Modular Multiple Dispatch with Multiple Inheritance Mechanized in Coq

Sukyoung Ryu

Joint work with Jieung Kim, Victor Luchangco, and Guy L. Steele Jr.

Programming Language Research Group (PL알지!) KAIST

July 26, 2012

Jieung at Yale

Multiple Inheritance

by Traits and Objects

```
trait String extends { StandardTotalOrder[[String]],
                       ZeroIndexed[[Char]] }
  ...
end
```

```
object FlatString extends { String,
                            DelegatedIndexed[[Char, ℤ32]] }
...
end
```

Multiple Dispatch

by 3 Kinds of Functionals

Dotted methods

```
trait ℤ
  asString() : String = ...
  add(x : ℤ) : ℤ = ...
  subtract(x : ℤ) : ℤ = ...
  multiply(x : ℤ) : ℤ = ...
  ...
end
```

3. *multiply*(5)

Multiple Dispatch

by 3 Kinds of Functionals

Top-level Functions

$$
\begin{array}{l}
\texttt{trait } \mathbb{Z} \\
\quad \mathit{asString}() = \ldots \\
\texttt{end} \\
\mathit{add}(x : \mathbb{Z}, y : \mathbb{Z}) : \mathbb{Z} = \ldots \\
\mathit{subtract}(x : \mathbb{Z}, y : \mathbb{Z}) : \mathbb{Z} = \ldots \\
\mathit{multiply}(x : \mathbb{Z}, y : \mathbb{Z}) : \mathbb{Z} = \ldots \\
\ldots \\
\mathit{multiply}(3, 5)
\end{array}
$$

Multiple Dispatch

by 3 Kinds of Functionals

Functional Methods

$$
\begin{array}{l}
\texttt{trait } \mathbb{Z} \\
\quad \mathit{asString}() = \ldots \\
\quad \mathit{add}(\texttt{self}, y : \mathbb{Z}) : \mathbb{Z} = \ldots \\
\quad \mathit{subtract}(\texttt{self}, y : \mathbb{Z}) : \mathbb{Z} = \ldots \\
\quad \mathit{multiply}(\texttt{self}, y : \mathbb{Z}) : \mathbb{Z} = \ldots \\
\quad \ldots \\
\texttt{end} \\
\\
\mathit{multiply}(3, 5)
\end{array}
$$

Multiple Dispatch

by 3 Kinds of Functionals

Functional Methods (Operators)

```
trait ℤ
  asString() = ...
  opr +(self, y : ℤ) : ℤ = ...
  opr −(self, y : ℤ) : ℤ = ...
  opr ·(self, y : ℤ) : ℤ = ...
  ...
end
3 · 5
```

Multiple Dispatch

by 3 Kinds of Functionals

Overloaded Declarations

$$\texttt{trait } \mathbb{Z}$$
$$\quad \mathit{add}(\texttt{self}, y : \mathbb{Z}) : \mathbb{Z} = \ldots$$
$$\quad \ldots$$
$$\texttt{end}$$

$$\mathit{add}(x : \mathrm{String}, y : \mathrm{String}) : \mathrm{String} = \ldots$$

Multiple Dispatch

by 3 Kinds of Functionals

Function Dispatch

```
trait ℤ
  add(self, y : ℤ) : ℤ = ...
  ...
end
```

$add(x : \mathrm{String}, y : \mathrm{String}) : \mathrm{String} = \ldots$

$add(3, 5)$

$add(\text{“Hello”}, \text{“ Bye”})$

Modularity

by Components and Selective Imports

```
component IntegerToString
  trait ℤ
    asString(self) = ...
    ...
  end
  trait ℕ extends ℤ... end
end
```

```
component MyProgram
  import IntegerToString.{ asString }
  asString(x: Boolean): String = ...
  run() = println(asString(42))
end
```

Mechanization

by Coq

```
(** ** Type Safety *)
(** corollary to the preservation theorem and the progress theorem *)
Module Type Safety (H: Hyps).
  Parameter preservation : forall E e e' fty,
    expTy E e fty ->
    evalRule e e' ->
    wideTyping E e' fty.

  Parameter progress : forall e fty,
    expTy nil e fty ->
    val e \/ (exists e', evalRule e e').
End Safety.
```

Ambiguity

due to Multiple Inheritance

```
trait Vector
  opr ·(r : ℝ, self): Vector
  opr ·(self, v: Vector): ℝ
end
```

Ambiguity

due to Multiple Inheritance

```
trait Vector
  opr ·(r : ℝ, self): Vector
  opr ·(self, v: Vector): ℝ
end
```

```
trait Matrix
  opr ·(r : ℝ, self): Matrix
  opr ·(self, m : Matrix): Matrix
end
```

Ambiguity

due to Multiple Inheritance

```
trait Vector
  opr ·(r : ℝ, self): Vector
  opr ·(self, v: Vector): ℝ
end
```

```
trait Matrix
  opr ·(r : ℝ, self): Matrix
  opr ·(self, m : Matrix): Matrix
end
```

```
object vm extends { Vector, Matrix } end
```

Ambiguity

due to Multiple Inheritance

```
trait Vector
  opr ·(r : ℝ, self): Vector
  opr ·(self, v: Vector): ℝ
end
```

```
trait Matrix
  opr ·(r : ℝ, self): Matrix
  opr ·(self, m : Matrix): Matrix
end
```

```
object vm extends { Vector, Matrix } end
```

$8 \cdot vm$ (*) Ambiguity due to Multiple Inheritance

Ambiguity

due to Multiple Dispatch

```
trait Matrix
  opr ·(r : ℝ, self): Matrix
  opr ·(self, m : Matrix): Matrix
  opr ·(self, v : Vector): Vector
  opr ·(v : Vector, self): Vector
end
```

```
object vm extends { Vector, Matrix } end
```

Ambiguity

due to Multiple Dispatch

```
trait Matrix
  opr ·(r : ℝ, self): Matrix
  opr ·(self, m : Matrix): Matrix
  opr ·(self, v : Vector): Vector
  opr ·(v : Vector, self): Vector
end
```

```
object vm extends { Vector, Matrix } end
```

$vm \cdot vm$ (*) Ambiguity due to Multiple Dispatch

Ambiguity

due to Modularity

```
component MatrixLibrary
  trait Matrix end
  object UnitMatrix extends Matrix end
end
```

```
component MyMatrixLibrary
  import MatrixLibrary.{ Matrix }
  object UnitMatrix extends Matrix end
  object GenUnitMatrix
    gen() = UnitMatrix
  end
end
```

Ambiguity

due to Modularity

```
component MatrixLibrary
  trait Matrix end
  object UnitMatrix extends Matrix end
end
```

```
component MyMatrixLibrary
  import MatrixLibrary.{ Matrix }
  object UnitMatrix extends Matrix end
  object GenUnitMatrix
    gen() = UnitMatrix
  end
end
```

Ambiguity due to Modularity

```
component MatrixLibrary
  trait Matrix end
  object UnitMatrix extends Matrix end
end
```

```
component MyMatrixLibrary
  import MatrixLibrary.{ Matrix }
  object UnitMatrix extends Matrix end
  object GenUnitMatrix
    gen() = UnitMatrix
  end
end
```

Ambiguity due to Modularity

```
component MatrixLibrary
  trait Matrix end
  object UnitMatrix extends Matrix end
end
```

```
component MyMatrixLibrary
  import MatrixLibrary.{ Matrix }
  object UnitMatrix extends Matrix end
  object GenUnitMatrix
    gen() = UnitMatrix
  end
end
```

```
import MatrixLibrary.{ Matrix, UnitMatrix }
import MyMatrixLibrary.{ GenUnitMatrix }
toString(x : Matrix) : String = "Matrix"
toString(x : UnitMatrix) : String = "UnitMatrix"
toString(GenUnitMatrix.gen()) (*) Ambiguity due to Modularity
```

Subtype Rule

```
trait Matrix end
object UnitMatrix extends Matrix end
```

$$toString(x : \text{Matrix}) : \text{String} = \text{“Matrix”}$$
$$toString(x : \text{UnitMatrix}) : \text{String} = \text{“UnitMatrix”}$$

$$toString(\text{UnitMatrix})$$

Subtype Rule

```
trait Matrix end
object UnitMatrix extends Matrix end
```

$$toString(x : \mathrm{Matrix}) : \mathrm{String} = \text{“Matrix”}$$
$$toString(x : \mathrm{UnitMatrix}) : \mathrm{String} = \text{“UnitMatrix”}$$
$$toString(\mathrm{UnitMatrix})$$

$$\frac{|\overline{C^a}| = |\overline{C^{a'}}| \qquad j = k \qquad actualTy_p(M, C)\ \overline{actualTy_p(M, C^a)} \neq actualTy_p(M', C')\ \overline{actualTy_p(M', C^{a'})} \qquad p \vdash (M', C') <: (M, C) \qquad p \vdash \overline{(M', C^{a'}) <: (M, C^a)} \qquad p \vdash (M', C^{r'}) <: (M, C^r)}{p \vdash valid(m, M, C, \overline{C^a} \to C^r, j, M', C', \overline{C^{a'}} \to C^{r'}, k, S)}$$

Meet Rule

```
trait Matrix end
object UnitMatrix extends Matrix end
```

$$\mathit{toString}(x : \mathrm{Matrix}, y : \mathrm{UnitMatrix}) : \mathrm{String} = \text{“MU”}$$
$$\mathit{toString}(x : \mathrm{UnitMatrix}, y : \mathrm{Matrix}) : \mathrm{String} = \text{“UM”}$$

$$\mathit{toString}(\mathrm{UnitMatrix}, \mathrm{UnitMatrix})$$

Meet Rule

```
trait Matrix end
object UnitMatrix extends Matrix end
```

$$toString(x : \text{Matrix}, y : \text{UnitMatrix}) : \text{String} = \text{“MU”}$$
$$toString(x : \text{UnitMatrix}, y : \text{Matrix}) : \text{String} = \text{“UM”}$$
$$toString(x : \text{UnitMatrix}, y : \text{UnitMatrix}) : \text{String} = \text{“UU”}$$
$$toString(\text{UnitMatrix}, \text{UnitMatrix})$$

$$\frac{\begin{array}{c} I = |\overline{C^a}| = |\overline{C^{a'}}| \qquad C_0^a = C \qquad C_0^{a'} = C' \\ actualTy_p(M, C)\ \overline{actualTy_p(M, C^a)} \neq actualTy_p(M', C')\ \overline{actualTy_p(M', C^{a'})} \\ \exists (M'', C_0^{a''}, md'') \in S. \\ (md'' = m(\_ : \overline{C^{a''}}) : \_) \wedge (j = k = selfIndex(md'')) \wedge (I = |\overline{C^{a''}}|) \wedge \\ (\forall\ 0 \leq n \leq I.\ p \vdash meet(\{actualTy_p(M, C_n^a),\ actualTy_p(M', C_n^{a'})\}, actualTy_p(M'', C_n^{a''}))) \end{array}}{p \vdash valid(m, M, C, \overline{C^a} \to C^r, j, M', C', \overline{C^{a'}} \to C^{r'}, k, S)}$$

Exclusion Rule

```
trait Vector end
trait Matrix
  opr ·(self, v: Vector): Vector
  opr ·(v: Vector, self): Vector
end

object v extends Vector end
object m extends Matrix end

(v · m) + (m · v)
```

Exclusion Rule

```
trait Vector end
trait Matrix excludes Vector
  opr ·(self, v: Vector): Vector
  opr ·(v: Vector, self): Vector
end

object v extends Vector end
object m extends Matrix end
```

$(v \cdot m) + (m \cdot v)$

$$\frac{|\overline{C^a}| = |\overline{C^{a'}}| \qquad p \vdash (\overline{(M, C^a)}) \Diamond (\overline{(M', C^{a'})})}{p \vdash \mathit{valid}(m, M, C, \overline{C^a} \to C^r, j, M', C', \overline{C^{a'}} \to C^{r'}, k, S)}$$

Functions vs Functional Methods

Overloaded Declarations

$$\texttt{trait}\ \mathbb{Z}$$
$$\quad add(\texttt{self}, y : \mathbb{Z}) : \mathbb{Z} = \ldots$$
$$\quad \ldots$$
$$\texttt{end}$$
$$add(x : \mathrm{Number}, y : \mathrm{Number}) : \mathrm{Number} = \ldots$$

$$\frac{\begin{array}{l}\forall (M, C, fd) \in visibleTopFunc_p(M^m).\\ \quad \forall (M', C', fd') \in visibleFuncMeth_p(M^m).\\ \qquad fd = m(\overline{\_ : C^a}) : C^r\ \_ \ \wedge\ fd' = m(\overline{\_ : C^{a'}}) : C^{r'}\ \_ \implies\\ \qquad\quad (|\overline{C^a}| \neq |\overline{C^{a'}}|) \vee\\ \qquad\quad (p \vdash (\overline{(M, C^a)}) \diamond (\overline{(M', C^{a'})})) \vee\\ \qquad\quad (\overline{actualTy_p(M, C^a)} \neq \overline{actualTy_p(M', C^{a'})} \wedge\\ \qquad\qquad p \vdash \overline{(M', C^{a'})} <: \overline{(M, C^a)} \ \wedge\ p \vdash (M', C^{r'}) <: (M, C^r))\end{array}}{p \vdash validBtwFunc(M^m)}$$

Type Identity

```
component MatrixLibrary
  trait Matrix end
  object UnitMatrix extends Matrix end
end
```

```
component MyMatrixLibrary
  import MatrixLibrary.{ Matrix }
  object UnitMatrix extends Matrix end
  object GenUnitMatrix
    gen() = UnitMatrix
  end
end
```

$$actualTy_p(M, C) = \begin{cases} (\text{Top}, \text{Object}) & \text{if } C = \text{Object} \\ (M, C) & \text{if } \_\ C\ \_\ \texttt{end} \in classTable_p(M) \\ actualTy_p(M', C) & \text{if } \texttt{component}\ M\ \overline{\texttt{import}\ M.\{\bar{i}\}}\ \_\ \texttt{end} \in p \wedge \\ & \quad M'.C \in \overline{M.\{\bar{i}\}} \end{cases}$$

MFFMM in Coq

```
(** ** Type Safety *)
(** corollary to the preservation theorem and the progress theorem *)
Module Type Safety (H: Hyps).
  Parameter preservation : forall E e e' fty,
    expTy E e fty ->
    evalRule e e' ->
    wideTyping E e' fty.

  Parameter progress : forall e fty,
    expTy nil e fty ->
    val e \/ (exists e', evalRule e e').
End Safety.
```

MFFMM in Coq

```
(** ** Type Safety *)
(** corollary to the preservation theorem and the progress theorem *)
Module Type Safety (H: Hyps).
  Parameter preservation : forall E e e' fty,
    expTy E e fty ->
    evalRule e e' ->
    wideTyping E e' fty.

  Parameter progress : forall e fty,
    expTy nil e fty ->
    val e \/ (exists e', evalRule e e').
End Safety.
```

"Well-typed programs do not have any undefined or ambiguous calls at run time."

Questions ?

Sukyoung Ryu
sryu.cs@kaist.ac.kr