A Conceptual Framework for Safe Object Initialization

Clément Blaudeau, Inria & Université de Paris Cité, France

Fengyun Liu, Oracle Labs, Switzerland

Splash/Oopsla 2023, 27/11/23

Examples

```scala
class A {
  var y = 42 :: this.x
  var x = List()
}
```

```scala
class A {
  var x : List[Int] = this.m()

  def m() = 42::this.x
 }
```

```scala
class A {
  var b = new B(this)
  var x = List()
}
class B (a:A) {
  var y = 42 :: a.x
}
```

Initialization errors

- Early field access
- Early method call
- Incorrect escaping

Key issue

Objects *under initialization* do not fulfill their class specification yet

→ Breaks the key assumption of OOP!

Complex initializations

Cyclic data structures

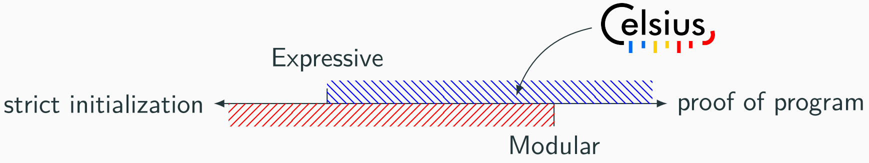
```
class A () {
  var b = new B(this)
  var c = this.b.c
}
class B (arg:A) {
  var a = arg
  var c = new C(this)
}
class C (arg:B) {
  var a = arg.a
  var b = arg
}
```

Early method call

```
class Server (a: Address) {
  var address = a
  var _ = this.broadcast("Init")
  ... // other fields

  def broadcast(m: String) = {
   ... // sends a message
  }
}
```

Design space



Sound

- No access to uninitialized field

Expressive

- Authorize controlled escaping

Modular

- Class-by-class analysis
- Limited footprint

Usable

- Understandable principles
- Inference

Plan

In this presentation

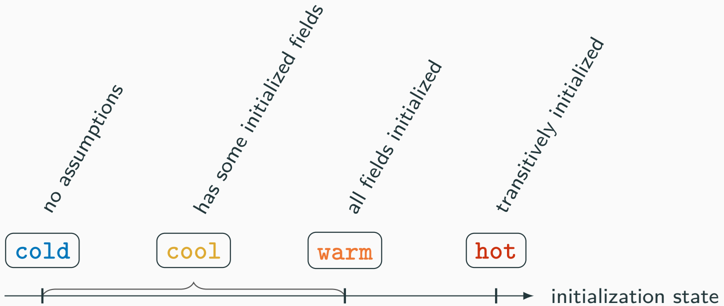
1. The Celsius model of initialization
2. The Core principles
    - High-level, language agnostic
    - Design choices a the minimal calculus with type annotations
3. Local reasoning (overview)

In the paper

- The minimal calculus
- The typing system (with temperature annotations)
- The (modular) soundness proof (based on the principles)
- The Scala implementation
- The Coq artifact

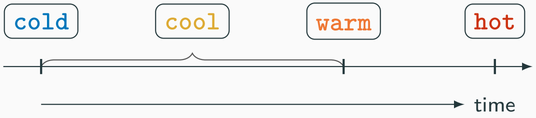
The Celsius Model

The Celsius model



The core principles

Principle 1/4: Monotonicity



Partial monotonicity $\preceq$

Fields cannot be un-initialized

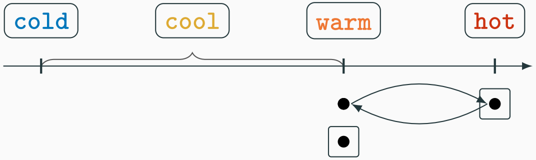
Perfect monotonicity $\preccurlyeq$

Initialization state *of every field* cannot decrease

Design choices (for the calculus)

- No de-initialization (language design)
- Update fields only with hot values (typing)

Principle 2/4: Authority



Local vision of the initialization state might differ between aliases

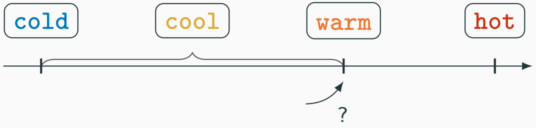
Authority

State updates are only authorized on a distinguished alias : `this`

Design choices

- Type updates (up to warm) only inside the constructor
- Distinguish 1$^{\text{fst}}$ assignment / update

Principle 3/4: Stackability



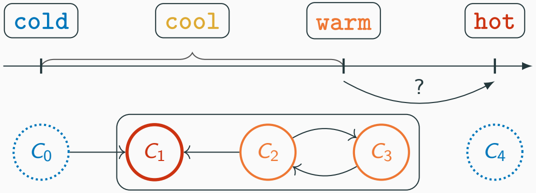
Stackability

All fields must be initialized at the end of their constructor
→ constructors form a *call stack*

Design choices

- Mandatory field initializers
- No control effects

Principle 4/4: Scopability



Nested/parallel initializations → Control the accessible part of the heap

Scopability

Access to *objects under initialization* must go through controlled channels

Design choices

- No global variables (see Liu 2023)
- Over-approximate reachable objects

Local reasoning

Theorem (Local reasoning)

Executing an expression in a hot environment results in a hot object

Proof.

In the resulting memory, accessible objects are either

- new and therefore warm (by stackability)
- old, so already accessible in the execution environment (by scopability), and therefore still hot (by monotonicity and authority)

$\square$

$\rightarrow$ gives rises to a typing system with *hot-bypasses*: you can safely ignore initialization issues when handling hot objects

Examples in Celsius syntax

```
class A () {
  var b: B@warm = new B(this)
  var c: C@warm = this.b.c
}
class B (arg: A@cold) {
  var a: A@cold = arg
  var c: C@warm = new C(this)
}
class C (arg: B@cool(a)) {
  var a: A@cold = arg.a
  var b: B@cool(a) = arg
}
```

```
class Main () {
  var a: A@hot = new A()
  // ignore initialization checks
}
```

Take away

A **conceptual framework** for safe initialization

→ *a simple interface to a subtle problem*

- the Celsius model (`cold`, `cool`, `warm`, `hot`)
- Four language agnostic principles
- Local reasoning

See the paper for precise definitions, typing system, soundness proof, implementation in Scala!

`github.com/clementblaudeau/celsius`