Implement a dependently typed core language in Haskell

Linus Yanpeng Yang

Apr. 9, 2015

Outline

1. **Introduction**
2. **Implement simply typed λ-calculus**
   - Type system of $\lambda_{\rightarrow}$
   - Bidirectional type inference
   - Locally nameless representation
   - Implementation
3. **Implement λ-calculus with dependent types**
   - Extend $\lambda_{\rightarrow}$ to λ_{Π}
   - Implement natural numbers & vectors for λ_{Π}

Introduction

- Dependent type is a type that depends on a value: Vec α n
- Used in some proof assistants: Coq, Agda
- Dependent types make the implementation complex: type checking Vec α 2 and Vec α $(1+1)$
- Incremental development of the core language: from $\lambda_{\rightarrow}$ to λ_{Π}

$\lambda_{\rightarrow}$: Abstract syntax

Types

$$\begin{array}{rcll} \tau & ::= & \alpha & \text{base type} \\ & | & \tau \rightarrow \tau' & \text{function type} \end{array}$$

Terms

$$\begin{array}{rcll} e & ::= & x & \text{variable} \\ & | & e\ e' & \text{application} \\ & | & \lambda x.e & \text{lambda abstraction} \end{array}$$

$\lambda_{\to}$: Typing rules

T-Var

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

T-App

$$\frac{\Gamma \vdash e : \tau \to \tau' \qquad \Gamma \vdash e' : \tau}{\Gamma \vdash e\ e' : \tau'}$$

T-Lam

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

How to implement the rule T-Lam?

T-Lam

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

Pseudocode

```
type ctx (Lam x e) =
  case (type ((x, ???) :: ctx) e) of
    Some t' => Some (Fun ??? t')
    None    => None
```

How do we decide the type as the ??? placeholder?

- Explicitly typed completely, i.e. use the Church-style λ-calculus: $\lambda x : \tau.e$
- Partially annotated and use the bidirectional type inference

Bidirectional type inference

- Separate expressions into two categories, checkable and inferable
- Separate judgments:
  - Given a type τ, verify a checkable expression e has that type: $\Gamma \vdash e \downarrow \tau$
  - Deduce the type of an inferable expression e as a result of τ: $\Gamma \vdash e \uparrow \tau$
- Make typing rules syntax-directed

Pros allowing many type annotations to be inferred automatically from the context of a term

Cons having more typing rules, and some type annotations must be written explicitly

Bidirectional style $\lambda_{\rightarrow}$

Inferable Terms

$$
\begin{array}{lcll}
e_{\uparrow} & ::= & e_{\downarrow} : \tau & \text{annotated term} \\
 & | & x & \text{variable} \\
 & | & e_{\uparrow}\ e'_{\downarrow} & \text{application}
\end{array}
$$

Checkable Terms

$$
\begin{array}{lcll}
e_{\downarrow} & ::= & e_{\uparrow} & \text{inferable term} \\
 & | & \lambda x . e_{\downarrow} & \text{abstraction}
\end{array}
$$

T-Ann

$$\frac{\Gamma \vdash e \downarrow \tau}{\Gamma \vdash (e : \tau) \uparrow \tau}$$

T-Var

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x \uparrow \tau}$$

T-Chk

$$\frac{\Gamma \vdash e \uparrow \tau}{\Gamma \vdash e \downarrow \tau}$$

T-App

$$\frac{\Gamma \vdash e \uparrow \tau \rightarrow \tau' \qquad \Gamma \vdash e' \downarrow \tau}{\Gamma \vdash e\ e' \uparrow \tau'}$$

T-Lam

$$\frac{\Gamma, x : \tau \vdash e \downarrow \tau'}{\Gamma \vdash \lambda x . e \downarrow \tau \rightarrow \tau'}$$

Representation of variables

- *Absolute references* (i.e. names) for free variables in terms
- *De Bruijn indices* for bound variables:
  - Use numbers to indicate how many binders occur between its binder and the occurrence
  - Variables never need to be renamed, i.e. α-equality of terms reduces to syntactic equality of terms

Examples

$$\begin{aligned} &\lambda xy.x && \rightarrow \lambda\ \lambda\ 1 \\ &\lambda xyz.xz(yz) && \rightarrow \lambda\ \lambda\ \lambda\ 2\ 0\ (1\ 0) \end{aligned}$$

Data types

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Term}_{\uparrow}\\
&\quad = \mathit{Ann}\quad \mathsf{Term}_{\downarrow}\ \mathsf{Type}\\
&\quad \mid \mathit{Bound}\ \mathsf{Int}\\
&\quad \mid \mathit{Free}\quad \mathsf{Name}\\
&\quad \mid \mathsf{Term}_{\uparrow}\ {:}@{:}\ \mathsf{Term}_{\downarrow}\\
&\ \ \textbf{deriving}\ (\mathit{Show}, \mathit{Eq})
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Term}_{\downarrow}\\
&\quad = \mathit{Inf}\quad \mathsf{Term}_{\uparrow}\\
&\quad \mid \mathit{Lam}\ \mathsf{Term}_{\downarrow}\\
&\ \ \textbf{deriving}\ (\mathit{Show}, \mathit{Eq})
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Name}\\
&\quad = \mathit{Global}\ \mathsf{String}\\
&\quad \mid \mathit{Local}\quad \mathsf{Int}\\
&\quad \mid \mathit{Quote}\quad \mathsf{Int}\\
&\ \ \textbf{deriving}\ (\mathit{Show}, \mathit{Eq})
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Type}\\
&\quad = \mathit{TFree}\ \mathsf{Name}\\
&\quad \mid \mathit{Fun}\quad \mathsf{Type}\ \mathsf{Type}\\
&\ \ \textbf{deriving}\ (\mathit{Show}, \mathit{Eq})
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Value}\\
&\quad = \mathit{VLam}\quad (\mathsf{Value} \to \mathsf{Value})\\
&\quad \mid \mathit{VNeutral}\ \mathsf{Neutral}
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Neutral}\\
&\quad = \mathit{NFree}\ \mathsf{Name}\\
&\quad \mid \mathit{NApp}\quad \mathsf{Neutral}\ \mathsf{Value}
\end{aligned}
$$

Evaluation

$$
\begin{array}{l}
\textbf{data}\ \mathsf{Term}_\uparrow \\
\quad = \mathit{Ann}\ \ \mathsf{Term}_\downarrow\ \mathsf{Type} \\
\quad \mid\ \mathit{Bound}\ \mathsf{Int} \\
\quad \mid\ \mathit{Free}\ \ \mathsf{Name} \\
\quad \mid\ \mathsf{Term}_\uparrow\ \text{:@:}\ \mathsf{Term}_\downarrow \\
\quad \textbf{deriving}\ (\mathit{Show}, \mathit{Eq})
\end{array}
\qquad
\begin{array}{l}
\textbf{data}\ \mathsf{Term}_\downarrow \\
\quad = \mathit{Inf}\ \ \mathsf{Term}_\uparrow \\
\quad \mid\ \mathit{Lam}\ \mathsf{Term}_\downarrow \\
\quad \textbf{deriving}\ (\mathit{Show}, \mathit{Eq})
\end{array}
\qquad
\begin{array}{l}
\textbf{data}\ \mathsf{Value} \\
\quad = \mathit{VLam}\ \ (\mathsf{Value} \to \mathsf{Value}) \\
\quad \mid\ \mathit{VNeutral}\ \mathsf{Neutral}
\end{array}
\qquad
\begin{array}{l}
\textbf{data}\ \mathsf{Neutral} \\
\quad = \mathit{NFree}\ \mathsf{Name} \\
\quad \mid\ \mathit{NApp}\ \ \mathsf{Neutral}\ \mathsf{Value}
\end{array}
$$

$$
\begin{array}{l}
\textbf{type}\ \mathsf{Env} = [\,\mathsf{Value}\,] \\[4pt]
\mathit{eval}_\uparrow :: \mathsf{Term}_\uparrow \to \mathsf{Env} \to \mathsf{Value} \\
\mathit{eval}_\uparrow\ (\mathit{Ann}\ \ e\ \_)\ d = \mathit{eval}_\downarrow\ e\ d \\
\mathit{eval}_\uparrow\ (\mathit{Free}\ x)\quad d = \mathit{vfree}\ x \\
\mathit{eval}_\uparrow\ (\mathit{Bound}\ i)\ \ d = d\ !!\ i \\
\mathit{eval}_\uparrow\ (e\ \text{:@:}\ e')\ \ d = \mathit{vapp}\ (\mathit{eval}_\uparrow\ e\ d)\ (\mathit{eval}_\downarrow\ e'\ d) \\[4pt]
\mathit{vapp} :: \mathsf{Value} \to \mathsf{Value} \to \mathsf{Value} \\
\mathit{vapp}\ (\mathit{VLam}\ f)\qquad v = f\ v \\
\mathit{vapp}\ (\mathit{VNeutral}\ n)\ v = \mathit{VNeutral}\ (\mathit{NApp}\ n\ v) \\[4pt]
\mathit{eval}_\downarrow :: \mathsf{Term}_\downarrow \to \mathsf{Env} \to \mathsf{Value} \\
\mathit{eval}_\downarrow\ (\mathit{Inf}\ \ \ i)\ \ d = \mathit{eval}_\uparrow\ i\ d \\
\mathit{eval}_\downarrow\ (\mathit{Lam}\ e)\ d = \mathit{VLam}\ (\lambda x \to \mathit{eval}_\downarrow\ e\ (x : d))
\end{array}
$$

Substitution and Quotation

$$
\begin{aligned}
&\mathit{subst}_{\uparrow} :: \mathsf{Int} \to \mathsf{Term}_{\uparrow} \to \mathsf{Term}_{\uparrow} \to \mathsf{Term}_{\uparrow}\\
&\mathit{subst}_{\uparrow}\ i\ r\ (\mathit{Ann}\ e\ \tau) = \mathit{Ann}\ (\mathit{subst}_{\downarrow}\ i\ r\ e)\ \tau\\
&\mathit{subst}_{\uparrow}\ i\ r\ (\mathit{Bound}\ j) = \mathbf{if}\ i == j\ \mathbf{then}\ r\ \mathbf{else}\ \mathit{Bound}\ j\\
&\mathit{subst}_{\uparrow}\ i\ r\ (\mathit{Free}\ y) \quad = \mathit{Free}\ y\\
&\mathit{subst}_{\uparrow}\ i\ r\ (e :@: e') = \mathit{subst}_{\uparrow}\ i\ r\ e :@: \mathit{subst}_{\downarrow}\ i\ r\ e'
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{subst}_{\downarrow} :: \mathsf{Int} \to \mathsf{Term}_{\uparrow} \to \mathsf{Term}_{\downarrow} \to \mathsf{Term}_{\downarrow}\\
&\mathit{subst}_{\downarrow}\ i\ r\ (\mathit{Inf}\ e) \quad = \mathit{Inf}\ (\mathit{subst}_{\uparrow}\ i\ r\ e)\\
&\mathit{subst}_{\downarrow}\ i\ r\ (\mathit{Lam}\ e) \quad = \mathit{Lam}\ (\mathit{subst}_{\downarrow}\ (i+1)\ r\ e)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{quote}_0 :: \mathsf{Value} \to \mathsf{Term}_{\downarrow}\\
&\mathit{quote}_0 = \mathit{quote}\ 0\\
&\mathit{quote} :: \mathsf{Int} \to \mathsf{Value} \to \mathsf{Term}_{\downarrow}\\
&\mathit{quote}\ i\ (\mathit{VLam}\ f) \quad = \mathit{Lam}\ (\mathit{quote}\ (i+1)\ (f\ (\mathit{vfree}\ (\mathit{Quote}\ i))\\
&\mathit{quote}\ i\ (\mathit{VNeutral}\ n) = \mathit{Inf}\ (\mathit{neutralQuote}\ i\ n)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{neutralQuote} :: \mathsf{Int} \to \mathsf{Neutral} \to \mathsf{Term}_{\uparrow}\\
&\mathit{neutralQuote}\ i\ (\mathit{NFree}\ x) \quad = \mathit{boundfree}\ i\ x\\
&\mathit{neutralQuote}\ i\ (\mathit{NApp}\ n\ v) = \mathit{neutralQuote}\ i\ n :@: \mathit{quote}\ i\ v
\end{aligned}
$$

Type checking

$$\begin{aligned}
&\textbf{data}\ \mathsf{Kind} = \mathit{Star}\\
&\quad \textbf{deriving}\ (\mathit{Show})\\
&\textbf{data}\ \mathsf{Info}\\
&\quad = \mathit{HasKind}\ \mathsf{Kind}\\
&\quad \mid\ \mathit{HasType}\ \mathsf{Type}\\
&\quad \textbf{deriving}\ (\mathit{Show})\\
&\textbf{type}\ \mathsf{Context} = [\,(\mathsf{Name}, \mathsf{Info})\,]
\end{aligned}$$

For inferable terms

$\mathsf{type}_{\uparrow} :: \mathsf{Int} \to \mathsf{Context} \to \mathsf{Term}_{\uparrow} \to \mathsf{Result\ Type}$

For checkable terms

$\mathsf{type}_{\downarrow} :: \mathsf{Int} \to \mathsf{Context} \to \mathsf{Term}_{\downarrow} \to \mathsf{Type} \to \mathsf{Result}\ ()$

Type checking for inferable terms

T-Ann

$$\frac{\Gamma \vdash e \downarrow \tau}{\Gamma \vdash (e : \tau) \uparrow \tau}$$

T-Var

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x \uparrow \tau}$$

T-App

$$\frac{\Gamma \vdash e \uparrow \tau \to \tau' \qquad \Gamma \vdash e' \downarrow \tau}{\Gamma \vdash e\ e' \uparrow \tau'}$$

$$
\begin{array}{l}
\mathit{type}_{\uparrow}\ i\ \Gamma\ (\mathit{Ann}\ e\ \tau) \\
\quad = \textbf{do}\ \mathit{kind}_{\downarrow}\ \Gamma\ \tau\ \mathit{Star} \\
\qquad\quad\ \ \mathit{type}_{\downarrow}\ i\ \Gamma\ e\ \tau \\
\qquad\quad\ \ \mathit{return}\ \tau \\
\mathit{type}_{\uparrow}\ i\ \Gamma\ (\mathit{Free}\ x) \\
\quad = \textbf{case}\ \mathit{lookup}\ x\ \Gamma\ \textbf{of} \\
\qquad\quad \mathit{Just}\ (\mathit{HasType}\ \tau) \to \mathit{return}\ \tau \\
\qquad\quad \mathit{Nothing} \qquad\qquad\quad\ \to \mathit{throwError}\ \texttt{"unknown identifier"} \\
\mathit{type}_{\uparrow}\ i\ \Gamma\ (e : @ : e') \\
\quad = \textbf{do}\ \sigma \leftarrow \mathit{type}_{\uparrow}\ i\ \Gamma\ e \\
\qquad\quad\ \ \textbf{case}\ \sigma\ \textbf{of} \\
\qquad\qquad \mathit{Fun}\ \tau\ \tau' \to \textbf{do}\ \mathit{type}_{\downarrow}\ i\ \Gamma\ e'\ \tau \\
\qquad\qquad\qquad\qquad\qquad \mathit{return}\ \tau' \\
\qquad\qquad \_ \qquad\quad\ \ \to \mathit{throwError}\ \texttt{"illegal application"}
\end{array}
$$

Type checking for checkable terms

T-Chk

$$\frac{\Gamma \vdash e \uparrow \tau}{\Gamma \vdash e \downarrow \tau}$$

T-Lam

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

$$
\begin{aligned}
& \mathit{type}_{\downarrow}\ i\ \Gamma\ (\mathit{Inf}\ e)\ \tau \\
& \quad = \mathbf{do}\ \tau' \leftarrow \mathit{type}_{\uparrow}\ i\ \Gamma\ e \\
& \qquad\qquad \mathit{unless}\ (\tau == \tau')\ (\mathit{throwError}\ \texttt{"type mismatch"}) \\
& \mathit{type}_{\downarrow}\ i\ \Gamma\ (\mathit{Lam}\ e)\ (\mathit{Fun}\ \tau\ \ \tau') \\
& \quad = \mathit{type}_{\downarrow}\ (i+1)\ ((\mathit{Local}\ i, \mathit{HasType}\ \tau) : \Gamma) \\
& \qquad\qquad (\mathit{subst}_{\downarrow}\ 0\ (\mathit{Free}\ (\mathit{Local}\ i))\ e)\ \tau' \\
& \mathit{type}_{\downarrow}\ i\ \Gamma\ \_\ \_ \\
& \quad = \mathit{throwError}\ \texttt{"type mismatch"}
\end{aligned}
$$

Extend $\lambda_{\rightarrow}$ with dependent types

- Polymorphic functions allow types to abstract over types:

$$\begin{aligned} id &:: \forall \alpha . \alpha \rightarrow \alpha \\ id &= \lambda \alpha : \star . x : \alpha . x \end{aligned}$$

$$id \ \mathsf{Bool} \ \textit{True} : \mathsf{Bool}$$
$$id \ \mathsf{Int} \ 42 : \mathsf{Int}$$

- Dependent types allow types to abstract over values:

$$\begin{aligned} id &:: \Pi(\alpha : \star).\Pi(n : \mathsf{Nat}).\Pi(v : \mathsf{Vec}\ \alpha\ n).\mathsf{Vec}\ \alpha\ n \\ &\equiv \Pi(\alpha : \star).\Pi(n : \mathsf{Nat}).\mathsf{Vec}\ \alpha\ n \rightarrow \mathsf{Vec}\ \alpha\ n \end{aligned}$$

Everything is a term

- Allowing values to appear freely in types breaks the separation of terms, types, and kinds: 0, Nat and $\star$ now are all terms

$$0 : \mathsf{Nat}, \mathsf{Nat} : \star$$

- We can have abstraction and application of types and kinds

λ_Π: Type system

Inferable Terms

$$
\begin{array}{rcll}
e_\uparrow & ::= & e_\downarrow : \tau & \text{annotated term} \\
& | & x & \text{variable} \\
& | & \star & \text{type of types} \\
& | & \Pi x : e.e' & \text{dependent fun.} \\
& | & e_\uparrow\ e'_\downarrow & \text{application}
\end{array}
$$

Checkable Terms

$$
\begin{array}{rcll}
e_\downarrow & ::= & e_\uparrow & \text{checkable term} \\
& | & \lambda x.e_\downarrow & \text{abstraction}
\end{array}
$$

$$
\frac{\Gamma \vdash \rho ::_\downarrow * \qquad \rho \Downarrow \tau \qquad \Gamma \vdash e ::_\downarrow \tau}{\Gamma \vdash (e :: \rho) ::_\uparrow \tau}\ (\text{ANN})
\qquad
\frac{}{\Gamma \vdash * ::_\uparrow *}\ (\text{STAR})
\qquad
\frac{\Gamma \vdash \rho ::_\downarrow * \qquad \rho \Downarrow \tau \qquad \Gamma, x :: \tau \vdash \rho' ::_\downarrow *}{\Gamma \vdash \forall x :: \rho.\rho' ::_\uparrow *}\ (\text{PI})
$$

$$
\frac{\Gamma(x) = \tau}{\Gamma \vdash x ::_\uparrow \tau}\ (\text{VAR})
\qquad
\frac{\Gamma \vdash e ::_\uparrow \forall x :: \tau.\tau' \qquad \Gamma \vdash e' ::_\downarrow \tau \qquad \tau'[x \mapsto e'] \Downarrow \tau''}{\Gamma \vdash e\ e' ::_\uparrow \tau''}\ (\text{APP})
$$

$$
\frac{\Gamma \vdash e ::_\uparrow \tau}{\Gamma \vdash e ::_\downarrow \tau}\ (\text{CHK})
\qquad
\frac{\Gamma, x :: \tau \vdash e ::_\downarrow \tau'}{\Gamma \vdash \lambda x \to e ::_\downarrow \forall x :: \tau.\tau'}\ (\text{LAM})
$$

λ_Π: Change of data types

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Term}_\uparrow\\
&\quad = \mathit{Ann}\ \mathsf{Term}_\downarrow\ \mathsf{Term}_\downarrow\\
&\quad \mid \mathit{Star}\\
&\quad \mid \mathit{Pi}\ \mathsf{Term}_\downarrow\ \mathsf{Term}_\downarrow\\
&\quad \mid \mathit{Bound}\ \mathsf{Int}\\
&\quad \mid \mathit{Free}\ \mathsf{Name}\\
&\quad \mid \mathsf{Term}_\uparrow\ \mathsf{:@:}\ \mathsf{Term}_\downarrow\\
&\textbf{deriving}\ (\mathit{Show}, \mathit{Eq})
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Value}\\
&\quad = \mathit{VLam}\ (\mathsf{Value} \to \mathsf{Value})\\
&\quad \mid \mathit{VStar}\\
&\quad \mid \mathit{VPi}\ \mathsf{Value}\ (\mathsf{Value} \to \mathsf{Value})\\
&\quad \mid \mathit{VNeutral}\ \mathsf{Neutral}
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{type}\ \mathsf{Type} &&= \mathsf{Value}\\
&\textbf{type}\ \mathsf{Context} &&= [\,(\mathsf{Name}, \mathsf{Type})\,]
\end{aligned}
$$

λ_Π: Change of eval, subst and quote

$$
\begin{aligned}
&eval_\uparrow\ Star \qquad\quad d = VStar\\
&eval_\uparrow\ (Pi\ \tau\ \tau')\ d = VPi\ (eval_\downarrow\ \tau\ d)\ (\lambda x \to eval_\downarrow\ \tau'\ (x : d))
\end{aligned}
$$

$$
\begin{aligned}
&subst_\uparrow\ i\ r\ (Ann\ e_\downarrow\ \tau) = Ann\ (subst_\downarrow\ i\ r\ e_\downarrow)\ (subst_\downarrow\ i\ r\ \tau)\\
&subst_\uparrow\ i\ r\ Star \qquad\quad\ \ = Star\\
&subst_\uparrow\ i\ r\ (Pi\ \tau\ \tau') \quad\ = Pi\ (subst_\downarrow\ i\ r\ \tau)\ (subst_\downarrow\ (i+1)\ r\ \tau')
\end{aligned}
$$

$$
\begin{aligned}
&quote\ i\ VStar = Inf\ Star\\
&quote\ i\ (VPi\ v\ f)\\
&\quad = Inf\ (Pi\ (quote\ i\ v)\ (quote\ (i+1)\ (f\ (vfree\ (Quote\ i)))))
\end{aligned}
$$

λ_Π: Change of $\uparrow$ typing rules

$$
\begin{array}{l}
\mathit{type}_{\uparrow}\ i\ \Gamma\ (\mathit{Ann}\ e\ \rho) \\
\quad = \mathbf{do}\ \mathit{type}_{\downarrow}\ i\ \Gamma\ \rho\ \mathit{VStar} \\
\qquad\quad \mathbf{let}\ \tau = \mathit{eval}_{\downarrow}\ \rho\ [\,] \\
\qquad\quad \mathit{type}_{\downarrow}\ i\ \Gamma\ e\ \tau \\
\qquad\quad \mathit{return}\ \tau \\
\mathit{type}_{\uparrow}\ i\ \Gamma\ \mathit{Star} \\
\quad = \mathit{return}\ \mathit{VStar} \\
\mathit{type}_{\uparrow}\ i\ \Gamma\ (\mathit{Pi}\ \rho\ \rho') \\
\quad = \mathbf{do}\ \mathit{type}_{\downarrow}\ i\ \Gamma\ \rho\ \mathit{VStar} \\
\qquad\quad \mathbf{let}\ \tau = \mathit{eval}_{\downarrow}\ \rho\ [\,] \\
\qquad\quad \mathit{type}_{\downarrow}\ (i+1)\ ((\mathit{Local}\ i, \tau) : \Gamma) \\
\qquad\qquad\quad (\mathit{subst}_{\downarrow}\ 0\ (\mathit{Free}\ (\mathit{Local}\ i))\ \rho')\ \mathit{VStar} \\
\qquad\quad \mathit{return}\ \mathit{VStar}
\end{array}
$$

$$
\begin{array}{l}
\mathit{type}_{\uparrow}\ i\ \Gamma\ (e :@: e') \\
\quad = \mathbf{do}\ \sigma \leftarrow \mathit{type}_{\uparrow}\ i\ \Gamma\ e \\
\qquad\quad \mathbf{case}\ \sigma\ \mathbf{of} \\
\qquad\qquad \mathit{VPi}\ \tau\ \tau' \rightarrow \mathbf{do}\ \mathit{type}_{\downarrow}\ i\ \Gamma\ e'\ \tau \\
\qquad\qquad\qquad\qquad\quad \mathit{return}\ (\tau'\ (\mathit{eval}_{\downarrow}\ e'\ [\,])) \\
\qquad\qquad \_ \qquad\quad\ \rightarrow \mathit{throwError}\ \texttt{"illegal application"}
\end{array}
$$

λ_Π: Change of $\downarrow$ typing rules

$$
\begin{aligned}
& \mathit{type}_\downarrow\ i\ \Gamma\ (\mathit{Inf}\ e)\ v \\
& \quad = \mathbf{do}\ v' \leftarrow \mathit{type}_\uparrow\ i\ \Gamma\ e \\
& \qquad\quad \mathit{unless}\ (\mathit{quote}_0\ v == \mathit{quote}_0\ v')\ (\mathit{throwError}\ \texttt{"type mismatch"}) \\
& \mathit{type}_\downarrow\ i\ \Gamma\ (\mathit{Lam}\ e)\ (\mathit{VPi}\ \tau\ \tau') \\
& \quad = \mathit{type}_\downarrow\ (i+1)\ ((\mathit{Local}\ i, \tau) : \Gamma) \\
& \qquad\qquad (\mathit{subst}_\downarrow\ 0\ (\mathit{Free}\ (\mathit{Local}\ i))\ e)\ (\tau'\ (\mathit{vfree}\ (\mathit{Local}\ i)))
\end{aligned}
$$

λ_Π: Demo of polymorphism

$\rangle\!\rangle$ **let** $id = (\lambda\alpha\ x \rightarrow x) :: \forall(\alpha :: *).\alpha \rightarrow \alpha$
$id :: \forall(x :: *)\ (y :: x).x$
$\rangle\!\rangle$ **assume** $(\mathsf{Bool} :: *)\ (False :: \mathsf{Bool})$

$\rangle\!\rangle$ $id\ \mathsf{Bool}$
$\lambda x \rightarrow x :: \forall x :: \mathsf{Bool}.\mathsf{Bool}$
$\rangle\!\rangle$ $id\ \mathsf{Bool}\ False$
$False :: \mathsf{Bool}$

Add data types into λ_Π

- $foldNat :: \forall\alpha :: *.\alpha \to (\alpha \to \alpha) \to \mathsf{Nat} \to \alpha$

- $$\begin{aligned} natElim :: \forall m :: \mathsf{Nat} \to *. \quad & m\ Zero \\ \to & (\forall l :: \mathsf{Nat}.m\ l \to m\ (Succ\ l)) \\ \to & \forall k :: \mathsf{Nat}.m\ k \end{aligned}$$

- $foldr :: \forall\alpha :: *.\forall m :: *.m \to (\alpha \to m \to m) \to [\alpha] \to m$

- $$\begin{aligned} & vecElim :: \forall\alpha :: *.\forall m :: (\forall k :: \mathsf{Nat}.\mathsf{Vec}\ \alpha\ k \to *). \\ & \qquad m\ Zero\ (Nil\ \alpha) \\ & \quad \to (\forall l :: \mathsf{Nat}.\forall x :: \alpha.\forall xs :: \mathsf{Vec}\ \alpha\ l. \\ & \qquad m\ l\ xs \to m\ (Succ\ l)\ (Cons\ \alpha\ l\ x\ xs)) \\ & \quad \to \forall k :: \mathsf{Nat}.\forall xs :: \mathsf{Vec}\ \alpha\ k.m\ k\ xs \end{aligned}$$

Implement plus and append

- $$\begin{aligned}
&\rangle\!\rangle\ \textbf{let}\ plus = natElim\ (\lambda_ \to \mathsf{Nat} \to \mathsf{Nat})\\
&\qquad\qquad\qquad\qquad (\lambda n \to n)\\
&\qquad\qquad\qquad\qquad (\lambda k\ rec\ n \to Succ\ (rec\ n))\\
&plus :: \forall (x :: \mathsf{Nat})\ (y :: \mathsf{Nat}).\mathsf{Nat}
\end{aligned}$$
- $$\begin{aligned}
&\rangle\!\rangle\ \textbf{let}\ append =\\
&\quad (\lambda\alpha \to vecElim\ \alpha\\
&\qquad\qquad (\lambda m\ _ \to \forall (n :: \mathsf{Nat}).\mathsf{Vec}\ \alpha\ n \to \mathsf{Vec}\ \alpha\ (plus\ m\ n))\\
&\qquad\qquad (\lambda_\ v \to v)\\
&\qquad\qquad (\lambda m\ v\ vs\ rec\ n\ w \to Cons\ \alpha\ (plus\ m\ n)\ v\ (rec\ n\ w)))\\
&\quad :: \forall (\alpha :: *)\ (m :: \mathsf{Nat})\ (v :: \mathsf{Vec}\ \alpha\ m)\ (n :: \mathsf{Nat})\ (w :: \mathsf{Vec}\ \alpha\ n).\\
&\qquad \mathsf{Vec}\ \alpha\ (plus\ m\ n)
\end{aligned}$$

λ_Π: Demo of dependent types

$$
\begin{array}{l}
\rangle\!\rangle\ \textbf{assume}\ (\alpha :: *)\ (x :: \alpha)\ (y :: \alpha) \\
\rangle\!\rangle\ \mathit{append}\ \alpha\ 2\ (\mathit{Cons}\ \alpha\ 1\ x\ (\mathit{Cons}\ \alpha\ 0\ x\ (\mathit{Nil}\ \alpha))) \\
\qquad\qquad\quad 1\ (\mathit{Cons}\ \alpha\ 0\ y\ (\mathit{Nil}\ \alpha)) \\
\mathit{Cons}\ \alpha\ 2\ x\ (\mathit{Cons}\ \alpha\ 1\ x\ (\mathit{Cons}\ \alpha\ 0\ y\ (\mathit{Nil}\ \alpha))) :: \mathsf{Vec}\ \alpha\ 3
\end{array}
$$

Link: `https://www.fpcomplete.com/user/linusyang/types`

Summary and future work

- Bidirectional typing inference helps the implementation of the core language
- Incremental development can simplify the process of adding functions into the core language
- The current language is still rather small and even *unsound*. Consider to add subtyping, recursive types into the type system

References

- Andres, Conor McBride, and Wouter Swierstra. "A tutorial implementation of a dependently typed lambda calculus." Fundamenta informaticae 102.2 (2010): 177-207.
- Jones, Simon Peyton, et al. "Practical type inference for arbitrary-rank types." Journal of functional programming 17.01 (2007): 1-82.
- Dunfield, Joshua, and Neelakantan R. Krishnaswami. "Complete and easy bidirectional typechecking for higher-rank polymorphism." ACM SIGPLAN Notices 48.9 (2013): 429-442.

Questions?

Thank you!