\overset{\leq}{=} \#\{\ldots, c_i : \tau_i, \ldots\} \dashv \Gamma_n}$$

Symmetric right instantiation.

Sum and Product Types

$$\frac{\Gamma \vdash r \text{ row}}{\Gamma \vdash [r]} \qquad \frac{\Gamma \vdash r \text{ row}}{\Gamma \vdash \{r\}}$$

Subtyping on rows (no fanciness).

$$\frac{\Gamma \vdash r <: r' \dashv \Delta}{\Gamma \vdash [r] <: [r'] \dashv \Delta} \qquad \frac{\Gamma \vdash r <: r' \dashv \Delta}{\Gamma \vdash \{r\} <: \{r'\} \dashv \Delta}$$

Sums

Eliminators. Collection of functions:

$$\frac{\ldots \quad \Gamma_{i-1} \vdash e_i \Leftarrow \tau_i \to B \dashv \Gamma_i \quad \ldots}{\Gamma_0 \vdash [\ldots, c_i \to e_i, \ldots] \Leftarrow [\ldots, c_i : \tau_i, \ldots] \to B \dashv \Gamma_n}$$

$$\frac{\begin{array}{ll} \Gamma_0, \hat{\beta}, \hat{\alpha}_1, \ldots, \hat{\alpha}_n \vdash e_1 \Leftarrow \hat{\alpha}_1 \to \beta \dashv \Gamma_1 & \ldots \\ \Gamma_{i-1} \vdash e_i \Leftarrow \hat{\alpha}_i \to \hat{\beta} \dashv \Gamma_i & \ldots \end{array}}{\Gamma_0 \vdash [\ldots, c_i \to e_i, \ldots] \Rightarrow [\ldots, c_i : \hat{\alpha}_i, \ldots] \to \hat{\beta} \dashv \Gamma_n}$$

Injection:

$$\frac{\Gamma \vdash e \Leftarrow \tau_k \dashv \Delta}{\Gamma \vdash c_k \cdot e \Leftarrow [\ldots, c_k : \tau_k, \ldots] \dashv \Delta}$$

No synthesis for injection (can build yourself).
Possibilities: Existential row variables, polymorphic variants.

Products

Records:

$$\frac{\ldots \quad \Gamma_{i-1} \vdash e_i \Leftarrow \tau_i \dashv \Gamma_i \quad \ldots}{\Gamma_0 \vdash \{\ldots, c_i : e_i, \ldots\} \Leftarrow \{\ldots, c_i : \tau_i, \ldots\} \dashv \Gamma_n}$$

$$\frac{\ldots \quad \Gamma_{i-1} \vdash e_i \Rightarrow A_i \dashv \Gamma_i \quad \ldots}{\Gamma_0 \vdash \{\ldots, c_i : e_i, \ldots\} \Rightarrow \{\ldots, c_i : A_i, \ldots\} \dashv \Gamma_n}$$

Projection:

$$\frac{\Gamma \vdash e \Rightarrow \{\ldots, c_k : A_k, \ldots\} \dashv \Delta}{\Gamma \vdash e \cdot c_k \Rightarrow A_k \dashv \Delta}$$

Have to know type of e (can write own projection function).

Recursive Types

Inspired by *Practical Foundations for Programming Languages* by Bob Harper:

$$\tau ::= \ldots \mid \mu r.\tau$$

$$e ::= \ldots \mid \text{fold } e \mid \text{unfold } e \mid \text{cata } e\ e$$

Also: “X marks the spot-mapping”:

$$e ::= \ldots \mid \text{map}\{X.\tau\}\ e\ e$$

e.g.

$$\text{map}\{X.\textsf{list } X\},$$

$$\text{map}\{X.[\textsf{none: } \{\}|\textsf{some: } X]\},$$

Part III

Abstract Binding Trees

De Bruijn indices

$$\frac{\ldots\ [y/x]A\ \ldots}{\ldots\ \forall x.A\ \ldots}$$

M'colleague Bob Atkey once memorably described the capacity to put up with De Bruijn indices as a Cylon detector, the kind of reverse Turing Test that the humans in Battlestar Galactica invent, the better to recognize one another by their common inadequacies. He had a point.

— Conor McBride, "I am not a number, I am a classy hack"

Operators

Using ABT library in OCaml (port of CMU's SML library).

```
type op =
  (* Rows *)
  | Vec of int
  | Tag of Tag.t
  (* Types *)
  | Basic of basic
  | Exi of ExiVar.t
  | Arr
  | All
  | Sum
  | Prod
  | Mu
  (* Terms *)
  | Lit of literal
  | Ann
  | App
  | Lam of typed
  | Let
  (* Explicit polymorphism *)
  | Gen
  | Inst
  (* Datatypes *)
  | Inj of Tag.t * typed
  | Proj of Tag.t * typed
  | Elim of typed
  | Build of typed
  | Map of typed
  (* Recursive datatypes *)
  | Fold of typed
  | Unfold
  | Cata of typed
```

Arities

```
let arity op =
  match op with
  | Vec n -> List.init
      ~f:(const 0) n

  | Tag _ -> [0]

  | Basic _ -> []
  | Exi _   -> []
  | Arr   -> [0; 0]
  | All   -> [1]
  | Sum   -> [0]
  | Prod  -> [0]
  | Mu    -> [1]
```

```
  | Lit _ -> []
  | Ann    -> [0; 0]
  | App    -> [0; 0]
  | Lam Untyped -> [1]
  | Lam Typed    -> [1; 0]
  | Let   -> [0; 1]
  | Gen   -> [1]
  | Inst -> [0; 0]

  | Inj  (_, Untyped) -> [0]
  | Inj  (_, Typed)    -> [0; 0]
  | Proj (_, Untyped) -> [0]
  | Proj (_, Typed)    -> [0; 0]
  ...
```

Out

Build a term ($\lambda x.x$):

```
let x = Syntax.Var.named "x" in
Lam $$ [x ^^ (!! x)]
```

`Syntax.out e` returns one of:

VarView x

AbsView $x.e$ (where x is fresh and free in e)

AppView $op(e_1, \ldots, e_n)$ (corresponding to the arity of op)

`$$`, `out` etc. throw errors on arity mismatch.
Also get: `subst`, `aequiv`.

Note: We give up on some static help from the compiler.
Also: Not fast.

Example

$$\frac{\Gamma[\hat{\alpha}], \blacktriangleright_{\hat{\beta}}, \hat{\beta} \vdash [\hat{\beta}/\beta]B :\overset{\leq}{=} \hat{\alpha} \dashv \Delta, \blacktriangleright_{\hat{\beta}}, \Theta}{\Gamma[\hat{\alpha}] \vdash \forall \beta. B :\overset{\leq}{=} \hat{\alpha} \dashv \Delta}$$

```
(* InstRAll *)
| AppView (All, [t]) ->
  (match Syntax.out t with
   | AbsView (b, t) ->
     let b' = fresh_exi () in
     let ctx = ctx +> Marker b' +> ExiVar b'
     and inst = Syntax.subst (Exi b' $$ []) b t in
     let%bind ctx = instr ctx ~typ:inst ~var:a in
     add_inst inst;
     return (Ctx.until (Marker b') ctx)
   | _ -> raise Syntax.Malformed)
```