Coma: an intermediate verification language with explicit abstraction barriers

joint work with Paul Patault and Jean-Christophe Filliâtre

LMF, Université Paris-Saclay • Toccata, Inria Saclay

IFIP WG 1.9/2.15 • July 2, 2025

piecewise specification

```
type tree = Node tree elt tree
          | Empty

let removeRoot (t: tree): tree
```

```
= match t with
  | Node l _ r ⟶ mergeTree l r
  | Empty ⟶ fail
```

piecewise specification

```
type tree = Node tree elt tree
          | Empty

let removeRoot (t: tree): tree
  requires  t ≠ Empty
  ensures   match t with
            | Node l _ r → ∀e. e ∈ result ↔ e ∈ l ∨ e ∈ r
            | Empty → false
= match t with
  | Node l _ r → mergeTree l r
  | Empty → fail
```

piecewise specification

```
type tree = Node tree elt tree
          | Empty

removeRoot (Node l _ r)
  ensures ∀e. e ∈ result ↔ e ∈ l ∨ e ∈ r
= mergeTree l r

removeRoot Empty = fail
```

key ideas

1. Mix specification with the code
   - in-code assertions assume the role of preconditions and postconditions
   - traditional contracts in the surface language are automatically translated

key ideas

1. Mix specification with the code
   - in-code assertions assume the role of preconditions and postconditions
   - traditional contracts in the surface language are automatically translated
2. Explicit abstraction barriers separate interface from implementation
   - instructions and assertions above the barrier are verified on every call
   - everything below the barrier is verified once on the definition site
   - surface syntax: attaching a contract to a code block introduces a barrier

key ideas

1. Mix specification with the code
   - in-code assertions assume the role of preconditions and postconditions
   - traditional contracts in the surface language are automatically translated
2. Explicit abstraction barriers separate interface from implementation
   - instructions and assertions above the barrier are verified on every call
   - everything below the barrier is verified once on the definition site
   - surface syntax: attaching a contract to a code block introduces a barrier

```
let wrapExit (r:int)
  requires 0 ≤ r < 256
= Unix.exit r
```

```
let wrapExit (r:int)
= assert 0 ≤ r < 256;
  hide Unix.exit r
```

key ideas

1. Mix specification with the code
   - in-code assertions assume the role of preconditions and postconditions
   - traditional contracts in the surface language are automatically translated
2. Explicit abstraction barriers separate interface from implementation
   - instructions and assertions above the barrier are verified on every call
   - everything below the barrier is verified once on the definition site
   - surface syntax: attaching a contract to a code block introduces a barrier

On call: $\lambda r.\, 0 \leq r < 256$

On def: $\forall r.\, 0 \leq r < 256 \longrightarrow \mathcal{P}re(\texttt{Unix.exit})\ r$

```
let wrapExit (r:int)
= assert 0 ≤ r < 256;
  hide Unix.exit r
```

key ideas

1. Mix specification with the code
   - in-code assertions assume the role of preconditions and postconditions
   - traditional contracts in the surface language are automatically translated
2. Explicit abstraction barriers separate interface from implementation
   - instructions and assertions above the barrier are verified on every call
   - everything below the barrier is verified once on the definition site
   - surface syntax: attaching a contract to a code block introduces a barrier

```
let triple (x:int): int
  ensures result = 3·x
= x + x + x
```

```
let triple (x:int) (ret:int→⊥)
= let out y = assert y = 3·x;
              hide ret y
  in hide out (x + x + x)
```

key ideas

1. Mix specification with the code
   - in-code assertions assume the role of preconditions and postconditions
   - traditional contracts in the surface language are automatically translated

2. Explicit abstraction barriers separate interface from implementation
   - instructions and assertions above the barrier are verified on every call
   - everything below the barrier is verified once on the definition site
   - surface syntax: attaching a contract to a code block introduces a barrier

On call: $\forall y.\, y = 3 \cdot x \longrightarrow \mathcal{P}re(\texttt{ret})\ y$

On def: $\forall x.\, x + x + x = 3 \cdot x$

```
let triple (x:int) (ret:int→⊥)
= let out y = assert y = 3·x;
              hide ret y
  in hide out (x + x + x)
```

the code is the contract

```
let removeRoot (t: tree) (ret: tree → ⊥) = match t with
  | Node l _ r →
      let out s = assert ∀e. e ∈ s ↔ e ∈ l ∨ e ∈ r;
                  hide ret s
      in hide out (mergeTree l r)
  | Empty → fail
```

the code is the contract

```
let removeRoot (t: tree) (ret: tree → ⊥) = match t with
  | Node l _ r →
      let out s = assert ∀e. e ∈ s ↔ e ∈ l ∨ e ∈ r;
                  hide ret s
      in hide out (mergeTree l r)
  | Empty → fail
```

On call: `match t with` for given `t` and `ret`

```
        | Node l _ r →
            ∀s. (∀e. e ∈ s ↔ e ∈ l ∨ e ∈ r) → 𝒫re(ret) s
        | Empty → ⊥
```

the code is the contract

```
let removeRoot (t: tree) (ret: tree → ⊥) = match t with
  | Node l _ r →
      let out s = assert ∀e. e ∈ s ↔ e ∈ l ∨ e ∈ r;
                  hide ret s
      in hide out (mergeTree l r)
  | Empty → fail
```

On call: `match t with` — for given `t` and `ret`

```
  | Node l _ r →
      ∀s. (∀e. e ∈ s ↔ e ∈ l ∨ e ∈ r) → 𝒫re(ret) s
  | Empty → ⊥
```

On def: `match t with` — for all possible `t`

```
  | Node l _ r →
      ∀e. e ∈ mergeTree l r ↔ e ∈ l ∨ e ∈ r
  | Empty → ⊤
```

x, y, z — variable

$s, t \quad ::= \quad x \mid 0 \ldots \mid s + t \ldots$ — data term

data / code

h, g, f — handler symbol

$k, o \quad ::= \quad h \mid \bar{x}\ \bar{g} \to d$ — (un)named handler

$e, d \quad ::= \quad k\ \bar{s}\ \bar{o}$ — handler invocation

$\mid \quad e\ \texttt{where}\ h\ \bar{x}\ \bar{g} = d$ — handler definition

COMA code is written in multibarrel continuation-passing style

- handlers return control by calling their handler parameters
- can express conditionals, loops, function calls, exceptions

x, y, z			variable
s, t	$::=$	$x \mid 0 \ldots \mid s + t \ldots$	data term
φ, ψ	$::=$	$s > t \ldots \mid \varphi \wedge \psi \ldots$	property

data

---

code

h, g, f			handler symbol
k, o	$::=$	$h \mid \bar{x}\ \bar{g} \to d$	(un)named handler
e, d	$::=$	$k\ \bar{s}\ \bar{o}$	handler invocation
	$\mid$	e **where** $h\ \bar{x}\ \bar{g} = d$	handler definition
	$\mid$	**assert** φ ; e	blocking assertion
	$\mid$	**hide** e	abstraction barrier

$$\mathcal{C}(h) \triangleq$$

$$\mathcal{C}(\bar{x}\ \bar{g} \to d) \triangleq$$

$$\mathcal{C}(k\ \bar{s}\ \bar{o}) \triangleq$$

$$\mathcal{C}(e\ \textbf{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq$$

$$\mathcal{C}(\textbf{assert}\ \varphi\ ;\ e) \triangleq$$

$$\mathcal{C}(\textbf{hide}\ e) \triangleq$$

$$\mathcal{C}(h) \triangleq h$$

$$\mathcal{C}(\bar{x}\ \bar{g} \to d) \triangleq$$

$$\mathcal{C}(k\ \bar{s}\ \bar{o}) \triangleq$$

$$\mathcal{C}(e\ \texttt{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq$$

VC generation maps continuations to propositions:

- handler symbols become predicate variables

$$\mathcal{C}(\texttt{floor} : \texttt{real} \to (\texttt{int} \to \bot) \to \bot) =$$

$$\texttt{floor} : \texttt{real} \to (\texttt{int} \to \text{Prop}) \to \text{Prop}$$

$$\mathcal{C}(h) \triangleq h$$

$$\mathcal{C}(\bar{x}\ \bar{g} \to d) \triangleq$$

$$\mathcal{C}(k\ \bar{s}\ \bar{o}) \triangleq$$

$$\mathcal{C}(e \texttt{ where } h\ \bar{x}\ \bar{g} = d) \triangleq$$

VC generation maps continuations to propositions:

- handler symbols become predicate variables

  $\mathcal{C}(\texttt{floor} : \texttt{real} \to (\texttt{int} \to \bot) \to \bot) =$

  $\texttt{floor} : \texttt{real} \to (\texttt{int} \to \text{Prop}) \to \text{Prop}$

- predicate variables carry the specification of handlers

  $\mathcal{C}(\texttt{if}) = \texttt{if} \triangleq \lambda \texttt{cfg}.(\texttt{c} \longrightarrow \texttt{f}) \wedge (\neg \texttt{c} \longrightarrow \texttt{g})$

$$\mathcal{C}(h) \triangleq h$$

$$\mathcal{C}(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.\mathcal{C}(d)$$

$$\mathcal{C}(k\ \bar{s}\ \bar{o}) \triangleq$$

$$\mathcal{C}(e \texttt{ where } h\ \bar{x}\ \bar{g} = d) \triangleq$$

VC generation maps continuations to propositions:

- handler symbols become predicate variables

  $\mathcal{C}(\texttt{floor} : \texttt{real} \to (\texttt{int} \to \bot) \to \bot) =$

  $\texttt{floor} : \texttt{real} \to (\texttt{int} \to \text{Prop}) \to \text{Prop}$

- predicate variables carry the specification of handlers

  $\mathcal{C}(\texttt{if}) = \texttt{if} \triangleq \lambda \texttt{cfg}.(\texttt{c} \longrightarrow \texttt{f}) \wedge (\neg \texttt{c} \longrightarrow \texttt{g})$

$$\mathcal{C}(h) \triangleq h$$

$$\mathcal{C}(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.\mathcal{C}(d)$$

$$\mathcal{C}(k\ \bar{s}\ \bar{o}) \triangleq \mathcal{C}(k)\ \ \bar{s}\ \ \mathcal{C}(o_1)\ \cdots\ \mathcal{C}(o_n)$$

$$\mathcal{C}(e \texttt{ where } h\ \bar{x}\ \bar{g} = d) \triangleq$$

VC generation maps continuations to propositions:

- handler symbols become predicate variables

  $\mathcal{C}(\texttt{floor} : \texttt{real} \to (\texttt{int} \to \bot) \to \bot) =$

  $\texttt{floor} : \texttt{real} \to (\texttt{int} \to \text{Prop}) \to \text{Prop}$

- predicate variables carry the specification of handlers

  $\mathcal{C}(\texttt{if}) = \texttt{if} \triangleq \lambda \texttt{cfg}.(\texttt{c} \longrightarrow \texttt{f}) \wedge (\neg \texttt{c} \longrightarrow \texttt{g})$

$$\mathcal{C}(h) \triangleq h$$

$$\mathcal{C}(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.\mathcal{C}(d)$$

$$\mathcal{C}(k\ \bar{s}\ \bar{o}) \triangleq \mathcal{C}(k)\ \ \bar{s}\ \ \mathcal{C}(o_1)\ \cdots\ \mathcal{C}(o_n)$$

$$\mathcal{C}(e\ \mathbf{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \mathbf{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \mathbf{in}\ \mathcal{C}(e) \wedge \forall \bar{x} \bar{g}.\mathcal{B}(d)$$

VC generation maps continuations to propositions:

- handler symbols become predicate variables

  $\mathcal{C}(\texttt{floor} : \texttt{real} \to (\texttt{int} \to \bot) \to \bot) =$

  $\texttt{floor} : \texttt{real} \to (\texttt{int} \to \text{Prop}) \to \text{Prop}$

- predicate variables carry the specification of handlers

  $\mathcal{C}(\texttt{if}) = \texttt{if} \triangleq \lambda \texttt{cfg}.(\texttt{c} \longrightarrow \texttt{f}) \wedge (\neg \texttt{c} \longrightarrow \texttt{g})$

$$\mathcal{C}(h) \triangleq h$$

$$\mathcal{C}(\bar{x}\ \bar{g} \twoheadrightarrow d) \triangleq \lambda \bar{x} \bar{g}.\mathcal{C}(d)$$

$$\mathcal{C}(k\ \bar{s}\ \bar{o}) \triangleq \mathcal{C}(k)\ \ \bar{s}\ \ \mathcal{C}(o_1)\ \cdots\ \mathcal{C}(o_n)$$

$$\mathcal{C}(e\ \mathbf{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \mathbf{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \mathbf{in}\ \mathcal{C}(e) \wedge \forall \bar{x} \bar{g}.\mathcal{B}(d)$$

$$\mathcal{C}(\mathbf{assert}\ \varphi\ ;\ e) \triangleq \varphi\ \&\ \mathcal{C}(e)$$

neutralizable conjunction

$$\mathcal{C}(h) \triangleq h$$

$$\mathcal{C}(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.\mathcal{C}(d)$$

$$\mathcal{C}(k\ \bar{s}\ \bar{o}) \triangleq \mathcal{C}(k)\ \ \bar{s}\ \ \mathcal{C}(o_1)\ \cdots\ \mathcal{C}(o_n)$$

$$\mathcal{C}(e\ \mathbf{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \mathbf{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \mathbf{in}\ \mathcal{C}(e) \wedge \forall \bar{x} \bar{g}.\mathcal{B}(d)$$

$$\mathcal{C}(\mathbf{assert}\ \varphi\ ;\ e) \triangleq \varphi\ \&\ \mathcal{C}(e)$$ neutralizable conjunction

$$\mathcal{C}(\mathbf{hide}\ e) \triangleq \mathcal{C}(e)$$ only relevant for $\mathcal{A}$ and $\mathcal{B}$

$$\mathcal{C}(h) \triangleq h$$

$$\mathcal{C}(\bar{x}\,\bar{g} \to d) \triangleq \lambda \bar{x}\bar{g}.\mathcal{C}(d)$$

$$\mathcal{C}(k\,\bar{s}\,\bar{o}) \triangleq \mathcal{C}(k)\ \ \bar{s}\ \ \mathcal{C}(o_1)\ \cdots\ \mathcal{C}(o_n)$$

$$\mathcal{C}(e\ \textbf{where}\ h\,\bar{x}\,\bar{g} = d) \triangleq \textbf{let}\ h\,\bar{x}\,\bar{g} = \mathcal{A}(d)\ \textbf{in}\ \mathcal{C}(e) \wedge \forall \bar{x}\bar{g}.\mathcal{B}(d)$$

$$\mathcal{C}(\textbf{assert}\ \varphi\ ;\ e) \triangleq \varphi\ \&\ \mathcal{C}(e)$$ neutralizable conjunction

$$\mathcal{C}(\textbf{hide}\ e) \triangleq \mathcal{C}(e)$$ only relevant for $\mathcal{A}$ and $\mathcal{B}$

$$\texttt{halt} \triangleq \top$$

$$\texttt{fail} \triangleq \bot\ \&\ \top = \mathcal{C}(\texttt{assert}\ \bot\texttt{; halt})$$

$$\texttt{if} \triangleq \lambda \texttt{cfg}.(\texttt{c} \longrightarrow \texttt{f}) \wedge (\neg \texttt{c} \longrightarrow \texttt{g})$$

$$\texttt{unTree} \triangleq \lambda \texttt{tfg}.(\forall \texttt{lvr}.\texttt{t} = \texttt{Node l v r} \longrightarrow \texttt{f l v r}) \wedge (\texttt{t} = \texttt{Empty} \longrightarrow \texttt{g})$$

$$
\begin{aligned}
C(h) &\triangleq h \\
C(\bar{x}\,\bar{g} \to d) &\triangleq \lambda \bar{x}\bar{g}.C(d) \\
C(k\,\bar{s}\,\bar{o}) &\triangleq C(k)\ \ \bar{s}\ \ C(o_1)\ \cdots\ C(o_n) \\
C(e\ \texttt{where}\ h\,\bar{x}\,\bar{g} = d) &\triangleq \texttt{let}\ h\,\bar{x}\,\bar{g} = \mathcal{A}(d)\ \texttt{in}\ C(e) \wedge \forall \bar{x}\bar{g}.\mathcal{B}(d) \\
C(\texttt{assert}\ \varphi\ ;\ e) &\triangleq \varphi\ \&\ C(e) \\
C(\texttt{hide}\ e) &\triangleq C(e)
\end{aligned}
$$

verify above barrier

---

verify below barrier

$$
\begin{aligned}
C(h) &\triangleq h \\
C(\bar{x}\,\bar{g} \to d) &\triangleq \lambda \bar{x}\bar{g}.C(d) \\
C(k\,\bar{s}\,\bar{o}) &\triangleq C(k)\ \ \bar{s}\ \ C(o_1)\ \cdots\ C(o_n) \\
C(e\ \texttt{where}\ h\,\bar{x}\,\bar{g} = d) &\triangleq \texttt{let}\ h\,\bar{x}\,\bar{g} = \mathcal{A}(d)\ \texttt{in}\ C(e) \wedge \forall \bar{x}\bar{g}.\mathcal{B}(d) \\
C(\texttt{assert}\ \varphi\ ;\ e) &\triangleq \varphi\ \&\ C(e) \\
C(\texttt{hide}\ e) &\triangleq C(e)
\end{aligned}
$$

$$C(h) \triangleq h$$

$$C(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.C(d)$$

$$C(k\ \bar{s}\ \bar{o}) \triangleq C(k)\ \ \bar{s}\ \ C(o_1)\ \cdots\ C(o_n)$$

$$C(e\ \texttt{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \texttt{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \texttt{in}\ C(e) \wedge \forall \bar{x} \bar{g}.\mathcal{B}(d)$$

$$C(\texttt{assert}\ \varphi\ ;\ e) \triangleq \varphi\ \&\ C(e)$$

$$\mathcal{A}(\texttt{hide}\ e) \triangleq \top$$

verify above barrier

---

verify below barrier

$$C(h) \triangleq h$$

$$C(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.C(d)$$

$$C(k\ \bar{s}\ \bar{o}) \triangleq C(k)\ \ \bar{s}\ \ C(o_1)\ \cdots\ C(o_n)$$

$$C(e\ \texttt{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \texttt{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \texttt{in}\ C(e) \wedge \forall \bar{x} \bar{g}.\mathcal{B}(d)$$

$$C(\texttt{assert}\ \varphi\ ;\ e) \triangleq \varphi\ \&\ C(e)$$

$$\mathcal{B}(\texttt{hide}\ e) \triangleq C(e)$$

$$C(h) \triangleq h$$

$$C(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.C(d)$$

$$C(k\ \bar{s}\ \bar{o}) \triangleq C(k)\ \ \bar{s}\ \ C(o_1)\ \cdots\ C(o_n)$$

$$C(e\ \texttt{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \texttt{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \texttt{in}\ C(e) \wedge \forall \bar{x} \bar{g}.\mathcal{B}(d)$$

$$\mathcal{A}(\textbf{assert}\ \varphi\ ;\ e) \triangleq \varphi\ \&\ \mathcal{A}(e)$$

$$\mathcal{A}(\textbf{hide}\ e) \triangleq \top$$

verify above barrier

---

verify below barrier

$$C(h) \triangleq h$$

$$C(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.C(d)$$

$$C(k\ \bar{s}\ \bar{o}) \triangleq C(k)\ \ \bar{s}\ \ C(o_1)\ \cdots\ C(o_n)$$

$$C(e\ \texttt{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \texttt{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \texttt{in}\ C(e) \wedge \forall \bar{x} \bar{g}.\mathcal{B}(d)$$

$$\mathcal{B}(\textbf{assert}\ \varphi\ ;\ e) \triangleq \varphi \to \mathcal{B}(e)$$

$$\mathcal{B}(\textbf{hide}\ e) \triangleq C(e)$$

$$C(h) \triangleq h$$

$$C(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.C(d)$$

$$C(k\ \bar{s}\ \bar{o}) \triangleq C(k)\ \ \bar{s}\ \ C(o_1)\ \cdots\ C(o_n)$$

$$\mathcal{A}(e\ \textbf{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \textbf{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \textbf{in}\ \mathcal{A}(e) \wedge \forall \bar{x} \bar{g}.\mathcal{B}(d)$$

$$\mathcal{A}(\textbf{assert}\ \varphi\ ;\ e) \triangleq \varphi\ \&\ \mathcal{A}(e)$$

$$\mathcal{A}(\textbf{hide}\ e) \triangleq \top$$

verify above barrier

---

verify below barrier

$$C(h) \triangleq h$$

$$C(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.C(d)$$

$$C(k\ \bar{s}\ \bar{o}) \triangleq C(k)\ \ \bar{s}\ \ C(o_1)\ \cdots\ C(o_n)$$

$$\mathcal{B}(e\ \textbf{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \textbf{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \textbf{in}\ \mathcal{B}(e)$$

$$\mathcal{B}(\textbf{assert}\ \varphi\ ;\ e) \triangleq \varphi \to \mathcal{B}(e)$$

$$\mathcal{B}(\textbf{hide}\ e) \triangleq C(e)$$

$$C(h) \triangleq h$$

$$C(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.C(d)$$

$$\mathcal{A}(k\ \bar{s}\ \bar{o}) \triangleq \mathcal{A}(k)\ \ \bar{s}\ \ \mathcal{A}(o_1)\ \cdots\ \mathcal{A}(o_n)$$

$$\mathcal{A}(e\ \textbf{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \textbf{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \textbf{in}\ \mathcal{A}(e) \wedge \forall \bar{x} \bar{g}.\mathcal{B}(d)$$

$$\mathcal{A}(\textbf{assert}\ \varphi\ ;\ e) \triangleq \varphi\ \&\ \mathcal{A}(e)$$

$$\mathcal{A}(\textbf{hide}\ e) \triangleq \top$$

verify above barrier

---

verify below barrier

$$C(h) \triangleq h$$

$$C(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.C(d)$$

$$\mathcal{B}(k\ \bar{s}\ \bar{o}) \triangleq \mathcal{B}(k)\ \ \bar{s}\ \ \mathcal{B}(o_1)\ \cdots\ \mathcal{B}(o_n)$$

$$\mathcal{B}(e\ \textbf{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \textbf{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \textbf{in}\ \mathcal{B}(e)$$

$$\mathcal{B}(\textbf{assert}\ \varphi\ ;\ e) \triangleq \varphi \to \mathcal{B}(e)$$

$$\mathcal{B}(\textbf{hide}\ e) \triangleq C(e)$$

$$\mathcal{A}(h) \triangleq h$$

$$C(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.C(d)$$

$$\mathcal{A}(k\ \bar{s}\ \bar{o}) \triangleq \mathcal{A}(k)\ \ \bar{s}\ \ \mathcal{A}(o_1)\ \cdots\ \mathcal{A}(o_n)$$

$$\mathcal{A}(e\ \mathbf{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \mathbf{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \mathbf{in}\ \mathcal{A}(e) \wedge \forall \bar{x} \bar{g}.\mathcal{B}(d)$$

$$\mathcal{A}(\mathbf{assert}\ \varphi\ ;\ e) \triangleq \varphi\ \&\ \mathcal{A}(e)$$

$$\mathcal{A}(\mathbf{hide}\ e) \triangleq \top$$

verify above barrier

---

verify below barrier

$$\mathcal{B}(h) \triangleq \natural h$$

under $\natural$, every & turns into $\longrightarrow$

$$C(\bar{x}\ \bar{g} \to d) \triangleq \lambda \bar{x} \bar{g}.C(d)$$

$$\mathcal{B}(k\ \bar{s}\ \bar{o}) \triangleq \mathcal{B}(k)\ \ \bar{s}\ \ \mathcal{B}(o_1)\ \cdots\ \mathcal{B}(o_n)$$

$$\mathcal{B}(e\ \mathbf{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \mathbf{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \mathbf{in}\ \mathcal{B}(e)$$

$$\mathcal{B}(\mathbf{assert}\ \varphi\ ;\ e) \triangleq \varphi \longrightarrow \mathcal{B}(e)$$

$$\mathcal{B}(\mathbf{hide}\ e) \triangleq C(e)$$

$$\mathcal{A}(h) \triangleq h$$

$$\mathcal{A}(\bar{x}\ \bar{g} \to d) \triangleq (\lambda \bar{x}\bar{g}.\mathcal{A}(d)) \wedge \natural(\lambda \bar{x}\bar{g}.\mathcal{B}(d))$$

$$\mathcal{A}(k\ \bar{s}\ \bar{o}) \triangleq \mathcal{A}(k)\ \ \bar{s}\ \ \mathcal{A}(o_1)\ \cdots\ \mathcal{A}(o_n)$$

$$\mathcal{A}(e\ \mathbf{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \mathbf{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \mathbf{in}\ \mathcal{A}(e) \wedge \forall \bar{x}\bar{g}.\mathcal{B}(d)$$

$$\mathcal{A}(\mathbf{assert}\ \varphi\ ;\ e) \triangleq \varphi\ \&\ \mathcal{A}(e)$$

$$\mathcal{A}(\mathbf{hide}\ e) \triangleq \top$$

verify above barrier

---

verify below barrier

$$\mathcal{B}(h) \triangleq \natural h$$

under $\natural$, every & turns into $\longrightarrow$

$$\mathcal{B}(\bar{x}\ \bar{g} \to d) \triangleq (\lambda \bar{x}\bar{g}.\mathcal{B}(d)) \wedge \natural(\lambda \bar{x}\bar{g}.\mathcal{A}(d))$$

$$\mathcal{B}(k\ \bar{s}\ \bar{o}) \triangleq \mathcal{B}(k)\ \ \bar{s}\ \ \mathcal{B}(o_1)\ \cdots\ \mathcal{B}(o_n)$$

$$\mathcal{B}(e\ \mathbf{where}\ h\ \bar{x}\ \bar{g} = d) \triangleq \mathbf{let}\ h\ \bar{x}\ \bar{g} = \mathcal{A}(d)\ \mathbf{in}\ \mathcal{B}(e)$$

$$\mathcal{B}(\mathbf{assert}\ \varphi\ ;\ e) \triangleq \varphi \longrightarrow \mathcal{B}(e)$$

$$\mathcal{B}(\mathbf{hide}\ e) \triangleq C(e)$$

Fully applied VCs reduce to first-order formulas:

- $\forall h.\Phi \triangleq \mathbf{let}\ h\ \bar{x}\ \bar{f} = \bot\ \&\ \bigwedge_f \forall \bar{z}\bar{g}.\, f\ \bar{z}\ \bar{g}\ \mathbf{in}\ \Phi$
- β-reduction eliminates bound predicate variables
- non-neutralized conjunctions $\Phi\ \&\ \Psi$ become $\Phi \wedge \Psi$

Fully applied VCs reduce to first-order formulas:

- $\forall h.\Phi \triangleq \mathtt{let}\ h\ \bar{x}\ \bar{f} = \bot\ \&\ \bigwedge_f \forall \bar{z}\bar{g}.\, f\ \bar{z}\ \bar{g}\ \mathtt{in}\ \Phi$
- β-reduction eliminates bound predicate variables
- non-neutralized conjunctions $\Phi\ \&\ \Psi$ become $\Phi \wedge \Psi$
- on-the-fly factorization of selected predicate applications:
  - no factorized subgoals $\approx$ traditional weakest preconditions
  - factorize all eligible subgoals $\approx$ compact VCs à la Flanagan-Saxe

Fully applied VCs reduce to first-order formulas:

- $\forall h.\Phi \triangleq \mathtt{let}\ h\ \bar{x}\ \bar{f} = \bot\ \&\ \bigwedge_f \forall \bar{z}\bar{g}.\, f\ \bar{z}\ \bar{g}\ \mathtt{in}\ \Phi$
- β-reduction eliminates bound predicate variables
- non-neutralized conjunctions $\Phi\ \&\ \Psi$ become $\Phi \wedge \Psi$
- on-the-fly factorization of selected predicate applications:
  - no factorized subgoals $\approx$ traditional weakest preconditions
  - factorize all eligible subgoals $\approx$ compact VCs à la Flanagan-Saxe

Curious logical properties:

$$\natural\Phi \equiv \top \text{ when } \Phi : \mathrm{Prop}$$

$$\Phi\Psi \equiv \Phi(\natural\Psi) \wedge (\natural\Phi)\Psi$$

$$\Upsilon(\Phi \wedge \Psi) \equiv \Upsilon\Phi \wedge \Upsilon\Psi \text{ when } \natural\Phi \equiv \natural\Psi$$

$$C(e) \equiv \mathcal{A}(e) \wedge \mathcal{B}(e) \qquad \text{and more...}$$

takeaways

1. COMA decouples the abstraction barrier from the entry-exit boundary
   - code above the barrier becomes part of the function's specification
   - functions without barriers are seamlessly inlined in the caller's VC

2. Minimalistic IVL: λ + recursive definitions + multibarrel CPS
   - nicely captures a variety of control structures (loops, exceptions, etc.)
   - *I for Invisible*: efficiently transpiled from human-written source code
   - small verification kernel producing natural first-order VCs
   - non-trivial case studies: regexp engine, ASM sort routine

3. Working implementation in WHY3
   - first-class alias-free mutable variables with effect inference
   - contract extraction: autogenerated $\mathcal{P}re$ and $\mathcal{P}ost$ predicates
   - used as the backend for CREUSOT: RUST $\Rightarrow$ COMA $\Rightarrow$ WHY3

```
... where h x g = assert P x ; hide ...                    COMA
                  where out z = assert Q x z ; hide g z

      let h x g = (P x & ⊤) ∧ (∀z. Q x z ⟶ g z) in ...      VC
```

$$\ldots \ \textbf{where}\ h\ x\ g\ =\ \textbf{assert}\ P\ x\ ;\ \textbf{hide}\ \ldots \qquad \textsc{Coma}$$
$$\textbf{where}\ \mathsf{out}\ z = \textbf{assert}\ Q\ x\ z\ ;\ \textbf{hide}\ g\ z$$

$$\textbf{let}\ h\ x\ g\ =\ (P\ x\ \&\ \top) \wedge (\forall z.\, Q\ x\ z \longrightarrow g\ z)\ \textbf{in}\ \ldots \qquad \mathrm{VC}$$

$$\natural h\ =\ \lambda x g.\, (P\ x \longrightarrow \top) \wedge (\forall z.\, Q\ x\ z \longrightarrow g\ z) \qquad \natural\mathrm{VC}$$
$$=\ \lambda x g. \forall z.\, Q\ x\ z \longrightarrow g\ z$$

$$\ldots \ \textbf{where}\ h\ x\ g\ =\ \textbf{assert}\ P\ x\ ;\ \textbf{hide}\ \ldots \qquad \textsc{Coma}$$
$$\textbf{where}\ \mathsf{out}\ z = \textbf{assert}\ Q\ x\ z\ ;\ \textbf{hide}\ g\ z$$

$$\textbf{let}\ h\ x\ g\ =\ (P\ x\ \&\ \top) \wedge (\forall z.\, Q\ x\ z \longrightarrow g\ z)\ \textbf{in}\ \ldots \qquad \mathrm{VC}$$

$$\natural h\ =\ \lambda x g.\, (P\ x \longrightarrow \top) \wedge (\forall z.\, Q\ x\ z \longrightarrow g\ z) \qquad \natural\mathrm{VC}$$
$$=\ \lambda x g. \forall z.\, Q\ x\ z \longrightarrow g\ z$$

$$\mathcal{P}re(h)\ =\ \lambda x.\, h\ x\ (\lambda z.\top)\ \equiv\ \lambda x.\, P\ x$$

$$\ldots \ \textbf{where}\ h\ x\ g\ =\ \textbf{assert}\ P\ x\ ;\ \textbf{hide}\ \ldots \qquad \textsc{Coma}$$
$$\textbf{where}\ \mathsf{out}\ z = \textbf{assert}\ Q\ x\ z\ ;\ \textbf{hide}\ g\ z$$

$$\textbf{let}\ h\ x\ g\ =\ (P\ x\ \&\ \top) \wedge (\forall z.\, Q\ x\ z \longrightarrow g\ z)\ \textbf{in}\ \ldots \qquad \mathrm{VC}$$

$$\natural h\ =\ \lambda x g.\, (P\ x \longrightarrow \top) \wedge (\forall z.\, Q\ x\ z \longrightarrow g\ z) \qquad \natural\mathrm{VC}$$
$$=\ \lambda x g. \forall z.\, Q\ x\ z \longrightarrow g\ z$$

$$\mathcal{P}re(h)\ =\ \lambda x.\, h\ x\ (\lambda z.\top)\ \equiv\ \lambda x.\, P\ x$$

$$\overline{\mathcal{P}ost}(h)\ =\ \lambda x y.\, (\natural h)\ x\ (\lambda u.\, u \neq y\ \&\ \top)$$

$$\ldots \ \textbf{where}\ h\ x\ g\ =\ \textbf{assert}\ P\ x\ ;\ \textbf{hide}\ \ldots \qquad \textsc{Coma}$$
$$\textbf{where}\ \mathsf{out}\ z = \textbf{assert}\ Q\ x\ z\ ;\ \textbf{hide}\ g\ z$$

$$\textbf{let}\ h\ x\ g\ =\ (P\ x\ \&\ \top) \wedge (\forall z.\, Q\ x\ z \longrightarrow g\ z)\ \textbf{in}\ \ldots \qquad \mathrm{VC}$$

$$\natural h\ =\ \lambda x g.\, (P\ x \longrightarrow \top) \wedge (\forall z.\, Q\ x\ z \longrightarrow g\ z) \qquad \natural\mathrm{VC}$$
$$=\ \lambda x g. \forall z.\, Q\ x\ z \longrightarrow g\ z$$

$$\mathcal{P}re(h)\ =\ \lambda x.\, h\ x\ (\lambda z.\top)\ \equiv\ \lambda x.\, P\ x$$

$$\overline{\mathcal{P}ost}(h)\ =\ \lambda x y.\, (\natural h)\ x\ (\lambda u.\, u \neq y\ \&\ \top)$$

$$\equiv\ \lambda x y. \forall z.\, Q\ x\ z \longrightarrow z \neq y$$

$$\ldots \ \textbf{where}\ h\ x\ g\ =\ \textbf{assert}\ P\ x\ ;\ \textbf{hide}\ \ldots \qquad \textsc{Coma}$$
$$\textbf{where}\ \mathsf{out}\ z = \textbf{assert}\ Q\ x\ z\ ;\ \textbf{hide}\ g\ z$$

$$\textbf{let}\ h\ x\ g\ =\ (P\ x\ \&\ \top) \wedge (\forall z.\, Q\ x\ z \longrightarrow g\ z)\ \textbf{in}\ \ldots \qquad \mathrm{VC}$$

$$\natural h\ =\ \lambda x g.\, (P\ x \longrightarrow \top) \wedge (\forall z.\, Q\ x\ z \longrightarrow g\ z) \qquad \natural\mathrm{VC}$$
$$=\ \lambda x g. \forall z.\, Q\ x\ z \longrightarrow g\ z$$

$$\mathcal{P}re(h)\ =\ \lambda x.\, h\ x\ (\lambda z.\top)\ \equiv\ \lambda x.\, P\ x$$

$$\overline{\mathcal{P}ost}(h)\ =\ \lambda x y.\, (\natural h)\ x\ (\lambda u.\, u \neq y\ \&\ \top)$$

$$\equiv\ \lambda x y. \forall z.\, z = y \longrightarrow \neg\, Q\ x\ z$$

$$\ldots \ \textbf{where}\ h\ x\ g\ =\ \textbf{assert}\ P\ x\ ;\ \textbf{hide}\ \ldots \qquad \textsc{Coma}$$
$$\textbf{where}\ \mathsf{out}\ z = \textbf{assert}\ Q\ x\ z\ ;\ \textbf{hide}\ g\ z$$

$$\textbf{let}\ h\ x\ g\ =\ (P\ x\ \&\ \top) \wedge (\forall z.\, Q\ x\ z \longrightarrow g\ z)\ \textbf{in}\ \ldots \qquad \mathrm{VC}$$

$$\natural h\ =\ \lambda x g.\, (P\ x \longrightarrow \top) \wedge (\forall z.\, Q\ x\ z \longrightarrow g\ z) \qquad \natural\mathrm{VC}$$
$$=\ \lambda x g. \forall z.\, Q\ x\ z \longrightarrow g\ z$$

$$\mathcal{P}re(h)\ =\ \lambda x.\, h\ x\ (\lambda z.\top)\ \equiv\ \lambda x.\, P\ x$$

$$\overline{\mathcal{P}ost}(h)\ =\ \lambda x y.\, (\natural h)\ x\ (\lambda u.\, u \neq y\ \&\ \top)$$

$$\equiv\ \lambda x y.\, \neg\, Q\ x\ y$$