Dependent Object Types

Towards a foundation for Scala's type system

Nada Amin, Adriaan Moors, Martin Odersky

FOOL 2012

October 22, 2012

DOT: Dependent Object Types

The DOT calculus proposes a new *type-theoretic foundation* for Scala and languages like it. It models

- path-dependent types
- abstract type members
- mixture of nominal and structural typing via refinement types

It does not model

- inheritance and mixin composition
- what's currently in Scala

DOT normalizes Scala's type system by

- unifying the constructs for type members
- providing classical intersection and union types

Classical Intersection and Union Types

- form a lattice wrt subtyping
- simplify glb and lub computations

```
trait A { type T <: A }
trait B { type T <: B }
trait C extends A with B { type T <: C }
trait D extends A with B { type T <: D }
// in Scala, lub(C, D) is an infinite sequence
A with B { type T <: A with B { type T <: A with B {
  type T <: ...
}}}
// type inference needs to compute glbs and lubs
if (cond) ((a: A) => c: C) else ((b: B) => d: D)
// lub(A => C, B => D) <: glb(A, B) => lub(C, D)
```

Constructs for Type Members

```
trait Food
trait Animal {
  // in DOT, abstract Meal: Bot .. Food
  type Meal <: Food
  def eat(meal: Meal) {}
}
// in Dot, concrete Grass: Bot .. Food
trait Grass extends Food
trait Cow extends Animal {
  // in DOT, abstract Meal: Grass .. Grass
  type Meal = Grass
}
val a = new Animal {}
val c = new Cow {}
val g = new Grass {}
a.eat(???) // ???.type <: a.Meal so ???.type <: Bot
c.eat(g) // g.type <: c.Meal so g.type <: Grass
```

DOT: Syntax

- terms

variables	x, y, z
selections	$t.l$
method invocations	$t.m(t)$
object creations	$\textbf{val}\ y = \textbf{new}\ c;\ t'$
	c is a constructor $T_c\left\{\overline{l = v}\ \overline{m(x) = t}\right\}$

- types

type selections	$p.L$
refinement types	$T\left\{z \Rightarrow \overline{D}\right\}$
type intersections	$T \wedge T'$
type unions	$T \vee T'$
a top type	$\top$
a bottom type	$\bot$

DOT: Judgments

Typing Judgments

- type assignment
  $\Gamma \vdash t : T$
- subtyping
  $\Gamma \vdash S <: T$
- well-formedness
  $\Gamma \vdash T \ \mathbf{wf}$
- membership
  $\Gamma \vdash t \ni D$
- expansion
  $\Gamma \vdash T \prec_z \overline{D}$

Small-Step Operational Semantics

- reduction
  $t \mid s \ \rightarrow \ t' \mid s'$

Functions as Sugar

$$
\begin{aligned}
S \rightarrow_s T &\iff \top\,\{z \Rightarrow \mathit{apply} : S \rightarrow T\} \\
\textbf{fun}\ (x : S)\ T\ t &\iff \textbf{val}\ z = \textbf{new}\ S \rightarrow_s T\,\{\mathit{apply}(x) = t\}\,;\ z \\
(\textbf{app}\ f\ x) &\iff f.\mathit{apply}(x) \\
(\textbf{cast}\ \ T\ \ t) &\iff (\textbf{app}\ (\textbf{fun}\ (x : T)\ T\ x)\ t)
\end{aligned}
$$

Revisiting LUB Computation

- Suppose f has type $T_f = (A \rightarrow_s C) \vee (B \rightarrow_s D)$
- $T_f = \top \{z \Rightarrow apply : A \rightarrow C\} \vee \top \{z \Rightarrow apply : B \rightarrow D\}$
- Let's type-check $y = (\textbf{app}\ f\ x) = f.apply(x)$
- $T_f \prec_f \{apply : A \wedge B \rightarrow C \vee D\}$
- $f \ni apply : A \wedge B \rightarrow C \vee D$
- $T_x <: A \wedge B$
- $T_y = C \vee D$

Revisiting Refined Type Members

```
trait Food
trait Animal {
  // in DOT, abstract Meal: Bot .. Food
  type Meal <: Food
  def eat(meal: Meal) {}
}
// in Dot, concrete Grass: Bot .. Food
trait Grass extends Food
trait Cow extends Animal {
  // in DOT, abstract Meal: Grass .. Grass
  type Meal = Grass
}
```

$$Cow \prec_c \{Meal : Grass \vee Bot..Grass \wedge Food, eat : c.Meal \rightarrow Unit\}$$
$$Cow \prec_c \{Meal : Grass..Grass, eat : c.Meal \rightarrow Unit\}$$

Why not alias Meal to Food in Animal?

```
trait Food
trait Animal {
  // in DOT, abstract Meal: Food .. Food
  type Meal = Food
  def eat(meal: Meal) {}
}

trait Grass extends Food
trait Cow extends Animal {
  // in DOT, abstract Meal: Grass .. Grass
  type Meal = Grass
}
```

$$Cow \prec_c \{Meal : Grass \vee Food..Grass \wedge Food, eat : c.Meal \to Unit\}$$

$$Cow \prec_c \{Meal : Food..Grass, eat : c.Meal \to Unit\}$$

Type-Safety?

- Type safety usually proven as a corollary of the standard theorems of preservation and progress.
- In DOT, preservation (also known as subject reduction) doesn't hold, because of
  - narrowing: after substitution, a term can have a more precise type
  - need for path-equality provisitions

Counterexample: TERM-$\ni$ Restriction

Let X be a shorthand for the type:

$$\top\{z \Rightarrow \ L_a : \top..\top \ l : z.L_a\}$$

Let Y be a shorthand for the type:

$$\top\{z \Rightarrow \ l : \top\}$$

Now, consider the term

$$\textbf{val } u = \textbf{new } X \, \{l = u\} \, ;$$
$$(\textbf{app } (\textbf{fun } (y : \top \rightarrow_s Y) \ Y \ (\textbf{app } y \ u)) \ (\textbf{fun } (d : \top) \ Y \ (\textbf{cast } X \ u))).l$$

- How to type (**cast** X u).l?

Counterexample: (Expansion and) Well-Formedness Lost

$$
\begin{array}{l}
\textbf{val } v = \textbf{new } \top\,\{z \Rightarrow L : \bot..\top\,\{z \Rightarrow A : \bot..\top, B : z.A..z.A\}\,\}\,\{\}\,; \\
(\textbf{app } (\textbf{fun } (x : \top\,\{z \Rightarrow L : \bot..\top\,\{z \Rightarrow A : \bot..\top, B : \bot..\top\}\}) \top \\
\quad \textbf{val } z = \textbf{new } \top\{z \Rightarrow \\
\qquad\qquad l : x.L \wedge \top\,\{z \Rightarrow A : z.B..z.B, B : \bot..\top\} \rightarrow \top\}\{ \\
\qquad\qquad l(y) = \textbf{fun } (a : y.A) \top\ a\}; \\
\quad (\textbf{cast } \top\ z)) \\
\ v)
\end{array}
$$

Counterexample: Path Equality

$$
\begin{array}{lll}
\textbf{val } b = \textbf{new } \top\{z \Rightarrow & X : \top..\top & \\
 & l : z.X & \}\,\{l = b\}\,; \\
\textbf{val } a = \textbf{new } \top\{z \Rightarrow i : \top\{z \Rightarrow & & \\
 & X : \bot..\top & \\
 & l : z.X\} & \}\,\{i = b\}\,; \\
(\textbf{cast } \top\ (\textbf{cast } a.i.X\ a.i.l)) & &
\end{array}
$$

- $a.i.l$ reduces to $b.l$.
- $b.l$ has type $b.X$, so we need $b.X <: a.i.X$.

Type-Safety

A well-typed term doesn't get stuck:

- If
  - $\emptyset \vdash t : T$ and
  - $t \mid \emptyset \rightarrow^* t' \mid s'$
- then
  - t' is a value or
  - $\exists t'', s''. t' \mid s' \rightarrow t'' \mid s''$.

Observations:

- Type-safety is stronger than progress, which states that a well-typed term can take *a* step or is a value.
- But progress + preservation is stronger than type-safety. In particular, we don't need to type-check intermediary terms for type-safety!
- But how to get a strong enough induction hypothesis without preservation? Use logical relations.

DOT: Dependent Object Types

- DOT is a core calculus for path-dependent types.
- DOT aims to normalize Scala's type system.
- DOT does not satisfy the standard theorem of preservation. Can and should we live with that?
- See the paper for the entire formalism, examples, counterexamples to preservation, and discussion of type-safety, design decisions and variants.

Extra Slides

Some DOT Rules

$$\frac{\Gamma \vdash p \ni L : S..U \ , \ S <: U \ , \ S' <: S}{\Gamma \vdash S' <: p.L} \quad (\text{<:-TSEL})$$

$$\frac{\Gamma \vdash p \ni L : S..U \ , \ S <: U \ , \ U <: U'}{\Gamma \vdash p.L <: U'} \quad (\text{TSEL-<:})$$

$$\frac{\Gamma \vdash T \prec_z \overline{D'}}{\Gamma \vdash T \left\{ z \Rightarrow \overline{D} \right\} \prec_z \overline{D'} \wedge \overline{D}} \quad (\text{RFN-}\prec)$$

$$\begin{aligned}(D \wedge D')(L) = & L : (S \vee S')..(U \wedge U') \\ & \text{if } (L : S..U) \in \overline{D} \text{ and } (L : S'..U') \in \overline{D'} \\ = & D(L) \text{ if } L \notin dom(\overline{D'}) \\ = & D'(L) \text{ if } L \notin dom(\overline{D})\end{aligned}$$

Some more DOT Rules

$$\frac{\Gamma \vdash p : T \ , \ T \prec_z \overline{D}}{\Gamma \vdash p \ni [p/z]D_i} \qquad (\textsc{Path-}\ni)$$

$$\frac{z \notin fn(D_i) \qquad \Gamma \vdash t : T \ , \ T \prec_z \overline{D}}{\Gamma \vdash t \ni D_i} \qquad (\textsc{Term-}\ni)$$

$$\frac{\Gamma \vdash p \ni L : S..U \ , \ U \prec_z \overline{D}}{\Gamma \vdash p.L \prec_z \overline{D}} \qquad (\textsc{TSel-}\prec)$$

$$\frac{\Gamma \vdash p \ni L : S..U \ , \ S \ \mathbf{wf} \ , \ U \ \mathbf{wf}}{\Gamma \vdash p.L \ \mathbf{wf}} \qquad (\textsc{TSel-WF}_1)$$

$$\frac{\Gamma \vdash p \ni L : \bot..U}{\Gamma \vdash p.L \ \mathbf{wf}} \qquad (\textsc{TSel-WF}_2)$$

Example: Class Hierarchies

```
object pets {
  trait Pet
  trait Cat extends Pet
  trait Dog extends Pet
  trait Poodle extends Dog
  trait Dalmatian extends Dog
}
```

$$
\begin{aligned}
&\textbf{val } \mathit{pets} = \textbf{new } \top\{z \Rightarrow \\
&\qquad \mathit{Pet}_c : \bot..\top \\
&\qquad \mathit{Cat}_c : \bot..z.\mathit{Pet}_c \\
&\qquad \mathit{Dog}_c : \bot..z.\mathit{Pet}_c \\
&\qquad \mathit{Poodle}_c : \bot..z.\mathit{Dog}_c \\
&\qquad \mathit{Dalmatian}_c : \bot..z.\mathit{Dog}_c \\
&\}\{\};
\end{aligned}
$$

Example: Abstract Type Members

```
object choices {
  trait Alt {
    type C
    type A <: C
    type B <: C
    val choose : A => B => C
  }
}
```

$$
\begin{aligned}
&\textbf{val } \mathit{choices} = \textbf{new } \top\{z \Rightarrow \\
&\qquad \mathit{Alt}_c : \bot..\top\{a \Rightarrow \\
&\qquad\qquad C : \bot..\top \\
&\qquad\qquad A : \bot..a.C \\
&\qquad\qquad B : \bot..a.C \\
&\qquad\qquad \mathit{choose} : a.A \to a.B \to_s a.A \vee a.B \\
&\qquad\} \\
&\}\{\};
\end{aligned}
$$

Subtyping of *choices*

$$
\begin{aligned}
& \mathit{choices}.\mathit{Alt}_c \{a \Rightarrow C : \bot..\mathit{pets}.\mathit{Dog}_c\} \\
<: \quad & \mathit{choices}.\mathit{Alt}_c \{a \Rightarrow C : \bot..\mathit{pets}.\mathit{Pet}_c\}
\end{aligned}
$$

but

$$
\begin{aligned}
& \mathit{choices}.\mathit{Alt}_c \{a \Rightarrow C : \mathit{pets}.\mathit{Dog}_c..\mathit{pets}.\mathit{Dog}_c\} \\
\not<: \quad & \mathit{choices}.\mathit{Alt}_c \{a \Rightarrow C : \mathit{pets}.\mathit{Pet}_c..\mathit{pets}.\mathit{Pet}_c\}
\end{aligned}
$$

Example: F-bounded Quantification

```
trait MetaAlt extends choices.Alt {
  type C = MetaAlt
  type A = C
  type B = C
}
```

$$
\begin{aligned}
&\textbf{val } m = \textbf{new } \top\{m \Rightarrow \\
&\quad MetaAlt_c : \bot..choices.Alt_c\{a \Rightarrow \\
&\qquad C : m.MetaAlt_c..m.MetaAlt_c \\
&\qquad A : a.C..a.C \\
&\qquad B : a.C..a.C \\
&\quad \} \\
&\}\{\};
\end{aligned}
$$

Example: Polymorphic Operators as Sugar

We translate

$$\textbf{val } x^a = \textbf{pickLast}(T^C, T^A, T^B); e^a$$

to

$$\begin{aligned}
&\textbf{val } x^a = \textbf{new } \mathit{choices}.\mathit{Alt}_c\{x^a \Rightarrow \\
&\qquad C : T^C..T^C \\
&\qquad A : T^A..T^A \\
&\qquad B : T^B..T^B \\
&\qquad \mathit{choose} : x^a.A \rightarrow x^a.B \rightarrow_s x^a.B \\
&\}\,\{\mathit{choose}(a) = \textbf{fun } (b : x^a.B)\ x^a.B\ b\}\,; \\
&e^a
\end{aligned}$$

Some $MetaAlt_c$ instances

```
val f = new MetaAlt {
  val choose: C => C => C = a => b => a
}
val rl = new MetaAlt {
  val choose: C => C => C = a => b => b.choose(a)(b)
}
```

$$\begin{aligned}
&\textbf{val } f = \textbf{new } m.MetaAlt_c\{\\
&\qquad choose(a) = \textbf{fun } (b : m.MetaAlt_c)\ m.MetaAlt_c\ a\};\\
&\textbf{val } rl = \textbf{new } m.MetaAlt_c\{\\
&\qquad choose(a) = \textbf{fun } (b : m.MetaAlt_c)\ m.MetaAlt_c\\
&\qquad\qquad (\textbf{app } b.choose(a)\ b)\};
\end{aligned}$$