# Safety of Mutable References & Quantified Types

## Definitions

Sergio Domínguez

Faculty of Science
Radboud University

December 8th, 2023

# Brief recap

Syntax of $\lambda^M$

$$\begin{array}{rl} \text{Expr} & e ::= x \mid l \mid \lambda x.e \mid ee \mid \mathsf{new}(e) \mid !e \mid e := e \\ & \mid \Lambda.e \mid e[] \mid \mathsf{pack}\, e \mid \mathsf{unpack}\, e \;\mathsf{as}\; x \;\mathsf{in}\; e \\ \text{Val} & v ::= l \mid \mathsf{unit} \mid \lambda x : A.e \mid \Lambda.e \mid \mathsf{pack}\, v \end{array}$$

Store $S$ $\longleftrightarrow$ function from locations to values.

Store typing $\Psi$ $\longleftrightarrow$ function from locations to types.

State $(S, e)$ $\longleftrightarrow$ tuple of store $S$ and expression $e$.

# Safety

## Definition (Safe)

A state $(S, e)$ is safe for $k$ steps if for any reduction $(S, e) \longmapsto^j_M (S', e')$ of $j < k$ steps, either $e'$ is a value or another step is possible.

$$\begin{aligned} \mathsf{safen}(k, S, e) \stackrel{\text{def}}{=} \forall j, S', e'.(j < k, (S, e) \longmapsto^j_M (S', e')) \\ \implies (\mathsf{val}(e') \vee \exists S'', e''.(S', e') \longmapsto_M (S'', e'') \end{aligned}$$

A state $(S, e)$ is called safe if it is safe for all $k \geq 0$ steps.

$$\mathsf{safe}(S, e) \stackrel{\text{def}}{=} \forall\, k \geq 0.\ \mathsf{safen}(k, S, e)$$

### Type elements

- Types cannot be defined as sets of tuples of the form $\langle \Psi, v \rangle$.

# Modeling Stratified Types as Sets

## Type elements

- Types cannot be defined as sets of tuples of the form $\langle \Psi, v \rangle$.
- Add an index $k$ to the tuple: $\langle k, \Psi, v \rangle$, where $k$ is the level in the hierarchy.

### Definition ($k$-approximation)

The $k$-aproximation of a *set* is the subset of its elements whose index is less than $k$.

$$\lfloor \tau \rfloor_k \stackrel{\text{def}}{=} \{\langle j, \Psi, v\rangle \mid \; j < k \wedge \langle j, \Psi, v\rangle \in \tau\}$$

# Modeling Stratified Types as Sets

## Definition of Approximation

**Definition ($k$-approximation)**

The $k$-aproximation of a *set* is the subset of its elements whose index is less than $k$.

$$\lfloor \tau \rfloor_k \stackrel{\text{def}}{=} \{ \langle j, \Psi, v \rangle \mid \ j < k \wedge \langle j, \Psi, v \rangle \in \tau \}$$

This notion is extended pointwise to store typings

$$\lfloor \Psi \rfloor_k \stackrel{\text{def}}{=} \{ (l \mapsto \lfloor \tau \rfloor_k) \mid \Psi(l) = \tau \}$$

## Stratification Invariant

All type definitions obey the following: The definition of $(k+1)$-approximation of a type $\tau$ cannot consider any type beyond approximation $k$.

## Stratification Invariant

All type definitions obey the following: The definition of $(k+1)$-approximation of a type $\tau$ cannot consider any type beyond approximation $k$.

$$
\begin{aligned}
\tau \in \mathsf{Type}_0 \quad &\longleftrightarrow \quad \tau = \{\} \\
\tau \in \mathsf{Type}_{k+1} \quad &\longleftrightarrow \quad \forall \langle j, \Psi, v \rangle \in \tau.\ j \leq k \wedge \Psi \in \mathsf{StoreType}_j \\
\Psi \in \mathsf{StoreType}_k \quad &\longleftrightarrow \quad \forall l \in \mathsf{dom}(\Psi).\ \Psi(l) \in \mathsf{Type}_k
\end{aligned}
$$

## Stratification Invariant

All type definitions obey the following: The definition of $(k+1)$-approximation of a type $\tau$ cannot consider any type beyond approximation $k$.

$$
\begin{aligned}
\tau \in \mathsf{Type}_0 &\quad\longleftrightarrow\quad \tau = \{\} \\
\tau \in \mathsf{Type}_{k+1} &\quad\longleftrightarrow\quad \forall \langle j, \Psi, v\rangle \in \tau.\ j \leq k \wedge \Psi \in \mathsf{StoreType}_j \\
\Psi \in \mathsf{StoreType}_k &\quad\longleftrightarrow\quad \forall l \in \mathsf{dom}(\Psi).\ \Psi(l) \in \mathsf{Type}_k
\end{aligned}
$$

$$
\begin{aligned}
\tau \in \mathsf{Type} &\quad\longleftrightarrow\quad \forall k.\ \lfloor \tau \rfloor_k \in \mathsf{Type}_k \\
\Psi \in \mathsf{StoreType} &\quad\longleftrightarrow\quad \forall k.\ \lfloor \tau \rfloor_k \in \mathsf{StoreType}_k
\end{aligned}
$$

The *hypothetical* definition of ref $\tau$ is as follows:

$$(\text{ref } \tau)^k \overset{\text{something like}}{=} \{\langle \Psi^{k-1}, l \rangle \mid \Psi^{k-1}(l) = \tau^{k-1}\}$$

# Modeling Stratified Types as Sets

## Definition of ref $\tau$

The *hypothetical* definition of ref $\tau$ is as follows:

$$(\text{ref } \tau)^k \overset{\text{something like}}{=} \{\langle \Psi^{k-1}, l \rangle \mid \Psi^{k-1}(l) = \tau^{k-1}\}$$

Now, we can define the *actual* definition of ref $\tau$ as follows:

$$\text{ref } \tau \overset{\text{def}}{=} \{\langle k, \Psi, l \rangle \mid \lfloor \Psi \rfloor_k (l) = \lfloor \tau \rfloor_k\}$$

**Note**

The actual definition of ref $\tau$ satisfies the stratification invariant.

| | |
|---|---|
| | `%` $S_0 = \Psi_0 = \{\}$ |
| `let` $x_1$ `= new(true) in` | `%` $S_1 = S_0[l_1 \mapsto \texttt{true}], \Psi_1 = \lfloor \Psi_0 \rfloor_{10}[l_1 \mapsto \lfloor \texttt{bool} \rfloor_{10}]$ |
| | `%` $x_1 \mapsto l_1$ `, 10 more steps` |
| `let` $x_2$ `= ... in` | `%` $S_2 = ..., \Psi_2 = ..., x_1 \mapsto l_1$ |
| $\vdots$ | |
| `let` $x_n$ `= ... in` | `%` $S_n = ..., \Psi_n = ..., x_1 \mapsto l_1$ `, 6 more steps` |
| `let` $y$ `= !`$x_1$ `in` | `%` $S_{n+1} = ..., \Psi_{n+1} = ...$ |
| $e_{rest}$ | |

# Properties of Types

**Definition (State Extensions)**

For any two machine states $(S, e)$ and $(S', e')$, and store typings $\Psi$ and $\Psi$', such that $S$ satisfies $\Psi$ for $k$ steps and $S'$ satisfies $\Psi$' for $j \leq k$ steps, if $(S, e) \longmapsto_M^{k-j} (S', e')$, then the relation between $\Psi$ and $\Psi$' is as follows:

$$(k, \Psi) \sqsubseteq (j, \Psi') \stackrel{\text{def}}{=} j \leq k \wedge (\forall l \in \mathrm{dom}(\Psi).\ \lfloor \Psi' \rfloor_j (l) = \lfloor \Psi \rfloor_j (l))$$

# Properties of Types

**Definition (Type)**

A type is a set $\tau$ of tuples of the form $\langle k, \Psi, v\rangle$ where $v$ is a value, $k$ is a natural number, and $\Psi$ is a store typing, and where the set $\tau$ is *closed under state extension*; that is,

$$\mathsf{type}(\tau) \stackrel{\mathsf{def}}{=} \forall \langle k, \Psi, v\rangle \in \tau.\ (k, \Psi) \sqsubseteq (j, \Psi') \implies \langle j, \Psi', v\rangle \in \tau$$

## Definition (Base Types)

A base type $\tau_{\text{base}}$ is given by a set of values $V$.

$$\tau_{\text{base}} \stackrel{\text{def}}{=} \{\langle k, \Psi, v\rangle \mid v \in V\}$$

## Definition (Base Types)

A base type $\tau_{\text{base}}$ is given by a set of values $V$.

$$\tau_{\text{base}} \stackrel{\text{def}}{=} \{\langle k, \Psi, v\rangle \mid v \in V\}$$

## Important notes

- The store typing $\Psi$ is irrelevant for base types.
- The index $k$ is irrelevant for base types.
- The set is closed under state extension.

# Type Definitions

## Base Types

### Definition (Base Types)

A base type $\tau_{\textsf{base}}$ is given by a set of values $V$.

$$\tau_{\textsf{base}} \stackrel{\text{def}}{=} \{\langle k, \Psi, v\rangle \mid v \in V\}$$

### Important notes

- The store typing $\Psi$ is irrelevant for base types.
- The index $k$ is irrelevant for base types.
- The set is closed under state extension.

unit, bool, and int are examples of base types.

# Type Definitions

## Function Types

Concerns when deciding if $\lambda x.e$ has type $\tau_1 \to \tau_2$ for $k$ steps for a store typing $\Psi$:

Concerns when deciding if $\lambda x.e$ has type $\tau_1 \to \tau_2$ for $k$ steps for a store typing $\Psi$:

- The program may not apply $\lambda x.e$ to a value of type $\tau_1$ until some time in the future (where the store typing may have changed).
- The number of steps required to reach that future point.

Concerns when deciding if $\lambda x.e$ has type $\tau_1 \to \tau_2$ for $k$ steps for a store typing $\Psi$:

- The program may not apply $\lambda x.e$ to a value of type $\tau_1$ until some time in the future (where the store typing may have changed).
- The number of steps required to reach that future point.

The intuitive definition of $\tau_1 \to \tau_2$ is as follows:

$$\tau_1 \to \tau_2 \overset{\text{something like}}{=} \{\langle k, \Psi, \lambda x.e\rangle \mid \forall v, \Psi', j < k.\ (k, \Psi) \sqsubseteq (j, \Psi') \wedge \langle j, \Psi', v\rangle \in \tau_1 \implies \text{evaluating } e\,[x := v] \text{ gives a value in } \tau_2 \text{ in less than } j \text{ steps}\}$$

## Definition (Well-Typed Store)

A store $S$ is well-typed to approximation $k$ with respect to a store typing $\Psi$ iff dom$(\Psi) \subseteq$ dom$(S)$ and the contents of each $l \in$ dom$(\Psi)$ has the type $\Psi(l)$ to aproximation $k$.

$$S \;:_k\; \Psi \quad \stackrel{\text{def}}{=} \quad \begin{array}{l} \mathsf{dom}(\Psi) \subseteq \mathsf{dom}(S) \land \\ \forall j < k.\, l \in \mathsf{dom}(\Psi).\ \langle j, \lfloor \Psi \rfloor_j, S(l) \rangle \in \lfloor \Psi \rfloor_k(l) \end{array}$$

## Definition (Well-Typed Store)

A store $S$ is well-typed to approximation $k$ with respect to a store typing $\Psi$ iff $\mathsf{dom}(\Psi) \subseteq \mathsf{dom}(S)$ and the contents of each $l \in \mathsf{dom}(\Psi)$ has the type $\Psi(l)$ to aproximation $k$.

$$S :_k \Psi \stackrel{\text{def}}{=} \mathsf{dom}(\Psi) \subseteq \mathsf{dom}(S) \wedge \forall j < k.\, l \in \mathsf{dom}(\Psi).\ \langle j, \lfloor\Psi\rfloor_j, S(l)\rangle \in \lfloor\Psi\rfloor_k(l)$$

## Important notes

- $j < k$ to avoid circularity.
- $\mathsf{dom}(\Psi) = \mathsf{dom}(S)$, though not incorrect, is too restrictive.

# Type Definitions

## Function Types

### Definition (Expr:Type)

For any closed expression $e$ and type $\tau$, $e \;:_{k,\Psi}\; \tau$ iff whenever $S \;:_k\; \Psi$, $(S, e) \longmapsto^j_M (S', e')$ for $j < k$, and $(S', e')$ is irreducible, then there exists a store typing $\Psi'$ such that $(j, \Psi) \sqsubseteq (k - j, \Psi')$, $S' \;:_{k-j}\; \Psi'$, and $\langle k - j, \Psi', e' \rangle \in \tau$.

$$
\begin{aligned}
e \;:_{k,\Psi}\; \tau \quad \stackrel{\text{def}}{=} \quad & \forall j, S, S', e'.(0 \leq j < k \wedge S \;:_k\; \Psi \\
& \qquad\qquad\quad \wedge (S, e) \longmapsto^j_M (S', e') \wedge \text{irred}((S', e')) \\
& \Longrightarrow \exists \Psi'.\, (j, \Psi) \sqsubseteq (k - j, \Psi') \\
& \qquad\qquad\quad \wedge S' \;:_{k-j}\; \Psi' \wedge \langle k - j, \Psi', e' \rangle \in \tau)
\end{aligned}
$$

# Type Definitions

## Function Types

### Definition (Expr:Type)

For any closed expression $e$ and type $\tau$, $e :_{k,\Psi} \tau$ iff whenever $S :_k \Psi$, $(S, e) \longmapsto^j_M (S', e')$ for $j < k$, and $(S', e')$ is irreducible, then there exists a store typing $\Psi'$ such that $(j, \Psi) \sqsubseteq (k - j, \Psi')$, $S' :_{k-j} \Psi'$, and $\langle k - j, \Psi', e' \rangle \in \tau$.

$$
\begin{aligned}
e :_{k,\Psi} \tau \quad \stackrel{\text{def}}{=} \quad & \forall j, S, S', e'.(0 \le j < k \wedge S :_k \Psi \\
& \qquad \wedge (S, e) \longmapsto^j_M (S', e') \wedge \mathrm{irred}((S', e')) \\
& \implies \exists \Psi'.\ (j, \Psi) \sqsubseteq (k - j, \Psi') \\
& \qquad \wedge S' :_{k-j} \Psi' \wedge \langle k - j, \Psi', e' \rangle \in \tau)
\end{aligned}
$$

### Important note

If $v$ is a value of type $\tau$ and $k > 0$, then $v :_{k,\Psi} \tau \longleftrightarrow \langle k, \Psi, v \rangle \in \tau$.

## Definition (Function type)

The semantics of function types is defined as follows:

$$\begin{aligned}\tau_1 \to \tau_2 \stackrel{\text{def}}{=} \{\langle k, \Psi, \lambda x.e\rangle \mid \forall v, \Psi', j < k.&\\ ((k, \Psi) \sqsubseteq (j, \Psi') \wedge \langle j, \Psi', v\rangle \in \tau_1)&\\ \implies e\,[x := v]\ :_{j,\Psi'}\ \tau_2\}&\end{aligned}$$

## Note

The definition satisfies the stratification invariant.

### Important note

Instead of writing $\forall \alpha.\tau$ and $\exists \alpha.\tau$, I write $\forall F$ and $\exists F$ where $\alpha$ is the only free type variable in $\tau$.

# Type Definitions

## Quantified Types

**Important note**

Instead of writing $\forall\alpha.\tau$ and $\exists\alpha.\tau$, I write $\forall F$ and $\exists F$ where $\alpha$ is the only free type variable in $\tau$.

- Same expressive power.
- Unconventional notation, but it leads to simpler semantics as we don't need to bookkeep the free type variables.

## Definition (Universal type)

$$\begin{aligned}\forall F \overset{\mathsf{def}}{=} \{\langle k, \Psi, \Lambda.e\rangle \mid \forall \tau, \Psi', j < k.\ & \\ & ((k,\Psi) \sqsubseteq (j, \Psi') \wedge \mathsf{type}(\lfloor \tau \rfloor_j)) \\ & \implies \forall i < j.\ e :_{i, \lfloor \Psi' \rfloor_i} F(\tau)\}\end{aligned}$$

# Type Definitions

## Quantified Types

### Definition (Universal type)

$$\begin{aligned}\forall F \stackrel{\text{def}}{=} \{\langle k, \Psi, \Lambda.e\rangle \mid \forall \tau, \Psi', j < k.\ & \\ & ((k, \Psi) \sqsubseteq (j, \Psi') \land \mathsf{type}(\lfloor \tau \rfloor_j)) \\ & \implies \forall i < j.\ e :_{i, \lfloor \Psi' \rfloor_i} F(\tau)\}\end{aligned}$$

### Note

The definition satisfies the stratification invariant.

## Definition (Existential type)

$$\exists F \stackrel{\text{def}}{=} \{\langle k, \Psi, \text{pack } v\rangle \mid \exists \tau.(\text{type}(\lfloor \tau \rfloor_k) \wedge \forall j < k.\langle j, \lfloor \Psi' \rfloor_j, v\rangle \in F(\tau))\}$$

## Note

The definition satisfies the stratification invariant.

## Definition (Semantics of Judgments)

For any type environment $\Gamma$ and value environment $\sigma$ I write $\sigma :_{k,\Psi} \Gamma$ if for all variables $x \in \mathsf{dom}(\Gamma)$ we have $\sigma(x) :_{k,\Psi} \Gamma(x)$; that is

$$\sigma :_{k,\Psi} \Gamma \overset{\text{def}}{=} \forall x \in \mathsf{dom}(\Gamma).\ \sigma(x) :_{k,\Psi} \Gamma(x)$$

## Note

Value environment $\sigma$ is a mapping from variables to values.
Type environment $\Gamma$ is a mapping from variables to types.

## Definition (Semantics of Judgments (Cont.))

I write $\Gamma \vDash^k_M e : \tau$ iff $\mathsf{FV}(e) \subseteq \mathsf{dom}(\Gamma)$ and

$$\forall \sigma, \Psi. \ (\sigma :_{k,\Psi} \Gamma \implies \sigma(e) \ :_{k,\Psi} \ \tau)$$

where $\sigma(e)$ is the result of substituting the free variables in $e$ with their values under $\sigma$.

## Definition (Semantics of Judgments (Cont.))

I write $\Gamma \vDash_M^k e : \tau$ iff $\mathsf{FV}(e) \subseteq \mathsf{dom}(\Gamma)$ and

$$\forall \sigma, \Psi.\ (\sigma :_{k,\Psi} \Gamma \implies \sigma(e)\ :_{k,\Psi}\ \tau)$$

where $\sigma(e)$ is the result of substituting the free variables in $e$ with their values under $\sigma$.

We remove the $k$ superscript when the property holds for all $k \geq 0$ and we remove $\Gamma$ when it is empty.

## Notes

$\Gamma \vDash_M^k e : \tau$ can be obtained from our semantics of a similar judgement of closed expressions.

$$\frac{}{\Gamma \vDash_M x : \Gamma(x)} \texttt{(M-var)} \qquad \frac{}{\Gamma \vDash_M \texttt{unit} : \textsf{unit}} \texttt{(M-unit)}$$

$$\frac{\Gamma\left[x \mapsto \tau_1\right] \vDash_M e : \tau_2}{\Gamma \vDash_M \lambda x.e : \tau_1 \rightarrow \tau_2} \texttt{(M-abs)}$$

$$\frac{\Gamma \vDash_M e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vDash_M e_2 : \tau_1}{\Gamma \vDash_M (e_1\, e_2) : \tau_2} \texttt{(M-app)}$$

$$\frac{\Gamma \vDash_M e : \tau}{\Gamma \vDash_M \texttt{new}(e) : \textsf{ref}\ \tau} \texttt{(M-new)} \qquad \frac{\Gamma \vDash_M e : \textsf{ref}\ \tau}{\Gamma \vDash_M\, !\, e : \tau} \texttt{(M-deref)}$$

$$\frac{\Gamma \vDash_M e_1 : \textsf{ref}\ \tau \qquad \Gamma \vDash_M e_2 : \tau}{\Gamma \vDash_M e_1 := e_2 : \textsf{unit}} \texttt{(M-assign)}$$

$$\frac{\forall \tau.\, \mathrm{type}(\tau) \Longrightarrow \Gamma \vDash_M e : F(\tau)}{\Gamma \vDash_M \Lambda.e : \forall F} \texttt{(M-tabs)} \qquad \frac{\mathrm{type}(\tau) \qquad \Gamma \vDash_M e : \forall F}{\Gamma \vDash_M e[\,] : F(\tau)} \texttt{(M-tapp)}$$

# Judgments, Typing Rules and Safety

## Typing Rules (Cont.)

$$\frac{\text{type}(\tau) \qquad \Gamma \vDash_M e : F(\tau)}{\Gamma \vDash_M \texttt{pack}\, e : \exists F} \;\texttt{(M-pack)}$$

$$\frac{\Gamma \vDash_M e_1 : \exists F \qquad \forall \tau.\, \text{type}(\tau) \implies \Gamma\,[x \mapsto F(\tau)] \vDash_M e_2 : \tau_2}{\Gamma \vDash_M \texttt{unpack}\, e_1 \,\texttt{as}\, x \,\texttt{in}\, e_2 : \tau_2} \;\texttt{(M-unpack)}$$

# Safety

**Theorem (Safety)**

*If* $\vDash e : \tau$, $\tau$ *is a type, and* $S$ *is a store, then* $(S, e)$ *is safe.*

# Relation with S4 Modal Logic

First, one specifies a set of possible worlds $W$:

$$W = \{\langle k, \Psi\rangle \mid \forall k \geq 0 \wedge l \in \mathsf{dom}(\Psi).\ (\forall\langle j, \Psi', v\rangle \in \Psi(l).\ j < k)\}$$

Second, a binary relation Acc $\subseteq W \times W$ which corresponds to:

$$\text{State Extension } (k, \Psi) \sqsubseteq (j, \Psi')$$

Third, a label function $\mathsf{L} : W \to \mathcal{P}(\texttt{Atoms})$ which corresponds to:

$$\mathsf{L}(k, \Psi) = \{(l, \tau) \mid \lfloor\Psi\rfloor_k(l) = \tau\}$$

# Well Founded and Nonexpansive Type Functions

## Definition

A nonexpansive functional is a function $F$ from types to types such that for any type $\tau$ and $k \geq 0$ we have:

$$\lfloor F(\tau) \rfloor_k = \lfloor F(\lfloor \tau \rfloor_k) \rfloor_k$$

# Well Founded and Nonexpansive Type Functions

## Definition

A nonexpansive functional is a function $F$ from types to types such that for any type $\tau$ and $k \geq 0$ we have:

$$\lfloor F(\tau) \rfloor_k = \lfloor F(\lfloor \tau \rfloor_k) \rfloor_k$$

## Definition (Well Founded)

A well founded functional is a function $F$ from types to types such that for any type $\tau$ and $k \geq 0$ we have:

$$\lfloor F(\tau) \rfloor_{k+1} = \lfloor F(\lfloor \tau \rfloor_k) \rfloor_{k+1}$$

# Thank you!