Path Dependent Types with Path-Equality

Jaemin Hong, Jihyeok Park, and Sukyoung Ryu from PLRG@KAIST

jaemin.hong@kaist.ac.kr

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.α(0), B.α(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 πDOT

2.1 Syntax

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

$$
\begin{array}{llll}
\text{term} & s,t,u & ::= & x \mid t.f \mid \text{new}\ (\overline{I}) \mid \text{let}\ x : T = t\ \text{in}\ t \\
\text{member init.} & I & ::= & \tau \mid \text{val}\ f = t \\
\text{type member decl.} & \tau & ::= & \text{type}\ L = T\ ..\ T \mid \text{type}\ L <: T \\
\text{type} & S,T,U & ::= & \{\overline{D}\} \mid p.L \\
\text{member decl.} & D & ::= & \tau \mid \text{val}\ f : T \\
\text{path} & p,q & ::= & x \mid p.f
\end{array}
$$

2.2 Dynamic Semantics

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

$$\frac{\Sigma = \_, x : v, \_}{\Sigma \vdash x \Downarrow v}$$

$$\frac{\Sigma = \overline{y : v} \quad \forall\ 1 \le i \le |\overline{y}|.\ x \neq y_i}{\Sigma \vdash x \Downarrow \text{stuck}}$$

2.3 Static Semantics

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

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

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

$$\frac{\psi \in \Psi \quad p, q \in \psi}{\Gamma; \Psi \vdash p.L <: q.L}$$

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

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

$$\frac{t \neq p}{t.f\ \text{implies}\ \langle \cdot \equiv \text{null} \rangle}$$

$$\frac{\overline{I\ \text{implies}\ \langle \overline{\rho \equiv \pi} \rangle}}{\text{new}\ (\overline{I})\ \text{implies}\ \langle \overline{\overline{\rho \equiv \pi}}, \cdot \equiv \text{null} \rangle}$$

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

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

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
```

KAIST

PLRG