A True Higher-Order Module System

George Kuan

Dissertation Defense
May 3, 2010

Higher-Order Modules

signature T = **sig type** t **end**
functor Apply(**functor** F(X:T):T) (M:T) = F(M)

Higher-Order Modules

signature T = **sig type** t **end**
functor Apply(**functor** F(X:T):T) (M:T) = F(M)

functor Id(X:**sig type** t **end**) = X
structure N = Apply (**functor** F=Id) (**struct type** t = int **end**)

Higher-Order Modules

signature T = **sig type** t **end**
functor Apply(**functor** F(X:T):T) (M:T) = F(M)

functor Id(X:**sig type** t **end**) = X
structure N = Apply (**functor** F=Id) (**struct type** t = int **end**)

Expect N.t = int

Higher-Order Modules

signature T = **sig type** t **end**
functor Apply(**functor** F(X:T):T) (M:T) = F(M)

functor Id(X:**sig type** t **end**) = X
structure N = Apply (**functor** F=Id) (**struct type** t = int **end**)

Expect N.t = int

functor Const(X:**sig type** t **end**) = **struct type** t = bool **end**
structure R = Apply (**functor** F=Const) (**struct type** t = int **end**)

Higher-Order Modules

signature T = **sig type** t **end**
functor Apply(**functor** F(X:T):T) (M:T) = F(M)

functor Id(X:**sig type** t **end**) = X
structure N = Apply (**functor** F=Id) (**struct type** t = int **end**)

Expect N.t = int

functor Const(X:**sig type** t **end**) = **struct type** t = bool **end**
structure R = Apply (**functor** F=Const) (**struct type** t = int **end**)

Expect R.t = bool

Approaches (1)

1. Syntactic (Applicative Functors [Leroy 1995])
   functor Apply(functor F(X:T):T) (M:T)
   : sig type t= F(M).t end = F(M)

Approaches (1)

1. Syntactic (Applicative Functors [Leroy 1995])
   functor Apply(functor F(X:T):T) (M:T)
   : sig type t= F(M).t end = F(M)

   can only be treated superficially

Approaches (1)

1. Syntactic (Applicative Functors [Leroy 1995])
   functor Apply(functor F(X:T):T) (M:T)
   : sig type t= F(M).t end = F(M)
   can only be treated superficially
2. Semantic [MacQueen-Tofte 1994]
   functor Apply(functor F(X:T):T) (M:T)
   : sig type t end = F(M)

   Dependence of result type t on F and M is inferred by the compiler

Approaches (2)

```
functor F(functor G(X:T):T) =
struct
  datatype s = S of int
  structure M = G(struct type t = s end)
  type u = M.t
end
```

Approaches (2)

functor F(**functor** G(X:T):T) =
struct
  datatype s = S **of** int
  structure M = G(**struct** **type** t = s **end**)
  type u = M.t
end

Syntactic approach breaks down.

Approaches (2)

```
functor F(functor G(X:T):T) =
struct
  datatype s = S of int
  structure M = G(struct type t = s end)
  type u = M.t
end
```

Syntactic approach breaks down.
A descriptive signature would have to involve static effects and the actions taken by formal functor G.

True Higher-Order Semantics

Functor Action

A function action is the way in which a functor computes its output types from its parameter types, namely:

1. type generativity
2. functor actions of formal functors

Motivation

Syntactic Approaches

All module type information is syntactic

1. Give up non-syntactic module type information
2. Try to express more module type information syntactically

Semantic Approach

Some module type information is semantic (functor actions)

Motivation

1. Restricting to syntactic module types is analogous to restricting a language to λ-terms where each λ is given a really powerful dependent type that computes the result of the λ
2. An abstract model of current SML/NJ implementation of higher-order modules
3. A more detailed and realistic expansion of MacQueen-Tofte 1994 and Shao 1998
4. True higher-order semantics without re-elaboration

Outline

1 Type of a Structure

2 Entity Calculus

3 Elaboration

4 Soundness

Type of a Structure

Big Question: What is the "type" of a structure?

Signatures

structure A = struct
  datatype α t = c of α
  structure M = struct datatype t = d val x = c d end
end

Syntactic Signature

sig
  type α t
  structure M : sig type t val x: ?? end
end

Signatures

structure A = struct
  datatype α t = c of α
  structure M = struct datatype t = d val x = c d end
end

Semantic Signature

sig
  type α t ρ_0
  structure M ρ_M: sig type t ρ_1 val x: $\rho_0(\rho_1)$ end
end

Because type names can shadow, syntactic names are insufficient

Signatures

structure A = struct
  datatype α t = c of α
  structure M = struct datatype t = d val x = c d end
end

Semantic Signature

sig
  type α t ρ_0
  structure M ρ_M: sig type t ρ_1 val x: $\rho_0(\rho_1)$ end
end

Need unshadowable entity variables (*aka* internal names [Harper-Lillibridge 94]) and entity paths (*e.g.*, $\rho_M \rho_1$)

Signatures

structure A = struct
  datatype α t = c of α
  structure M = struct datatype t = d val x = c d end
end

Semantic Signature

sig
  type α t ρ_0
  structure M ρ_M: sig type t ρ_1 val x: $\rho_0(\rho_1)$ end
end

relativized types

Signatures

structure A = struct
  datatype α t = c of α
  structure M = struct datatype t = d val x = c d end
end

Semantic Signature

sig
  type α t ρ_0
  structure M ρ_M: sig type t ρ_1 val x: $\rho_0(\rho_1)$ end
end

Abbreviated:

$$\left\{ \begin{array}{l} t : (\rho_0, 1), \\ M : (\rho_M, \{t : (\rho_1, 0),\ x : \rho_0(\rho_1)\}) \end{array} \right\}$$

Type of a Structure

```
functor F() = struct
  structure M = struct datatype t end
  val x : M.t = ...
end

structure A = F()
```

What is the module "type" for A?

A semantic signature is not the complete module "type".

Type of a Structure

```
functor F() = struct
  structure M = struct datatype t end
  val x : M.t = ...
end

structure A = F()
```

What is the module “type” for A?

$\{\ M : (\rho_M, \{t : (\rho_t, 0)\}),\ x : \rho_M \rho_t\ \}$

Need an entity environment mapping entity variables to static entities (tycons, structures, and functors) $\{\rho_M \mapsto \{\rho_t \mapsto \tau^0\}\}$

τ^0 is a fresh atomic semantic tycon

Entity Environments

```
functor F() = struct
  structure M = struct datatype t end
  val x : M.t = ...
end
```

structure A = F() $\quad\quad \rho_A \mapsto \{\rho_M \mapsto \{\rho_t \mapsto \tau_a^0\}\}$

Entity Environments

```
functor F() = struct
  structure M = struct datatype t end
  val x : M.t = ...
end
```

structure A = F() $\qquad \rho_A \mapsto \{\rho_M \mapsto \{\rho_t \mapsto \tau_a^0\}\}$

structure B = F() $\qquad \rho_B \mapsto \{\rho_M \mapsto \{\rho_t \mapsto \tau_b^0\}\}$

Each time F is applied, we get a fresh atomic tycon

Full Signature

signature (fixed) realization (volatile)

$$\{\ M : (\rho_M, \{t : (\rho_t, 0)\}),\ x : \rho_M \rho_t\ \} + \{\rho_M \mapsto \{\rho_t \mapsto \tau^0\}\}$$

Full Signature

signature (fixed) realization (volatile)

$$\{\ M : (\rho_M, \{t : (\rho_t, 0)\}),\ x : \rho_M \rho_t\ \} + \{\rho_M \mapsto \{\rho_t \mapsto \tau^0\}\}$$

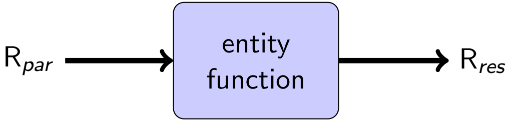
$$\text{M.t} \xrightarrow{\text{signature}} \rho_M \rho_t \xrightarrow{\text{entity env}} \tau^0$$

Type of a Structure

But what is a functor entity?

Type of a Structure

But what is a functor entity?



Entity Calculus (1)

```
datatype v
functor F(X:sig type t end) = struct
  datatype (α, β) u
  type s = (X.t,v) u
end
```

Entity Calculus (1)

```
datatype v
functor F(X:sig type t end) = struct
  datatype (α, β) u
  type s = (X.t,v) u
end
```

A **tycon entity** is either

- an atomic tycon (*e.g.*, τ_u^2)
- or normal form semantic tycon (*e.g.*, $\lambda().\tau_u^2(\tau_t^0, \tau_v^0)$)

Entity Calculus (1)

```
datatype v
functor F(X:sig type t end) = struct
  datatype (α, β) u
  type s = (X.t,v) u
end
```

A **structure entity** R is a pair of entity environments
$\langle\{\rho_u \mapsto \tau_u^2,\ \rho_s \mapsto \lambda().\tau_u^2(\tau_t^0, \tau_v^0)\},\ \ \{\rho_v \mapsto \tau_v^0\}\rangle$

a local one defining all entities in the structure and a closure environment

τ_t^0 is a dummy atomic tycon to stand in for the tycon in the functor argument

Entity Calculus (1)

```
datatype v
functor F(X:sig type t end) = struct
  datatype (α, β) u
  type s = (X.t,v) u
end
```

A **functor entity** is a closure: a λ-expression mapping structure entity to an expression that evaluates to a structure entity
$\lambda\rho_x.\{\rho_u = \mathrm{new}(2), \rho_s = \lambda().\rho_u(\rho_x\rho_t, \rho_v)\} + \{\rho_v \mapsto \tau_v^0\}$

Entity Calculus (2)

Tycon entity expression:

$$\zeta ::= \mathsf{new}(\mathit{n}) \mid \mathbb{C}^{\lambda}(\text{relativized tycons})$$

Structure entity expression:

$$\varphi ::= \vec{\rho} \mid \{\eta\} \mid \theta(\varphi) \mid \mathsf{let}\ \eta\ \mathsf{in}\ \varphi$$

Functor entity expression:

$$\theta ::= \vec{\rho} \mid \lambda\rho.\varphi \mid \lambda\rho.\Sigma$$

Entity declaration:

$$\eta ::= \circ \mid \rho = \zeta, \eta \mid \rho = \varphi, \eta \mid \rho = \theta, \eta$$

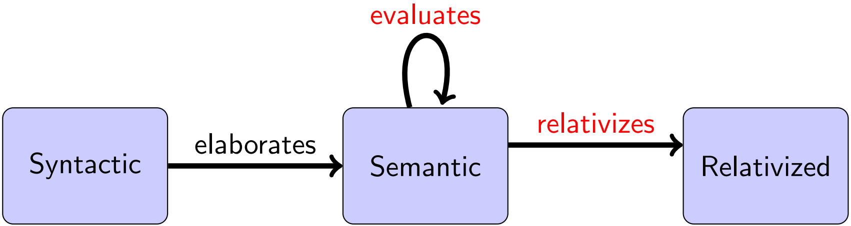
Life-Cycle of a Type

Syntactic

Life-Cycle of a Type

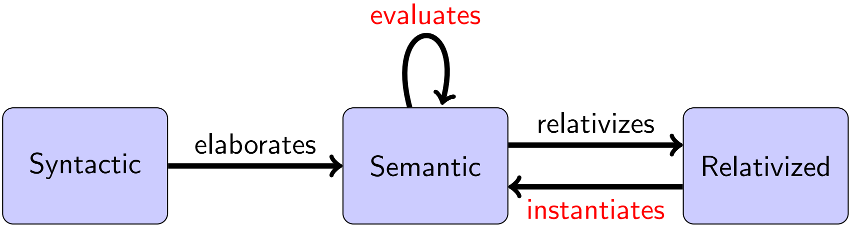
static env

Life-Cycle of a Type



static env	semantic sig
entity env	entity expr

Life-Cycle of a Type



static env

semantic sig

entity env

entity expr

Functor Actions

```
functor Apply(X:sig functor F(Y:sig type t end):sig type t end
                    structure M: sig type t end
                end) =
struct structure R = X.F(X.M) end
```

Functor Actions

functor Apply(X:**sig functor** F(Y:**sig type** t **end**):**sig type** t **end**
structure M: **sig type** t **end**
end) =
struct structure R = X.F(X.M) **end**

$$\lambda\rho_x.\{\rho_r = \rho_x\rho_f(\rho_x\rho_m)\}$$

Functor Actions

functor Apply(X:**sig functor** F(Y:**sig type** t **end**):**sig type** t **end**
structure M: **sig type** t **end**
end) =
struct structure R = X.F(X.M) **end**

$$\lambda\rho_x.\{\rho_r = \rho_x\rho_f(\rho_x\rho_m)\}$$

functor G() = **struct datatype** t **end**

Functor Actions

functor Apply(X:**sig functor** F(Y:**sig type** t **end**):**sig type** t **end**
structure M: **sig type** t **end**
end) =
struct structure R = X.F(X.M) **end**

$$\lambda\rho_x.\{\rho_r = \rho_x\rho_f(\rho_x\rho_m)\}$$

functor G() = **struct datatype** t **end**

$$\lambda().\{\rho_t = new(0)\}$$

Full Functor Signature

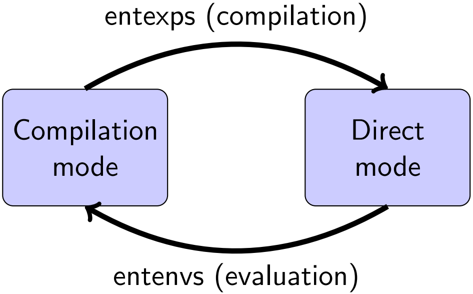
datatype v
functor F() = **struct** **datatype** t **val** a : v **end**

Semantic functor signature + Entity function (closure)

$\Pi().\{t : (\rho_t, 0),\ a : \rho_v\}$ $\langle \lambda().\{\rho_t = \mathrm{new}(0)\},\ \{\rho_v \mapsto \tau^0\}\rangle$

Constructing Full Signatures and Entity Expressions

Full signatures and entity expressions are produced by elaboration in two interweaving modes:



Structure Elaboration

$\Gamma, \Upsilon \vdash \mathit{strexp} \Rightarrow_{\mathit{str}} (M, \varphi)$

1. Γ is the static environment mapping symbols to semantic tycons, full signatures, full functor signatures

Structure Elaboration

$\Gamma, \Upsilon \vdash \mathit{strexp} \Rightarrow_{str} (M, \varphi)$

1. Γ is the static environment mapping symbols to semantic tycons, full signatures, full functor signatures
2. Υ is the entity environment

Structure Elaboration

$\Gamma, \Upsilon \vdash \mathit{strexp} \Rightarrow_{\mathit{str}} (M, \varphi)$

1. Γ is the static environment mapping symbols to semantic tycons, full signatures, full functor signatures
2. Υ is the entity environment
3. Full signature $M = \langle \Sigma, R \rangle$ where Σ is the semantic signature and R is the structure entity

Structure Elaboration

$\Gamma, \Upsilon \vdash \mathit{strexp} \Rightarrow_{\mathit{str}} (M, \varphi)$

1. Γ is the static environment mapping symbols to semantic tycons, full signatures, full functor signatures
2. Υ is the entity environment
3. Full signature $M = \langle \Sigma, R \rangle$ where Σ is the semantic signature and R is the structure entity
4. Structure entity expression φ: Evaluates to an R' isomorphic to R under the current entity environment Υ.

Functor Application

$$\frac{\begin{array}{c}\Gamma(p) = (\vec{\rho}, (\Pi X : \Sigma_{par}.\Sigma_{body}, \langle \theta, \Upsilon' \rangle)) \\ \Gamma, \Upsilon \vdash strexp \Rightarrow_{str} (M, \varphi) \\ \Upsilon \vdash (M, \varphi) : \Sigma_{par} \Rightarrow_{match} (R_c, \varphi_c) \\ \varphi_{app} = \theta(\varphi_c) \qquad \Upsilon'\Upsilon \vdash \varphi_{app} \Downarrow_{str} R_{app}\end{array}}{\Gamma, \Upsilon \vdash p(strexp) \Rightarrow_{str} ((\Sigma_{body}, R_{app}), \varphi_{app})}$$

1. Lookup symbolic path p in static environment

Functor Application

$$\frac{\begin{array}{c}\Gamma(p) = (\vec{\rho}, (\Pi X : \Sigma_{par}.\Sigma_{body}, \langle \theta, \Upsilon' \rangle)) \\ \Gamma, \Upsilon \vdash strexp \Rightarrow_{str} (M, \varphi) \\ \Upsilon \vdash (M, \varphi) : \Sigma_{par} \Rightarrow_{match} (R_c, \varphi_c) \\ \varphi_{app} = \theta(\varphi_c) \qquad \Upsilon'\Upsilon \vdash \varphi_{app} \Downarrow_{str} R_{app}\end{array}}{\Gamma, \Upsilon \vdash p(strexp) \Rightarrow_{str} ((\Sigma_{body}, R_{app}), \varphi_{app})}$$

1. Lookup symbolic path p in static environment
2. Elaborate argument

Functor Application

$$\frac{\begin{array}{c}\Gamma(p) = (\vec{\rho}, (\Pi X : \Sigma_{par}.\Sigma_{body}, \langle \theta, \Upsilon' \rangle)) \\ \Gamma, \Upsilon \vdash strexp \Rightarrow_{str} (M, \varphi) \\ \Upsilon \vdash (M, \varphi) : \Sigma_{par} \Rightarrow_{match} (R_c, \varphi_c) \\ \varphi_{app} = \theta(\varphi_c) \qquad \Upsilon'\Upsilon \vdash \varphi_{app} \Downarrow_{str} R_{app}\end{array}}{\Gamma, \Upsilon \vdash p(strexp) \Rightarrow_{str} ((\Sigma_{body}, R_{app}), \varphi_{app})}$$

1. Lookup symbolic path p in static environment
2. Elaborate argument
3. Coerce argument to formal parameter form

Functor Application

$$\frac{\begin{array}{c}\Gamma(p) = (\vec{\rho}, (\Pi X : \Sigma_{par}.\Sigma_{body}, \langle \theta, \Upsilon' \rangle)) \\ \Gamma, \Upsilon \vdash strexp \Rightarrow_{str} (M, \varphi) \\ \Upsilon \vdash (M, \varphi) : \Sigma_{par} \Rightarrow_{match} (R_c, \varphi_c) \\ \varphi_{app} = \theta(\varphi_c) \qquad \Upsilon'\Upsilon \vdash \varphi_{app} \Downarrow_{str} R_{app}\end{array}}{\Gamma, \Upsilon \vdash p(strexp) \Rightarrow_{str} ((\Sigma_{body}, R_{app}), \varphi_{app})}$$

1. Lookup symbolic path p in static environment
2. Elaborate argument
3. Coerce argument to formal parameter form
4. Form entity expression

Functor Application

$$\frac{\begin{array}{c}\Gamma(p) = (\vec{\rho}, (\Pi X : \Sigma_{par}.\Sigma_{body}, \langle \theta, \Upsilon' \rangle)) \\ \Gamma, \Upsilon \vdash strexp \Rightarrow_{str} (M, \varphi) \\ \Upsilon \vdash (M, \varphi) : \Sigma_{par} \Rightarrow_{match} (R_c, \varphi_c) \\ \varphi_{app} = \theta(\varphi_c) \qquad \Upsilon'\Upsilon \vdash \varphi_{app} \Downarrow_{str} R_{app}\end{array}}{\Gamma, \Upsilon \vdash p(strexp) \Rightarrow_{str} ((\Sigma_{body}, R_{app}), \varphi_{app})}$$

1. Lookup symbolic path p in static environment
2. Elaborate argument
3. Coerce argument to formal parameter form
4. Form entity expression
5. Evaluate entity expression (no re-elaboration of functor)

Signature Matching

$$\Upsilon \vdash ((\Sigma_a, R_a), \varphi) : \Sigma_s \Rightarrow_{match} (R_c, \varphi_c)$$

1. Coerces full signature (Σ_a, R_a) to form of spec Σ_s and produces a coerced structure entity expression φ_c from φ

Signature Matching

$$\Upsilon \vdash ((\Sigma_a, R_a), \varphi) : \Sigma_s \Rightarrow_{match} (R_c, \varphi_c)$$

1. Coerces full signature (Σ_a, R_a) to form of spec Σ_s and produces a coerced structure entity expression φ_c from φ
2. Fill in (*i.e.*, instantiate) open tycons with actuals in R_a

Signature Matching

$$\Upsilon \vdash ((\Sigma_a, R_a), \varphi) : \Sigma_s \Rightarrow_{match} (R_c, \varphi_c)$$

1. Coerces full signature (Σ_a, R_a) to form of spec Σ_s and produces a coerced structure entity expression φ_c from φ
2. Fill in (*i.e.*, instantiate) open tycons with actuals in R_a
3. Verify type definitional specs

Signature Matching

$$\Upsilon \vdash ((\Sigma_a, R_a), \varphi) : \Sigma_s \Rightarrow_{match} (R_c, \varphi_c)$$

1. Coerces full signature (Σ_a, R_a) to form of spec Σ_s and produces a coerced structure entity expression φ_c from φ
2. Fill in (*i.e.*, instantiate) open tycons with actuals in R_a
3. Verify type definitional specs
4. Functor signature matching

Signature Matching

$$\Upsilon \vdash ((\Sigma_a, R_a), \varphi) : \Sigma_s \Rightarrow_{match} (R_c, \varphi_c)$$

1. Coerces full signature (Σ_a, R_a) to form of spec Σ_s and produces a coerced structure entity expression φ_c from φ
2. Fill in (*i.e.*, instantiate) open tycons with actuals in R_a
3. Verify type definitional specs
4. Functor signature matching
5. Construct a coercion that rebinds actual ρ's to spec variables
   actual: $((\{t : (\rho'_t, 0)\},\ \langle\{\rho'_t \mapsto \tau^0\}, \emptyset\}),\ \ \{\rho'_t = \text{new}(0)\})$
   spec: $\{t : (\rho_t, 0)\}$
   coercion: let $\rho_{raw} = \{\rho'_t = \text{new}(0)\}$ in $\{\rho_t = \rho_{raw}\rho'_t\}$

Signature Matching (2)

If $\Upsilon \vdash ((\Sigma_a, R_a), \varphi) : \Sigma_s \Rightarrow_{match} (R_c, \varphi_c)$, then for all $x : s \in \Sigma_s$, there exists $x : s' \in \Sigma_a$ such that $R_c(s) = R_a(s')$.

Signature Matching (2)

If $\Upsilon \vdash ((\Sigma_a, R_a), \varphi) : \Sigma_s \Rightarrow_{match} (R_c, \varphi_c)$, then for all $x : s \in \Sigma_s$, there exists $x : s' \in \Sigma_a$ such that $R_c(s) = R_a(s')$.

If $\Upsilon \vdash ((\Sigma_a, R_a), \varphi) : \Sigma_s \Rightarrow_{match} (R_c, \varphi_c)$, then $\Upsilon \vdash \varphi_c \Downarrow_{str} R'$ such that R' and R_c are isomorphic.

Other Elaboration Semantics

1. Base structure: extract a semantic signature from a static environment by *relativizing* types/type expressions (also relevant during signature elaboration)

Other Elaboration Semantics

1. Base structure: extract a semantic signature from a static environment by *relativizing* types/type expressions (also relevant during signature elaboration)
2. Functor declaration and opaque ascription: instantiation of formal parameter

Translation

1. To show soundness, use a translation to System F_ω

Translation

1. To show soundness, use a translation to System F_ω
2. A translation of elaborated module language to a standard System F_ω enriched with records and new

Translation

1. To show soundness, use a translation to System F_ω
2. A translation of elaborated module language to a standard System F_ω enriched with records and new
3. Factors structures into static and value parts (phase separation [Harper, Mitchell, Moggi 1990])

Translation

1. To show soundness, use a translation to System F_ω
2. A translation of elaborated module language to a standard System F_ω enriched with records and new
3. Factors structures into static and value parts (phase separation [Harper, Mitchell, Moggi 1990])
4. Constructs type- and value-level coercions (signature matching)

Translation

```
functor F(X:sig type s end) =
struct
  datatype w
  val n : w -> w = fn z : w => z
end
```

let tyc $\widehat{f} = \lambda\widehat{x} :: \{s :: \Omega\}.\{w = \text{new}(0)\}$
in let $f = \Lambda\widehat{x} :: \{s :: \Omega\}.\Lambda\widehat{f_{res}} :: \{w :: \Omega\}.\lambda x :: \{\}.\{n = \lambda z : (\widehat{f_{res}}.w).z\}$
in ...

$\widehat{\bullet}$ indicates type part of $\bullet$.

(Relative) Soundness

$$\begin{tikzcd}
\text{strexp} \arrow[r, "\text{translation}"] \arrow[d, "\text{elaboration}"] & e_\omega \arrow[d, "F_\omega\ \text{typing}"] \\
M \arrow[r, "\text{type translation}"] & t
\end{tikzcd}$$

Proof: Induction on a strengthened version of the above. The proof depends on the correctness of type and value coercion, signature matching, and signature instantiation.

Main Ideas

1. Factoring module type information (full signature) into semantic signature and realization
2. Entity calculus encodes functor actions
3. Elaboration semantics in compilation and direct modes
4. Coercive signature matching

Related Work: Syntactic Approach

- CMU
  - Harper-Lillibridge 1994, Harper-Stone 1997
  - Dreyer-Crary-Harper 2003
  - Harper-Pierce 2005
  - Dreyer 2005, 2007
  - Dreyer-Blume 2007
  - Dreyer-Rossberg 2009
  - Rossberg-Russo-Dreyer 2010
- Leroy 1994, 1995, 1996, 2000
- Biswas 1995, Russo 2000
- Shao 1998, 1999
- Govereau 2005
- Montagu and Rémy 2009

Related Work: Semantic Approach

- MacQueen-Tofte 1994
- Crégut and MacQueen 1994
- Shao 1998
- Kuan and MacQueen 2009

Future Work

1. Relationship to type classes
2. Exceptions and modules
3. Type inference and modules

Conclusion

- HO module semantics is analogous to β-reduction semantics
- Module types are and should be semantic
- One neither has to give up generative datatypes/functors nor true higher-order semantics for a practical semantics

Thank You

Syntactic Signature

Syntactic signatures are comprised of specs:

type (α, β) t

type s = int

structure M : sig type u end

val a : (M.u, s) t

functor F(X:T) : T

Syntactic Signature

Syntactic signatures are comprised of specs:

type (α, β) t — open tycon

type s = int

structure M : sig type u end

val a : (M.u, s) t

functor F(X:T) : T

Syntactic Signature

Syntactic signatures are comprised of specs:

type (α, β) t	open tycon
type s = int	type definition
structure M : sig type u end	
val a : (M.u, s) t	
functor F(X:T) : T	

Syntactic Signature

Syntactic signatures are comprised of specs:

type (α, β) t	open tycon
type s = int	type definition
structure M : sig type u end	structure
val a : (M.u, s) t	
functor F(X:T) : T	

Syntactic Signature

Syntactic signatures are comprised of specs:

type (α, β) t	open tycon
type s = int	type definition
structure M : sig type u end	structure
val a : (M.u, s) t	value
functor F(X:T) : T	

Syntactic Signature

Syntactic signatures are comprised of specs:

type (α, β) t	open tycon
type s = int	type definition
structure M : sig type u end	structure
val a : (M.u, s) t	value
functor F(X:T) : T	functor

Syntactic Signature

Syntactic signatures are comprised of specs:

type (α, β) t	open tycon
type s = int	type definition
structure M : sig type u end	structure
val a : (M.u, s) t	value
functor F(X:T) : T	functor

Syntactic Signature

Syntactic signatures are comprised of specs:

type (α, β) t	open tycon
type s = int	type definition
structure M : sig type u end	structure
val a : (M.u, s) t	value
functor F(X:T) : T	functor

Syntactic Signature

Syntactic signatures are comprised of specs:

type (α, β) t	open tycon
type s = int	type definition
structure M : sig type u end	structure
val a : (M.u, s) t	value
functor F(X:T) : T	functor

Syntactic Signature

Syntactic signatures are comprised of specs:

type (α, β) t	open tycon
type s = int	type definition
structure M : sig type u end	structure
val a : (M.u, s) t	value
functor F(X:T) : T	functor

Relativization

```
functor f() =
struct
  datatype t = S of int
  type u = t
  val x : u = S 1
end
```

How to represent u in the signature?

$$\lambda().\{\rho_t = \mathsf{new}(0), \rho_u = \rho_t\}$$

$$x : \rho_u$$

Relativization

1. Value and definitional tycon bindings must be relativized
2. Look up first occurrence of atomic tycons in entity environment, the entity path mapping to that occurrence is the canonical entity path
3. Replace atomic tycons with canonical entity path, entity paths which always point to the current instantiation given the current entity environment

Signature Extraction

struct
datatype t
structure A = **struct type** u = t **end**
functor F(X:**sig type** s **end**) = **struct type** v = X.s **end**
end

$$
\begin{aligned}
&t \mapsto (\rho_0, \tau^0)\\
&A \mapsto \{u \mapsto (\rho_1, \rho_0)\}\\
&F \mapsto (\rho_2, \Pi\rho_3 : \{s \mapsto (\rho_4, 0)\}.\{v \mapsto \lambda().\rho_3\rho_4\})
\end{aligned}
$$

Elaboration

- $\Gamma, \Upsilon, \Sigma \vdash \textit{sigexp} \Rightarrow_{sig} \Sigma'$ signature elaboration

Elaboration

- $\Gamma, \Upsilon, \Sigma \vdash \mathit{sigexp} \Rightarrow_{sig} \Sigma'$ signature elaboration
- $\Gamma, \Upsilon, \Sigma \vdash \mathit{fsgexp} \Rightarrow_{fsg} \Sigma^f$ functor signature elaboration

Elaboration

- $\Gamma, \Upsilon, \Sigma \vdash sigexp \Rightarrow_{sig} \Sigma'$ signature elaboration
- $\Gamma, \Upsilon, \Sigma \vdash fsgexp \Rightarrow_{fsg} \Sigma^f$ functor signature elaboration
- $\Upsilon^{clo}, \Upsilon^{lcl} \vdash \Sigma \uparrow \Upsilon^{lcl}$ signature instantiation

Elaboration

- $\Gamma, \Upsilon, \Sigma \vdash \mathit{sigexp} \Rightarrow_{\mathit{sig}} \Sigma'$ signature elaboration
- $\Gamma, \Upsilon, \Sigma \vdash \mathit{fsgexp} \Rightarrow_{\mathit{fsg}} \Sigma^{f}$ functor signature elaboration
- $\Upsilon^{\mathit{clo}}, \Upsilon^{\mathit{lcl}} \vdash \Sigma \uparrow \Upsilon^{\mathit{lcl}}$ signature instantiation
- $\Upsilon \vdash \Gamma \hookrightarrow \Sigma$ signature extraction

Elaboration

- $\Gamma, \Upsilon, \Sigma \vdash \mathit{sigexp} \Rightarrow_{\mathit{sig}} \Sigma'$ signature elaboration
- $\Gamma, \Upsilon, \Sigma \vdash \mathit{fsgexp} \Rightarrow_{\mathit{fsg}} \Sigma^{f}$ functor signature elaboration
- $\Upsilon^{\mathit{clo}}, \Upsilon^{\mathit{lcl}} \vdash \Sigma \uparrow \Upsilon^{\mathit{lcl}}$ signature instantiation
- $\Upsilon \vdash \Gamma \hookrightarrow \Sigma$ signature extraction
- $\Upsilon \vdash (M, \varphi) : \Sigma \Rightarrow_{\mathit{match}} (M_c, \varphi_c)$ signature matching

Elaboration

- $\Gamma, \Upsilon, \Sigma \vdash \mathit{sigexp} \Rightarrow_{sig} \Sigma'$ signature elaboration
- $\Gamma, \Upsilon, \Sigma \vdash \mathit{fsgexp} \Rightarrow_{fsg} \Sigma^f$ functor signature elaboration
- $\Upsilon^{clo}, \Upsilon^{lcl} \vdash \Sigma \uparrow \Upsilon^{lcl}$ signature instantiation
- $\Upsilon \vdash \Gamma \hookrightarrow \Sigma$ signature extraction
- $\Upsilon \vdash (M, \varphi) : \Sigma \Rightarrow_{match} (M_c, \varphi_c)$ signature matching
- $\Upsilon \vdash F \preceq \Sigma^f \Rightarrow_{fsgmtch} (\psi_c, \theta_c)$ functor signature matching

Elaboration

- $\Gamma, \Upsilon, \Sigma \vdash \mathit{sigexp} \Rightarrow_{sig} \Sigma'$ signature elaboration
- $\Gamma, \Upsilon, \Sigma \vdash \mathit{fsgexp} \Rightarrow_{fsg} \Sigma^f$ functor signature elaboration
- $\Upsilon^{clo}, \Upsilon^{lcl} \vdash \Sigma \uparrow \Upsilon^{lcl}$ signature instantiation
- $\Upsilon \vdash \Gamma \hookrightarrow \Sigma$ signature extraction
- $\Upsilon \vdash (M, \varphi) : \Sigma \Rightarrow_{match} (M_c, \varphi_c)$ signature matching
- $\Upsilon \vdash F \preceq \Sigma^f \Rightarrow_{fsgmtch} (\psi_c, \theta_c)$ functor signature matching
- $\Gamma, \Upsilon \vdash d^m \Rightarrow_{decl} (\eta, \Gamma', \Upsilon')$ module declaration elaboration

Elaboration

- $\Gamma, \Upsilon, \Sigma \vdash \mathit{sigexp} \Rightarrow_{sig} \Sigma'$ signature elaboration
- $\Gamma, \Upsilon, \Sigma \vdash \mathit{fsgexp} \Rightarrow_{fsg} \Sigma^f$ functor signature elaboration
- $\Upsilon^{clo}, \Upsilon^{lcl} \vdash \Sigma \uparrow \Upsilon^{lcl}$ signature instantiation
- $\Upsilon \vdash \Gamma \hookrightarrow \Sigma$ signature extraction
- $\Upsilon \vdash (M, \varphi) : \Sigma \Rightarrow_{match} (M_c, \varphi_c)$ signature matching
- $\Upsilon \vdash F \preceq \Sigma^f \Rightarrow_{fsgmtch} (\psi_c, \theta_c)$ functor signature matching
- $\Gamma, \Upsilon \vdash d^m \Rightarrow_{decl} (\eta, \Gamma', \Upsilon')$ module declaration elaboration
- $\Gamma, \Upsilon \vdash \mathit{strexp} \Rightarrow_{str} (M, \varphi)$ structure expression elaboration