System $\mathbf{F}_i$: a Higher-Order Polymorphic λ-calculus with Erasable Term Indices

Portland State UNIVERSITY

Association for Computing Machinery

Advancing Computing as a Science & Profession

Ki Yung Ahn kya@cs.pdx.edu Department of Computer Science, Portland State University

colloaborators: **Tim Sheard** sheard@pdx.edu
Marcelo Fiore and **Andrew M. Pitts** {Marcelo.Fiore,Andy.Pitts}@cl.cam.ac.uk

1. Indexed Datatypes (Lightweight Dependent Types)

- Indexed datatypes are datatypes with *static (compile-time) dependencies*. Also known as higher-kinded datatypes, higher-rank datatypes, or lightweight dependent types
- Use of indexed datatypes in programming languages, or the *lightweight approach*, has become popular over the past decade even in real-world functional programming due to the GADT extension in the Glasgow Haskell Compiler.

2. Examples of Indexed Datatypes

(c.f.) Regular datatypes `data List a = Cons a (List a) | Nil`

- Type-indexed datatypes (example code in Haskell)
  - Nested datatypes `data Powl a = PCons a (Powl(a,a)) | PNil`
  - Representation types in datatype generic programming

```
data Rep t where
  RInt  :: Rep Int
  RBool :: Rep Bool
  RPair :: Rep a → Rep b → Rep (a,b)
  RFun  :: Rep a → Rep b → Rep (a → b)
```

- Term-indexed datatypes (example code in Nax)
  - Length-indexed lists

```
data Vec (a :: *) {n :: Nat} where
  VCons :: a → Vec a {i} → Vec a {Succ i}
  VNil  :: Vec a {Zero}
```

  - de Bruijn terms indexed by size-indexed contexts

```
data BTerm (c :: Nat → *) {n :: Nat} where
  BVar :: c {i} → BTerm c {i}
  BApp :: BTerm c {i} → BTerm {i} → BTerm c {i}
  BAbs :: BTerm c {Succ i} → BTerm c {i}
```

4. Motivating example: embedding datatypes

- Regular datatypes are embeddable in System **F**

$\texttt{List}\ A \triangleq \forall X.X \to (A \to X \to X) \to X$

- Type-indexed datatypes are embeddable in System $\mathbf{F}_\omega$

$\texttt{Powl} \triangleq \lambda A^{*}.\forall X^{*\to *}.X A \to (A \to X(A \times A) \to X A) \to X A$

- Term-indexed datatypes are embeddable in System $\mathbf{F}_i$

$\texttt{Vec} \triangleq \lambda A^{*}.\lambda i^{\texttt{Nat}}.\forall X^{\texttt{Nat}\to *}.$
$X\{\texttt{Z}\} \to (\forall i^{\texttt{Nat}}.A \to X\{i\} \to X\{\texttt{S}\ i\}) \to X\{i\}$

New features of $\mathbf{F}_i$ not found in $\mathbf{F}_\omega$

- index-arrow kinds
- index abstraction
- index polymorphism
- index application

3. Limitations of the Lightweight approach

Although extending existing programming languages with indexed datatypes has been useful in practice, but it still suffers from the following problems (so far):

- Increase *confidence* but no *guarantee* of correctness

```
loop :: ∀ a . a     -- logically inconsistent type system
loop = loop         -- allows proof of falsity
```

- Faked term indices in implementations (until recently)

```
data Zero        -- code duplication at type level
data Succ n      -- and cannot prevent (Succ Bool)
```

- Type checking/inference may be undecidable/impossible
  - type equality check over term-indexed types relies on term equality, which is undecidable when diverging terms exist
  - need *annotation* for inference, but *how much* and *where*?

System $\mathbf{F}_i$ (details submitted to *POPL '13*) resolves all of above, except where and how much annotations are needed for type inference. Sequel to this work, we are developing Nax (to appear in *IFL '12*), a programming language based on System $\mathbf{F}_i$, which supports type inference from small amount of systematic type annotation.

5. $\mathbf{F}_i$ = Curry-style $\mathbf{F}_\omega$ + {erasable term indices}

Variables x, i Type constructor variables X

Terms r, s, t ::= $x \mid \lambda x.t \mid r\ s$ (Curry-style terms)

Kinds $\kappa ::= * \mid \kappa \to \kappa \mid A \to \kappa$

Type Constructors

$A,B,F,G ::= X \mid A \to B \mid \lambda X^{\kappa}.F \mid F\,G \mid \forall X^{\kappa}.B \mid \lambda i^{A}.F \mid F\,\{s\} \mid \forall i^{A}.B$

Contexts $\Delta ::= \cdot \mid \Delta, X^{\kappa} \mid \Delta, i^{A}$ $\quad \Gamma ::= \cdot \mid \Gamma, x : A$

Typing rules $(:)\dfrac{(x : A) \in \Gamma \quad \Delta \vdash \Gamma}{\Delta;\Gamma \vdash x : A}$ $\quad (:i)\dfrac{i^{A} \in \Delta \quad \Delta \vdash \Gamma}{\Delta;\Gamma \vdash i : A}$

Kinding rules $(@i)\dfrac{\Delta \vdash F : A \to \kappa \quad \Delta;\cdot \vdash s : A}{\Delta \vdash F\,\{s\} : \kappa}$

Index Erasure $(A \to \kappa)^\circ = \kappa^\circ$ $\quad (\Delta, i^{A})^\circ = \Delta^\circ$

$(\lambda i^{A}.F)^\circ = F^\circ$ $\quad (F\,\{s\})^\circ = F^\circ$ $\quad (\forall i^{A}.F)^\circ = F^\circ$

Index Erasure Theorem (for terms without index variables)

$$\dfrac{\Gamma;\Delta \vdash t : A \text{ derivable in } \mathbf{F}_i}{\Delta^\circ;\Gamma^\circ \vdash t : A^\circ \text{ derivable in } \mathbf{F}_\omega}(\mathrm{FV}(t) \cap \mathbf{dom}(\Delta) = \emptyset)$$

- **Strong Normalization**: Index Erasure and Strong Normalization of $\mathbf{F}_\omega$
- **Logical Consistency**: strict subset of a logically consistent calculus

6. Contribution and Ongoing work

- Identifying the features needed for polymorphic λ-calculi to embed term-indexed datatypes in isolation with other requirements
- Design of a calculus useful for studying properties of term-indexed datatypes (e.g., proof using $\mathbf{F}_i$ that the eliminators for indexed datatypes in *Ahn & Sheard, ICFP '11* are indeed normalizing)
- Proof that the calculus enjoys a simple erasure property and inherits metatheoretic results from well-known calculi ($\mathbf{F}_\omega$, ICC)
- Ongoing work: Leibniz equality on term-indices, type inference in Nax, handling non-logical language constructs (ideas from Trellys project)

A poster submitted to the *ACM Student Research Competition* hosted by *the 17th ACM SIGPLAN International Conference on Functional Programming* (*ICFP '12*), Copenhagen, Denmark; September 10-12th, 2012