A Conceptual Framework for Safe Object Initialization

Clément Blaudeau, Inria & Université de Paris Cité, France

Fengyun Liu, Oracle Labs, Switzerland

Cambium Seminar 2023, 16/10/23

Celsius

Examples

Examples

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

Initialization errors

- Early field access

Examples

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

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

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

Initialization errors

- Early field access
- Early method call

Examples

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

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

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

```
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

Examples

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

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

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

```
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

Examples

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

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

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

```
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

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
}
```

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

Design space

Sound

- No access to uninitialized field

Design space

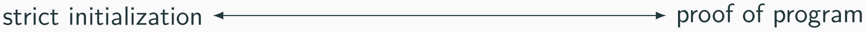
strict initialization

proof of program

Sound

- No access to uninitialized field

Design space



strict initialization

proof of program

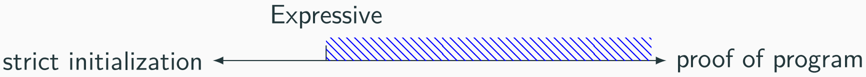
Sound

- No access to uninitialized field

Expressive

- Authorize controlled escaping

Design space



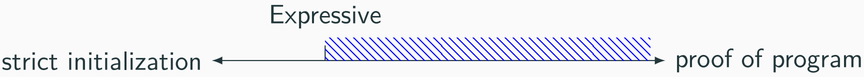
Sound

- No access to uninitialized field

Expressive

- Authorize controlled escaping

Design space



Sound

- No access to uninitialized field

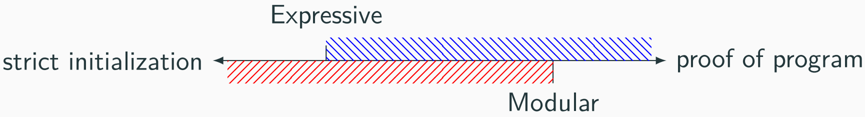
Modular

- Class by class analysis
- Limited footprint

Expressive

- Authorize controlled escaping

Design space



Sound

- No access to uninitialized field

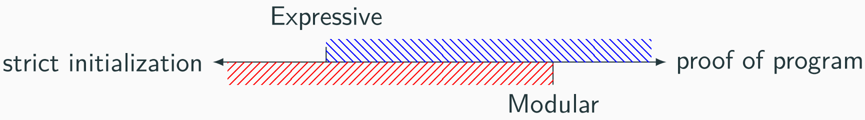
Expressive

- Authorize controlled escaping

Modular

- Class by class analysis
- Limited footprint

Design space



Sound

- No access to uninitialized field

Expressive

- Authorize controlled escaping

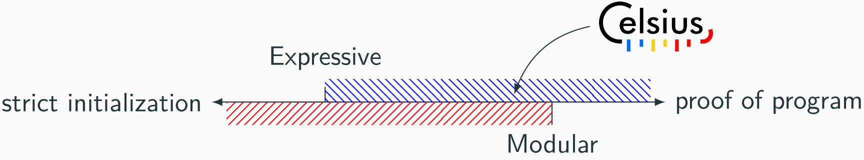
Modular

- Class by class analysis
- Limited footprint

Usable

- Understandable annotations
- Inference

Design space



Sound

- No access to uninitialized field

Expressive

- Authorize controlled escaping

Modular

- Class by class analysis
- Limited footprint

Usable

- Understandable annotations
- Inference

Plan

Plan

In this presentation

1. The Celsius model of initialization

Plan

In this presentation

1. The Celsius model of initialization
2. The Core principles

Plan

In this presentation

1. The Celsius model of initialization
2. The Core principles
    - High-level, language agnostic

Plan

In this presentation

1. The Celsius model of initialization
2. The Core principles
   - High-level, language agnostic
   - Design choices for the minimal calculus

Plan

In this presentation

1. The Celsius model of initialization
2. The Core principles
   - High-level, language agnostic
   - Design choices for the minimal calculus
3. Local reasoning and soundness (overview)

Plan

In this presentation

1. The Celsius model of initialization
2. The Core principles
   - High-level, language agnostic
   - Design choices for the minimal calculus
3. Local reasoning and soundness (overview)

Plan

In this presentation

1. The Celsius model of initialization
2. The Core principles
    - High-level, language agnostic
    - Design choices for the minimal calculus
3. Local reasoning and soundness (overview)

In the paper

- The minimal calculus

Plan

In this presentation

1. The Celsius model of initialization
2. The Core principles
   - High-level, language agnostic
   - Design choices for the minimal calculus
3. Local reasoning and soundness (overview)

In the paper

- The minimal calculus
- The semantic interpretation of the principles

Plan

In this presentation

1. The Celsius model of initialization
2. The Core principles
   - High-level, language agnostic
   - Design choices for the minimal calculus
3. Local reasoning and soundness (overview)

In the paper

- The minimal calculus
- The semantic interpretation of the principles
- The typing system inspired by the principles

Plan

In this presentation

1. The Celsius model of initialization
2. The Core principles
    - High-level, language agnostic
    - Design choices for the minimal calculus
3. Local reasoning and soundness (overview)

In the paper

- The minimal calculus
- The semantic interpretation of the principles
- The typing system inspired by the principles
- The (modular) soundness proof

Plan

In this presentation

1. The Celsius model of initialization
2. The Core principles
   - High-level, language agnostic
   - Design choices for the minimal calculus
3. Local reasoning and soundness (overview)

In the paper

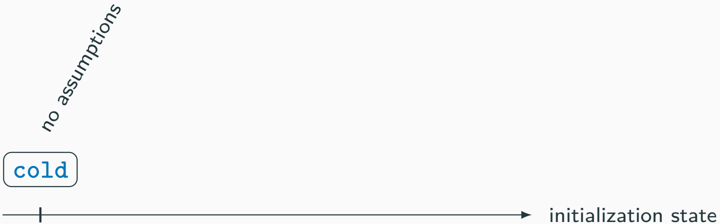
- The minimal calculus
- The semantic interpretation of the principles
- The typing system inspired by the principles
- The (modular) soundness proof
- The Coq artifact

The Celsius Model

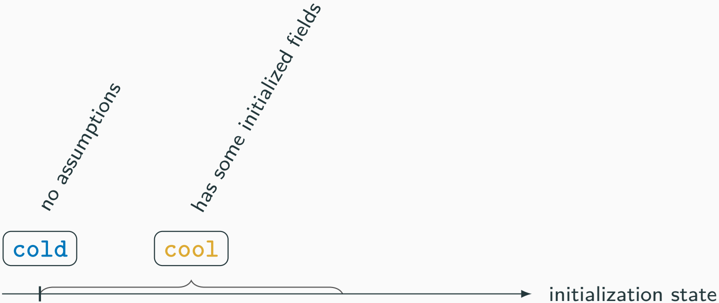
The Celsius model

initialization state

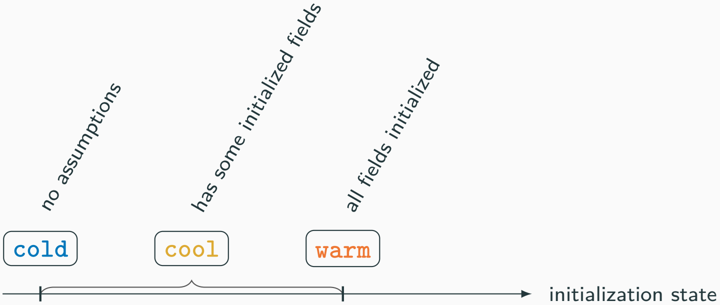
The Celsius model



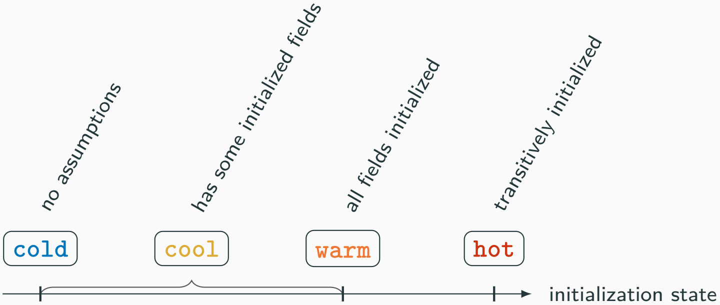
The Celsius model



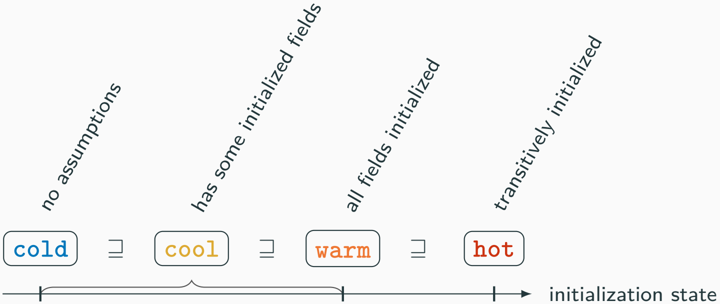
The Celsius model



The Celsius model

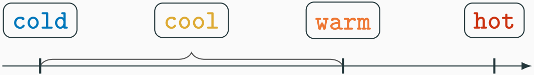


The Celsius model

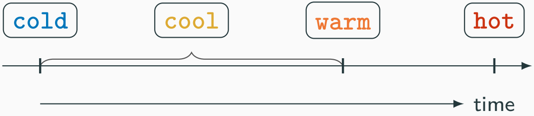


The core principles

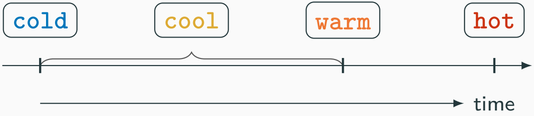
Principle 1/4: Monotonicity



Principle 1/4: Monotonicity



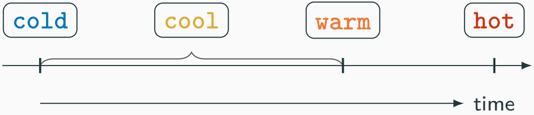
Principle 1/4: Monotonicity



Partial monotonicity $\preceq$

Fields cannot be un-initialized

Principle 1/4: Monotonicity



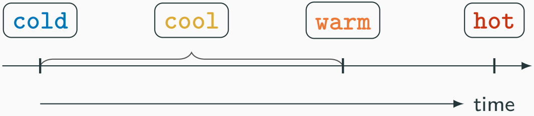
Partial monotonicity $\preceq$

Fields cannot be un-initialized

Perfect monotonicity $\preccurlyeq$

Initialization state *of every field* cannot decrease

Principle 1/4: Monotonicity



Partial monotonicity $\preceq$

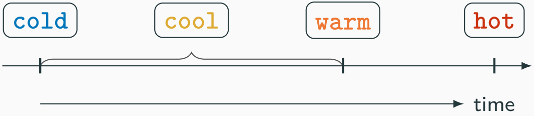
Fields cannot be un-initialized

Perfect monotonicity $\preccurlyeq$

Initialization state *of every field* cannot decrease

Design choices (for the calculus)

Principle 1/4: Monotonicity



Partial monotonicity $\preceq$

Fields cannot be un-initialized

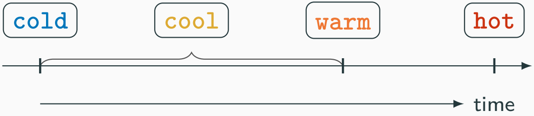
Perfect monotonicity $\preccurlyeq$

Initialization state *of every field* cannot decrease

Design choices (for the calculus)

- No de-initialization

Principle 1/4: Monotonicity



Partial monotonicity $\preceq$

Fields cannot be un-initialized

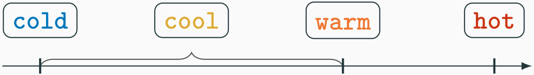
Perfect monotonicity $\preccurlyeq$

Initialization state *of every field* cannot decrease

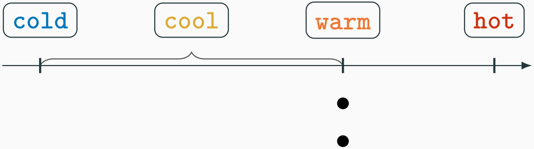
Design choices (for the calculus)

- No de-initialization
- Update fields only with hot values

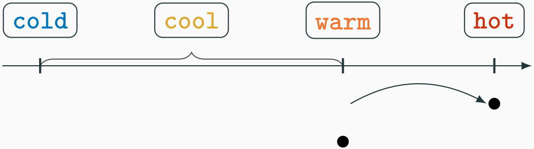
Principle 2/4: Authority



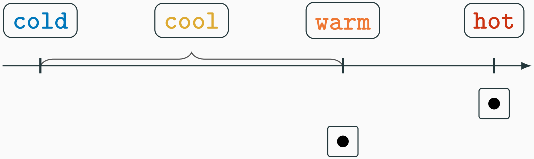
Principle 2/4: Authority



Principle 2/4: Authority

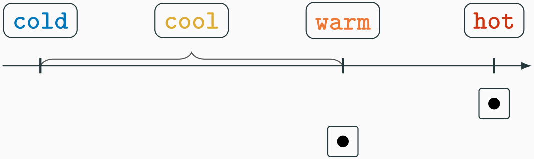


Principle 2/4: Authority



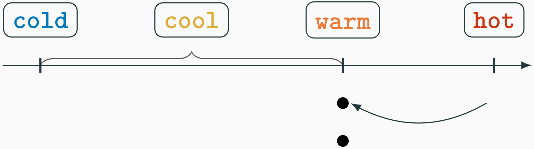
Local vision of the initialization state might differ between aliases

Principle 2/4: Authority



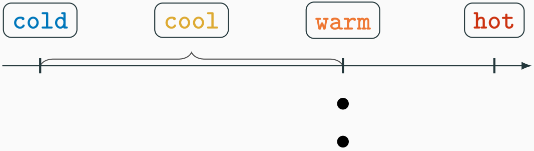
Local vision of the initialization state might differ between aliases

Principle 2/4: Authority



Local vision of the initialization state might differ between aliases

Principle 2/4: Authority

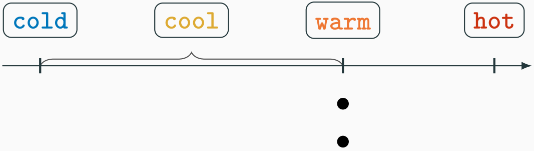


Local vision of the initialization state might differ between aliases

Authority

State updates are only authorized on a distinguished alias

Principle 2/4: Authority

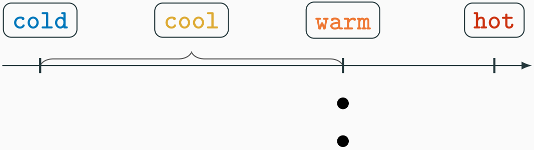


Local vision of the initialization state might differ between aliases

Authority

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

Principle 2/4: Authority



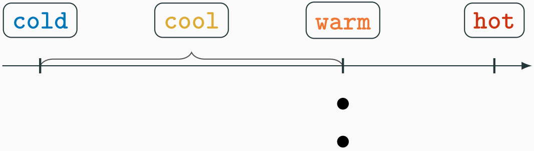
Local vision of the initialization state might differ between aliases

Authority

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

Design choices

Principle 2/4: Authority



Local vision of the initialization state might differ between aliases

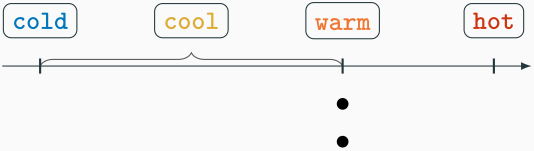
Authority

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

Design choices

- Distinguish 1$^{\text{fst}}$ assignment / update

Principle 2/4: Authority



Local vision of the initialization state might differ between aliases

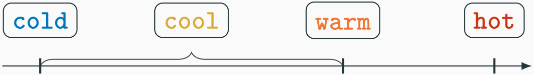
Authority

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

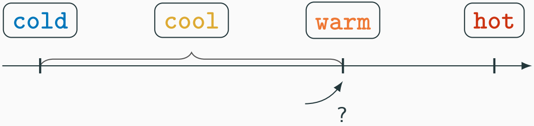
Design choices

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

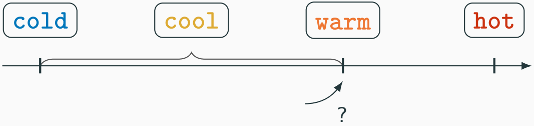
Principle 3/4: Stackability



Principle 3/4: Stackability

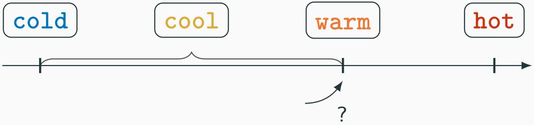


Principle 3/4: Stackability



Stackability

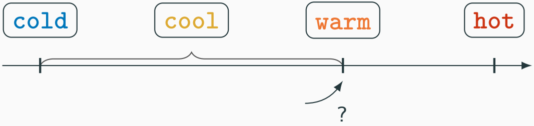
Principle 3/4: Stackability



Stackability

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

Principle 3/4: Stackability



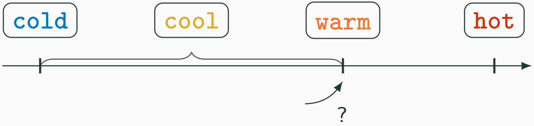
Stackability

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

Design choices

- Mandatory field initializers

Principle 3/4: Stackability



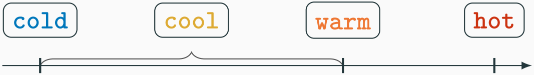
Stackability

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



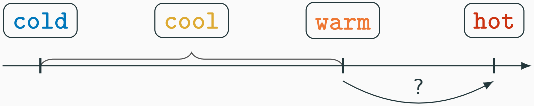
cold

cool

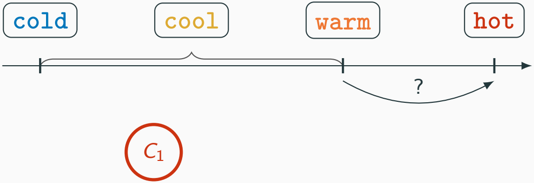
warm

hot

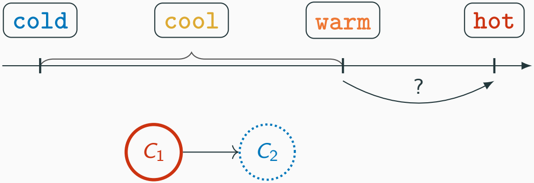
Principle 4/4: Scopability



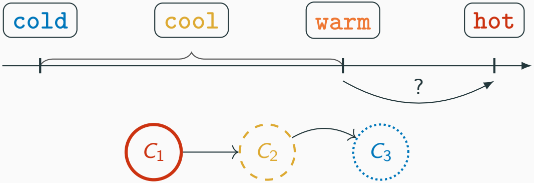
Principle 4/4: Scopability



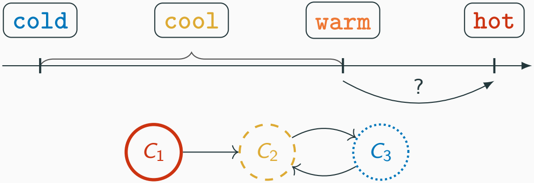
Principle 4/4: Scopability



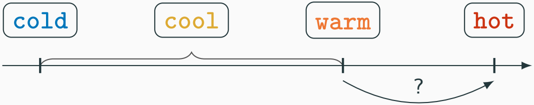
Principle 4/4: Scopability

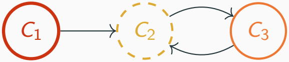


Principle 4/4: Scopability

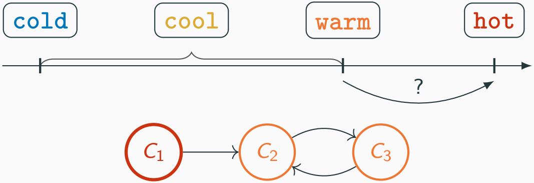


Principle 4/4: Scopability

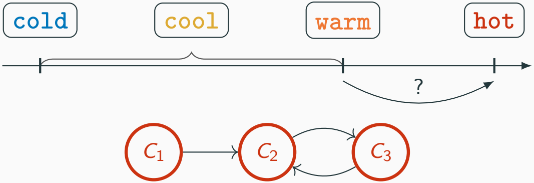




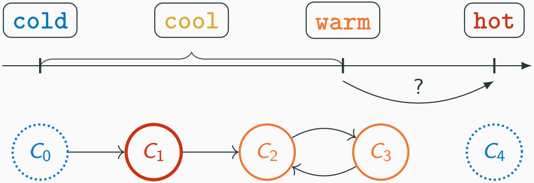
Principle 4/4: Scopability



Principle 4/4: Scopability

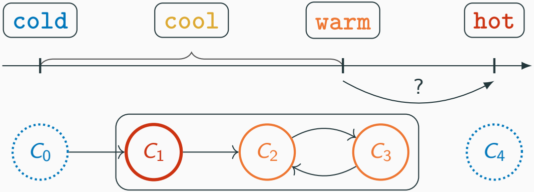


Principle 4/4: Scopability



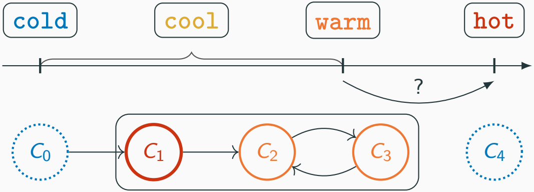
Nested/parallel initializations

Principle 4/4: Scopability



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

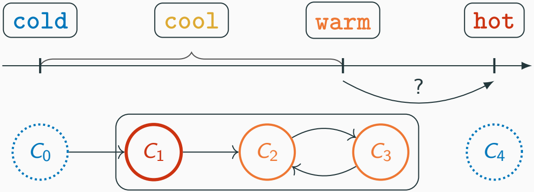
Principle 4/4: Scopability



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

Scopability

Principle 4/4: Scopability

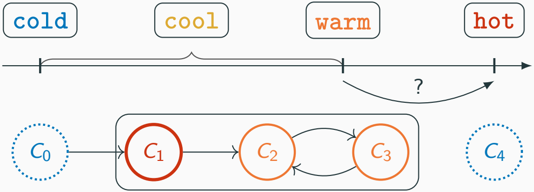


Nested/parallel initializations → Control the accessible part of the heap

Scopability

Access to *objects under initialization* must go through controlled channels, i.e. be controlled by static scoping

Principle 4/4: Scopability



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

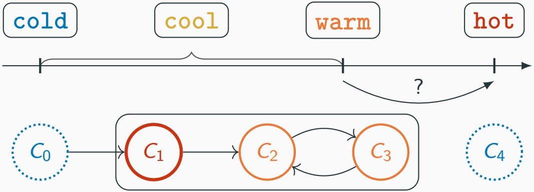
Scopability

Access to *objects under initialization* must go through controlled channels, i.e. be controlled by static scoping

Design choices

- No global variables (see Liu 2023)

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, i.e. be controlled by static scoping

Design choices

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

Local reasoning

Local reasoning

Theorem (Local reasoning)

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

Local reasoning

Theorem (Local reasoning)

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

Proof.

Local reasoning

Theorem (Local reasoning)

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

Proof.

In the resulting memory, accessible objects are either

Local reasoning

Theorem (Local reasoning)

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

Proof.

In the resulting memory, accessible objects are either

- new and therefore warm (by stackability)

Local reasoning

Theorem (Local reasoning)

Executing an expression in an hot environment results in an 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),

Local reasoning

Theorem (Local reasoning)

Executing an expression in an hot environment results in an 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),

Local reasoning

Theorem (Local reasoning)

Executing an expression in an hot environment results in an 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)

□

Local reasoning

Theorem (Local reasoning)

Executing an expression in an hot environment results in an 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)

□

→ gives rises to a typing system with *hot-bypasses*:

Local reasoning

Theorem (Local reasoning)

Executing an expression in an hot environment results in an 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)

□

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

Take away

Take away

A **conceptual framework** for safe initialization based on four principles

- the Celsius model (`cold`, `cool`, `warm`, `hot`)

Take away

A **conceptual framework** for safe initialization based on four principles

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

Take away

A **conceptual framework** for safe initialization based on four principles

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

Take away

A **conceptual framework** for safe initialization based on four principles

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

Take away

A **conceptual framework** for safe initialization based on four principles

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

See the paper for precise definitions, typing system and soundness proof!

The Celsius calculus

Grammar

Grammar

Expressions

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

Mode

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

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

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

Field

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

Method

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

Program

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

Examples in Celsius syntax

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

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)
}
```

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
}
```

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 Server (a: Address@hot) {
  var address : Address@hot = a
  var _ = this.broadcast("Init");
  ... // other fields

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

Semantics - Big steps

Semantics - Big steps

Store

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

Semantics - Big steps

Store

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

Expressions

Semantics - Big steps

Store

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

Expressions

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

Semantics - Big steps

Store

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

Expressions

e expression

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

Semantics - Big steps

Store

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

Expressions

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

e expression

σ store

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)

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`)

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`**)

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

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

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

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`)

Semantics - Big steps

Store

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

Expressions

$$\llbracket e \rrbracket(\sigma, \rho, \psi) \longrightarrow (v, \sigma')$$

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

Initialization

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

ψ	current object (`this`)
i	number of initialized fields

Semantics - Big steps

Store

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

Expressions

$$\llbracket e \rrbracket(\sigma, \rho, \psi) \longrightarrow (v, \sigma')$$

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)

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

Semantic rules

E-New

$$\frac{}{[\![\mathsf{new}\ C\ (\overline{e_a})]\!](\sigma, \rho, \psi) \longrightarrow (\quad , \quad)}$$

Semantic rules

$$\frac{\llbracket \overline{e_a} \rrbracket(\sigma, \rho, \psi) \longrightarrow (\overline{l_a}, \sigma_1)}{\llbracket \mathsf{new}\ C\ (\overline{e_a}) \rrbracket(\sigma, \rho, \psi) \longrightarrow (\quad , \quad)}\ \textsc{E-New}$$

Semantic rules

E-NEW

$$\frac{[\![\overline{e_a}]\!](\sigma, \rho, \psi) \longrightarrow (\overline{l_a}, \sigma_1) \qquad l_{\mathsf{fresh}} \notin dom\,(\sigma_1)}{[\![\mathsf{new}\; C\,(\overline{e_a})]\!](\sigma, \rho, \psi) \longrightarrow (\qquad , \quad)}$$

Semantic rules

$$\dfrac{\textsc{E-New}\quad [\![\overline{e_a}]\!](\sigma,\rho,\psi) \longrightarrow (\overline{l_a},\sigma_1) \qquad l_{\mathsf{fresh}} \notin dom\,(\sigma_1) \qquad \mathsf{args}(C) = \overline{(x : T)}}{[\![\mathsf{new}\ C\,(\overline{e_a})]\!](\sigma,\rho,\psi) \longrightarrow (\qquad,\quad)}$$

Semantic rules

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

Semantic rules

E-New

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

Semantic rules

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

Semantic rules

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

$$\textsc{E-Init-Cons}\quad \frac{}{\mathsf{init}_C(\psi, i, \rho, \sigma) \longrightarrow}$$

Semantic rules

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

$$\textsc{E-Init-Cons}\quad \frac{\mathsf{fields}(C)(i) = \mathsf{var}\, f_i : T = e}{\mathsf{init}_C(\psi, i, \rho, \sigma) \longrightarrow}$$

Semantic rules

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

$$\textsc{E-Init-Cons}\;\frac{\mathsf{fields}(C)(i) = \mathsf{var}\; f_i : T = e \qquad [\![e]\!](\sigma,\rho,\psi) \longrightarrow (l_1,\sigma_1)}{\mathsf{init}_C(\psi, i, \rho, \sigma) \longrightarrow}$$

Semantic rules

E-New

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

E-Init-Cons

$$\dfrac{\begin{array}{l}\text{fields}(C)(i) = \text{var}\ f_i : T = e \qquad [\![e]\!](\sigma,\rho,\psi) \longrightarrow (l_1,\sigma_1) \\ \sigma_1(\psi) = (C,\omega) \\ \\ \end{array}}{\text{init}_C(\psi, i, \rho, \sigma) \longrightarrow}$$

Semantic rules

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

$$\textsc{E-Init-Cons}\quad \frac{\mathsf{fields}(C)(i) = \mathsf{var}\; f_i : T = e \qquad [\![e]\!](\sigma, \rho, \psi) \longrightarrow (l_1, \sigma_1) \\ \sigma_1(\psi) = (C, \omega) \qquad \sigma_2 = [\psi \mapsto (C, \omega \cup (f_i \mapsto l_1))]\,\sigma_1}{\mathsf{init}_C(\psi, i, \rho, \sigma) \longrightarrow}$$

Semantic rules

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

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

Semantic rules

E-New

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

E-Init-Cons

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

Semantic rules

E-New

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

E-Init-Cons

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

E-Init-End

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

Typing and soundness

Soundness (1/3)

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$
\begin{aligned}
[\![e]\!](\sigma, \rho, \psi, n) = r \quad &\textit{with} \quad r \coloneqq \mathsf{success}(v, \sigma') \mid \mathsf{error} \mid \mathsf{timeout} \\
\mathsf{init}_C(\psi, i, \rho, \sigma, n) = r \quad &\textit{with} \quad r \coloneqq \mathsf{success}(\sigma') \quad\ \mid \mathsf{error} \mid \mathsf{timeout}
\end{aligned}
$$

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$\begin{aligned} [\![e]\!](\sigma,\rho,\psi,n) = r \quad &\textit{with} \quad r \coloneqq \mathsf{success}(v,\sigma') \mid \mathsf{error} \mid \mathsf{timeout} \\ \mathsf{init}_C(\psi,i,\rho,\sigma,n) = r \quad &\textit{with} \quad r \coloneqq \mathsf{success}(\sigma') \quad\ \mid \mathsf{error} \mid \mathsf{timeout} \end{aligned}$$

$$[\![e]\!](\sigma,\rho,\psi) \longrightarrow (v,\sigma') \iff \exists n.\, [\![e]\!](\sigma,\rho,\psi,n) = \mathsf{success}(v,\sigma') \tag{1}$$

$$\mathsf{init}_C(\psi,i,\rho,\sigma) \longrightarrow \sigma' \iff \exists n.\, \mathsf{init}_C(\psi,i,\rho,\sigma,n) = \mathsf{success}(\sigma') \tag{2}$$

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$\begin{aligned} [\![e]\!](\sigma, \rho, \psi, n) = r \quad & \textit{with} \quad r \coloneqq \mathsf{success}(v, \sigma') \mid \mathsf{error} \mid \mathsf{timeout} \\ \mathsf{init}_C(\psi, i, \rho, \sigma, n) = r \quad & \textit{with} \quad r \coloneqq \mathsf{success}(\sigma') \quad\ \mid \mathsf{error} \mid \mathsf{timeout} \end{aligned}$$

$$[\![e]\!](\sigma, \rho, \psi) \longrightarrow (v, \sigma') \iff \exists n.\, [\![e]\!](\sigma, \rho, \psi, n) = \mathsf{success}(v, \sigma') \tag{1}$$

$$\mathsf{init}_C(\psi, i, \rho, \sigma) \longrightarrow \sigma' \iff \exists n.\, \mathsf{init}_C(\psi, i, \rho, \sigma, n) = \mathsf{success}(\sigma') \tag{2}$$

Soundness invariant (structure)

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$\begin{aligned} [\![e]\!](\sigma, \rho, \psi, n) = r \quad &\textit{with} \quad r \coloneqq \mathsf{success}(v, \sigma') \mid \mathsf{error} \mid \mathsf{timeout} \\ \mathsf{init}_C(\psi, i, \rho, \sigma, n) = r \quad &\textit{with} \quad r \coloneqq \mathsf{success}(\sigma') \quad \mid \mathsf{error} \mid \mathsf{timeout} \end{aligned}$$

$$[\![e]\!](\sigma, \rho, \psi) \longrightarrow (v, \sigma') \iff \exists n.\, [\![e]\!](\sigma, \rho, \psi, n) = \mathsf{success}(v, \sigma') \tag{1}$$

$$\mathsf{init}_C(\psi, i, \rho, \sigma) \longrightarrow \sigma' \iff \exists n.\, \mathsf{init}_C(\psi, i, \rho, \sigma, n) = \mathsf{success}(\sigma') \tag{2}$$

Soundness invariant (structure)

$$\left.\begin{aligned} &[\![e]\!](\sigma, \rho, \psi, n) = r \\ &r \neq \mathsf{timeout} \\ & \end{aligned}\right\} \qquad \left\{ \begin{aligned} & \\ & \\ & \end{aligned}\right.$$

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$
\begin{array}{rll}
[\![e]\!](\sigma, \rho, \psi, n) = r & \textit{with} & r \coloneqq \mathsf{success}(v, \sigma') \mid \mathsf{error} \mid \mathsf{timeout} \\
\mathsf{init}_C(\psi, i, \rho, \sigma, n) = r & \textit{with} & r \coloneqq \mathsf{success}(\sigma') \quad\;\; \mid \mathsf{error} \mid \mathsf{timeout}
\end{array}
$$

$$
[\![e]\!](\sigma, \rho, \psi) \longrightarrow (v, \sigma') \iff \exists n.\, [\![e]\!](\sigma, \rho, \psi, n) = \mathsf{success}(v, \sigma') \tag{1}
$$

$$
\mathsf{init}_C(\psi, i, \rho, \sigma) \longrightarrow \sigma' \iff \exists n.\, \mathsf{init}_C(\psi, i, \rho, \sigma, n) = \mathsf{success}(\sigma') \tag{2}
$$

Soundness invariant (structure)

$$
\left.
\begin{array}{l}
[\![e]\!](\sigma, \rho, \psi, n) = r \\
r \neq \mathsf{timeout} \\
\vdash e : T
\end{array}
\right\}
\qquad\qquad
\left\{
\begin{array}{l}
\\
\\
\\
\\
\end{array}
\right.
$$

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$
\begin{aligned}
[\![e]\!](\sigma, \rho, \psi, n) = r \quad & \textit{with} \quad r \coloneqq \mathsf{success}(v, \sigma') \mid \mathsf{error} \mid \mathsf{timeout} \\
\mathsf{init}_C(\psi, i, \rho, \sigma, n) = r \quad & \textit{with} \quad r \coloneqq \mathsf{success}(\sigma') \quad \mid \mathsf{error} \mid \mathsf{timeout}
\end{aligned}
$$

$$[\![e]\!](\sigma, \rho, \psi) \longrightarrow (v, \sigma') \iff \exists n.\, [\![e]\!](\sigma, \rho, \psi, n) = \mathsf{success}(v, \sigma') \tag{1}$$

$$\mathsf{init}_C(\psi, i, \rho, \sigma) \longrightarrow \sigma' \iff \exists n.\, \mathsf{init}_C(\psi, i, \rho, \sigma, n) = \mathsf{success}(\sigma') \tag{2}$$

Soundness invariant (structure)

$$
\left.
\begin{array}{l}
[\![e]\!](\sigma, \rho, \psi, n) = r \\
r \neq \mathsf{timeout} \\
\vdash e : T
\end{array}
\right\}
\implies \exists \sigma', v. \left\{ \begin{array}{l} \\ \\ \\ \end{array} \right.
$$

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$\begin{aligned} [\![e]\!](\sigma,\rho,\psi,n) = r &\quad \textit{with} \quad r \coloneqq \mathsf{success}(v,\sigma') \mid \mathsf{error} \mid \mathsf{timeout} \\ \mathsf{init}_C(\psi,i,\rho,\sigma,n) = r &\quad \textit{with} \quad r \coloneqq \mathsf{success}(\sigma') \quad\ \mid \mathsf{error} \mid \mathsf{timeout} \end{aligned}$$

$$[\![e]\!](\sigma,\rho,\psi) \longrightarrow (v,\sigma') \iff \exists n.\, [\![e]\!](\sigma,\rho,\psi,n) = \mathsf{success}(v,\sigma') \tag{1}$$

$$\mathsf{init}_C(\psi,i,\rho,\sigma) \longrightarrow \sigma' \iff \exists n.\, \mathsf{init}_C(\psi,i,\rho,\sigma,n) = \mathsf{success}(\sigma') \tag{2}$$

Soundness invariant (structure)

$$\left.\begin{array}{l} [\![e]\!](\sigma,\rho,\psi,n) = r \\ r \neq \mathsf{timeout} \\ \vdash e : T \end{array}\right\} \implies \exists \sigma', v. \left\{\begin{array}{l} r = \mathsf{success}(v,\sigma') \\ \\ \\ \\ \\ \end{array}\right.$$

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$\begin{aligned}
[\![e]\!](\sigma, \rho, \psi, n) = r \quad &\textit{with} \quad r \coloneqq \mathrm{success}(v, \sigma') \mid \mathrm{error} \mid \mathrm{timeout} \\
\mathrm{init}_C(\psi, i, \rho, \sigma, n) = r \quad &\textit{with} \quad r \coloneqq \mathrm{success}(\sigma') \quad\ \mid \mathrm{error} \mid \mathrm{timeout}
\end{aligned}$$

$$[\![e]\!](\sigma, \rho, \psi) \longrightarrow (v, \sigma') \iff \exists n.\, [\![e]\!](\sigma, \rho, \psi, n) = \mathrm{success}(v, \sigma') \tag{1}$$

$$\mathrm{init}_C(\psi, i, \rho, \sigma) \longrightarrow \sigma' \iff \exists n.\, \mathrm{init}_C(\psi, i, \rho, \sigma, n) = \mathrm{success}(\sigma') \tag{2}$$

Soundness invariant (structure)

$$\left.\begin{array}{l}
[\![e]\!](\sigma, \rho, \psi, n) = r \\
r \neq \mathrm{timeout} \\
\vdash e : T
\end{array}\right\} \implies \exists \sigma', v. \begin{cases}
r = \mathrm{success}(v, \sigma') \\
\vDash v : T \\
\\
\\
\\
\end{cases}$$

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$\begin{aligned} [\![e]\!](\sigma,\rho,\psi,n) &= r \quad \textit{with} \quad r \coloneqq \mathsf{success}(v,\sigma') \mid \mathsf{error} \mid \mathsf{timeout} \\ \mathsf{init}_C(\psi,i,\rho,\sigma,n) &= r \quad \textit{with} \quad r \coloneqq \mathsf{success}(\sigma') \quad\ \mid \mathsf{error} \mid \mathsf{timeout} \end{aligned}$$

$$[\![e]\!](\sigma,\rho,\psi) \longrightarrow (v,\sigma') \iff \exists n.\, [\![e]\!](\sigma,\rho,\psi,n) = \mathsf{success}(v,\sigma') \tag{1}$$

$$\mathsf{init}_C(\psi,i,\rho,\sigma) \longrightarrow \sigma' \iff \exists n.\, \mathsf{init}_C(\psi,i,\rho,\sigma,n) = \mathsf{success}(\sigma') \tag{2}$$

Soundness invariant (structure)

$$\left.\begin{array}{l} [\![e]\!](\sigma,\rho,\psi,n) = r \\ r \neq \mathsf{timeout} \\ \vdash e : T \end{array}\right\} \implies \exists \sigma', v.\ \begin{cases} r = \mathsf{success}(v,\sigma') \\ \vDash v : T \\ \text{Monotonicity} \\ \\ \\ \end{cases}$$

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$\begin{aligned}
[\![e]\!](\sigma,\rho,\psi,n) = r \quad & \textit{with} \quad r \coloneqq \mathsf{success}(v,\sigma') \mid \mathsf{error} \mid \mathsf{timeout} \\
\mathsf{init}_C(\psi,i,\rho,\sigma,n) = r \quad & \textit{with} \quad r \coloneqq \mathsf{success}(\sigma') \quad\;\; \mid \mathsf{error} \mid \mathsf{timeout}
\end{aligned}$$

$$[\![e]\!](\sigma,\rho,\psi) \longrightarrow (v,\sigma') \iff \exists n.\, [\![e]\!](\sigma,\rho,\psi,n) = \mathsf{success}(v,\sigma') \tag{1}$$

$$\mathsf{init}_C(\psi,i,\rho,\sigma) \longrightarrow \sigma' \iff \exists n.\, \mathsf{init}_C(\psi,i,\rho,\sigma,n) = \mathsf{success}(\sigma') \tag{2}$$

Soundness invariant (structure)

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

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$\begin{aligned}
[\![e]\!](\sigma, \rho, \psi, n) = r \quad & \textit{with} \quad r \coloneqq \text{success}(v, \sigma') \mid \text{error} \mid \text{timeout} \\
\text{init}_C(\psi, i, \rho, \sigma, n) = r \quad & \textit{with} \quad r \coloneqq \text{success}(\sigma') \quad\ \mid \text{error} \mid \text{timeout}
\end{aligned}$$

$$[\![e]\!](\sigma, \rho, \psi) \longrightarrow (v, \sigma') \iff \exists n.\, [\![e]\!](\sigma, \rho, \psi, n) = \text{success}(v, \sigma') \qquad (1)$$

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

Soundness invariant (structure)

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

Soundness (1/3)

Definitional interpreter [Amin and Rompf(2017)]

$$\begin{aligned} [\![e]\!](\sigma,\rho,\psi,n) = r &\quad \textit{with} \quad r \coloneqq \mathsf{success}(v,\sigma') \mid \mathsf{error} \mid \mathsf{timeout} \\ \mathsf{init}_C(\psi,i,\rho,\sigma,n) = r &\quad \textit{with} \quad r \coloneqq \mathsf{success}(\sigma') \quad\ \mid \mathsf{error} \mid \mathsf{timeout} \end{aligned}$$

$$[\![e]\!](\sigma,\rho,\psi) \longrightarrow (v,\sigma') \iff \exists n.\, [\![e]\!](\sigma,\rho,\psi,n) = \mathsf{success}(v,\sigma') \tag{1}$$

$$\mathsf{init}_C(\psi,i,\rho,\sigma) \longrightarrow \sigma' \iff \exists n.\, \mathsf{init}_C(\psi,i,\rho,\sigma,n) = \mathsf{success}(\sigma') \tag{2}$$

Soundness invariant (structure)

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

Reachability and Scopability

Reachability and Scopability

Reachability and Scopability

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

Reachability and Scopability

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

Transitive closure of field access

Reachability and Scopability

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

Transitive closure of field access

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

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:

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

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

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 - $\mu \sqsubseteq \mu'$

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

Typing (1/2) - typing rules

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

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

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

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

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

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

Typing (1/2) - typing rules

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

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

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

- Local environment
- Ambient subtyping

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

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)

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

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

T-E-New

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

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)
- Hot shortcuts

T-E-Fld-Hot

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

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 (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)
- Hot shortcuts
- Monotonicity (T-E-Block)

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

Typing (2/2) - Store typing

Typing (2/2) - Store typing

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

Prevent cyclic dependencies

Typing (2/2) - Store typing

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

Prevent cyclic dependencies

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

Lower bound of actual initialization state

Typing (2/2) - Store typing

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

Prevent cyclic dependencies

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

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

Lower bound of actual initialization state

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 - $\Sigma \vDash \sigma$

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

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

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)

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

Soundness (2/3)

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

Soundness (2/3)

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

Soundness (2/3)

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

Soundness (2/3)

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

Soundness (2/3)

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

Soundness (2/3)

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

Soundness (2/3)

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

Soundness (2/3)

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

Core principles for typing

Core principles for typing

Monotonicity

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

Core principles for typing

Monotonicity

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

Stackability

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

Core principles for typing

Monotonicity

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

Stackability

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

Authority

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

Soundness (3/3)

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

Soundness (3/3)

$$\left.\begin{array}{l}
\llbracket e \rrbracket(\sigma, \rho, \psi, n) = r \\
r \neq \text{timeout} \\
\Sigma \models \sigma \\
\Sigma \models \rho : \Gamma \\
\Sigma \models \psi : T_{\texttt{this}} \\
(\Gamma, T_{\texttt{this}}) \vdash e : T
\end{array}\right\} \implies \exists \sigma',\, v,\, \Sigma'.\, \left\{\begin{array}{l}
r = \text{success}(v, \sigma') \\
\Sigma' \models \sigma' \\
\Sigma' \models v : T \\
\Sigma \preccurlyeq \Sigma' \\
\Sigma \triangleright \Sigma' \\
\Sigma \ll \Sigma'
\end{array}\right. \tag{3}$$

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

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

Take-away

- Four principles for the safe initialization of objects
    - Monotonicity (invariants progress)

Take-away

- Four principles for the safe initialization of objects
  - Monotonicity (invariants progress)
  - Authority (distinguished alias)

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)

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)

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

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

References i

N. Amin and T. Rompf.
Type soundness proofs with definitional interpreters, 2017.
Pages: 666–679 Publisher: ACM.