A Core Calculus for Scala Type Checking

Rafał Łasocha

Wrocław, 11th June 2014

Motivation

- Featherweight Scala
- problem of decidability of Scala type checking
- mostly I want to show how may look complete system for type checking and reduction for such language like Scala

Featherweight Scala

- a minimal core calculus of classes that captures an essential set of features of Scala's type system
- subset of Scala (except explicit `self` names)
- classes can have types, values, methods and other classes as members
- types, methods and values can be abstract
- call-by-name evaluation
- deduction rules are syntax-directed

Peano numbers

Peano numbers

```
trait Any extends { this0 | }
trait Nat extends Any { this0 |
  def isZero(): Boolean
  def pred(): Nat
  trait Succ extends Nat { this1 |
    def isZero(): Boolean = false
    def pred(): Nat = this0
  }
  def succ(): Nat = ( val result = new this0.Succ; result )
  def add(other : Nat): Nat = (
    if (this0.isZero()) other
    else this0.pred().add(other.succ()))
}
val zero = new Nat { this0 |
  def isZero(): Boolean = true
  def pred(): Nat = error(  zero  . p r e d )
}
```

List class hierarchy

List class hierarchy

```
trait List extends Any { this0 |
  type Elem
  type ListOfElem = List { this1 | type Elem = this0.Elem }
  def isEmpty(): Boolean
  def head(): this0.Elem
  def tail(): this0.ListOfElem }

trait Nil extends List { this0 |
  def isEmpty(): Boolean = true
  def head(): this0.Elem = error("Nil.head")
  def tail(): this0.ListOfElem = error("Nil.tail") }
trait Cons extends List { this0 |
  val hd : this0.Elem
  val tl : this0.ListOfElem
  def isEmpty(): Boolean = false
  def head(): this0.Elem = hd
  def tail(): this0.ListOfElem = tl }
```

List class hierarchy

```
val nilOfNat = new Nil { type Elem = Nat }

val list2 = new Cons { this0 |
  type Elem = Nat
  val hd : Nat = zero.succ().succ()
  val tl : this0.ListOfElem = nilOfNat
}

val list12 = new Cons { this0 |
  type Elem = Nat
  val hd : Nat = zero.succ()
  val tl : this0.ListOfElem = list2
}
```

First class functions

First class functions

```
trait Function extends Any { this0 |
  type Dom
  type Range
  def apply(x : this0.Dom): this0.Range
}

val inc = new Function { this0 |
  type Dom = Nat
  type Range = Nat
  def apply(x : this0.Dom): this0.Range = x.succ()
}
```

Mapper class (implementation of map function)

Mapper class (implementation of map function)

```
trait Mapper extends Any { t0 |
  type A
  type B
  def map(f: Function { type Dom = t0.A; type Range = t0.B },
      xs: List { type Elem = t0.A }): List { type Elem = t0.B } =
    if (xs.isEmpty()) (
      val result = new Nil {
        type Elem = t0.B
      }; result
    ) else (
      val result = new Cons {
        type Elem = t0.B
        val hd: t0.B = f.apply(xs.head())
        val tl: List { type Elem = t0.B } = t0.map(f, xs.tail())
      }; result
    )
}
```

Mapper class usage

```
val list23 : List { type Elem = Nat } = (
  val mapper = new Mapper { type A = Nat; type B = Nat };
  mapper.map(inc, list12)
)
```

Syntax

- each member in class is associated with unique integer n (it's used for detecting cycles during the static analysis)
- value of fields and methods, type fields may be abstract
- concrete type field is also called type alias

Syntax

x, y, z	Variable
a	Value label
A	Type label
$P ::= \{x \mid \overline{M}\ t\}$	Program
$M, N ::=$	Member decl
$\quad \texttt{val}_n a : T (= t)^?$	Field decl
$\quad \texttt{def}_n a(\overline{y : S}) : T (= t)^?$	Method decl
$\quad \texttt{type}_n A (= T)^?$	Type decl
$\quad \texttt{trait}_n A \texttt{ extends } (T)\{\varphi \mid \overline{M}\}$	Class decl
$s, t, u ::=$	Term
$\quad x$	Variable
$\quad t.a$	Field selection
$\quad s.a(\overline{t})$	Method call
$\quad \texttt{val } x = \texttt{new } T; t$	Object creation

Syntax (paths)

$p ::=$	Path
$\quad x$	Variable
$\quad p.a$	Field selection
$T, U ::=$	Type
$\quad p.A$	Type selection
$\quad p.\mathbf{type}$	Singleton type
$\quad (\overline{T})\ \{\varphi \mid \overline{M}\}$	Type signature

Reduction

$$\frac{\mathtt{val}_n a : T = t \in \Sigma(x)}{\Sigma \; ; \; x.a \to \Sigma \; ; \; t} \text{ (RED-VALUE)}$$

$$\frac{\mathtt{def}_n a(\overline{z : S}) : T = t \in \Sigma(x)}{\Sigma \; ; \; x.a(\bar{y}) \to \Sigma \; ; \; [\bar{y}/\bar{z}]t} \text{ (RED-METHOD)}$$

$$\frac{\Sigma \vdash T \prec_x \overline{M}}{\Sigma \; ; \; \mathtt{val}\ x = \mathtt{new}\ T; t \to \Sigma, x : \overline{M} \; ; \; t} \text{ (RED-NEW)}$$

$$\frac{\Sigma \; ; \; t \to \Sigma' \; ; \; t'}{\Sigma \; ; \; e[t] \to \Sigma' \; ; \; e[t']} \text{ (RED-CONTEXT)}$$

Evaluation contexts

$e ::=$ **term evaluation context**

$\langle\rangle$

$e.a$

$e.a(t)$

$x.a(\overline{s}, e, \overline{u})$

$\texttt{val}\ x = \texttt{new}\ E; t$

$E ::=$ **type evaluation context**

$e.A$

$(\overline{T}, E, \overline{U})\ \{\varphi \mid \overline{M}\}$

Lookup

$$\frac{\texttt{type}_n A = T \in \Sigma(y) \qquad \Sigma \vdash T \prec_\varphi \overline{M}}{\Sigma \vdash y.A \prec_\varphi \overline{M}} \text{ (LOOKUP-ALIAS)}$$

$$\frac{\texttt{trait}_n A \ \texttt{extends}\ (\overline{T})\{\varphi \mid \overline{M}\} \in \Sigma(y) \qquad \Sigma \vdash (\overline{T})\{\varphi \mid \overline{M}\} \prec_\varphi \overline{N}}{\Sigma \vdash y.A \prec_\varphi \overline{N}} \text{ (LOOKUP-CLASS)}$$

$$\frac{\forall_i, \Sigma \vdash T_i \prec_\varphi \overline{N_i}}{\Sigma \vdash (\overline{T})\ \{\varphi \mid \overline{M}\} \prec_\varphi (\biguplus_i \overline{N_i}) \uplus \overline{M}} \text{ (LOOKUP-SIG)}$$

Path Typing

$$\frac{x : T \in \Gamma}{S, \Gamma \vdash_{path} x : T} \text{ (PATH-VAR)}$$

$$\frac{S, \Gamma \vdash p.\mathbf{type} \ni \mathtt{val}_n\ a : T(= t)^?}{S, \Gamma \vdash_{path} p.a : T} \text{ (PATH-SELECT)}$$

Type Assignment

$$\frac{S, \Gamma \vdash_{path} p : T}{S, \Gamma \vdash p : p.\textbf{type}} \text{(PATH)}$$

$$\frac{S, \Gamma \vdash t : U \qquad t \text{ is not a path} \qquad S, \Gamma \vdash U \ni \mathtt{val}_n\ a : T(= u)^?}{S, \Gamma \vdash t.a : T} \text{(SELECT)}$$

$$\frac{\begin{array}{c} S, \Gamma \vdash s : V \qquad S, \Gamma \vdash \overline{t} : \overline{T'} \qquad S, \Gamma \vdash \overline{T'} <: \overline{T} \\ S, \Gamma \vdash V \ni \mathtt{def}_n a(\overline{x : T}) : U(= u)^? \end{array}}{S, \Gamma \vdash s.a(\overline{t}) : U} \text{(METHOD)}$$

$$\frac{S, \Gamma, x : T \vdash t : U \qquad S, \Gamma \vdash T \prec_{\varphi} \overline{M_c} \qquad x \notin fn(U) \qquad S, \Gamma \vdash T \ \mathtt{WF}}{S, \Gamma \vdash \mathtt{val}\ x = \mathtt{new}\ T; t : U} \text{(NEW)}$$

Expansion

$$\frac{n \notin S \qquad \{n\} \cup S, \Gamma \vdash (\overline{T})\ \{\varphi \mid \overline{M}\} \prec_{\varphi} \overline{N} \qquad S, \Gamma \vdash p.\mathbf{type} \ni \mathtt{trait}_n A\ \mathtt{extends}\ (T)\{\varphi \mid \overline{M}\}}{S, \Gamma \vdash p.A \prec_{\varphi} \overline{N}}\ (\prec\text{-CLASS})$$

$$\frac{n \notin S \qquad \{n\} \cup S, \Gamma \vdash T \prec_{\varphi} \overline{M} \qquad S, \Gamma \vdash p.\mathbf{type} \ni \mathtt{type}_n A = T}{S, \Gamma \vdash p.A \prec_{\varphi} \overline{M}}\ (\prec\text{-TYPE})$$

$$\frac{\forall_i, S, \Gamma \vdash T_i \prec_{\varphi} \overline{N_i}}{S, \Gamma \vdash (\overline{T})\ \{\varphi \mid \overline{M}\} \prec_{\varphi} (\biguplus_i \overline{N_i}) \uplus \overline{M}}\ (\prec\text{-SIGNATURE})$$

Membership

$$\frac{S, \Gamma \vdash p \simeq q \qquad S, \Gamma \vdash_{path} q : T \qquad \psi(p) \cup S, \Gamma \vdash T \prec_{\varphi} \overline{M} \qquad \psi(p) \nsubseteq S}{S, \Gamma \vdash p.\mathbf{type} \ni [p/\varphi]M_i} \; (\ni\text{-SINGLETON})$$

$$\frac{T \text{ is not a singleton type} \qquad S, \Gamma \vdash T \prec_{\varphi} \overline{M} \qquad \varphi \notin fn(M_i)}{S, \Gamma \vdash T \ni M_i} \; (\ni\text{-OTHER})$$

Type Alias Expansion

$$\frac{}{S, \Gamma \vdash p.\textbf{type} \simeq p.\textbf{type}}\ (\simeq\text{-SINGLETON})$$

$$\frac{}{S, \Gamma \vdash (\overline{T})\{\varphi \mid \overline{M}\} \simeq (\overline{T})\{\varphi \mid \overline{M}\}}\ (\simeq\text{-SIGNATURE})$$

$$\frac{S, \Gamma \vdash p.\textbf{type} \ni \texttt{trait}_n A\ \texttt{extends}\ (T)\{\varphi \mid \overline{M}\}}{S, \Gamma \vdash p.A \simeq p.A}\ (\simeq\text{-CLASS})$$

$$\frac{S, \Gamma \vdash p.\textbf{type} \ni \texttt{type}_n A}{S, \Gamma \vdash p.A \simeq p.A}\ (\simeq\text{-ABSTYPE})$$

$$\frac{S, \Gamma \vdash p.\textbf{type} \ni \texttt{type}_n A = T \qquad \{n\} \cup S, \Gamma \vdash T \simeq U \qquad n \notin S}{S, \Gamma \vdash p.A \simeq U}\ (\simeq\text{-TYPE})$$

Path Alias Expansion

$$\frac{S, \Gamma \vdash_{path} p : q.\mathbf{type} \qquad \psi(p) \cup S, \Gamma \vdash q \simeq q' \qquad \psi(p) \not\subseteq S}{S, \Gamma \vdash p \simeq q'} \ (\simeq\text{-STEP})$$

$$\frac{T \text{ is not a singleton type} \qquad S, \Gamma \vdash_{path} p : T}{S, \Gamma \vdash p \simeq p} \ (\simeq\text{-REFL})$$

Path Alias Expansion: Example

```
trait D { x|
  val a : y.type = (val y = new (...) {...}; y)
}

// let's name "(...) {...}" as YSYG signature
```

$$
\cfrac{
  \cfrac{
    \cfrac{S,\Gamma \vdash_{path} x : D}{S,\Gamma \vdash x \simeq x}\,(\simeq\text{-REFL}) \quad S,\Gamma \vdash_{path} x : D \quad \ldots
  }{S,\Gamma \vdash x.\mathbf{type} \ni \mathtt{val}_n a : y.\mathbf{type}}\,(\ni\text{-SINGLETON})
}{S,\Gamma \vdash_{path} x.a : y.\mathbf{type}}\,(\text{PATH-SELECT})
\quad
\cfrac{S,\Gamma \vdash_{path} y : YSYG}{S,\Gamma \vdash y \simeq y}\,(\simeq\text{-REFL})
$$

$$
\cfrac{S,\Gamma \vdash_{path} x.a : y.\mathbf{type} \qquad S,\Gamma \vdash y \simeq y}{S,\Gamma \vdash x.a \simeq y}\,(\simeq\text{-STEP})
$$

Algorithmic Subtyping

$$\frac{S, \Gamma \vdash T \simeq T' \qquad S, \Gamma \vdash U \simeq U' \qquad S, \Gamma \vdash_* T' <: U'}{S, \Gamma \vdash T <: U} \ (<:\text{-UNALIAS})$$

$$\frac{S, \Gamma \vdash p \simeq p' \qquad S, \Gamma \vdash q \simeq p'}{S, \Gamma \vdash_* p.\mathbf{type} <: q.\mathbf{type}} \ (<:\text{-SINGLETON-RIGHT})$$

$$\frac{\begin{array}{cc} U \text{ is not singleton type} & S, \Gamma \vdash T <: U \\ S, \Gamma \vdash_{path} q : T & S, \Gamma \vdash p \simeq q \end{array}}{S, \Gamma \vdash_* p.\mathbf{type} <: U} \ (<:\text{-SINGLETON-LEFT})$$

Algorithmic Subtyping

$$\frac{S, \Gamma \vdash p \simeq p' \qquad S, \Gamma \vdash q \simeq p'}{S, \Gamma \vdash_* p.A <: q.A} \text{ (<:-PATHS)}$$

$$\frac{\begin{array}{c} A \neq A' \qquad n \notin S \qquad \{n\} \cup S, \Gamma \vdash T_i <: p'.A' \\ S, \Gamma \vdash p.\mathbf{type} \ni \mathtt{trait}_n A \ \mathtt{extends} \ (\overline{T})\{\varphi \mid \overline{M}\} \end{array}}{S, \Gamma \vdash_* p.A <: p'.A'} \text{ (<:-CLASS)}$$

$$\frac{S, \Gamma \vdash T_i <: p.A}{S, \Gamma \vdash_* (\overline{T})\{\varphi \mid \overline{M}\} <: p.A} \text{ (<:-SIG-LEFT)}$$

Algorithmic Subtyping

$$\frac{\begin{array}{ll} dom(\overline{M}) \subseteq dom(\overline{N}) & \\ S, \Gamma \vdash T \prec_{\varphi} \overline{N} & T \text{ is not a singleton type} \\ \forall_i, S, \Gamma \vdash T <: T_i & \varphi : (\overline{T})\{\varphi \mid \overline{M}\}, S, \Gamma \vdash \overline{N} \ll \overline{M} \end{array}}{S, \Gamma \vdash_* T <: (\overline{T})\{\varphi \mid \overline{M}\}} \;(\text{<:-SIG-RIGHT})$$

Algorithmic Subtyping

$$\frac{\begin{array}{ll} dom(\overline{M}) \subseteq dom(\overline{N}) & \\ S, \Gamma \vdash T \prec_{\varphi} \overline{N} & T \text{ is not a singleton type} \\ \forall_i, S, \Gamma \vdash T <: T_i & \varphi : (\overline{T})\{\varphi \mid \overline{M}\}, S, \Gamma \vdash \overline{N} \ll \overline{M} \end{array}}{S, \Gamma \vdash_* T <: (\overline{T})\{\varphi \mid \overline{M}\}} \; (<:\text{-SIG-RIGHT})$$

Definition

$$\overline{N} \ll \overline{N'} \Leftrightarrow (\forall (N, N') \in \overline{N} \times \overline{N'}, dom(N) = dom(N') \Rightarrow N <: N')$$

Member Subtyping

$$\frac{}{S, \Gamma \vdash \texttt{type}_n A = T <: \texttt{type}_n A (= T)^?} \ (<:\text{-MEMBER-TYPE})$$

$$\frac{S, \Gamma \vdash T <: T'}{S, \Gamma \vdash \texttt{val}_n a : T(= t)^? <: \texttt{val}_m a : T'(= t')^?} \ (<:\text{-MEMBER-FIELD})$$

$$\frac{(<:\text{-MEMBER-CLASS})}{S, \Gamma \vdash \texttt{trait}_n A \ \texttt{extends} \ (T)\{\varphi \mid \overline{M}\} <: \texttt{trait}_n A \ \texttt{extends} \ (T)\{\varphi \mid \overline{M}\}}$$

$$\frac{S, \Gamma \vdash \overline{S'} <: \overline{S} \qquad S, \Gamma \vdash T <: T' \qquad (<:\text{-MEMBER-METHOD})}{S, \Gamma \vdash \texttt{def}_n a(\overline{x : S}) : T(= t)^? <: \texttt{def}_n a(\overline{x : S'}) : T'(= t')^?}$$

Well-Formedness

$$\frac{S, \Gamma \vdash_{path} p : T \qquad \psi(p) \not\subseteq S \qquad \psi(p) \cup S, \Gamma \vdash T \ \texttt{WF}}{S, \Gamma \vdash p.\textbf{type} \ \texttt{WF}} \ \text{(WF-SINGLETON)}$$

$$\frac{S, \Gamma \vdash p.\textbf{type} \ni \texttt{trait}_n A \ \texttt{extends} \ (T)\{\varphi \mid \overline{M}\}}{S, \Gamma \vdash p.A \ \texttt{WF}} \ \text{(WF-CLASS)}$$

$$\frac{S, \Gamma, \varphi : (\overline{T})\{\varphi \mid \overline{M}\} \vdash (\overline{T})\{\varphi \mid \overline{M}\} \ \texttt{WF}_{\varphi}}{S, \Gamma \vdash (\overline{T})\{\varphi \mid \overline{M}\} \ \texttt{WF}} \ \text{(WF-SIGNATURE)}$$

$$\frac{S, \Gamma \vdash p.\textbf{type} \ni \texttt{type}_n A (= T)^? \qquad (\{n\} \cup S, \Gamma \vdash T \ \texttt{WF})^? \qquad (n \notin S)^?}{S, \Gamma \vdash p.A \ \texttt{WF}} \ \text{(WF-TYPE)}$$

Member Well-Formedness

$$\frac{S, \Gamma \vdash T \;\; \mathtt{WF} \qquad (S, \Gamma \vdash t : T')^{?} \qquad (S, \Gamma \vdash T' <: T)^{?}}{S, \Gamma \vdash \mathtt{val}_n a : T (= t)^{?} \;\; \mathtt{WF}_x} \text{(WF-X-FIELD)}$$

$$\frac{\begin{array}{c} S, \Gamma \vdash \overline{S}, T \;\; \mathtt{WF} \qquad (\overline{x : S}, S, \Gamma \vdash t : T')^{?} \qquad (S, \Gamma \vdash T' <: T)^{?} \\ \overline{S} \text{ does not contain singleton types} \end{array}}{S, \Gamma \vdash \mathtt{def}_n a(\overline{x : S}) : T (= t)^{?}} \text{(WF-X-METHOD)}$$

$$\frac{(S, \Gamma \vdash T \;\; \mathtt{WF})^{?}}{S, \Gamma \vdash \mathtt{type}_n A (= T)^{?} \;\; \mathtt{WF}_x} \text{(WF-X-TYPE)}$$

Member Well-Formedness

$$\frac{\varphi : x.A, S, \Gamma \vdash (\overline{T})\{\varphi \mid \overline{M}\} \;\; \mathtt{WF}_{\varphi}}{S, \Gamma \vdash \mathtt{trait}_n A \;\mathtt{extends}\; (\overline{T})\{\varphi \mid \overline{M}\} \;\; \mathtt{WF}_{x}} \text{(WF-X-CLASS)}$$

$$\frac{\begin{array}{cc} S, \Gamma \vdash \overline{M} \;\; \mathtt{WF}_{\varphi} & \forall_i, S, \Gamma \vdash T_i \prec_{\varphi} \overline{N_i} \\ S, \Gamma \vdash \overline{T} \;\; \mathtt{WF} & \forall_{(i,j)}, S, \Gamma \vdash (\overline{N_{i+j}}, \overline{M}) \ll \overline{N_i} \end{array}}{S, \Gamma \vdash (\overline{T})\{\varphi \mid \overline{M}\} \;\; \mathtt{WF}_{\varphi}} \text{(WF-X-SIGNATURE)}$$

Properties

Lemma

If a term t can be assigned a type T by the **Path Typing** *judgment, then it is unique.*

Properties

Lemma

If a term t can be assigned a type T by the **Path Typing** *judgment, then it is unique.*

Lemma

The calculus defines a deterministic algorithm.

Properties

Lemma

If a term t can be assigned a type T by the **Path Typing** *judgment, then it is unique.*

Lemma

The calculus defines a deterministic algorithm.

Lemma

The **Path Typing**, **Expansion**, **Membership** *and* **Path Alias Expansion** *judgments terminate on all inputs.*

Properties

Lemma

If a term t can be assigned a type T by the **Path Typing** *judgment, then it is unique.*

Lemma

The calculus defines a deterministic algorithm.

Lemma

The **Path Typing**, **Expansion**, **Membership** *and* **Path Alias Expansion** *judgments terminate on all inputs.*

Corollary

The **Type Alias Expansion** *judgment terminates on all inputs.*

Lemma

The **Algorithmic Subtyping** *and* **Member Subtyping** *judgments terminate on all inputs.*

Lemma

The **Type Assignment**, **Well-Formedness** *and* **Member Well-Formedness** *judgments terminate on all inputs.*

Conclusions

- FS type checking is decidable, we know how to construct a program which is performing type checking for FS
- Scala is **probably** decidable, it's not (AFAIK) proved. There are some problems with the lower/upper bounds of types, details are explained in the paper

Questions?

Homework

Info

Deadline: 29th June

Theoretical variant

Formulate CBV semantics and extend FS with mutable state (in pdf format). I am not sure, how hard this task is. It may be very easy or not. I am accepting solutions for this task for 5.0 grade, even if these are not complete, however you have to show me what you achieved and explain me where you got stuck.

Practical variant

Try to make a static analysis program for Featherweight Scala (with basic features like finding dead code, unused variables and so on). Of course in any functional and reasonable language.