

# Path Dependent Types with Path-Equality

In Proceedings of the 9th ACM SIGPLAN International Scala Symposium (Scala '18)

val authors: List[KAIST.PLRG] = Jaemin\_Hong :: Jihyeok\_Park :: Sukyoung\_Ryu :: Nil

## 1 Introduction

- In Scala, objects have type members and types depend on paths.

```
class C { type T }
val (c0, c1): (C, C) = (new C, new C)
val t: c0.T = ???
t: c1.T // type error: c1.T != c0.T
```

- Static analyzers use abstract domains, which summarize concrete domains.
- AbsDom, whose elements have type Elem, corresponds to domain V.

```
trait AbsDom[V] {
  type Elem <: ElemTrait
  trait ElemTrait { this: Elem => }
}
```

- Abstract domains can be implemented in various ways for the same concrete domain.

```
object A extends AbsDom[Int] { ... }
object B extends AbsDom[Int] { ... }

val (a, b): (A.Elem, B.Elem) = (A.alpha(0), B.alpha(1))
(a + a): A.Elem
a + b // type error: A.Elem != B.Elem
```

- Create a new domain by combining multiple abstract domains.
- The new domain is unuseful due to the lack of path-equality.

```
trait PairDom[L, R](
  val domL: AbsDom[L],
  val domR: AbsDom[R]
) { this: AbsDom[(L, R)] => ... }

object AB extends AbsDom[(Int, Int)]
  with PairDom[Int, Int](A, B) { ... }

val ab: AB.Elem = AB.alpha((0, 1))
val a: AB.domL.Elem = elem.left
a: A.Elem // type error: A.Elem != AB.domL.elem
```

## 2 Formalization of $\pi$ DOT

### 2.1 Syntax

- A type is a record type or a type selection on a path.

term  $s, t, u ::= x \mid t.f \mid \text{new } \bar{I} \mid \text{let } x : T = t \text{ in } t$   
member init.  $I ::= \tau \mid \text{val } f = t$   
type member decl.  $\tau ::= \text{type } L = T \dots T \mid \text{type } L <: T$   
type  $S, T, U ::= \{\bar{D}\} \mid p.L$   
member decl.  $D ::= \tau \mid \text{val } f : T$   
path  $p, q ::= x \mid p.f$

### 2.2 Dynamic Semantics

- Defined in a big-step style with the total evaluation rules.
- Has the normalization property.

$$\frac{\Sigma = \_ , x : v, \_}{\Sigma \vdash x \Downarrow v}$$
$$\frac{\Sigma = \bar{y} : \bar{v} \quad \forall 1 \leq i \leq |\bar{y}|. x \neq y_i}{\Sigma \vdash x \Downarrow \text{stuck}}$$

### 2.3 Static Semantics

- The typing rule for let binding updates a path environment.

$$\frac{\Gamma; \Psi \vdash t : T \quad t \text{ implies } \langle \bar{\rho} \equiv \bar{\pi} \rangle \quad \text{expand}(\Psi, x, \langle \bar{\rho} \equiv \bar{\pi} \rangle) = \Psi' \quad \Gamma, x : T; \Psi' \vdash s : U \quad \Gamma; \Psi \vdash U}{\Gamma; \Psi \vdash \text{let } x : T = t \text{ in } s : U}$$

- The subtyping rules check path-equality in a path environment.

$$\frac{\psi \in \Psi \quad p, q \in \psi}{\Gamma; \Psi \vdash p.L <: q.L}$$
$$\frac{\psi \in \Psi \quad p, q \in \psi \quad \Gamma; \Psi \vdash q : \{\text{type } L = S \dots U\} \quad \Gamma; \Psi \vdash U <: T}{\Gamma; \Psi \vdash p.L <: T}$$

- The implies function searches for the same paths from a given term.

$$\frac{t \neq p}{t.f \text{ implies } \langle \cdot \equiv \text{null} \rangle}$$
$$\frac{I \text{ implies } \langle \bar{\rho} \equiv \bar{\pi} \rangle}{\text{new } \bar{I} \text{ implies } \langle \bar{\rho} \equiv \bar{\pi}, \cdot \equiv \text{null} \rangle}$$

- The expand function adds the same paths to a path environment.

$$\frac{\text{make}(x; \rho') = p \quad \Psi = \{\bar{\psi}\} \quad \forall 1 \leq i \leq |\bar{\psi}|. \psi'_i = \psi_i \cup \{p' \mid \text{make}(q; \rho'') = q' \wedge q' \in \psi_i \wedge \text{make}(p; \rho'') = p'\} \quad \Psi' = \{\bar{\psi}'\} \quad \text{expand}(\Psi', x, \langle \bar{\rho} \equiv \bar{\pi} \rangle) = \Psi''}{\text{expand}(\Psi, x, \langle \bar{\rho} \equiv \bar{\pi}, \rho' \equiv q \rangle) = \Psi''}$$

### 2.4 Type Soundness

- Proof based on the subtype transitivity and the inversion lemmas.

**Theorem** (Type Soundness).

$$\frac{\Sigma : \Gamma; \Psi \quad \Gamma; \Psi \vdash t : T \quad \Sigma \vdash t \Downarrow r}{r = v \wedge \Gamma; \Psi \vdash v : T}$$

**Lemma** (Subtyping Transitivity).

$$\frac{\Gamma; \Psi \vdash S <: T \quad \Gamma; \Psi \vdash T <: U}{\Gamma; \Psi \vdash S <: U}$$

**Lemma** (Inversion Lemma for Variables).

$$\frac{\Gamma; \Psi \vdash x : T}{\Gamma = \_ , x : T', \_ \wedge \Gamma; \Psi \vdash T' <: T}$$

### 2.5 Implementation using Singleton Types

- In Scala, for a given path, singleton type, which contains only a value bound to the path, can be defined.
- $p.type$  is a singleton type for path  $p$ .
- $p.T$  equals to  $p.type\#T$ , which is a type projection from  $p.type$ .

```
trait PairDom[L, R,
  DL <: AbsDom[L] with Singleton,
  DR <: AbsDom[R] with Singleton](
  val domL: DL,
  val domR: DR
) { this: AbsDom[(L, R)] => ... }

object AB extends AbsDom[(Int, Int)] with
  PairDom[Int, Int, A.type, B.type](A, B) { ... }

val ab: AB.Elem = AB.alpha((0, 1))
val a: AB.domL.Elem = elem.left
a: A.Elem
```