Type Soundness for DOT
(Dependent Object Types)

Tiark Rompf *Nada Amin*

OOPSLA

November 3, 2016

DOT Syntax

$x ::=$	**variables**:
y	concrete var.
z	abstract var.
$t ::=$	**terms**:
x	variable
$\{z \Rightarrow \overline{d}\}$	new object
$t.m(t)$	meth. app.
$d ::=$	**init.**:
$L = T$	type mem.
$m(x : T) = t$	meth. mem.
$v ::=$	**values**:
y	store loc.

$S, T, U ::=$	**types**:
$\top$	top
$\bot$	bot.
$T \wedge T$	inter.
$T \vee T$	union
$L : S..U$	type mem.
$m(x : S) : U$	meth. mem.
$x.L$	sel.
$\{z \Rightarrow T\}$	rec. self
$\Gamma ::=$	**contexts**:
$\emptyset \mid \Gamma, x : T$	var. bind.
$\rho ::=$	**stores**:
$\emptyset \mid \rho, y : \overline{d}$	val. bind.

DOT Types

$S, T, U ::=$	
	subtyping lattice:
$\top$	top
$\bot$	bottom
$T \wedge T$	intersection
$T \vee T$	union
	structural member types:
$L : S..U$	type member
$m(x : S) : U$	(U may depend on x) method member
	path-dependent types:
$x.L$	type selection
	recursive type:
$\{z \Rightarrow T\}$	(T may depend on z) recursive self type

Semantic Intuition for Path-Dependent Types

$$\frac{\Gamma \vdash x : (L : S..U)}{\Gamma \vdash S <: x.L <: U} \quad \text{(Sel)}$$

Type Members, Path-Dependent Types

```scala
trait Keys {
  type Key
  def key(data: String): Key
}

object hashKeys extends Keys {
  type Key = Int
  def key(s: String) = s.hashCode
}

def mapKeys(k: Keys, ss: List[String]): List[k.Key] =
  ss.map(k.key)
```

Translucency

$$\frac{\Gamma \vdash x : (L : S..U)}{\Gamma \vdash S <: x.L <: U} \quad \text{(Sel)}$$

```
val abstracted: Keys = hashKeys
val transparent: Keys { type Key = Int } = haskKeys
val upperBounded: Keys { type Key <: Int } = hashKeys
val lowerBounded: Keys { type Key >: Int } = hashKeys

(1: lowerBounded.Key)
(upperBounded.key("a"): Int)
```

DOT as a type-theoretic foundation

- few yet powerful concepts,
  with uniform means of abstraction and combination
  e.g. quantification only over term, yet supports polymorphism
- “user-extensible” subtyping
- mixture of nominal and structural
- nominality is “scoped”
  e.g. no global class table
- no imposed notion of code sharing
  such as prototype vs class inheritance, mixins, ...

Impact on Practice, particularly Scala/Dotty

- suggesting simplifications,
  e.g. a core type system based on DOT
- lifting ad-hoc restrictions,
  e.g. recursive structural types are more powerful in DOT than in Scala
- characterizing soundness issues,
  e.g. type selection on `Null` or $\bot$ paths

Impact on Practice, particularly Scala/Dotty

- suggesting simplifications,
  e.g. a core type system based on DOT
- lifting ad-hoc restrictions,
  e.g. recursive structural types are more powerful in DOT than in Scala
- characterizing soundness issues,
  e.g. type selection on `Null` or $\perp$ paths

Java and Scala's Type Systems are Unsound:
The Existential Crisis of Null Pointers
N. Amin & R. Tate, OOPSLA'16
See Ross's talk tomorrow at 11:45 in Matterhorn 1. :)

The Quest for Soundness

Review of Negative Results (Amin et al. OOPSLA '14)

- (invertible) transitivity

$$\frac{\Gamma \vdash S <: T \ , \ T <: U}{\Gamma \vdash S <: U} \qquad \text{(<:-trans)}$$

- narrowing

$$\frac{\begin{array}{c}\Gamma^a, (x : U), \Gamma^b \vdash T <: T' \\ \Gamma^a \vdash S <: U\end{array}}{\Gamma^a, (x : S), \Gamma^b \vdash T <: T'} \qquad \text{(<:-narrow)}$$

- lattice collapse through bad bounds

$$\frac{\Gamma \vdash x : (L : \top..\bot)}{\Gamma \vdash \top <: x.L <: \bot} \qquad \text{(Sel)}$$

Main Breakthrough

distinction between concrete and abstract variables
(proof device)

Recursive Subtyping

$$\frac{\Gamma, z : T_1 \vdash T_1 <: T_2}{\Gamma \vdash \{z \Rightarrow T_1\} <: \{z \Rightarrow T_2\}} \qquad \text{(BindX)}$$

$$\frac{\Gamma, z : T_1 \vdash T_1 <: T_2 \qquad z \notin fv(T_2)}{\Gamma \vdash \{z \Rightarrow T_1\} <: T_2} \qquad \text{(Bind1)}$$

enables comparisons like

```
scalalib.List & { type Elem = Apple }
<:
{ z => type Elem <: Fruit; def head: z.Elem }
```

Mutual Dependencies in Lemmas!

- $z_1 : T_1,\ z_2 : T_2 \vdash \{z_3 \Rightarrow T_3\} <: \{z_3 \Rightarrow T_3'\}$

  $\ldots$

  $z_1 : T_1,\ z_2 : T_2,\ z_3 : T_3 \vdash z_2.L <: U$
- No one breakthrough or principle.
- Hard work refactoring the bottom-up proof to break cycles.
- Inversion lemmas, substitution lemmas, equivalence lemmas all got entangled.
- Unpacking self types as part of subtyping.
- Growing context leads to more proof obligations that may need to unpack, and grow context again.

Contractiveness Restrictions in Model

in typing for subtyping (now $:_!$ instead of $:$)

- no self packing
- context "popping" (left-to-right behavior) $\Gamma_{[x]}$
  $(z_1 : T_1,\ z_2 : T_2,\ z_3 : T_3)_{[z_2]} \equiv (z_1 : T_1,\ z_2 : T_2)$
- does not seem to matter much for most purposes, maybe affects semantics of strange loops, e.g. $\{z \Rightarrow z.L \ \wedge \ (L : S..U)\}$
- seems similar to restrictions on recursive structural types for decidability (as opposed to soundness)

Formal Semantics

DOT Subtyping $\boxed{\Gamma \vdash S <: U}$

Lattice structure

$$\Gamma \vdash \bot <: T \quad \text{(Bot)} \qquad\qquad \Gamma \vdash T <: \top \quad \text{(Top)}$$

$$\frac{\Gamma \vdash T_1 <: T}{\Gamma \vdash T_1 \wedge T_2 <: T} \text{(And11)} \qquad\qquad \frac{\Gamma \vdash T <: T_1}{\Gamma \vdash T <: T_1 \vee T_2} \text{(Or21)}$$

$$\frac{\Gamma \vdash T_2 <: T}{\Gamma \vdash T_1 \wedge T_2 <: T} \text{(And12)} \qquad\qquad \frac{\Gamma \vdash T <: T_2}{\Gamma \vdash T <: T_1 \vee T_2} \text{(Or22)}$$

$$\frac{\Gamma \vdash T <: T_1 \;,\; T <: T_2}{\Gamma \vdash T <: T_1 \wedge T_2} \text{(And2)} \qquad\qquad \frac{\Gamma \vdash T_1 <: T \;,\; T_2 <: T}{\Gamma \vdash T_1 \vee T_2 <: T} \text{(Or1)}$$

Properties

$$\Gamma \vdash T <: T \quad \text{(Refl)} \qquad\qquad \frac{\Gamma \vdash T_1 <: T_2 \;,\; T_2 <: T_3}{\Gamma \vdash T_1 <: T_3} \text{(Trans)}$$

DOT Subtyping $\boxed{\Gamma \vdash S <: U}$

Method and type members

$$\frac{\begin{array}{c}\Gamma \vdash S_2 <: S_1 \\ \Gamma, x : S_2 \vdash U_1 <: U_2\end{array}}{\Gamma \vdash m(x : S_1) : U_1 <: m(x : S_2) : U_2} \quad \text{(Fun)}$$

$$\frac{\Gamma \vdash S_2 <: S_1 \ , \ U_1 <: U_2}{\Gamma \vdash L : S_1..U_1 <: L : S_2..U_2} \quad \text{(Typ)}$$

DOT Subtyping $\boxed{\Gamma \vdash S <: U}$

Type selections

$$\frac{\Gamma_{[x]} \vdash x :_! (L : T..\top)}{\Gamma \vdash T <: x.L} \quad \text{(Sel2)} \qquad \frac{\Gamma_{[x]} \vdash x :_! (L : \bot..T)}{\Gamma \vdash x.L <: T} \quad \text{(Sel1)}$$

Recursive self types

$$\frac{\Gamma, z : T_1 \vdash T_1 <: T_2}{\Gamma \vdash \{z \Rightarrow T_1\} <: \{z \Rightarrow T_2\}} \quad \text{(BindX)}$$

$$\frac{\begin{array}{c}\Gamma, z : T_1 \vdash T_1 <: T_2 \\ z \notin fv(T_2)\end{array}}{\Gamma \vdash \{z \Rightarrow T_1\} <: T_2} \quad \text{(Bind1)}$$

DOT Typing $\boxed{\Gamma \vdash t :_{(!)} T}$

$$\frac{\Gamma(x) = T}{\Gamma \vdash x :_{(!)} T} \quad \text{(Var)} \qquad \frac{\Gamma \vdash t :_{(!)} T_1 \ , \ T_1 <: T_2}{\Gamma \vdash t :_{(!)} T_2} \quad \text{(Sub)}$$

$$\frac{\Gamma \vdash x : T}{\Gamma \vdash x : \{z \Rightarrow [x \mapsto z] T\}} \ \text{(VarPack)} \qquad \frac{\Gamma \vdash x :_{(!)} \{z \Rightarrow T\}}{\Gamma \vdash x :_{(!)} [z \mapsto x] T} \ \text{(VarUnpack)}$$

DOT Typing $\boxed{\Gamma \vdash t : T}$

$$\frac{\begin{array}{c}\Gamma \vdash t : (m(x : T_1) : T_2) \ , \ t_2 : T_1 \\ x \notin fv(T_2)\end{array}}{\Gamma \vdash t.m(t_2) : T_2} \qquad \text{(TApp)}$$

$$\frac{\Gamma \vdash t : (m(x : T_1) : T_2) \ , \ y : T_1}{\Gamma \vdash t.m(y) : [x \mapsto y] T_2} \qquad \text{(TAppVar)}$$

$$\frac{\begin{array}{c}\text{(labels disjoint)} \\ \Gamma, x : T_1 \wedge \ldots \wedge T_n \vdash d_i : T_i \qquad \forall i, 1 \leq i \leq n\end{array}}{\Gamma \vdash \{x \Rightarrow d_1 \ldots d_n\} : \{x \Rightarrow T_1 \wedge \ldots \wedge T_n\}} \qquad \text{(TNew)}$$

DOT Member Initialization $\boxed{\Gamma \vdash d : T}$

$$\frac{\Gamma \vdash T <: T}{\Gamma \vdash (L = T) : (L : T..T)} \quad \text{(DTyp)}$$

$$\frac{\Gamma, x : T_1 \vdash t : T_2}{\Gamma \vdash (m(x) = t) : (m(x : T_1) : T_2)} \quad \text{(DFun)}$$

DOT Small-Step Operational Semantics $\boxed{\rho\ t \ \longrightarrow\ t'\ \rho'}$

$$\rho \quad \{z \Rightarrow \overline{d}\} \quad \longrightarrow \quad y \quad \rho, (y : [z \mapsto y]\overline{d}) \quad \textbf{with } y \text{ fresh}$$

$$\rho \quad y_1.m(v_2) \quad \longrightarrow \quad [x \mapsto v_2]t \quad \rho \quad \textbf{if } \rho(y_1) \ni m(x) = t$$

$$\rho \quad e[t] \quad \longrightarrow \quad e[t'] \quad \rho' \quad \textbf{if } \rho\ t \ \longrightarrow\ t'\ \rho'$$

$$\textbf{where } e ::= [\,] \mid [\,].m(t) \mid v.m([\,])$$

DOT Store Typing

Subtyping... $\boxed{\rho\ \Gamma \vdash\ S <: U}$

$$\frac{\rho(y) \ni (L = U) \qquad \rho\ \emptyset \vdash\ T <: U}{\rho\ \Gamma \vdash\ T <: y.L} \tag{SSel2}$$

$$\frac{\rho(y) \ni (L = U) \qquad \rho\ \emptyset \vdash\ U <: T}{\rho\ \Gamma \vdash\ y.L <: T} \tag{SSel1}$$

Typing... $\boxed{\rho\ \Gamma \vdash\ t :_{(!)}\ T}$

$$\frac{\begin{array}{c}(y, \overline{d}) \in \rho \\ \text{(labels disjoint)} \qquad \forall i, 1 \leq i \leq n \\ \rho\ \emptyset, (x : T_1 \wedge \ldots \wedge T_n) \vdash\ [y \mapsto x]d_i : T_i\end{array}}{\rho\ \Gamma \vdash\ y : [x \mapsto y](T_1 \wedge \ldots \wedge T_n)} \tag{TLoc}$$

Some Unsound Variations on Typing Objects

1. Add subsumption to member initialization.

$$\frac{\Gamma \vdash d : T \quad \Gamma \vdash T <: U}{\Gamma \vdash d : U} \qquad \text{(DSub)}$$

$$\{x \Rightarrow L = \top\} : \{x \Rightarrow L : \top..\bot\}$$

2. Change type member initialization from $\{L = T\}$ to $\{L : S..U\}$.

$$\frac{\Gamma \vdash S <: U}{\{L : S..U\} : \{L : S..U\}} \qquad \text{(DTyp)}$$

$$\{x \Rightarrow L : \top..\bot\} : \{x \Rightarrow L : \top..\bot\}$$

Unsound Variation 1

Adding subsumption to definition type assignment...

$$\frac{\Gamma \vdash d : T \quad \Gamma \vdash T <: U}{\Gamma \vdash d : U} \qquad (\text{Def-Sub})$$

... would fail to flag this bad typing:

$$\{x \Rightarrow X = \top\} : \{x \Rightarrow X : \top..\bot\}$$

... because the following typing derivation tree would be accepted:

$$\dfrac{\dfrac{}{x : (X : \top..\bot) \vdash (X = \top) : (X : \top..\top)}\,\textbf{(DTyp)} \quad \dfrac{\dfrac{x : (X : \top..\bot) \vdash \top <: x.X <: \bot}{x : (X : \top..\bot) \vdash \top <: \bot}\,\textbf{(Trans)}}{x : (X : \top..\bot) \vdash (X : \top..\top) <: (X : \top..\bot)}\,\textbf{(Typ)}}{\dfrac{x : (X : \top..\bot) \vdash (X = \top) : (X : \top..\bot)}{\emptyset \vdash \{x \Rightarrow X = \top\} : \{x \Rightarrow X : \top..\bot\}}\,\textbf{(TObj)}}\,\textbf{(Def-Sub)}$$

Unsound Variation 2

Changing type definition from $\{L = T\}$ to $\{L : S..U\}$ ...

$$\frac{\Gamma \vdash \boxed{S <: U}}{\Gamma \vdash \boxed{\{L : S..U\} : \{L : S..U\}}} \qquad \text{(DTyp')}$$

... would fail to flag this bad typing:

$$\{x \Rightarrow X : \top..\bot\} : \{x \Rightarrow X : \top..\bot\}$$

... because the following typing derivation tree would be accepted:

$$\dfrac{\dfrac{\dfrac{x : (X : \top..\bot) \vdash \top <: x.X <: \bot}{x : (X : \top..\bot) \vdash \top <: \bot}\,\textbf{(Trans)}}{x : (X : \top..\bot) \vdash (X : \top..\bot) : (X : \top..\bot)}\,\textbf{(DTyp')}}{\emptyset \vdash \{x \Rightarrow X : \top..\bot\} : \{x \Rightarrow X : \top..\bot\}}\,\textbf{(TObj)}$$

Retrospective on Proving Soundness

A good proof is one that makes us wiser. – Yuri Manin

- Static semantics should be monotonic. All attempts to prevent bad bounds broke it.
- Embrace subsumption, don't requires precise calculations in arbitrary contexts.
- Inversion lemmas need only hold in empty abstract environment.
- Rely on precise types for runtime values. Distinguish between concrete and abstract identifiers.
- Create recursive objects concretely, enforcing good bounds and shape syntactically not semantically. Then subsume/abstract, if desired.
- Recursive subtyping introduce cycles in the proof. Entangle with care.
- Designing the calculus and proving its soundness are intertwined.
- Happy journey: sound calculus is simpler yet more regular, powerful and expressive than initial design.

Thank you and **SPLASH** community!

- **FOOL 2012, SPLASH** in Tucson, Arizona, USA
  Dependent Object Types, *N. Amin*, A. Moors, M. Odersky
- **OOPSLA 2014, SPLASH** in Portland, Oregon, USA
  Foundations of Path-Dependent Types, N. Amin, *T. Rompf*, M. Odersky
- **OOPSLA 2016, SPLASH** in Amsterdam, Netherlands
  Type Soundness for DOT, T. Rompf and *N. Amin*

- Purdue University Tech. Report 2015
  From F to DOT: Type Soundness Proofs with Definitional Interpreters,
  T. Rompf and N. Amin
- WadlerFest 2016 in Edinburgh, Scotland, UK
  The Essence of DOT, N. Amin, S. Grütter, *M. Odersky*, S. Stucki, T. Rompf
- EPFL Thesis, 2016
  Dependent Object Types, N. Amin
- POPL 2017 in Paris, France
  Type Soundness Proofs with Definitional Interpreters, N. Amin and T. Rompf

Q&A: Methods & Tools

- **FOOL 2012** logical relations, small-step, PLT Redex, Coq, Dafny
  Dependent Object Types, *N. Amin*, A. Moors, M. Odersky
- **OOPSLA 2014** big-step, Twelf
  Foundations of Path-Dependent Types, N. Amin, *T. Rompf*, M. Odersky
- **OOPSLA 2016** small-step, Coq
  Type Soundness for DOT, T. Rompf and *N. Amin*

- Purdue University Tech. Report 2015 big-step, small-step, Coq
  From F to DOT: Type Soundness Proofs with Definitional Interpreters,
  T. Rompf and N. Amin
- WadlerFest 2016 small-step, Pen&Paper, Coq, Agda
  The Essence of DOT, N. Amin, S. Grütter, *M. Odersky*, S. Stucki, T. Rompf
- EPFL Thesis, 2016
  Dependent Object Types, N. Amin
- POPL 2017 big-step, Coq
  Type Soundness Proofs with Definitional Interpreters, N. Amin and T. Rompf