OhML

(The "Zen Essence" of Methods-as-Functions)

(e.m) ⇝ (m e)

Arnaud Spiwack

ENS Cachan

joint work with

Luigi Liquori

INRIA Sophia

OhML

Motivations

Modeling Method Call via Function Application

- A modular integer would feature:
  - ⋆ An implicit type: `int`
  - ⋆ An addition function `plus` : `int->int->int`
  - ⋆ Addition will be performed like: `plus 3 5`

Modeling Method Call via Function Application

- A modular integer would feature:
  - ⋆ An implicit type: `int`
  - ⋆ An addition function `plus` : `int->int->int`
  - ⋆ Addition will be performed like: `plus 3 5`

- An objective integer would be:
  - ⋆ A class: `int`
  - ⋆ Featuring an addition method `plus` : `int->int`
  - ⋆ Addition will be performed like: `3.plus 5`
  - ⋆ Moreover `plus` is not "first-class", *i.e.* it cannot passed as an argument or be a result of a call

Modeling Method Call via Function Application

- A modular integer would feature:
  - ⋆ An implicit type: `int`
  - ⋆ An addition function `plus` : `int->int->int`
  - ⋆ Addition will be performed like: `plus 3 5`

- An objective integer would be:
  - ⋆ A class: `int`
  - ⋆ Featuring an addition method `plus` : `int->int`
  - ⋆ Addition will be performed like: `3.plus 5`
  - ⋆ Moreover `plus` is not "first-class", *i.e.* it cannot passed as an argument or be a result of a call

- Though they have the same semantics, they have a radically different syntax. And functions designed to work with one do not work straightforwadly with the other

Simple, Uniform, and Powerful

- Simple functional calculus *à la* ML with fixpoint, simple objects with `this`, no inheritance and message sending via functional application ... this means that for
- $$\texttt{point} \triangleq \texttt{obj this} \texttt{ -> data x} = 1 \texttt{ data getx } = (\texttt{x this})$$

  ... we have that:

$$\begin{aligned}
(\texttt{fun z -> (z point)})\ \underline{((\texttt{fun z -> z})\ \texttt{getx})} &\rightarrow \underline{(\texttt{fun z -> (z point)})\ \texttt{getx}} \\
&\rightarrow \underline{(\texttt{getx point})} \\
&\rightarrow \underline{(\texttt{x point})} \\
&\rightarrow 1
\end{aligned}$$

Simple, Uniform, and Powerful

- Simple functional calculus *à la* ML with fixpoint, simple objects with `this`, no inheritance and message sending via functional application ... this means that for
- $$\texttt{point} \triangleq \texttt{obj this} \texttt{ -> data x} = 1 \texttt{ data getx } = (\texttt{x this})$$

  ... we have that:

$$\begin{array}{rcl} (\texttt{fun z -> (z point)})\ \underline{((\texttt{fun z -> z})\ \texttt{getx})} & \rightarrow & \underline{(\texttt{fun z -> (z point)})\ \texttt{getx}} \\ & \rightarrow & \underline{(\texttt{getx point})} \\ & \rightarrow & \underline{(\texttt{x point})} \\ & \rightarrow & 1 \end{array}$$

- A sound (`message-not-found` preventing) first-order type system with width subtyping

Simple, Uniform, and Powerful

- Simple functional calculus *à la* ML with fixpoint, simple objects with `this`, no inheritance and message sending via functional application ... this means that for
- $$\texttt{point} \triangleq \texttt{obj this} \texttt{ -> data x} = 1 \texttt{ data getx } = (\texttt{x this})$$

  ... we have that:

$$\begin{array}{rcl} (\texttt{fun z -> (z point)})\ \underline{((\texttt{fun z -> z})\ \texttt{getx})} & \rightarrow & \underline{(\texttt{fun z -> (z point)})\ \texttt{getx}} \\ & \rightarrow & \underline{(\texttt{getx point})} \\ & \rightarrow & \underline{(\texttt{x point})} \\ & \rightarrow & 1 \end{array}$$

- A sound (`message-not-found` preventing) first-order type system with width subtyping
- Decidable (?) type inference (plug'n play with Damas-Milner W)

Summarizing

- An alternative to row-polymorphism *à la* Wand, just record-types and simple subtyping, no polymorphism (yet)
- Method names are first-class citizens, *i.e.* they can be passed as function arguments (in a single dispatch setting)

$$(\texttt{fun x -> x})\ \texttt{getx} \mapsto \texttt{getx}$$

Summarizing

- An alternative to row-polymorphism *à la* Wand, just record-types and simple subtyping, no polymorphism (yet)
- Method names are first-class citizens, *i.e.* they can be passed as function arguments (in a single dispatch setting)

$$(\texttt{fun x -> x})\ \texttt{getx} \mapsto \texttt{getx}$$

- Method call reduced to function application (plus a method lookup of course!)

$$\texttt{m}\ (\texttt{obj x -> o}) \mapsto \mathit{mbody}(\texttt{o}\,;\texttt{m})[\texttt{obj x -> o}/\texttt{x}]$$

- Inheritance is out of the scope of this paper; however, it would not be difficult to add an object-based inheritance
- A decidable type inference algorithm is our current challenge

Summarizing

- An alternative to row-polymorphism *à la* Wand, just record-types and simple subtyping, no polymorphism (yet)
- Method names are first-class citizens, *i.e.* they can be passed as function arguments (in a single dispatch setting)

$$(\texttt{fun x -> x})\ \texttt{getx} \mapsto \texttt{getx}$$

- Method call reduced to function application (plus a method lookup of course!)

$$\texttt{m}\ (\texttt{obj x -> o}) \mapsto \mathit{mbody}(\texttt{o}\,;\texttt{m})[\texttt{obj x -> o}/\texttt{x}]$$

- Inheritance is out of the scope of this paper; however, it would not be difficult to add an object-based inheritance
- A decidable type inference algorithm is our current challenge
- OhML use the most economic type theory for a rich list of features:
  Great value for money guaranteed! :-)

OhML

The Syntax

OhML's Syntax

$$\texttt{e} ::= \texttt{c} \mid \texttt{m} \mid \texttt{x} \mid \texttt{fun x} \texttt{->} \texttt{e} \mid \texttt{obj x} \texttt{->} \texttt{o} \mid \texttt{e e} \mid$$
$$\texttt{let x} = \texttt{e in e} \mid \texttt{fix}$$

$$\texttt{o} ::= \epsilon \mid \texttt{o data m} = \texttt{e}$$

OhML

One Step Semantics

Values and One Step

$$\texttt{v} ::= \texttt{c} \mid \texttt{x} \mid \texttt{m} \mid \texttt{fun x -> e} \mid \texttt{obj x -> o} \mid$$

$$\texttt{fix (fun x -> e)} \mid \texttt{fix} \mid \texttt{error}$$

Values and One Step

$$\mathtt{v} ::= \mathtt{c} \mid \mathtt{x} \mid \mathtt{m} \mid \mathtt{fun\ x} \mathtt{->} \mathtt{e} \mid \mathtt{obj\ x} \mathtt{->} \mathtt{o} \mid$$

$$\mathtt{fix\ (fun\ x} \mathtt{->} \mathtt{e)} \mid \mathtt{fix} \mid \mathtt{error}$$

$$(\mathtt{fun\ x} \mathtt{->} \mathtt{e})\ \mathtt{v} \mapsto \mathtt{e}[\mathtt{v}/\mathtt{x}] \qquad (\beta_{\mathrm{v}})$$

$$\mathtt{m}\ (\mathtt{obj\ x} \mathtt{->} \mathtt{o}) \mapsto \mathit{mbody}(\mathtt{o}\,;\mathtt{m})[\mathtt{obj} \ldots/\mathtt{x}] \qquad (\mathrm{call}_{\mathrm{v}})$$

$$\mathtt{let\ x} = \mathtt{v\ in\ e} \mapsto \mathtt{e}[\mathtt{v}/\mathtt{x}] \qquad (\mathrm{let}_{\mathrm{v}})$$

$$\mathtt{fix}\ \mathtt{v}_1\ \mathtt{v}_2 \mapsto \mathtt{v}_1\ (\mathtt{fix}\ \mathtt{v}_1)\ \mathtt{v}_2 \qquad (\delta_{\mathrm{fix}})$$

Lookup Algorithm (No Inheritance Yet)

$$\frac{}{\mathit{mbody}(\epsilon\,;\mathtt{m}) = \mathtt{error}}\ (\text{Empty})$$

$$\frac{}{\mathit{mbody}(\mathtt{o\ data\ m} = \mathtt{e}\,;\mathtt{m}) = \mathtt{e}}\ (\text{Top})$$

$$\frac{\mathtt{m} \neq \mathtt{n}}{\mathit{mbody}(\mathtt{o\ data\ m} = \mathtt{e}\,;\mathtt{n}) = \mathit{mbody}(\mathtt{o}\,;\mathtt{m})}\ (\text{Next})$$

Just try before I tell you the full story ...

```
let meth  = gtx in
let point = fun    x ->
            fun    y ->
            obj this -> data gtx = x data gty = y in
let p = (point 3 4) in (meth p)
```

Just try before I tell you the full story ...

```
let meth  = gtx in
let point = fun    x ->
            fun    y ->
            obj this -> data gtx = x data gty = y in
let p = (point 3 4) in (meth p)
 →
let point = fun    x ->
            fun    y ->
            obj this -> data gtx = x data gty = y in
let p = (point 3 4) in (gtx p)
```

Just try before I tell you the full story ...

```
let meth  = gtx in
let point = fun    x ->
            fun    y ->
            obj this -> data gtx = x data gty = y in
let p = (point 3 4) in (meth p)
 →
let point = fun    x ->
            fun    y ->
            obj this -> data gtx = x data gty = y in
let p = (point 3 4) in (gtx p)
 →
let p = ((fun    x -> fun  y ->
          obj this -> data gtx = x data gty = y) 3 4) in (gtx p)
```

Just try before I tell you the full story ...

```
let meth  = gtx in
let point = fun    x ->
            fun    y ->
            obj this -> data gtx = x data gty = y in
let p = (point 3 4) in (meth p)
 →
let point = fun    x ->
            fun    y ->
            obj this -> data gtx = x data gty = y in
let p = (point 3 4) in (gtx p)
 →
let p = ((fun    x -> fun  y ->
          obj this -> data gtx = x data gty = y) 3 4) in (gtx p)
 →
let p = obj this -> data gtx = 3 data gty = 4 in (gtx p)
```

Just try before I tell you the full story ...

```
let meth  = gtx in
let point = fun    x ->
            fun    y ->
            obj this -> data gtx = x data gty = y in
let p = (point 3 4) in (meth p)
 →
let point = fun    x ->
            fun    y ->
            obj this -> data gtx = x data gty = y in
let p = (point 3 4) in (gtx p)
 →
let p = ((fun    x -> fun  y ->
          obj this -> data gtx = x data gty = y) 3 4) in (gtx p)
 →
let p = obj this -> data gtx = 3 data gty = 4 in (gtx p)
 →
gtx (obj this -> data gtx = 3 data gty = 4)
```

Just try before I tell you the full story ...

```
let meth  = gtx in
let point = fun    x ->
            fun    y ->
            obj this -> data gtx = x data gty = y in
let p = (point 3 4) in (meth p)
 →
let point = fun    x ->
            fun    y ->
            obj this -> data gtx = x data gty = y in
let p = (point 3 4) in (gtx p)
 →
let p = ((fun    x -> fun  y ->
          obj this -> data gtx = x data gty = y) 3 4) in (gtx p)
 →
let p = obj this -> data gtx = 3 data gty = 4 in (gtx p)
 →
gtx (obj this -> data gtx = 3 data gty = 4)
 → 3
```

OhML

Types

Type Syntax Contexts and Judgments

$$\tau \ ::= \ \mathtt{b} \mid \alpha \mid \langle \overline{\mathtt{m}}{:}\overline{\tau} \rangle \mid \tau \to \tau \qquad \text{Types}$$

$$\Gamma \ ::= \ \epsilon \mid \Gamma, \mathtt{x}{:}\tau \mid \Gamma, \mathtt{c}{:}\tau \qquad \text{Contexts}$$

$$\Gamma \vdash \mathtt{e} : \tau \qquad \text{Typing judg.}$$

$$\tau_1 <: \tau_2 \qquad \text{Subtyping judg.}$$

Type Rules (I)

$$\frac{\mathtt{c}{:}\mathtt{b} \in \Gamma}{\Gamma \vdash \mathtt{c} : \mathtt{b}}\ (\text{Const}) \qquad \frac{\mathtt{x}{:}\tau \in \Gamma}{\Gamma \vdash \mathtt{x} : \tau}\ (\text{Var})$$

$$\frac{\Gamma, \mathtt{x}{:}\tau_1 \vdash \mathtt{e} : \tau_2}{\Gamma \vdash \mathtt{fun\ x\ ->\ e} : \tau_1 \to \tau_2}\ (\text{Fun})$$

$$\frac{\Gamma \vdash \mathtt{e}_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash \mathtt{e}_2 : \tau_1}{\Gamma \vdash \mathtt{e}_1\,\mathtt{e}_2 : \tau_2}\ (\text{Appl})$$

$$\frac{\Gamma \vdash \mathtt{e}_1 : \tau_1 \quad \Gamma, \mathtt{x}{:}\tau_1 \vdash \mathtt{e}_2 : \tau_2}{\Gamma \vdash \mathtt{let\ x} = \mathtt{e}_1\ \mathtt{in}\ \mathtt{e}_2 : \tau_2}\ (\text{Let})$$

Type Rules (II)

$$\frac{}{\Gamma \vdash \mathtt{m} : \langle \mathtt{m}{:}\tau \rangle \longrightarrow \tau} \text{(Meth)}$$

$$\frac{\tau \equiv \tau_1 \longrightarrow \tau_2}{\Gamma \vdash \mathtt{fix} : (\tau \longrightarrow \tau) \longrightarrow \tau} \text{(Fix)}$$

$$\frac{\Gamma \vdash \mathtt{e} : \tau_1 \quad \tau_1 <: \tau_2}{\Gamma \vdash \mathtt{e} : \tau_2} \text{(Sub)}$$

Type Rules (III)

$$\frac{\begin{array}{l}\mathtt{o} \triangleq \mathtt{data}\ \mathtt{m}_1 = \mathtt{e}_1\ \ldots\ \mathtt{data}\ \mathtt{m}_n = \mathtt{e}_n \\ \forall i, j = 1 \ldots n.\ \mathtt{m}_i \neq \mathtt{m}_j \\ \tau \triangleq \langle \mathtt{m}_1{:}\tau_1 \ldots \mathtt{m}_n{:}\tau_n \rangle \\ \Gamma, \mathtt{x}{:}\tau \vdash \mathtt{e}_i : \tau_i \qquad \forall i = 1 \ldots n\end{array}}{\Gamma \vdash \mathtt{obj\ x} \mathtt{->} \mathtt{o} : \tau} \ (\mathrm{Obj})$$

SubType Rules (IV)

$$\frac{}{\tau <: \tau}\ (\text{Refl}) \qquad \frac{(\mathsf{b}_1, \mathsf{b}_2) \in \mathcal{R}}{\mathsf{b}_1 <: \mathsf{b}_2}\ (\text{Taut})$$

$$\frac{\tau_1 <: \tau_2 \quad \tau_2 <: \tau_3}{\tau_1 <: \tau_3}\ (\text{Trans}) \qquad \frac{\tau_3 <: \tau_1 \quad \tau_2 <: \tau_4}{\tau_1 \longrightarrow \tau_2 <: \tau_3 \longrightarrow \tau_4}\ (\text{Arr})$$

$$\frac{h \geq k}{\langle \mathtt{m}_1{:}\tau_1 \ldots \mathtt{m}_h{:}\tau_h \rangle <: \langle \mathtt{m}_1{:}\tau_1 \ldots \mathtt{m}_k{:}\tau_k \rangle}\ (\text{Width})$$

OhML:

Examples

A simple functional point (with its typing)

Consider the point

```
let meth  = gtx in
let point = fun   x ->
            fun   y ->
            obj   this ->
            data  gtx = x
            data  gty = y
in let p = (point 3 4)
in (gtx p)
```

A simple functional point (with its typing)

Consider the point

```
let meth  = gtx in
let point = fun   x ->
            fun   y ->
            obj   this ->
            data  gtx = x
            data  gty = y
in let p = (point 3 4)
in (gtx p)
```

with typing

```
|- meth         : ⟨gtx:int gty:int⟩ -> int
|- point        : int -> int -> ⟨gtx:int gty:int⟩
|- (point 3 4)  : ⟨gtx:int gty:int⟩
|- gtx          : ⟨gtx:int gty:int⟩ -> int
|- (gtx p)      : int
```

OhML:

Metatheory

Properties

Spec If $\Gamma, \mathrm{x}{:}\tau_1 \vdash \mathsf{e} : \tau$ and $\tau_2 <: \tau_1$ then $\Gamma, \mathrm{x}{:}\tau_2 \vdash \mathsf{e} : \tau$.

Subst If $\Gamma, \mathrm{x}{:}\tau_1 \vdash \mathsf{e}_1 : \tau$ and $\Gamma \vdash \mathsf{e}_2{:}\tau_1$ then $\Gamma \vdash \mathsf{e}_1[\mathsf{e}_2/\mathrm{x}] : \tau$.

SR If $\Gamma \vdash \mathsf{e}_1 : \tau$, and $\mathsf{e}_1 \longrightarrow \mathsf{e}_2$, then $\Gamma \vdash \mathsf{e}_2 : \tau$.

Progress If $\Gamma \vdash \mathsf{e} : \tau$, then the evaluation of e cannot produce `error`, *i.e.* $\mathsf{e} \not\longrightarrow\!\!\!\!\!\rightarrow \mathsf{C}[\texttt{error}]$ for every context $\mathsf{C}[\cdot]$.

Progress If $\Gamma \vdash \mathsf{e}_1 : \tau$ then, either e_1 is a value, or there exists e_2 such that $\mathsf{e}_1 \longrightarrow\!\!\!\!\!\rightarrow \mathsf{e}_2$.

Conclusions and Agenda

- Already done:
  - ⋆ simple and uniform object-based language
  - ⋆ with subtyping (especially useful for traits)
  - ⋆ method as first-class citizens
  - ⋆ economic type system
- To be done:
  - ⋆ an inference algorithm
  - ⋆ featuring inheritance (and related issues)
  - ⋆ featuring polymorphism
  - ⋆ traits (featuring multiple inheritance)
  - ⋆ featuring references