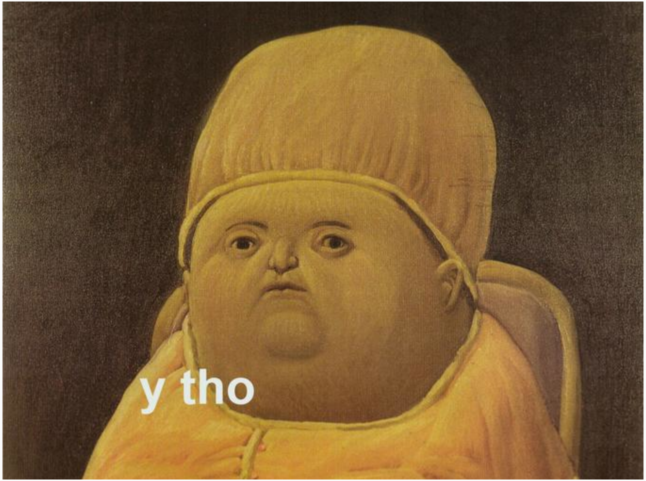
Eliminating Ghost code

One step forward, two steps backward

Noé De Santo

February 7, 2023

Ghost code?



A sparse matrix

```
class SparseMatrix {
    let data: Repr

    fun * (that: SparseMatrix): SparseMatrix {
        let resData = ... // Do the sparse multiplication
        return SparseMatrix {
            data = resData,
        }
    }
}
```

Testing a sparse matrix

```
class SparseMatrix {
    let data: Repr
    let full: Matrix

    fun * (that: SparseMatrix): SparseMatrix {
        let resData = ... // Do the sparse multiplication
        return SparseMatrix {
            data = resData,
            full = this.full * that.full
        }
    }.ensuring( res => res == SparseMatrix.from(res.full) )
}
```

Testing a sparse matrix for free

```
class SparseMatrix {
    let data: Repr
    let full: Ghost Matrix

    fun * (that: SparseMatrix): SparseMatrix {
        let resData = ... // Do the sparse multiplication
        return SparseMatrix {
            data = resData,
            full = this.full * that.full
        }
    }.ensuring( res => res == SparseMatrix.from(res.full) )
}
```

Testing a sparse matrix for free

```
class SparseMatrix {
    let data: Repr
    let full: Ghost Matrix

    fun * (that: SparseMatrix): SparseMatrix {
        let resData = ... // Do the sparse multiplication
        return SparseMatrix {
            data = resData,
            full = this.full * that.full
        }
    }.ensuring( res => res == SparseMatrix.from(res.full) )
}
```

Verifying a sparse matrix for free

```
class SparseMatrix {
    let data: Repr
    let full: Ghost Matrix

    @invariant
    Ghost fun fullMatch(): Ghost Boolean {
        this == SparseMatrix.from(res.full)
    }

    fun * (that: SparseMatrix): SparseMatrix { ...
    }.ensuring( res => res.full == this.full * that.full )
}
```

The calculus

Values

$$\begin{array}{rll} v ::= & k & \text{Constant} \\ \mid & l & \text{Location} \\ \mid & \lambda x{:}\,T.\ p & \text{Abstraction} \end{array}$$

Atoms

$$\begin{array}{rll} a ::= & v & \text{Value} \\ \mid & x & \text{Variable} \end{array}$$

Programs

$$\begin{array}{rll} p ::= & L(a) & \text{Filled context} \end{array}$$

Expressions

$$\begin{array}{rll} e ::= & a\,a & \text{Application} \\ \mid & !a & \text{Dereference} \\ \mid & \mathtt{new}\,[T]\ a & \text{Allocation} \end{array}$$

Let contexts

$$\begin{array}{rll} L ::= & a := a;\ L & \text{Assignment} \\ \mid & \mathtt{let}\,x{:}\,T = e;\ L & \text{Let-binding} \\ \mid & [] & \text{Hole} \end{array}$$

Terms

Values

$v ::=$		k	Constant
	$\mid$	l	Location
	$\mid$	$\lambda x\colon T.\ p$	Abstraction

Atoms

$a ::=$		v	Value
	$\mid$	x	Variable

Programs

$p ::=$		$L(a)$	Filled context

Expressions

$e ::=$		$a\ a$	Application
	$\mid$	$!a$	Dereference
	$\mid$	$\texttt{new}\,[T]\ a$	Allocation

Let contexts

$L ::=$		$a := a;\ L$	Assignment
	$\mid$	$\texttt{let}\, x\colon T = e;\ L$	Let-binding
	$\mid$	$[]$	Hole

Terms

Values

$v ::=$		k	Constant
	$\mid$	l	Location
	$\mid$	$\lambda x\colon T.\ p$	Abstraction

Atoms

$a ::=$		v	Value
	$\mid$	x	Variable

Programs

$p ::=$		$L(a)$	Filled context

Expressions

$e ::=$		$a\ a$	Application
	$\mid$	$!a$	Dereference
	$\mid$	$\mathtt{new}\,[T]\ a$	Allocation

Let contexts

$L ::=$		$a := a;\ L$	Assignment
	$\mid$	$\mathtt{let}\,x\colon T = e;\ L$	Let-binding
	$\mid$	$[]$	Hole

Terms[1]

$t ::= \quad p \mid e$

[1] Note that $v \subset a \subset p$.

A program

$$
\begin{aligned}
&\texttt{let}\ r\colon \textsc{Ref}\ \mathbb{N} = \texttt{new}\,[\mathbb{N}]\ 0;\\
&r := 1;\\
&\texttt{let}\ n\colon \mathbb{N} = {!r};\\
&n
\end{aligned}
$$

A program

$$\left.\begin{array}{l}\left.\begin{array}{l} \texttt{let}\; r\colon \textsc{Ref}\;\mathbb{N} = \texttt{new}\,[\mathbb{N}]\;0;\\ r := 1;\\ \texttt{let}\; n\colon \mathbb{N} = !r;\end{array}\right\} L \\ \left.\begin{array}{l} n \end{array}\right\} a\end{array}\right\} L(a) = p$$

Reduction rules

Definition (Reduction derivations)

$$\frac{}{(\lambda x{:}\, T.\ p)\ v \circ \mu \ \longrightarrow\ [x \mapsto v]\, p \circ \mu}\ \text{R-AppAbs}$$

$$\frac{l \in \operatorname{dom}(\mu)}{!l \circ \mu \ \longrightarrow\ \mu(l) \circ \mu}\ \text{R-Deref} \qquad \frac{l \in \operatorname{dom}(\mu)}{l := v;\ p \circ \mu \ \longrightarrow\ p \circ [l \mapsto v]\, \mu}\ \text{R-Assign}$$

$$\frac{l \notin \operatorname{dom}(\mu)}{\texttt{new}\,[\_]\ v \circ \mu \ \longrightarrow\ l \circ \mu + \{l \mapsto v\}}\ \text{R-New}$$

$$\frac{e \circ \mu \ \longrightarrow\ L(a) \circ \mu'}{\texttt{let}\, x{:}\, T = e;\ p \circ \mu \ \longrightarrow\ L([x \mapsto a]\, p) \circ \mu'}\ \text{R-Let}$$

Reduction rules

Definition (Reduction derivations)

$$\frac{}{(\lambda x{:}\, T.\ p)\ v \circ \mu \ \longrightarrow\ [x \mapsto v]\, p \circ \mu}\ \text{R-AppAbs}$$

$$\frac{l \in \mathrm{dom}\,(\mu)}{!l \circ \mu \ \longrightarrow\ \mu(l) \circ \mu}\ \text{R-Deref} \qquad \frac{l \in \mathrm{dom}\,(\mu)}{l := v;\ p \circ \mu \ \longrightarrow\ p \circ [l \mapsto v]\, \mu}\ \text{R-Assign}$$

$$\frac{l \notin \mathrm{dom}\,(\mu)}{\texttt{new}\,[\,\_\,]\ v \circ \mu \ \longrightarrow\ l \circ \mu + \{l \mapsto v\}}\ \text{R-New}$$

$$\frac{e \circ \mu \ \longrightarrow\ L(a) \circ \mu'}{\texttt{let}\, x{:}\, T = e;\ p \circ \mu \ \longrightarrow\ L([x \mapsto a]\, p) \circ \mu'}\ \text{R-Let}$$

Calling a function

```
(λx: ℕ.
    let a: ℕ = 2*x;
    let b: ℕ = a+3;
    b
)(0)
```

Calling a function

```
(λx: ℕ.
    let a: ℕ = 2*x;
    let b: ℕ = a+3;
    b
)(0)
```

$\xrightarrow{\text{R-AppAbs}}$

```
let a: ℕ = 2*0;
let b: ℕ = a+3;
b
```

Calling a function

```
(λx: ℕ.
    let a: ℕ = 2*x;
    let b: ℕ = a+3;
    b
)(0)
```

$\xrightarrow{\text{R-AppAbs}}$

```
let a: ℕ = 2*0;
let b: ℕ = a+3;
b
```

```
let n: ℕ = (λx: ℕ.
        let a: ℕ = 2*x;
        let b: ℕ = a+3;
        b
    )(0);
r := n;
```

Calling a function

```
(λx: ℕ.
    let a: ℕ = 2*x;
    let b: ℕ = a+3;
    b
)(0)
```

$$\xrightarrow{\text{R-AppAbs}}$$

```
let a: ℕ = 2*0;
let b: ℕ = a+3;
b
```

```
let n: ℕ = (λx: ℕ.
        let a: ℕ = 2*x;
        let b: ℕ = a+3;
        b
    )(0);
r := n;
```

$$\xrightarrow{\text{R-Let}}$$

```
let a: ℕ = 2*0;
let b: ℕ = a+3;
r := b;
b
```

Ghost in the calculus

Generalized ghost annotations

Every type is annotated with a (ghost) label, e.g. ${}_{\ell}\mathbb{N}$.

Generalized ghost annotations

Every type is annotated with a (ghost) label, e.g. ${}_{\ell}\mathbb{N}$.

Definition (Labels)

Labels are taken from a bounded lattice $(\mathcal{L}, \sqcap, \sqcup)$. The induced partial order is noted $\sqsubseteq$.

Generalized ghost annotations

Every type is annotated with a (ghost) label, e.g. ${}_\ell\mathbb{N}$.

Semantic (intuitively)

If $\ell \sqsubseteq h$:

1. ℓ is ``more important'' for run-time than h;
2. ``$h$-annotated data'' cannot flow into ``ℓ-annotated data''.

E.g. $R\textsc{egular} \sqsubseteq G\textsc{host}$.

Typing judgments

Definition (Typing judgment)

A typing judgment is of the form

$$\Gamma \circ \Sigma \vdash t \; : T \circ \ell$$

- Γ: Variable context;
- Σ: Store context;
- t: Term being typed;
- T: Determined type;
- ℓ: Determined effect bound.

Typing judgments

Definition (Typing judgment)

A typing judgment is of the form

$$\Gamma \circ \Sigma \vdash t \; : T \circ \ell$$

ℓ --- determined effect bound, e.g.:

- $\ell = \top$: No effect;
- $\ell = G_{\text{HOST}}$: Only ghost locations are allocated or written to;
- $\ell = R_{\text{EGULAR}}$: Anything can happen.

Types

Definition (Types)

$$\ell \in \mathcal{L} \qquad \text{Labels}$$

$$P ::= \mathcal{C} \mid \textsc{Top} \mid \textsc{Ref}\ T \mid T \xrightarrow{\ell} T \qquad \text{Pre-types}$$

$$T ::= {}_{\ell}P \qquad \text{Types}$$

Types

Definition (Types)

$$\ell \in \mathcal{L}$$ Labels

$$P ::= \mathcal{C} \mid \textsc{Top} \mid \textsc{Ref}\ T \mid T \xrightarrow{\ell} T$$ Pre-types

$$T ::= {}_{\ell}P$$ Types

Bound of the effect of the function *when called*.

Typing rules (part 1/∞)

Definition (Typing derivations --- atoms)

$$\frac{x \in \mathrm{dom}\,(\Gamma)}{\Gamma \circ \Sigma \vdash x \ : \Gamma(x) \circ \top}\ \text{AT-Var} \qquad \frac{}{\Gamma \circ \Sigma \vdash k \ : {}_{\bot}\mathcal{C} \circ \top}\ \text{AT-Const}$$

$$\frac{\Sigma(l) = T}{\Gamma \circ \Sigma \vdash l \ : {}_{\mathcal{L}_{\mathrm{of}}(T)}\textsc{Ref}\ T \circ \top}\ \text{AT-Loc} \qquad \frac{\Gamma + \{x\colon S\} \circ \Sigma \vdash p \ : T \circ m}{\Gamma \circ \Sigma \vdash \lambda x\colon S.\, p \ : {}_{\bot}\left(S \xrightarrow{m} T\right) \circ \top}\ \text{AT-Abs}$$

Typing rules (part 2/∞)

Definition (Typing derivations --- expressions)

$$\frac{\Gamma \circ \Sigma \vdash a_1 \;:\; \_\left(S \xrightarrow{\rho} T\right) \circ \top \quad \Gamma \circ \Sigma \vdash a_2 \;: S' \circ \top \quad S' <: S}{\Gamma \circ \Sigma \vdash a_1\, a_2 \;: T \circ \rho}\text{AT-App}$$

$$\frac{\Gamma \circ \Sigma \vdash a \;:\; \_\ \textsc{Ref}\ T \circ \top}{\Gamma \circ \Sigma \vdash\ !a \;: T \circ \top}\text{AT-Deref} \qquad \frac{\Gamma \circ \Sigma \vdash a \;: S \circ \top \quad S <: {}_{\rho}P}{\Gamma \circ \Sigma \vdash \mathtt{new}\left[{}_{\rho}P\right]\ a \;:\; {}_{\rho}\textsc{Ref}\ {}_{\rho}P \circ \rho}\text{AT-New}$$

Definition (Typing derivations --- let contexts)

$$\frac{\Gamma \circ \Sigma \vdash a_1 \;:\; \_\ \textsc{Ref}\ {}_{\rho}P \circ \top \quad \Gamma \circ \Sigma \vdash a_2 \;: S \circ \top \quad \Gamma \circ \Sigma \vdash p \;: T \circ n \quad S <:\ {}_{\rho}P}{\Gamma \circ \Sigma \vdash a_1 := a_2;\ p \;: T \circ \rho \sqcap n}\ \text{AT-AssSeq}$$

$$\frac{\Gamma \circ \Sigma \vdash e \;: S \circ m \quad S <: U \quad \Gamma + \{x{:}\, U\} \circ \Sigma \vdash p \;: T \circ n}{\Gamma \circ \Sigma \vdash \texttt{let}\, x{:}\, U = e;\ p \;: T \circ m \sqcap n}\ \text{AT-LetSeq}$$

Typing rules (part 4/∞)

Definition (Subtyping derivations)

$$\frac{\ell \sqsubseteq h}{{}_{\ell}\mathcal{C} <: {}_{h}\mathcal{C}}\text{AS-Const} \qquad \frac{\ell \sqsubseteq h}{{}_{\ell}P <: {}_{h}\textsc{Top}}\text{AS-Top} \qquad \frac{}{{}_{\ell}\textsc{Ref}\ T <: {}_{\ell}\textsc{Ref}\ T}\text{AS-Ref}$$

$$\frac{S' <: S \quad T <: T' \quad m' \sqsubseteq m \quad \ell \sqsubseteq h}{{}_{\ell}\left(S \xrightarrow{m} T\right) <: {}_{h}\left(S' \xrightarrow{m'} T'\right)}\text{AS-Arrow}$$

Type health

Definition (Healthy types)

A type T is healthy if it satisfies:

- $T = {}_{\ell}\mathcal{C} \implies \mathbb{1}$
- $T = {}_{\ell}\textsc{Top} \implies \mathbb{1}$
- $T = {}_{\ell}\textsc{Ref}\ {}_{m}P \implies \ell = m$
- $T = {}_{\ell}\left(T \xrightarrow{m} {}_{n}P\right) \implies (\ell \sqsubseteq m \land \ell \sqsubseteq n)$

Type health

Definition (Healthy types)

A type T is healthy if it satisfies:

- $T = {}_{\ell}\textsc{Ref}\ {}_{m}P \implies \ell = m$
- $T = {}_{\ell}\left(T \xrightarrow{m} {}_{n}P\right) \implies (\ell \sqsubseteq m \wedge \ell \sqsubseteq n)$

- Intuitively: a ghost function cannot return a retained value, or have a retained effect.

Type health

Definition (Healthy types)

A type T is healthy if it satisfies:

- $T = {}_{\ell}\textsc{Ref}\ {}_{m}P \implies \ell = m$
- $T = {}_{\ell}\Big(T \xrightarrow{m} {}_{n}P\Big) \implies (\ell \sqsubseteq m \wedge \ell \sqsubseteq n)$

- Intuitively: a ghost function cannot return a retained value, or have a retained effect.
- All rules (typing, subtyping) include it implicitly as a side-condition.

Erasure

Relevance

We fix a threshold $\tau \in \mathcal{L}$.

Definition (Relevance)

A label ℓ is relevant, noted $\mathcal{R}^{\tau}(\ell)$, if $\ell \sqsubseteq \tau$.

A type T is relevant, noted $\mathcal{R}^{\tau}(T)$, if $T = {}_{\ell}P \wedge \mathcal{R}^{\tau}(\ell)$.

Relevance

We fix a threshold $\tau \in \mathcal{L}$.

Definition (Relevance)

A label ℓ is relevant, noted $\mathcal{R}^\tau(\ell)$, if $\ell \sqsubseteq \tau$.

A type T is relevant, noted $\mathcal{R}^\tau(T)$, if $T = {}_{\ell}P \wedge \mathcal{R}^\tau(\ell)$.

Intuitively: something is relevant if we want to keep it (e.g. R_{EGULAR}), and irrelevant if we want it to disappear after erasure (e.g. G_{HOST}).

THE Erasure function (part 1/∞)

Definition (Erasure function)

If $\Gamma \circ \Sigma \vdash t\ : T \circ \ell$ and $T <: A$, $\mathcal{E}^\tau$ is defined by case analysis on t:

$$\mathcal{E}^\tau(\Gamma, \Sigma, x, A) ::= \begin{cases} x & \text{if } \mathcal{R}^\tau(A) \\ () & \text{else} \end{cases}$$

$$\mathcal{E}^\tau(\Gamma, \Sigma, k, A) ::= \begin{cases} k & \text{if } \mathcal{R}^\tau(A) \\ () & \text{else} \end{cases}$$

$$\mathcal{E}^\tau(\Gamma, \Sigma, l, A) ::= \begin{cases} l & \text{if } \mathcal{R}^\tau(A) \\ () & \text{else} \end{cases}$$

THE Erasure function (part 2/∞)

Definition (Erasure function)

If $\Gamma \circ \Sigma \mapsto t \ : T \circ \ell$ and $T <: A$, $\mathcal{E}^\tau$ is defined by case analysis on t:

$$\mathcal{E}^\tau(\Gamma, \Sigma, \lambda x\colon S.\ p_1, A) ::= \begin{cases} \lambda x\colon \mathcal{E}^\tau(S)\,.\ \mathcal{E}^\tau(\Gamma + \{x\colon S\}\,, \Sigma, p_1, T') & \text{if } \mathcal{R}^\tau(A) \\ () & \text{else} \end{cases}$$

$$\text{where } \Gamma + \{x\colon S\} \circ \Sigma \mapsto p_1 \ : T' \circ \_$$

$$\mathcal{E}^\tau(\Gamma, \Sigma, a_1\, a_2, A) ::= (\mathcal{E}^\tau(\Gamma, \Sigma, a_1, T'))\ \ (\mathcal{E}^\tau(\Gamma, \Sigma, a_2, S'))$$

$$\text{where } \Gamma \circ \Sigma \mapsto a_1 \ : T' \circ \_$$

$$\text{and } T' = \ \_\,(S' \xrightarrow{\_} \_)$$

if $\mathcal{R}^\tau(T')$

Definition (Erasure function)

If $\Gamma \circ \Sigma \vdash t \; : T \circ \ell$ and $T <: A$, $\mathcal{E}^{\tau}$ is defined by case analysis on t:

$$\begin{aligned}
\mathcal{E}^{\tau}(\Gamma, \Sigma, !a_1, A) ::= {} & ! \left(\mathcal{E}^{\tau}\left(\Gamma, \Sigma, a_1, T'\right)\right) \\
& \text{where } \Gamma \circ \Sigma \vdash a_1 \; : T' \circ _ \\
& \textbf{if } \mathcal{R}^{\tau}\left(T'\right)
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}^{\tau}(\Gamma, \Sigma, \texttt{new}\,[S]\; a_1, A) ::= {} & \texttt{new}\,[\mathcal{E}^{\tau}(S)]\; \left(\mathcal{E}^{\tau}\left(\Gamma, \Sigma, a_1, T'\right)\right) \\
& \text{where } \Gamma \circ \Sigma \vdash a_1 \; : T' \circ _ \\
& \textbf{if } \mathcal{R}^{\tau}(S)
\end{aligned}$$

THE Erasure function (part 4/∞)

Definition (Erasure function)

If $\Gamma \circ \Sigma \vdash t \;: T \circ \ell$ and $T <: A$, $\mathcal{E}^\tau$ is defined by case analysis on t:

$$\mathcal{E}^\tau(\Gamma, \Sigma, a_1 := a_2;\; p_3, A) ::= \begin{cases} (\mathcal{E}^\tau(\Gamma, \Sigma, a_1, T')) := (\mathcal{E}^\tau(\Gamma, \Sigma, a_2, S'));\; (\mathcal{E}^\tau(\Gamma, \Sigma, p_3, A)) \\ \qquad \text{if } \mathcal{R}^\tau(S') \\ \mathcal{E}^\tau(\Gamma, \Sigma, p_3, A) \\ \qquad \text{else} \end{cases}$$

$$\text{where } \Gamma \circ \Sigma \vdash a_1 \;: T' \circ \_$$

$$\text{and } T' = \;\_\; \textsc{Ref}\; S'$$

THE Erasure function (part 5/∞)

Definition (Erasure function)

If $\Gamma \circ \Sigma \vdash t \ : T \circ \ell$ and $T <: A$, $\mathcal{E}^\tau$ is defined by case analysis on t:

$$\mathcal{E}^\tau(\Gamma, \Sigma, \texttt{let}\, x{:}\, S = e_1;\ p_2, A) ::= \begin{cases} \texttt{let}\, x{:}\, \mathcal{E}^\tau(S) = (\mathcal{E}^\tau(\Gamma, \Sigma, e_1, T'));\ p_2' \\ \qquad \text{if } (\mathcal{R}^\tau(T) \wedge x \in \mathrm{FV}(p_2')) \vee \mathcal{R}^\tau(\ell) \\ p_2' \\ \qquad \text{else} \end{cases}$$

$$\text{where } p_2' = \mathcal{E}^\tau(\Gamma + \{x{:}\, S\}, \Sigma, p_2, A)$$

$$\text{and } \Gamma \circ \Sigma \vdash e_1 \ : \ \_ \circ \ell$$

THE Erasure function (part 6/∞)

Definition (Erasure function)

If $\Gamma \circ \Sigma \vdash t \ : T \circ \ell$ and $T <: A$, $\mathcal{E}^{\tau}$ is defined by case analysis on t.

Note that $\mathcal{E}^{\tau}$:

- Is a partial function, but
- Is total on well-typed programs[2].

[2] This is a theorem, not an evidence.

THE Erasure function (part 7/∞)

Definition (Erasure function)

If $\Gamma \circ \Sigma \vdash t \; : T \circ \ell$ and $T <: A$, $\mathcal{E}^{\tau}$ is defined by case analysis on t

The function is also defined for

- Labels;
- Types;
- Variable contexts;
- Store contexts;
- Stores.

A peek at the results

Early results

Theorem (Reduction well-definedness)

Reduction is well-defined, in the sense that it yields terms which are in ANF.

Theorem (Progress)

If $t \circ \mu$ is well-typed (under the empty store context), either t is a value or there exists $t' \circ \mu'$ such that

$$t \circ \mu \longrightarrow t' \circ \mu'$$

Preservation

Theorem (Preservation)

If

$$\Gamma \circ \Sigma \vdash t \;: T \circ \ell \qquad \textit{and} \qquad \Gamma \circ \Sigma \vdash \mu \qquad \textit{and} \qquad t \circ \mu \longrightarrow t' \circ \mu'$$

then for some $\Sigma' \supseteq^{\sigma} \Sigma$

$$\Gamma \circ \Sigma' \vdash t' \;: T \circ \ell \qquad \textit{and} \qquad \Gamma \circ \Sigma' \vdash \mu$$

Preservation

Theorem (Preservation)

If

$$\Gamma \circ \Sigma \vdash t \ : T \circ \ell \qquad \textit{and} \qquad \Gamma \circ \Sigma \vdash \mu \qquad \textit{and} \qquad t \circ \mu \longrightarrow t' \circ \mu'$$

then for some[1] $\Sigma' \supseteq^{\sigma} \Sigma$

$$\Gamma \circ \Sigma' \vdash t' \ : T^{2} \circ \ell^{3} \qquad \textit{and} \qquad \Gamma \circ \Sigma' \vdash \mu$$

1. Need to explicitly refer to Σ' later;
2. Incorrect in the context of algorithmic typing;
3. Incorrect as well, as effects ``disappear'' once done.

Preservation

Theorem (Preservation)

If

$$\Gamma \circ \Sigma \vdash t \; : T \circ \ell \qquad \textit{and} \qquad \Gamma \circ \Sigma \vdash \mu \qquad \textit{and} \qquad t \circ \mu \longrightarrow t' \circ \mu'$$

then for $\Sigma' ::= ext^{\sigma}(\Sigma, t \circ \mu)$

$$\Gamma \circ \Sigma' \vdash t' \; : T^2 \circ \ell^3 \qquad \textit{and} \qquad \Gamma \circ \Sigma' \vdash \mu$$

2. Incorrect in the context of algorithmic typing;
3. Incorrect as well, as effects ``disappear'' once done.

Preservation

Theorem (Preservation)

If

$$\Gamma \circ \Sigma \vdash t \ : T \circ \ell \qquad \textit{and} \qquad \Gamma \circ \Sigma \vdash \mu \qquad \textit{and} \qquad t \circ \mu \longrightarrow t' \circ \mu'$$

then for $\Sigma' ::= ext^{\sigma}(\Sigma, t \circ \mu)$, $T' <: T$

$$\Gamma \circ \Sigma' \vdash t' \ : T' \circ \ell^{3} \qquad \textit{and} \qquad \Gamma \circ \Sigma' \vdash \mu$$

3. Incorrect as well, as effects ``disappear'' once done.

Preservation

Theorem (Preservation)

If

$$\Gamma \circ \Sigma \vdash t : T \circ \ell \qquad \textit{and} \qquad \Gamma \circ \Sigma \vdash \mu \qquad \textit{and} \qquad t \circ \mu \longrightarrow t' \circ \mu'$$

then for $\Sigma' ::= ext^{\sigma}(\Sigma, t \circ \mu)$, $T' <: T$ *and* $\ell \sqsubseteq \ell'$

$$\Gamma \circ \Sigma' \vdash t' : T' \circ \ell' \qquad \textit{and} \qquad \Gamma \circ \Sigma' \vdash \mu$$

Erasure & typing

Theorem

Let $\Gamma \circ \Sigma \vdash t \ : T \circ \ell$ *and* $T <: A$. *Then if* $\mathcal{E}^\tau(\Gamma, \Sigma, t, A)$ *is well-defined*

$$\mathcal{E}^\tau(\Gamma) \circ \mathcal{E}^\tau(\Sigma) \vdash \mathcal{E}^\tau(\Gamma, \Sigma, t, A) \ : T' \circ \ell'$$

where the following properties hold:

1. $\mathcal{E}^\tau(\ell) \sqsubseteq \ell' \sqsubseteq \tau$ *or* $\ell' = \tau$ *or* $\ell' = \top$;
2. $\mathcal{L}_{in}(T') \subseteq \tau{\downarrow} \cup \{\top\}$;
3. *If* $\mathcal{R}^\tau(A)$, *then* $T' <: \mathcal{E}^\tau(T)$;
4. *If* $\neg\mathcal{R}^\tau(A)$, *then* $T' <: {}_\tau\textsc{Top}$.

Erasure & typing

Theorem

Let $\Gamma \circ \Sigma \vdash t \; : T \circ \ell$ *and* $T <: A$*. Then if* $\mathcal{E}^\tau(\Gamma, \Sigma, t, A)$ *is well-defined*

$$\mathcal{E}^\tau(\Gamma) \circ \mathcal{E}^\tau(\Sigma) \vdash \mathcal{E}^\tau(\Gamma, \Sigma, t, A) \; : T' \circ \ell'$$

Corollary (Erasure well-typedness)

A well-typed term stays well-typed after erasure.

Erasure & typing

Theorem

Let $\Gamma \circ \Sigma \vdash t \;: T \circ \ell$ and $T <: A$. Then if $\mathcal{E}^\tau(\Gamma, \Sigma, t, A)$ is well-defined

$$\mathcal{E}^\tau(\Gamma) \circ \mathcal{E}^\tau(\Sigma) \vdash \mathcal{E}^\tau(\Gamma, \Sigma, t, A) \;: T' \circ \ell'$$

where the following properties hold:

1. $\mathcal{E}^\tau(\ell) \sqsubseteq \ell' \sqsubseteq \tau$ or $\ell' = \tau$ or $\ell' = \top$;
3. *If* $\mathcal{R}^\tau(A)$, *then* $T' <: \mathcal{E}^\tau(T)$;
4. *If* $\neg\mathcal{R}^\tau(A)$, *then* $T' <: {}_\tau\textsc{Top}$.

Corollary (Erasure relevance)

A well-typed erased term has a relevant type and effect.

Erasure & typing

Theorem

Let $\Gamma \circ \Sigma \vdash t \ : T \circ \ell$ *and* $T <: A$*. Then if* $\mathcal{E}^\tau(\Gamma, \Sigma, t, A)$ *is well-defined*

$$\mathcal{E}^\tau(\Gamma) \circ \mathcal{E}^\tau(\Sigma) \vdash \mathcal{E}^\tau(\Gamma, \Sigma, t, A) \ : T' \circ \ell'$$

where the following properties hold:

1. $\mathcal{E}^\tau(\ell) \sqsubseteq \ell' \sqsubseteq \tau$ *or* $\ell' = \tau$ *or* $\ell' = \top$;
2. $\mathcal{L}_{in}(T') \subseteq \tau{\downarrow} \cup \{\top\}$;

Corollary (Erasure projection)

A well-typed erased term lives in a subcalculus where the labels are restricted to

$$\{\ell \in \mathcal{L} \mid \ell \sqsubseteq \tau\} \cup \{\top\}$$

Erasure & reduction

Theorem (Erasure (proto) forward simulation)

If

$$\Gamma \circ \Sigma \vdash t \; : T \circ \ell \qquad \text{and} \qquad \Gamma \circ \Sigma \vdash \mu \qquad \text{and} \qquad T <: A$$

$$\mathcal{E}^{\tau}(\Gamma, \Sigma, t, A) \text{ is well-defined} \qquad \text{and} \qquad t \circ \mu \longrightarrow t' \circ \mu'$$

then

$$\mathcal{E}^{\tau}(\Gamma, \Sigma, t, A) \circ \mathcal{E}^{\tau}(\Gamma, \Sigma, \mu) \overset{=}{\longrightarrow} \mathcal{E}^{\tau}\left(\Gamma, \Sigma', t', A'\right) \circ \mathcal{E}^{\tau}\left(\Gamma, \Sigma', \mu'\right)$$

where $\Sigma' ::= ext^{\sigma}(\Sigma, t \circ \mu)$ *and* A' *is some type satisfying* $T <: A' <: A$.

Additionally,

- *If* t *is a program,* $A' = A$;
- *If* t *is an expression, the relation doesn't hold by reflexivity.*

Erasure & reduction

Corollary (Erasure forward simulation)

If

$$\Gamma \circ \Sigma \vdash p \;: T \circ \ell \qquad \text{and} \qquad \Gamma \circ \Sigma \vdash \mu \qquad \text{and} \qquad T <: A$$
$$\mathcal{E}^\tau(\Gamma, \Sigma, p, A) \text{ is well-defined} \qquad \text{and} \qquad p \circ \mu \longrightarrow p' \circ \mu'$$

then

$$\mathcal{E}^\tau(\Gamma, \Sigma, p, A) \circ \mathcal{E}^\tau(\Gamma, \Sigma, \mu) \overset{=}{\longrightarrow} \mathcal{E}^\tau(\Gamma, \Sigma', p', A) \circ \mathcal{E}^\tau(\Gamma, \Sigma', \mu')$$

where $\Sigma' ::= ext^\sigma(\Sigma, p \circ \mu)$.

Corollary (Erasure forward simulation)

If

$$\Gamma \circ \Sigma \vdash p \ : T \circ \ell \qquad \textit{and} \qquad \Gamma \circ \Sigma \vdash \mu \qquad \textit{and} \qquad T <: A$$
$$\mathcal{E}^{\tau}(\Gamma, \Sigma, p, A) \textit{ is well-defined} \qquad \textit{and} \qquad p \circ \mu \longrightarrow p' \circ \mu'$$

then

$$\mathcal{E}^{\tau}(\Gamma, \Sigma, p, A) \circ \mathcal{E}^{\tau}(\Gamma, \Sigma, \mu) \xrightarrow{=} \mathcal{E}^{\tau}(\Gamma, \Sigma', p', A) \circ \mathcal{E}^{\tau}(\Gamma, \Sigma', \mu')$$

where $\Sigma' ::= ext^{\sigma}(\Sigma, p \circ \mu)$.

Informal corollary (Erased computation)

p and its erasure ``compute the same thing''.

Going forward

Going forward

Implement this type system:

- Testing of potential improvements:
  - Erasure of other constructs;
  - More ``aggressive'' erasure;
  - ...
- Creation of more concrete examples of ghost code for testing.


My job here is done.

Overtime!

Encoding trick --- pairs

$$\frac{\ell \sqsubseteq h \quad T_1 <: S_1 \quad T_2 <: S_2}{{}_{\ell}(T_1, T_2) <: {}_{h}(S_1, S_2)}\text{AS-Pair}$$

$$T = {}_{\ell}({}_{m_1}P_1, {}_{m_2}P_2) \implies (\ell \sqsubseteq m_1) \wedge (\ell \sqsubseteq m_2)$$

$$\frac{\Gamma \circ \Sigma \vdash a_1 \;: T_1 \circ \top \quad \Gamma \circ \Sigma \vdash a_2 \;: T_2 \circ \top}{\Gamma \circ \Sigma \vdash (a_1, a_2) \;: {}_{\perp}(T_1, T_2) \circ \top}\text{AT-Pair}$$

$$\mathcal{E}^{\tau}(\Gamma, \Sigma, (a_1, a_2), A) ::= \begin{cases} (\mathcal{E}^{\tau}(\Gamma, \Sigma, a_1, T_1), \mathcal{E}^{\tau}(\Gamma, \Sigma, a_2, T_2)) & \text{if } \mathcal{R}^{\tau}(A) \\ () & \text{else} \end{cases}$$

Better pairs (?)

$$\mathcal{E}^\tau(\Gamma, \Sigma, (a_1, a_2), A) ::= \begin{cases} () & \text{if } \neg\mathcal{R}^\tau(A) \vee (\neg\mathcal{R}^\tau(T_1) \wedge \neg\mathcal{R}^\tau(T_2)) \\ a_1 & \text{if } \mathcal{R}^\tau(A) \wedge \mathcal{R}^\tau(T_1) \wedge \neg\mathcal{R}^\tau(T_2) \\ a_2 & \text{if } \mathcal{R}^\tau(A) \wedge \neg\mathcal{R}^\tau(T_1) \wedge \mathcal{R}^\tau(T_2) \\ \left(\mathcal{E}^\tau(\Gamma, \Sigma, a_1, T_1), \mathcal{E}^\tau(\Gamma, \Sigma, a_2, T_2)\right) & \text{else} \end{cases}$$

Better references (?)

$$\Sigma : l \to \ell \circ T$$

$$\frac{\ell \sqsubseteq h}{{}_{\ell}\textsc{Ref}\ T <: {}_{h}\textsc{Ref}\ T}\ \text{AS-Ref}$$

$$T = {}_{\ell}\textsc{Ref}\ {}_{m}P \implies \ell \sqsubseteq m$$

$$\frac{\Sigma(l) = \ell \circ T}{\Gamma \circ \Sigma \vdash l\ :\ {}_{\ell}\textsc{Ref}\ T \circ \top}\ \text{AT-Loc} \qquad \frac{\Gamma \circ \Sigma \vdash a\ : S' \circ \top \quad S' <: S}{\Gamma \circ \Sigma \vdash \texttt{new}\,[\rho, S]\ a\ :\ {}_{\rho}\textsc{Ref}\ S \circ \rho}\ \text{AT-New}$$

$$\frac{\Gamma \circ \Sigma \vdash a_1\ :\ {}_{\rho}\textsc{Ref}\ S \circ \top \quad \Gamma \circ \Sigma \vdash a_2\ : S' \circ \top \quad S' <: S \quad \Gamma \circ \Sigma \vdash p\ : T \circ n}{\Gamma \circ \Sigma \vdash a_1 := a_2;\ p\ : T \circ \rho \sqcap n}\ \text{AT-AssSeq}$$

~~Fixpoint combinator~~ Landin's knot

```
λf: (T → T) → (T → T). {
    let r: Ref (T → T) = new[T → T] (λx: T. x);
    r := f( λx: T. {
        let self: T → T = !r
        self x
    } )
    !r
}
```

~~Landin's~~ Noé's knot

```
λf: (Unit → T) → T. λdummy: T. {
    let r: Ref T = new[T] dummy;
    let a: Unit → T = λ_: Unit. !r
    r := f(a)
    !r
}
```