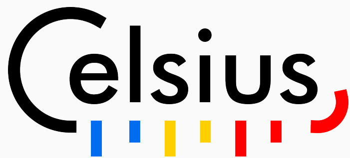
A Conceptual Framework for Safe Object Initialization

Clément Blaudeau, Inria & Université de Paris Cité, France
Fengyun Liu, Oracle Labs, Switzerland

Splash/Oopsla 2023, 27/11/23



Examples

```
1 class A {
2   var y = 42 :: this.x
3   var x = List()
4 }
```

```
1 class A {
2   var x : List[Int] = this.m()
3
4   def m() = 42::this.x
5  }
```

```
1 class A {
2   var b = new B(this)
3   var x = List()
4 }
5 class B (a:A) {
6   var y = 42 :: a.x
7 }
```

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

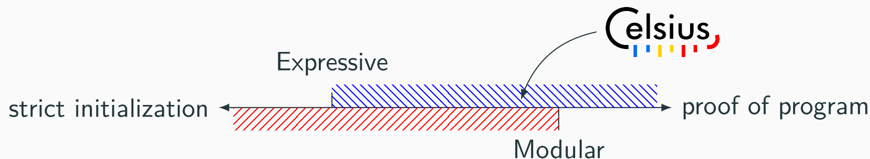
Cyclic data structures

```
1  class A () {
2    var b = new B(this)
3    var c = this.b.c
4  }
5  class B (arg:A) {
6    var a = arg
7    var c = new C(this)
8  }
9  class C (arg:B) {
10   var a = arg.a
11   var b = arg
12 }
```

Early method call

```
1  class Server (a: Address) {
2    var address = a
3    var _ = this.broadcast("Init")
4    ... // other fields
5
6    def broadcast(m: String) = {
7     ... // sends a message
8    }
9  }
```

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

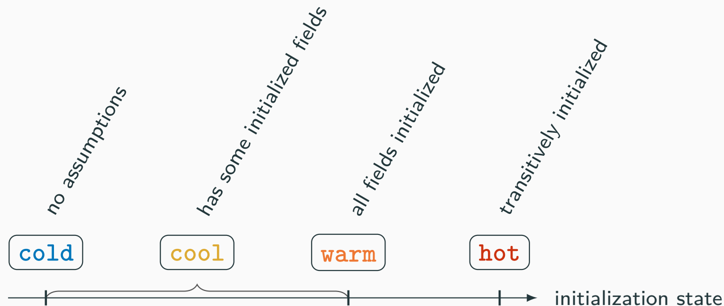
1. The Celsius model of initialization
2. The Core principles
    - High-level, language agnostic
    - Design choices a the minimal calculus with type annotations
3. Local reasoning (overview)

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

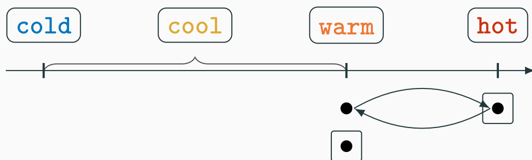
Perfect monotonicity $\preccurlyeq$

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^fst assignment / update

Principle 3/4: Stackability

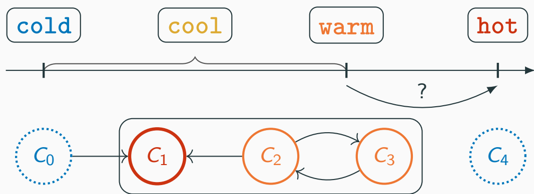
Stackability

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)

□

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

Examples in Celsius syntax

```
1  class A () {
2    var b: B@warm = new B(this)
3    var c: C@warm = this.b.c
4  }
5  class B (arg: A@cold) {
6    var a: A@cold = arg
7    var c: C@warm = new C(this)
8  }
9  class C (arg: B@cool(a)) {
10   var a: A@cold = arg.a
11   var b: B@cool(a) = arg
12 }
```

```
1  class Main () {
2    var a: A@hot = new A()
3    // ignore initialization checks
4  }
```

Take away

A **conceptual framework** for safe initialization

$\rightarrow$ *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`

The Celsius calculus

Grammar

Expressions

$$
\begin{aligned}
e ::= &\ x && \text{(Local variable)}\\
\mid &\ \text{this} && \text{(Self-reference)}\\
\mid &\ e.f && \text{(Field access)}\\
\mid &\ e.m\,(\overline{e}) && \text{(Method call)}\\
\mid &\ \text{new}\ C\,(\overline{e}) && \text{(Instance creation)}\\
\mid &\ e.f \leftarrow e; e && \text{(Assignment)}
\end{aligned}
$$

Mode

$$\mu ::= \text{cold} \mid \text{cool}\,\overline{f} \mid \text{warm} \mid \text{hot}$$

Type

$$T ::= C^{\mu}$$

Class

$$\mathbb{C} ::= \text{class}\ C(\overline{x : T})\{\ \text{fields} = \overline{\mathbb{F}}, \text{methods} = \overline{\mathbb{M}}\}$$

Field

$$\mathbb{F} ::= \text{var}\,f : T = e$$

Method

$$\mathbb{M} ::= @\mu\,\text{def}\,m(\overline{x : T}) : T = \{e\}$$

Program

$$\mathbb{P} ::= \{\text{ct} = \overline{\mathbb{C}}, \text{entry} = \mathbb{C}\}$$

Examples in Celsius syntax

```
1  class A () {
2    var b: B@warm = new B(this)
3    var c: C@warm = this.b.c
4  }
5  class B (arg: A@cold) {
6    var a: A@cold = arg
7    var c: C@warm = new C(this)
8  }
9  class C (arg: B@cool(a)) {
10   var a: A@cold = arg.a
11   var b: B@cool(a) = arg
12 }
```

```
1  class Server (a: Address@hot) {
2    var address : Address@hot = a
3    var _ = this.broadcast("Init");
4    ... // other fields
5
6    @cool(address)
7      def broadcast(m: String) = {
8     ... // sends a message
9    }
10 }
```

Semantics - Big steps

Store

$$\sigma : l \mapsto (C, \omega)$$

Expressions

$$[\![e]\!](\sigma, \rho, \psi) \longrightarrow (v, \sigma')$$

- e expression
- σ store
- ρ local environment (fields)
- ψ current object (`this`)

Initialization

$$\text{init}_C(\psi, i, \rho, \sigma) \longrightarrow \sigma'$$

- ψ current object (`this`)
- i number of initialized fields
- ρ local environment (args)
- σ store

Semantic rules

E-New

$$\frac{[\![\overline{e_a}]\!](\sigma,\rho,\psi) \longrightarrow (\overline{l_a},\sigma_1) \qquad l_{\text{fresh}} \notin dom\,(\sigma_1) \qquad \text{args}(C) = \overline{(x : T)} \qquad \text{init}_C(l_{\text{fresh}}, 0, \overline{(x \mapsto l_a)}, \sigma_1 \cup \{l_{\text{fresh}} \mapsto (C, \varnothing)\}) \longrightarrow \sigma_2}{[\![\text{new}\ C\,(\overline{e_a})]\!](\sigma,\rho,\psi) \longrightarrow (l_{\text{fresh}},\sigma_2)}$$

E-Init-Cons

$$\frac{\text{fields}(C)(i) = \text{var}\ f_i : T = e \qquad [\![e]\!](\sigma,\rho,\psi) \longrightarrow (l_1,\sigma_1) \qquad \sigma_1(\psi) = (C,\omega) \qquad \sigma_2 = [\psi \mapsto (C, \omega \cup (f_i \mapsto l_1))]\,\sigma_1 \qquad \text{init}_C(\psi, i+1, \rho, \sigma_2) \longrightarrow \sigma_3}{\text{init}_C(\psi, i, \rho, \sigma) \longrightarrow \sigma_3}$$

E-Init-End

$$\text{init}_C(\psi, \text{length}(\text{fields}(C)), \rho, \sigma) \longrightarrow \sigma$$

Typing and soundness

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$\llbracket e \rrbracket(\sigma, \rho, \psi, n) = r \quad \textit{with} \quad r \coloneqq \text{success}(v, \sigma') \mid \text{error} \mid \text{timeout}$$
$$\text{init}_C(\psi, i, \rho, \sigma, n) = r \quad \textit{with} \quad r \coloneqq \text{success}(\sigma') \quad \mid \text{error} \mid \text{timeout}$$

$$\llbracket e \rrbracket(\sigma, \rho, \psi) \longrightarrow (v, \sigma') \iff \exists n.\, \llbracket e \rrbracket(\sigma, \rho, \psi, n) = \text{success}(v, \sigma') \qquad (1)$$
$$\text{init}_C(\psi, i, \rho, \sigma) \longrightarrow \sigma' \iff \exists n.\, \text{init}_C(\psi, i, \rho, \sigma, n) = \text{success}(\sigma') \qquad (2)$$

Soundness invariant (structure)

$$\left.\begin{array}{l} \llbracket e \rrbracket(\sigma, \rho, \psi, n) = r \\ r \neq \text{timeout} \\ \vdash e : T \end{array}\right\} \implies \exists \sigma', v. \begin{cases} r = \text{success}(v, \sigma') \\ \models v : T \\ \text{Monotonicity} \\ \text{Authority} \\ \text{Stackability} \\ \text{Scopability} \end{cases}$$

Reachability and Scopability

Reachability - $\sigma \vDash l \rightsquigarrow l'$

Transitive closure of field access

Scopability - $(\sigma, L) \lessdot (\sigma', L')$

Every location reachable from L' in σ' is either new or already reachable from L in σ:

$$\forall l \in dom(\sigma), \sigma' \vDash L' \rightsquigarrow l \implies \sigma \vDash L \rightsquigarrow l$$

Scopability theorem

The heap reachable from the result location v is scoped in the result store σ' by the *execution environment* ($codom(\rho) \cup \{\psi\}$) in the starting store σ:

$$[\![e]\!](\sigma, \rho, \psi) \longrightarrow (v, \sigma') \implies (\sigma, \rho \cup \{\psi\}) \lessdot (\sigma', \{v\})$$

Typing (1/2) - typing rules

Mode lattice - $\mu \sqsubseteq \mu'$

$$\text{cold} \sqsubseteq \text{cool}\ \overline{f} \sqsubseteq \text{warm} \sqsubseteq \text{hot}$$

Typing - $(\Gamma, T_{\texttt{this}}) \vdash e : T$

- Local environment
- Ambient subtyping
- Stackability (T-E-New)
- Monotonicity (T-E-Block)

T-E-Var
$$\frac{\Gamma(x) = T}{(\Gamma, T_{\texttt{this}}) \vdash x : T}$$

T-E-This
$$(\Gamma, T_{\texttt{this}}) \vdash \text{this} : T_{\texttt{this}}$$

T-E-Sub
$$\frac{(\Gamma, T_{\texttt{this}}) \vdash e : C^{\mu} \qquad \mu \sqsubseteq \mu'}{(\Gamma, T_{\texttt{this}}) \vdash e : C^{\mu'}}$$

T-E-Fld
$$\frac{(\Gamma, T_{\texttt{this}}) \vdash e : D^{\text{cool}\ \overline{f}} \qquad f \in \overline{f} \qquad \text{fieldType}(D, f) = T}{(\Gamma, T_{\texttt{this}}) \vdash e.f : T}$$

T-E-Call
$$\frac{(\Gamma, T_{\texttt{this}}) \vdash e : C^{\mu} \qquad \text{lookup}(C, m) = @\mu\ \text{def}\ m : \overline{(x : T)} \to T \qquad (\Gamma, T_{\texttt{this}}) \vdash \overline{e_a} : \overline{T}}{(\Gamma, T_{\texttt{this}}) \vdash e.m\left(\overline{e_a}\right) : T}$$

T-E-New
$$\frac{\text{ct}(C) = \text{class}\ C(\overline{x : T})\ \{\ldots\} \qquad (\Gamma, T_{\texttt{this}}) \vdash \overline{e} : \overline{T}}{(\Gamma, T_{\texttt{this}}) \vdash \text{new}\ C\left(\overline{e}\right) : C^{\text{warm}}}$$

T-E-Block
$$\frac{(\Gamma, T_{\texttt{this}}) \vdash e_1.f : C^{\mu} \qquad (\Gamma, T_{\texttt{this}}) \vdash e_2 : C^{\text{hot}} \qquad (\Gamma, T_{\texttt{this}}) \vdash e_3 : T}{(\Gamma, T_{\texttt{this}}) \vdash e_1.f \leftarrow e_2; e_3 : T}$$

Typing (1/2) - typing rules - hot rules

Mode lattice - $\mu \sqsubseteq \mu'$

$$\text{cold} \sqsubseteq \text{cool}\ \overline{f} \sqsubseteq \text{warm} \sqsubseteq \text{hot}$$

Typing - $(\Gamma, T_{\texttt{this}}) \vdash e : T$

- Local environment
- Ambient subtyping
- Stackability (T-E-New)
- Monotonicity (T-E-Block)
- Hot shortcuts (also the only way of getting hot objects)

T-E-Fld-Hot

$$\frac{(\Gamma, T_{\texttt{this}}) \vdash e : D^{\,\text{hot}} \qquad \text{fieldType}\,(D, f) = C^{\,\mu}}{(\Gamma, T_{\texttt{this}}) \vdash e.f : C^{\,\text{hot}}}$$

T-E-Call-Hot

$$\frac{\begin{array}{c}(\Gamma, T_{\texttt{this}}) \vdash e : C_0^{\text{hot}} \\ \text{lookup}(C_0, m) = @\mu\ \text{def}\ m : \overline{(x : D^{\mu})} \to C^{\mu} \\ (\Gamma, T_{\texttt{this}}) \vdash \overline{e_a} : \overline{D^{\,\text{hot}}}\end{array}}{(\Gamma, T_{\texttt{this}}) \vdash e.m\,(\overline{e_a}) : C^{\,\text{hot}}}$$

T-E-New-Hot

$$\frac{\begin{array}{c}\text{ct}(C) = \text{class}\ C(\overline{x : D^{\,\mu}})\ \{\ldots\} \\ (\Gamma, T_{\texttt{this}}) \vdash \overline{e} : \overline{D^{\,\text{hot}}}\end{array}}{(\Gamma, T_{\texttt{this}}) \vdash \text{new}\ C\,(\overline{e}) : C^{\,\text{hot}}}$$

Typing (2/2) - Store typing

Store typing - $\Sigma : l \mapsto T$

Prevent cyclic dependencies

Object typing - $\Sigma \vDash (C, \omega) : (C, \mu)$

Lower bound of actual initialization state

Abstraction of the store - $\Sigma \vDash \sigma$

$$\frac{dom(\sigma) = dom(\Sigma) \qquad \forall l \in dom(\sigma) \,.\, \Sigma \vDash \sigma(l) : \Sigma(l)}{\Sigma \vDash \sigma}$$

Environment typing - $\Sigma \vDash \rho : \Gamma$

$$\Sigma \vDash \varnothing : \varnothing \qquad\qquad \frac{\Sigma \vDash \rho : \Gamma \qquad \Sigma \vDash l : T}{\Sigma \vDash (x \mapsto l) \cup \rho : (x \mapsto T) \cup \Gamma}$$

Soundness (2/3)

$$\left.\begin{array}{l} [\![e]\!](\sigma, \rho, \psi, n) = r \\ r \neq \text{timeout} \\ \Sigma \vDash \sigma \\ \Sigma \vDash \rho : \Gamma \\ \Sigma \vDash \psi : T_{\texttt{this}} \\ \vdash e : T \end{array}\right\} \implies \exists \sigma', v, \Sigma'. \begin{cases} r = \text{success}(v, \sigma') \\ \vDash v : T \\ \Sigma' \vDash \sigma' \\ \text{Monotonicity} \\ \text{Authority} \\ \text{Stackability} \\ \text{~~Scopability~~} \end{cases}$$

Core principles for typing

Monotonicity

$$\Sigma \preccurlyeq \Sigma' \coloneqq \forall l \in dom(\Sigma) \,.\, \Sigma'(l) <: \Sigma(l)$$

Stackability

$$\Sigma \ll \Sigma' \coloneqq \forall l \in dom(\Sigma') \setminus dom(\Sigma).\, \Sigma' \vDash l : \text{warm}$$

Authority

$$\Sigma \triangleright \Sigma' \coloneqq \forall l.\, \Sigma(l) = C^{\,\text{cool}\,\overline{f}} \implies \Sigma'(l) = \Sigma(l)$$

Soundness (3/3)

$$\left.\begin{aligned} &[\![e]\!](\sigma, \rho, \psi, n) = r \\ &r \neq \text{timeout} \\ &\Sigma \vDash \sigma \\ &\Sigma \vDash \rho : \Gamma \\ &\Sigma \vDash \psi : T_{\texttt{this}} \\ &(\Gamma, T_{\texttt{this}}) \vdash e : T \end{aligned}\right\} \implies \exists \sigma', v, \Sigma'. \begin{cases} r = \text{success}(v, \sigma') \\ \Sigma' \vDash \sigma' \\ \Sigma' \vDash v : T \\ \Sigma \preccurlyeq \Sigma' \\ \Sigma \triangleright \Sigma' \\ \Sigma \ll \Sigma' \end{cases} \tag{3}$$

$$\left.\begin{aligned} &\text{init}_C(\psi, i, \rho, \sigma, n) = r \\ &r \neq \text{timeout} \\ &\Sigma \vDash \sigma \\ &\Sigma \vDash \rho : \Gamma \\ &\Sigma(\psi) = \text{cool}\,\{f_0, \ldots, f_{i-1}\} \\ &\Gamma <: \Gamma_a \end{aligned}\right\} \implies \exists \sigma', \Sigma'. \begin{cases} r = \text{success}(\sigma') \\ \Sigma' \vDash \sigma' \\ \Sigma \preccurlyeq \Sigma' \\ \Sigma \ll \Sigma' \\ \left[\psi \mapsto C^{\,\text{cool}(\text{fields}(C))}\right] \Sigma \triangleright \Sigma' \end{cases} \tag{4}$$

Program Soundness

Theorem (Program Soundness)

A well typed program cannot run into an error

$$\vdash \mathbb{P} \implies \forall n, [\![\mathbb{P}]\!](n) \neq \mathit{error} \tag{5}$$

Conclusion

Take-away

- Four principles for the safe initialization of objects
  - Monotonicity (invariants progress)
  - Authority (distinguished alias)
  - Stackability (all fields initialized at the end of the constructor)
  - Scopability (control the access to un-initialized objects)
- A minimal calculus to illustrate the principles
- A modular proof, mechanized in Coq