

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
$\{z \Rightarrow \bar{d}\}$	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
\perp	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:
$\emptyset \mid \rho, y : \bar{d}$	val. bind.

DOT Types

$S, T, U ::=$

\top

\perp

$T \wedge T$

$T \vee T$

$L : S..U$

$m(x : S) : U$

$x.L$

$\{z \Rightarrow T\}$

subtyping lattice:

top

bottom

intersection

union

structural member types:

type member

(U may depend on x) method member

path-dependent types:

type selection

recursive type:

(T may depend on z) recursive self type

Semantic Intuition for Path-Dependent Types

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

(Sel)

Type Members, Path-Dependent Types

```
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

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

(Sel)

```
val abstracted: Keys = hashKeys
val transparent: Keys { type Key = Int } = hashKeys
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 \perp 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

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

- ▶ narrowing

$$\frac{\Gamma^a, (x : U), \Gamma^b \vdash T <: T' \quad \Gamma^a \vdash S <: U}{\Gamma^a, (x : S), \Gamma^b \vdash T <: T'} \quad (<:-\text{narrow})$$

- ▶ lattice collapse through bad bounds

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

Main Breakthrough

distinction between concrete and abstract variables
(proof device)

Recursive Subtyping

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

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

enables comparisons like

```
scalalib.List & { type Elem = Apple }
```

```
<:
```

```
{ z => type Elem <: Fruit; def head: z.Elem }
```

Mutual Dependencies in Lemmas!

- ▶ $z_1 : T_1, z_2 : T_2 \vdash \{z_3 \Rightarrow T_3\} <: \{z_3 \Rightarrow T'_3\}$
...
 $z_1 : T_1, z_2 : T_2, z_3 : T_3 \vdash z_2.L <: U$
- ▶ 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 $:\downarrow$ 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

$$\Gamma \vdash \perp <: T \quad (\text{Bot})$$

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

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

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

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

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

$$\frac{\Gamma \vdash T <: T_1, T <: T_2}{\Gamma \vdash T <: T_1 \wedge T_2} \quad (\text{And2})$$

$$\frac{\Gamma \vdash T_1 <: T, T_2 <: T}{\Gamma \vdash T_1 \vee T_2 <: T} \quad (\text{Or1})$$

Properties

$$\Gamma \vdash T <: T \quad (\text{Refl})$$

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

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

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

$$\frac{\Gamma_{[x]} \vdash x :! (L : \perp..T)}{\Gamma \vdash x.L <: T} \quad (\text{Sel1})$$

Recursive self types

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

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

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

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : (!) T} \quad (\text{Var})$$

$$\frac{\Gamma \vdash t : (!) T_1, T_1 <: T_2}{\Gamma \vdash t : (!) T_2} \quad (\text{Sub})$$

$$\frac{\Gamma \vdash x : T}{\Gamma \vdash x : \{z \Rightarrow [x \mapsto z] T\}} \quad (\text{VarPack})$$

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

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

$$\frac{\Gamma \vdash t : (m(x : T_1) : T_2) , t_2 : T_1 \quad x \notin \text{fv}(T_2)}{\Gamma \vdash t.m(t_2) : T_2} \quad (\text{TApp})$$

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

$$\frac{\text{(labels disjoint)} \quad \Gamma, x : T_1 \wedge \dots \wedge T_n \vdash d_i : T_i \quad \forall i, 1 \leq i \leq n}{\Gamma \vdash \{x \Rightarrow d_1 \dots d_n\} : \{x \Rightarrow T_1 \wedge \dots \wedge T_n\}} \quad (\text{TNew})$$

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 $\boxed{\rho \ t \longrightarrow t' \ \rho'}$

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

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

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

where $e ::= [] \mid [].m(t) \mid v.m([])$

DOT Store Typing

Subtyping...

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

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

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

Typing...

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

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

Some Unsound Variations on Typing Objects

1. Add subsumption to member initialization.

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

$$\{x \Rightarrow L = T\} : \{x \Rightarrow L : T.. \perp\}$$

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

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

$$\{x \Rightarrow L : T.. \perp\} : \{x \Rightarrow L : T.. \perp\}$$

Unsound Variation 1

Adding subsumption to definition type assignment...

$$\frac{\Gamma \vdash d : T \quad \Gamma \vdash T <: U}{\Gamma \vdash d : U}$$

(Def-Sub)

... would fail to flag this bad typing:

$$\{x \Rightarrow X = T\} : \{x \Rightarrow X : T.. \perp\}$$

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

$$\frac{\frac{\frac{\frac{x : (X : T.. \perp) \vdash T <: x.X <: \perp}{(Trans)}}{x : (X : T.. \perp) \vdash T <: \perp}}{\frac{\frac{x : (X : T.. \perp) \vdash (X = T) : (X : T.. T)}{(DTyp)} \quad \frac{x : (X : T.. \perp) \vdash (X : T.. T) <: (X : T.. \perp)}{(Typ)}}{x : (X : T.. \perp) \vdash (X = T) : (X : T.. \perp)} \quad (Def-Sub)}{\frac{x : (X : T.. \perp) \vdash (X = T) : (X : T.. \perp)}{(TObj)}}{\emptyset \vdash \{x \Rightarrow X = T\} : \{x \Rightarrow X : T.. \perp\}}$$

Unsound Variation 2

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

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

... would fail to flag this bad typing:

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

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

$$\frac{x : (X : \top.. \perp) \vdash \top <: x.X <: \perp}{x : (X : \top.. \perp) \vdash \top <: \perp} \quad (\text{Trans})$$
$$\frac{}{x : (X : \top.. \perp) \vdash (X : \top.. \perp) : (X : \top.. \perp)} \quad (\text{DTyp}'')$$
$$\frac{}{\emptyset \vdash \{x \Rightarrow X : \top.. \perp\} : \{x \Rightarrow X : \top.. \perp\}} \quad (\text{TObj})$$

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