DOT: Dependent Object Types

Semester Project, Spring 2012

Nada Amin

EPFL

DOT: Dependent Object Types

- type-theoretic foundation of Scala and languages like it
- models:
  - path-dependent types
  - abstract type members
  - mixture of nominal and structural typing via refinement types
- does not model:
  - inheritance and mixin composition
  - what's currently in Scala

DOT Syntax

term t

- variable
  x
- lambda abstraction
  $\lambda x : T.\ \ t$
- function application
  $t\ t'$
- field selection
  $t.l$
- object creation expression
  val $x =$
  new $T_c \left\{ \overline{l = v} \right\};\ t$

type T

- selection
  $p.L$
- refinement
  $T \left\{ z \Rightarrow \overline{D} \right\}$
- function
  $T \to T'$
- intersection
  $T \wedge T'$
- union
  $T \vee T'$
- $\top$, $\bot$

DOT Judgments

Typing Judgments

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

Small-Step Operational Semantics

- reduction
  $t \,|\, s \rightarrow t' \,|\, s'$

Basics

Booleans, Error, ...

$$
\begin{array}{l}
\textbf{val}\ \mathit{root} = \textbf{new}\ \top\{r \Rightarrow \\
\quad \mathit{Unit} : \bot..\top \\
\quad \mathit{unit} : \top \to r.\mathit{Unit} \\
\quad \mathit{Boolean} : \bot..\top\{z \Rightarrow \\
\qquad\quad \mathit{ifNat} : (r.\mathit{Unit} \to r.\mathit{Nat}) \to (r.\mathit{Unit} \to r.\mathit{Nat}) \to r.\mathit{Nat} \\
\qquad \} \\
\quad \mathit{false} : r.\mathit{Unit} \to r.\mathit{Boolean} \\
\quad \mathit{true} : r.\mathit{Unit} \to r.\mathit{Boolean} \\
\quad \mathit{error} : r.\mathit{Unit} \to \bot \\
\quad \ldots \\
\}\left\{\ldots \overline{(l = v)} \ldots\right\};
\end{array}
$$

Basics (Continued)

Booleans, Error, ...

$$
\begin{array}{l}
\{ \\
\quad unit = \lambda x : \top.\ \ \textbf{val}\ u = \textbf{new}\ root.Unit;\ u \\
\quad false = \lambda u : root.Unit. \\
\qquad \textbf{val}\ ff = \textbf{new}\ root.Boolean\{ \\
\qquad\quad ifNat = \lambda t : root.Unit \to root.Nat. \\
\qquad\qquad \lambda e : root.Unit \to root.Nat. \\
\qquad\qquad e\ root.unit \\
\qquad \}; \\
\qquad ff \\
\quad error = \lambda u : root.Unit.\ \ root.error\ u \\
\quad \ldots \\
\}
\end{array}
$$

Outline

No Subtyping Transitivity to No Preservation

1. Start with 3 types S, T, U st $S <: T$ and $T <: U$ but $S \not<: U$.
2. Create function of type $S \to S$.
3. Cast it to $S \to T$.
4. Cast it to $S \to U$.
5. After some reduction step, the first cast vanishes and we need to cast directly from $S \to S$ to $S \to U$.

Note: The 3 types don't need to be realizable but must be expressible within a realizable universe.

Code Recipe

$$
\begin{aligned}
&\textbf{val } u = \textbf{new } \ldots; \\
&((\lambda x{:}\top.\ x) \\
&\quad((\lambda f{:}S \to U.\ f) \\
&\qquad((\lambda f{:}S \to T.\ f) \\
&\qquad\quad((\lambda f{:}S \to S.\ f) \\
&\qquad\qquad \lambda x{:}S.\ x))))
\end{aligned}
$$

Non-Expanding Types and Subtyping Transitivity

$$
\begin{aligned}
&\top\{u \Rightarrow \\
&\quad Bad : \bot..u.Bad \\
&\quad Good : \top\{z \Rightarrow L : \bot..\top\}.. \\
&\qquad\qquad \top\{z \Rightarrow L : \bot..\top\} \\
&\quad Lower : u.Bad \wedge u.Good..u.Good \\
&\quad Upper : u.Good..u.Bad \vee u.Good \\
&\quad X : u.Lower..u.Upper \\
&\}
\end{aligned}
$$

$$S = u.Bad \wedge u.Good$$

$$T = u.Lower$$

$$U = u.X\{z \Rightarrow L : \bot..\top\}$$

tsel-$\prec$

$$\frac{\Gamma \vdash p \ni L : S..U \ , \ U \prec_z \overline{D}}{\Gamma \vdash p.L \prec_z \overline{D}}$$

<:-rfn

$$\frac{\Gamma \vdash S <: T \ , \ S \prec_z \overline{D'} \qquad \Gamma, z : S \vdash \overline{D' <: D}}{\Gamma \vdash S <: T\{z \Rightarrow \overline{D}\}}$$

Functions as Objects

$$// \; f \text{ is an undefined function!}$$

$$\textbf{val } u = \textbf{new } \top \{z \Rightarrow C : \top \to \top .. \top \to \top\} \{\};$$
$$\textbf{val } f = \textbf{new } u.C \{\};$$
$$(\lambda g : \top \to \top. \;\; g\,(\lambda x : \top. \;\; x))\, f$$

app

$$\frac{\Gamma \vdash t : S \to T \;,\; t' : T' \;,\; T' <: S}{\Gamma \vdash t\; t' : T}$$

(Expansion and) Well-Formedness Lost

// $y.A$ is not well-formed after v is substituted for x.

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

term-$\ni$ Restriction

$$X = \top\{z \Rightarrow L_a : \top..\top, l : z.L_a\}$$
$$Y = \top\{z \Rightarrow l : \top\}$$
$$\textbf{val } u = \textbf{new } X\,\{l = u\};$$
$$((\lambda y{:}\top \to Y.\ \ y\ u)$$
$$(\lambda d{:}\top.\ \ (\lambda x{:}X.\ \ x)\ u)).l$$

path-$\ni$

$$\frac{\Gamma \vdash p : T\ ,\ \ T \prec_z \overline{D}}{\Gamma \vdash p \ni [p/z]D_i}$$

term-$\ni$

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

Path Equality

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

$$\textbf{val } b = \textbf{new } \top\{z \Rightarrow \quad X : \top..\top \quad l : z.X \quad \}\{l = b\};$$

$$\textbf{val } a = \textbf{new } \top\{z \Rightarrow i : \top\{z \Rightarrow \quad X : \bot..\top \quad l : z.X\} \quad \}\{i = b\};$$

$$(\lambda x : \top.\ \ x)\ ((\lambda x : a.i.X.\ \ x)\ a.i.l)$$

Non-Expanding Types and Subtyping Transitivity

1. Introduce a new judgment form for whether a type is well-formed and expanding.

$\Gamma \vdash T$ wfe

$$\frac{\Gamma \vdash T \textbf{ wf} \ , \ T \prec_z \overline{D}}{\Gamma \vdash T \textbf{ wfe}}$$

2. Replace other uses of **wf** with **wfe**.
3. Make subtyping regular with respect to **wfe**.

Functions as Sugar

like in Scala

1. Introduce a new kind of label for methods with one parameter: $m : S \to U$.
2. Return types are now explicit.

Explicit Widening

1. Add widening terms $t : T$.
2. Add widening values $v : T$.
3. Replace relaxed subtyping in app and new with "equality".
4. Add the only typing rule with relaxed subtyping.

wid

$$\frac{\Gamma \vdash t : T' \ , \ T' <: T}{\Gamma \vdash (t : T) : T}$$

5. Reduction becomes complicated and dependent on typing judgments because type widenings need to be propagated outwards.

Path Equality

1. Add store to context of typing judgments.
2. Add path-equality provisions in subtyping.

<:-path

$$\frac{\Gamma, s \vdash p \to q\ ,\ T <: q.L}{\Gamma, s \vdash T <: p.L}$$

path-<:

$$\frac{\Gamma, s \vdash p \to q\ ,\ q.L <: T}{\Gamma, s \vdash p.L <: T}$$

3. Path reduction not quite like reduction, since it roughly skips over widenings.

The Final Blow

by the much needed path-$<:$

$$//d \text{ has type } \bot \text{ if ignoring the cast on } b$$

$$\textbf{val } a = \textbf{new } \top\{z \Rightarrow C : \bot..\top\{z \Rightarrow D : \bot..z.X, X : \bot..\top\}\};$$

$$\textbf{val } b = \textbf{new } a.C\{z \Rightarrow X : \bot..\bot\};$$

$$\textbf{val } c = \textbf{new } a.C;$$

$$\textbf{val } d = \textbf{new } (b : a.C).D;$$

$$(\lambda x : \bot.\ \ x.foo)\ d$$

Conclusion

1. No soundness proof but lots of soundness holes.
2. Patches to calculus affect its elegance.
3. Going forward by getting broader perspective with PL summer school and candidacy exam on dependent object types.
4. Will keep DOT in mind, and come back to it with fresh and broader perspective.