Scala Step-by-Step

Soundness for DOT with Step-Indexed Logical Relations in Iris

Paolo G. Giarrusso,[1,2] with Léo Stefanesco,[3] Amin Timany,[4] Lars Birkedal,[4] Robbert Krebbers[2]

[1]BedRock Systems, Inc. [2]Delft University of Technology, The Netherlands [3]IRIF, Université de Paris & CNRS, France
[4]Aarhus University, Denmark

ICFP 2020

Why study Scala and DOT?

- Scala “unifies FP and OOP?”
- **Expressive**:
  ML-like software modules $\Rightarrow$ 1st-class objects
  - Unlike typeclasses and ML modules
- Objects gain **impredicative type members** (!)
  - Relatives of $\mathsf{Type} : \mathsf{Type}$
  - Challenging to prove sound

Scala's Open Problem: Type Soundness

- First Scala version: 2003 [Odersky *et al.*]
- ✓ Soundness proven for DOT calculi, including:
  - WadlerFest DOT [2016, Amin, Grütter, Odersky, Rompf & Stucki]
  - OOPSLA DOT [2016, Rompf & Amin]
  - pDOT [2019, Rapoport & Lhoták]
- × abstract types / data abstraction / parametricity?
- × DOT lags behind Scala

Our Approach: Semantics-first Design

× Preservation & progress (syntactic)

✓ Logical relations model

+ Type soundness
+ Data abstraction

✓ Retrofit DOT over model $\Longrightarrow$ guarded DOT (gDOT):

− Guardedness restrictions (acceptable in our evaluation)
+ More extensible
+ Extra features (see later)

Our Approach: Semantics-first Design

× Preservation & progress (syntactic)

✓ Logical relations model

- + Type soundness
- + Data abstraction

✓ Retrofit DOT over model $\Longrightarrow$ guarded DOT (gDOT):

- − Guardedness restrictions (acceptable in our evaluation)
- + More extensible
- + Extra features (see later)

Our Approach: Semantics-first Design

× Preservation & progress (syntactic)

✓ Logical relations model

  + Type soundness
  + Data abstraction

✓ Retrofit DOT over model $\Rightarrow$ guarded DOT (gDOT):

  – Guardedness restrictions (acceptable in our evaluation)
  + More extensible
  + Extra features (see later)

Our Approach: Semantics-first Design

- × Preservation & progress (syntactic)
- ✓ Logical relations model
  - + Type soundness
  - + Data abstraction
- ✓ Retrofit DOT over model $\Rightarrow$ guarded DOT (gDOT):
  - – Guardedness restrictions (acceptable in our evaluation)
  - + More extensible
  - + Extra features (see later)

A Scala Example

Scala Example: 1st-class Validators

We want `Validator`s that:

- ✓ Validate `Input`s from users
- ✓ Provide:
  - Abstract type `Vld` of valid `Input`
  - Smart constructor `make : Input ⇒ Option[Vld]`
- New validators can be created at runtime
- Each with a distinct **abstract type** `Vld`
- Simplifications:
  - `Input = Int`
  - Input n is valid if greater than k

```scala
val solution = new {
  type Validator = {
    type Vld                    <: Int
    val make :  Int ⇒ Option[this.Vld] }
  val mkValidator : Int ⇒ Validator =
    k ⇒ new {
      type Vld = Int
      val make = n ⇒
        if (n > k) Some(n) else None }
  val pos           = mkValidator(0)
  val fails         = pos.make(-1)   // None
  val works         = pos.make(1)    // Some(1)
  val nope : pos.Vld = 1             // type error
  val legalAges     = mkValidator(   // runtime args!
    askUser("Legal age in your country?"))
}
```

```
val solution = new {
  type Validator = {
    type Vld >: Nothing <: Int
    val make :  Int ⇒ Option[this.Vld] }
  val mkValidator : Int ⇒ Validator =
    k ⇒ new {
      type Vld = Int
      val make = n ⇒
        if (n > k) Some(n) else None }
  val pos          = mkValidator(0)
  val fails        = pos.make(-1)   // None
  val works        = pos.make(1)    // Some(1)
  val nope : pos.Vld = 1            // type error
  val legalAges    = mkValidator(   // runtime args!
    askUser("Legal age in your country?"))
}
```

```scala
val solution = new {
  type Validator = {
    type Vld >: Nothing <: Int
    val make :  Int ⇒ Option[this.Vld] }
  val mkValidator : Int ⇒ Validator =
    k ⇒ new {
      type Vld = Int
      val make = n ⇒
        if (n > k) Some(n) else None }
  val pos            = mkValidator(0)
  val fails          = pos.make(-1)  // None
  val works          = pos.make(1)   // Some(1)
  val nope : pos.Vld = 1             // type error
  val legalAges      = mkValidator(  // runtime args!
    askUser("Legal age in your country?"))
}
```

```
val solution = new {
  type Validator = {
    type Vld >: Nothing <: Int
    val make :  Int ⇒ Option[this.Vld] }
  val mkValidator : Int ⇒ Validator =
    k ⇒ new {
      type Vld = Int
      val make = n ⇒
        if (n > k) Some(n) else None }
  val pos             = mkValidator(0)
  val fails           = pos.make(-1)  // None
  val works           = pos.make(1)   // Some(1)
  val nope : pos.Vld = 1             // type error
  val legalAges       = mkValidator(  // runtime args!
    askUser("Legal age in your country?"))
}
```

```scala
val solution = new {
  type Validator = {
    type Vld >: Nothing <: Int
    val make :  Int ⇒ Option[this.Vld] }
  val mkValidator : Int ⇒ Validator =
    k ⇒ new {
      type Vld = Int
      val make = n ⇒
        if (n > k) Some(n) else None }
  val pos            = mkValidator(0)
  val fails          = pos.make(-1)  // None
  val works          = pos.make(1)   // Some(1)
  val nope : pos.Vld = 1             // type error
  val legalAges      = mkValidator(  // runtime args!
    askUser("Legal age in your country?"))
}
```

Example Summary

- 1^{st}-class modules with abstract types $\mapsto$ Scala objects with (bounded) abstract type members:

$$\frac{\Gamma \vdash L <: T <: U}{\Gamma \vdash \{\texttt{type A} = T\} : \{\texttt{type A} >: L <: U\}}$$

- **impredicative** type members (!)
  - types (`Validator`) with nested type members (`Vld`) are regular types, not "large" types; *e.g.*, `Validator` can be a type member.

Sketching Our Soundness Proof

Logical relation models

$$\begin{array}{rcl}
\text{Type } T & \mapsto & \text{set of values } \mathcal{V}[\![T]\!]: \\
\mathcal{V}[\![S \wedge T]\!] & \triangleq & \mathcal{V}[\![S]\!] \cap \mathcal{V}[\![T]\!] \\
\\
\text{Syn. typing judgment } \vdash J & \mapsto & \textit{sem. typing judgment } \vDash J: \\
\vDash S <: T & \triangleq & \mathcal{V}[\![S]\!] \subseteq \mathcal{V}[\![T]\!] \\
\vDash e : T & \triangleq & e \textbf{ runs safely} \text{ with result in } \mathcal{V}[\![T]\!] \\
\\
\text{Typing rule} & \mapsto & \textit{typing lemma}: \\
\vDash S \wedge T <: S & \Leftrightarrow & \mathcal{V}[\![S \wedge T]\!] \subseteq \mathcal{V}[\![S]\!]
\end{array}$$

Result: extensible type soundness!

Types Members, Naively

$$\mathcal{V}[\![\{\texttt{type A} >: L <: U\}]\!] \triangleq \{v \mid \exists\varphi.\ v.\texttt{A} \searrow \varphi \wedge \mathcal{V}[\![L]\!] \subseteq \varphi \subseteq \mathcal{V}[\![U]\!]\}$$

Types Members, Naively

$$\mathcal{V}[\![\{\texttt{type A} >: L <: U\}]\!] \triangleq \{v \mid \exists\, \varphi.\ v.\texttt{A} \searrow \varphi \wedge \mathcal{V}[\![L]\!] \subseteq \varphi \subseteq \mathcal{V}[\![U]\!]\}$$

$$\mathrm{SemType} \triangleq \mathrm{SemVal} \to \mathrm{Prop}$$

$$\mathrm{SemVal} \cong \ldots + \left(\mathrm{Label} \xrightarrow{\text{fin}} (\mathrm{SemVal} + \mathrm{SemType})\right)$$

Types Members, Naively

$$\mathcal{V}[\![\, \{\texttt{type A} >: L <: U\} \,]\!] \triangleq \{v \mid \exists \varphi.\ v.\mathtt{A} \searrow \varphi \wedge \mathcal{V}[\![L]\!] \subseteq \varphi \subseteq \mathcal{V}[\![U]\!]\}$$

$$\boxed{\mathrm{SemType}} \triangleq \boxed{\mathrm{SemVal}} \rightarrow \mathrm{Prop}$$

$$\boxed{\mathrm{SemVal}} \cong \ldots + \left(\mathrm{Label} \xrightarrow{\mathrm{fin}} (\mathrm{SemVal} + \boxed{\mathrm{SemType}})\right)$$

Types Members, Naively

$$\mathcal{V}[\![\, \{\texttt{type A} >: L <: U\} \,]\!] \triangleq \{v \mid \exists\, \varphi.\ v.\texttt{A} \searrow \varphi \wedge \mathcal{V}[\![\, L \,]\!] \subseteq \varphi \subseteq \mathcal{V}[\![\, U \,]\!]\}$$

$$\boxed{\mathrm{SemType}} \triangleq \boxed{\mathrm{SemVal}} \rightarrow \mathrm{Prop}$$

$$\boxed{\mathrm{SemVal}} \cong \ldots + (\mathrm{Label} \xrightarrow{\mathrm{fin}} (\mathrm{SemVal} + \boxed{\mathrm{SemType}}))$$

$$\boxed{\mathrm{SemVal}} \cong \ldots + (\mathrm{Label} \xrightarrow{\mathrm{fin}} (\mathrm{SemVal} + (\boxed{\mathrm{SemVal}} \rightarrow \mathrm{Prop})))$$

- Unsound negative recursion!
- Exclusive to impredicative type members.

Types Members, Naively

$$\mathcal{V}[\![\{\texttt{type}\ \texttt{A} >: L <: U\}]\!] \triangleq \{v \mid \exists\,\varphi.\ v.\texttt{A} \searrow \varphi \land \mathcal{V}[\![L]\!] \subseteq \varphi \subseteq \mathcal{V}[\![U]\!]\}$$

$$\boxed{\mathrm{SemType}} \triangleq \boxed{\mathrm{SemVal}} \to \mathrm{Prop}$$

$$\boxed{\mathrm{SemVal}} \cong \ldots + (\mathrm{Label} \xrightarrow{\mathrm{fin}} (\mathrm{SemVal} + \boxed{\mathrm{SemType}}))$$

$$\boxed{\mathrm{SemVal}} \cong \ldots + (\mathrm{Label} \xrightarrow{\mathrm{fin}} (\mathrm{SemVal} + (\boxed{\mathrm{SemVal}} \to \mathrm{Prop})))$$

- Unsound negative recursion!
- Exclusive to impredicative type members.

Type Members, Soundly with Iris

$$\mathcal{V}[\![\{\texttt{type}\ \mathsf{A} >: L <: U\}]\!] \triangleq \{v \mid \exists \varphi.\ v.\mathsf{A} \searrow \varphi \wedge \boxed{\triangleright} \mathcal{V}[\![L]\!] \subseteq \boxed{\triangleright} \varphi \subseteq \boxed{\triangleright} \mathcal{V}[\![U]\!]\}$$

$$\boxed{\mathrm{SemType}} \triangleq \boxed{\mathrm{SemVal}} \to \mathrm{iProp}$$

$$\boxed{\mathrm{SemVal}} \cong \ldots + (\mathrm{Label} \xrightarrow{\mathrm{fin}} (\mathrm{SemVal} + \boxed{\blacktriangleright} \boxed{\mathrm{SemType}}))$$

+ Solution: Guard recursion, *i.e.*, "truncate" SemTypes with the later functor $\blacktriangleright$ from Iris.
+ Reason about solution using Iris logic, ignoring details of construction.

Type Members, Soundly with Iris

$$\mathcal{V}[\![\{\texttt{type A} >: L <: U\}]\!] \triangleq \{v \mid \exists\, \varphi.\ v.\texttt{A} \searrow \varphi \land \boxed{\triangleright}\, \mathcal{V}[\![L]\!] \subseteq \boxed{\triangleright}\, \varphi \subseteq \boxed{\triangleright}\, \mathcal{V}[\![U]\!]\}$$

$$\boxed{\mathrm{SemType}} \triangleq \boxed{\mathrm{SemVal}} \to \mathrm{iProp}$$

$$\boxed{\mathrm{SemVal}} \cong \ldots + \left(\mathrm{Label} \xrightarrow{\mathrm{fin}} (\mathrm{SemVal} + \boxed{\blacktriangleright}\boxed{\mathrm{SemType}})\right)$$

– Assertions about φ are weakened through later modality $\triangleright$

Retrofitting DOT over Model: gDOT

- Turn rules from pDOT/OOPSLA DOT into typing lemmas appropriate to the model; each proof is around 2-10 lines of Coq.
- Add type $\triangleright T$ with $\mathcal{V}[\![\triangleright T]\!] \triangleq \triangleright \mathcal{V}[\![T]\!]$ and associated typing rules (!)
  - \+ Stronger/additional rules
    - \+ Abstract types in nested objects (*mutual information hiding*), as in example
    - \+ Distributivity of $\wedge$, $\vee$, ...
    - \+ Subtyping for recursive types (beyond OOPSLA DOT)
  - \+ (Arguably) more principled restrictions

gDOT key typing rules

$$\frac{\Gamma \vdash_{\mathsf{p}} p : \{\mathsf{A} >: L <: U\}}{\Gamma \vdash \boxed{\triangleright} L <: p.\mathsf{A} <: \boxed{\triangleright} U} \ (\text{<:-}\textsc{Sel}, \textsc{Sel}\text{-<:})$$

$$\frac{\Gamma \vdash e : \triangleright T}{\Gamma \vdash \textbf{coerce}\ e : T} \ (\text{T-}\textsc{Coerce})$$

$$\frac{\Gamma \mid x : \boxed{\triangleright} T \vdash \{\bar{d}\} : T}{\Gamma \vdash \nu x.\, \{\bar{d}\} : \mu x.\, T} \ (\text{T-}\{\}\text{-I})$$

$$\frac{\Gamma, x : V \vdash v : T \qquad \cancel{\textbf{tight}\ T}}{\Gamma \mid x : V \vdash \{\mathsf{a} = v\} : \{\mathsf{a} : T\}} \ (\text{D-}\textsc{Val})$$

Contributions/In the paper

- Motivating examples for novel features
- Scale model to gDOT
  - μ-types, singleton types, path-dependent functions, paths(!), …
- Demonstrate expressivity despite guardedness restriction
- Data abstraction proofs
- Coq mechanization using Iris (soundness: $\approx$ 9200 LoC; examples: $\approx$ 5600 LoC)

Future work

- Type projections
- Higher-kinds
- Elaboration from calculi closer to Scala, and $\triangleright$-inference
- Applications to other type systems with impredicative type members/virtual classes

Conclusions

- Scala needs extensible type-soundness $\Rightarrow$ semantics-first
- Challenge: impredicative type members
- Iris enabled machine-checking solution conveniently in Coq