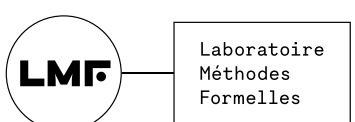
Comaleer

Paul Patault, Beatriz Rosas, Mário Pereira

June 15 2026 @ Gospel Plenary Meeting



Context

one month visit in Lisbon (Portugal)

to work on a new backend for Cameleer

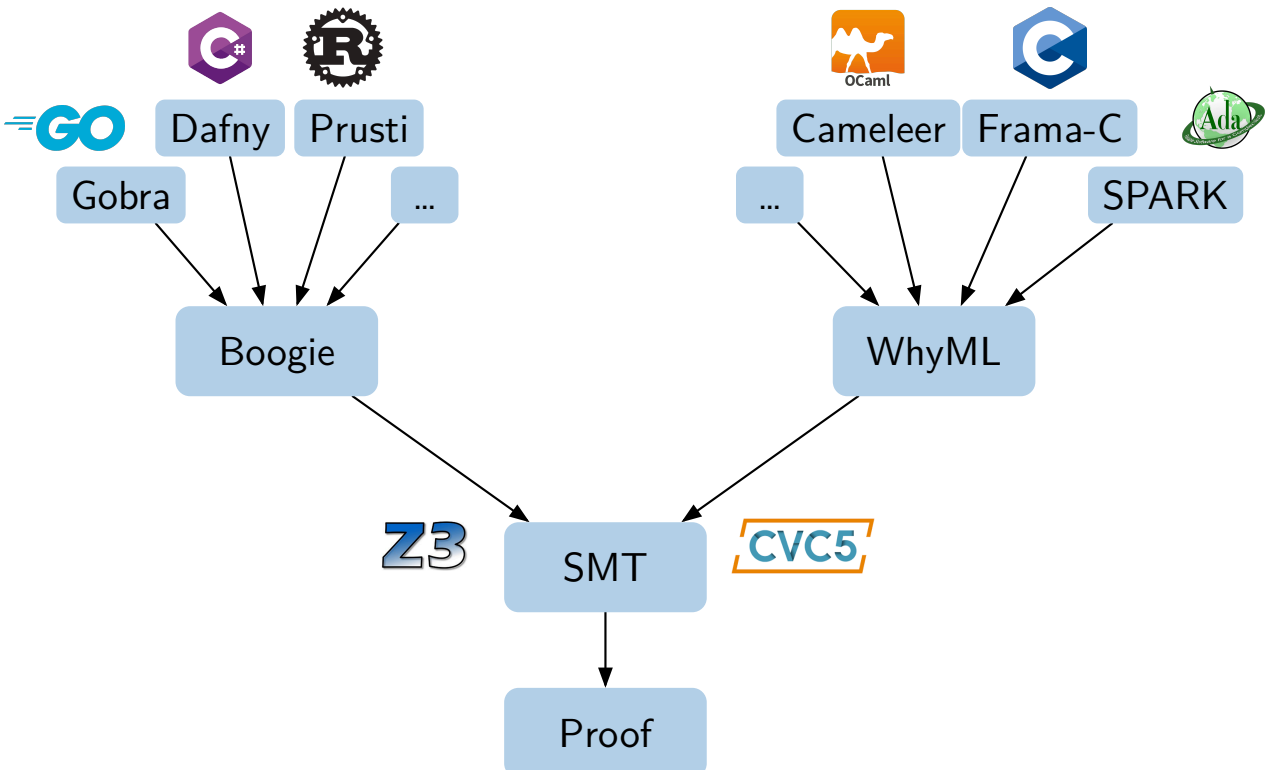
with Beatriz Rosas and Mário Pereira

Context

one month visit in Lisbon (Portugal)

to work on a new backend for Cameleer

with Beatriz Rosas and Mário Pereira

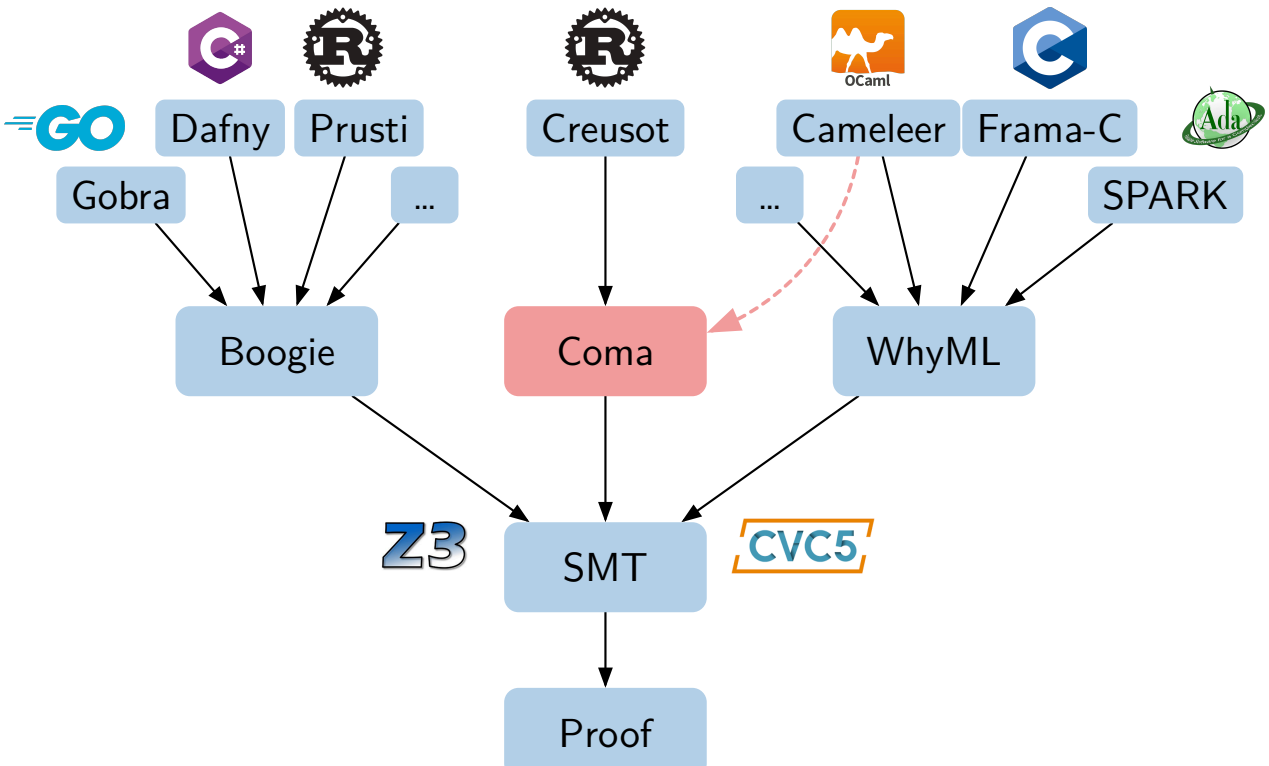


Context

one month visit in Lisbon (Portugal)

to work on a new backend for Cameleer

with Beatriz Rosas and Mário Pereira



Deductive verification

verifying a function

```
let f x̄
  requires P
  ensures  Q
= e
```

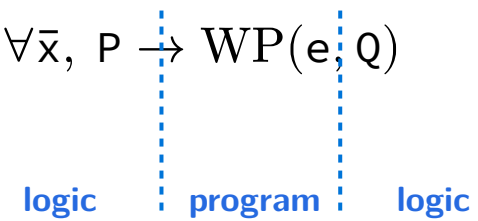
is proving $\forall \bar{x},\ \mathtt{P} \rightarrow \mathrm{WP}(\mathtt{e}, \mathtt{Q})$

Deductive verification

verifying a function

```
let f x̄
  requires P
  ensures  Q
= e
```

is proving $\forall \bar{x},\ \mathtt{P} \to \mathrm{WP}(\mathtt{e}, \mathtt{Q})$



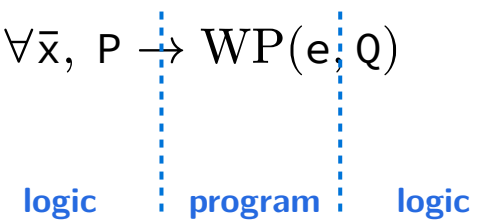
where there is a **strict separation** between specification and programs

Deductive verification

verifying a function

```
let f x̄
  requires P
  ensures  Q
= e
```

is proving $\forall \bar{x},\ \mathtt{P} \to \mathrm{WP}(\mathtt{e}, \mathtt{Q})$



where there is a **strict separation** between specification and programs

$\to$ **Coma**, a new IVL where we relax this separation.

Key ideas

1. Mix specification with the code
   - in-code assertions assume the role of preconditions and postconditions

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

Coma *[Paskevich, Patault, Filliâtre (ESOP 2025)]*

x, y, z				variable
s, t	$::=$	x	$\mid 0 \ldots \mid s + t \ldots$	term

pure data

---

control flow

h, g, f				handler symbol
k, o	$::=$	h	$\mid$ **fun** $\bar{x}\ \bar{g} \longrightarrow e$	(un)named handler
e, d	$::=$	$k\ \bar{s}\ \bar{o}$		handler call
		$\mid$ **let rec** $f\ \bar{s}\ \bar{o} = d$ **in** e		handler definition

Coma code is written in **multibarrel continuation-passing style** (CPS)

- handlers give back control by calling their continuation parameters
- can express conditionals, loops, function calls, exceptions

Coma *[Paskevich, Patault, Filliâtre (ESOP 2025)]*

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

pure data

---

control flow

h, g, f			handler symbol
k, o	$::=$	$h \mid \mathbf{fun}\ \bar{x}\ \bar{g} \longrightarrow e$	(un)named handler
e, d	$::=$	$k\ \bar{s}\ \bar{o}$	handler call
	$\mid$	$\mathbf{let\ rec}\ f\ \bar{s}\ \bar{o} = d\ \mathbf{in}\ e$	handler definition
	$\mid$	$\mathbf{assert}\ \varphi\ ;\ e$	blocking assertion
	$\mid$	$\mathbf{hide}\ e$	abstraction barrier

The code is the contract

```
let rec fib (n: int) (out: int ⟶ ⊥): ⊥
= if n < 0 then fail  else
  if n < 2 then out n else
  fib (n-2) (fun x ⟶
  fib (n-1) (fun y ⟶
  out (x+y) ))
```

The code is the contract

```
let rec fib (n: int) (out: int ⟶ ⊥): ⊥
= hide if n < 0 then fail  else
       if n < 2 then out n else
       fib (n-2) (fun x ⟶
       fib (n-1) (fun y ⟶
       out (x+y) ))
```

The code is the contract

```
let rec fib (n: int) (out: int ⟶ ⊥): ⊥
= assert n ⩾ 0 ;
  hide if n < 0 then fail  else
       if n < 2 then out n else
       fib (n-2) (fun x ⟶
       fib (n-1) (fun y ⟶
       out (x+y) ))
```

The code is the contract

```
let rec fib (n: int) (out: int ⟶ ⊥): ⊥
= assert n ⩾ 0 ;
  hide if n < 0 then fail  else
       if n < 2 then out n else
       fib (n-2) (fun x ⟶
       fib (n-1) (fun y ⟶
       assert (x+y) = F(n) ; out (x+y) ))
```

The code is the contract

```
let rec fib (n: int) (out: int ⟶ ⊥): ⊥
= let ret r = assert r = F(n) ;
              hide out r in
  assert n ⩾ 0 ;
  hide if n < 0 then fail  else
       if n < 2 then out n else
       fib (n-2) (fun x ⟶
       fib (n-1) (fun y ⟶
       ret (x+y) ))
```

The code is the contract

```
let rec fib (n: int) (out: int ⟶ ⊥): ⊥
= let ret r = assert r = F(n) ;
              hide out r in
  assert n ⩾ 0 ;
  if n < 0 then fail else
  hide if n < 2 then out n else
       fib (n-2) (fun x ⟶
       fib (n-1) (fun y ⟶
       ret (x+y) ))
```

The code is the contract

```
let rec fib (n: int) (out: int ⟶ ⊥): ⊥
= let ret r = assert r = F(n) ;
              hide out r in
  if n < 0 then fail else
  hide if n < 2 then out n else
       fib (n-2) (fun x ⟶
       fib (n-1) (fun y ⟶
       ret (x+y) ))
```

The code is the contract

```
let rec fib (n: int) (out: int ⟶ ⊥): ⊥
= let ret r = assert r = F(n) ;
              hide out r in
  if n < 0 then fail  else
  if n < 2 then out n else
  hide fib (n-2) (fun x ⟶
       fib (n-1) (fun y ⟶
       ret (x+y) ))
```

The code is the contract

```
let rec fib (n: int) (out: int ⟶ ⊥): ⊥
= let ret r = assert r = F(n) ;
              hide out r in
  if n < 0 then fail  else
  if n < 2 then out n else
  hide fib (n-2) (fun x ⟶
       fib (n-1) (fun y ⟶
       ret (x+y) ))
```

implementation

---

client

```
fib 42 (fun r ⟶ assert r > 10^8 ; halt)
```

The code is the contract

```
let rec fib (n: int) (out: int ⟶ ⊥): ⊥
= let ret r = assert r = F(n) ;
              hide out r in
  if n < 0 then fail  else
  if n < 2 then out n else
  hide fib (n-2) (fun x ⟶
       fib (n-1) (fun y ⟶
       ret (x+y) ))
```

implementation

VC of the client

```
fib 42 (fun r ⟶ assert r > 10^8 ; halt)
```

$$
\begin{aligned}
&(42 < 0 \longrightarrow \mathrm{false}) \wedge \\
&(0 \leqslant 42 < 2 \longrightarrow 42 > 10^8) \wedge \\
&(\forall r.\ r = F(42) \longrightarrow r > 10^8)
\end{aligned}
$$

The code is the contract

```
let rec fib (n: int) (out: int → ⊥): ⊥
= let ret r = assert r = F(n) ;
              hide out r in
  if n < 0 then fail  else
  if n < 2 then out n else
  hide fib (n-2) (fun x →
       fib (n-1) (fun y →
       ret (x+y) ))
```

implementation

VC of the definition

```
∀n. not n < 0 → not n < 2 →
  (n-2 < 0 → false) ∧
  (0 ⩽ n-2 < 2 →
   (n-1 < 0 → false) ∧
   (0 ⩽ n-1 < 2 → n-2 + n-1 = F(n))) ∧
  F(n-2) + F(n-1) = F(n)
```

Modal VCgen

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

$$\mathcal{C}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

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

$$\mathcal{C}(\texttt{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \texttt{in}\ \mathsf{e}) \triangleq$$

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

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

Modal VCgen

$$\mathcal{C}(\mathsf{h}) \triangleq \mathsf{h}$$
$$\mathcal{C}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$
$$\mathcal{C}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq$$
$$\mathcal{C}(\texttt{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \texttt{in}\ \mathsf{e}) \triangleq$$

VC generation maps continuations to propositions:

- *handler symbols* become *predicate variables*

$$\mathcal{C}(\mathtt{fib} : \mathtt{int} \to (\mathtt{int} \to \bot) \to \bot) =$$
$$\mathtt{fib} : \mathtt{int} \to (\mathtt{int} \to \mathrm{Prop}) \to \mathrm{Prop}$$

- *predicate variables* carry the specification of *handler symbols*

$$\mathcal{C}(\mathtt{if}) = \mathtt{if} = \lambda \mathtt{cfg}.\ (\mathtt{c} \to \mathtt{f}) \wedge (\neg \mathtt{c} \to \mathtt{g})$$

Modal VCgen

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

$$\mathcal{C}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq \lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{C}(\mathsf{e})$$

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

$$\mathcal{C}(\texttt{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \texttt{in}\ \mathsf{e}) \triangleq$$

VC generation maps continuations to propositions:

- *handler symbols* become *predicate variables*

$$\mathcal{C}(\texttt{fib} : \texttt{int} \to (\texttt{int} \to \bot) \to \bot) =$$
$$\texttt{fib} : \texttt{int} \to (\texttt{int} \to \mathrm{Prop}) \to \mathrm{Prop}$$

- *predicate variables* carry the specification of *handler symbols*

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

Modal VCgen

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

$$\mathcal{C}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq \lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{C}(\mathsf{e})$$

$$\mathcal{C}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{C}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{C}(\mathsf{o}_1) \cdots \mathcal{C}(\mathsf{o}_n)$$

$$\mathcal{C}(\texttt{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \texttt{in}\ \mathsf{e}) \triangleq$$

VC generation maps continuations to propositions:

- *handler symbols* become *predicate variables*

$$\mathcal{C}(\texttt{fib} : \texttt{int} \to (\texttt{int} \to \bot) \to \bot) =$$
$$\texttt{fib} : \texttt{int} \to (\texttt{int} \to \mathrm{Prop}) \to \mathrm{Prop}$$

- *predicate variables* carry the specification of *handler symbols*

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

Modal VCgen

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

$$\mathcal{C}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq \lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{C}(\mathsf{e})$$

$$\mathcal{C}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{C}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{C}(\mathsf{o}_1) \cdots \mathcal{C}(\mathsf{o}_n)$$

$$\mathcal{C}(\texttt{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \texttt{in}\ \mathsf{e}) \triangleq \qquad\qquad \mathcal{C}(\mathsf{e})$$

VC generation maps continuations to propositions:

- *handler symbols* become *predicate variables*

$$\mathcal{C}(\texttt{fib} : \texttt{int} \to (\texttt{int} \to \bot) \to \bot) =$$
$$\texttt{fib} : \texttt{int} \to (\texttt{int} \to \mathrm{Prop}) \to \mathrm{Prop}$$

- *predicate variables* carry the specification of *handler symbols*

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

Modal VCgen

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

$$\mathcal{C}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq \lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{C}(\mathsf{e})$$

$$\mathcal{C}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{C}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{C}(\mathsf{o}_1) \cdots \mathcal{C}(\mathsf{o}_n)$$

$$\mathcal{C}(\texttt{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \texttt{in}\ \mathsf{e}) \triangleq \texttt{let}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathcal{A}(\mathsf{d})\ \texttt{in}\ \mathcal{C}(\mathsf{e})$$

VC generation maps continuations to propositions:

- *handler symbols* become *predicate variables*

$$\mathcal{C}(\texttt{fib} : \texttt{int} \to (\texttt{int} \to \bot) \to \bot) =$$
$$\texttt{fib} : \texttt{int} \to (\texttt{int} \to \mathrm{Prop}) \to \mathrm{Prop}$$

- *predicate variables* carry the specification of *handler symbols*

$$\mathcal{C}(\texttt{if}) = \texttt{if} = \lambda \mathsf{c}\,\mathsf{f}\,\mathsf{g}.\ (\mathsf{c} \to \mathsf{f}) \wedge (\neg \mathsf{c} \to \mathsf{g})$$

Modal VCgen

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

$$\mathcal{C}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq \lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{C}(\mathsf{e})$$

$$\mathcal{C}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{C}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{C}(\mathsf{o}_1) \cdots \mathcal{C}(\mathsf{o}_n)$$

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

VC generation maps continuations to propositions:

- *handler symbols* become *predicate variables*

$$\mathcal{C}(\texttt{fib} : \texttt{int} \to (\texttt{int} \to \bot) \to \bot) =$$
$$\texttt{fib} : \texttt{int} \to (\texttt{int} \to \mathrm{Prop}) \to \mathrm{Prop}$$

- *predicate variables* carry the specification of *handler symbols*

$$\mathcal{C}(\texttt{if}) = \texttt{if} = \lambda \mathsf{c}\,\mathsf{f}\,\mathsf{g}.\ (\mathsf{c} \to \mathsf{f}) \wedge (\neg \mathsf{c} \to \mathsf{g})$$

Modal VCgen

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

$$\mathcal{C}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq \lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{C}(\mathsf{e})$$

$$\mathcal{C}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{C}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{C}(\mathsf{o}_1) \cdots \mathcal{C}(\mathsf{o}_n)$$

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

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

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

Modal VCgen

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

$$\mathcal{C}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq \lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{C}(\mathsf{e})$$

$$\mathcal{C}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{C}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{C}(\mathsf{o}_1) \cdots \mathcal{C}(\mathsf{o}_n)$$

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

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

neutralisable conjunction

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

Modal VCgen

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

$$\mathcal{C}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq \lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{C}(\mathsf{e})$$

$$\mathcal{C}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{C}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{C}(\mathsf{o}_1) \cdots \mathcal{C}(\mathsf{o}_n)$$

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

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

neutralisable conjunction

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

Modal VCgen

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

$$\mathcal{C}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq \lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{C}(\mathsf{e})$$

$$\mathcal{C}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{C}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{C}(\mathsf{o}_1) \cdots \mathcal{C}(\mathsf{o}_n)$$

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

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

neutralisable conjunction

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

standard library

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

$$\texttt{fail} \triangleq \bot\ \&\ \top$$

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

Modal VCgen

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

$$\mathcal{A}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

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

$$\mathcal{A}(\texttt{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \texttt{in}\ \mathsf{e}) \triangleq$$

$$\mathcal{A}(\texttt{assert}\ \varphi\ \texttt{;}\ \mathsf{e}) \triangleq$$

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

verify above barrier

---

verify below barrier

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

$$\mathcal{B}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

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

$$\mathcal{B}(\texttt{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \texttt{in}\ \mathsf{e}) \triangleq$$

$$\mathcal{B}(\texttt{assert}\ \varphi\ \texttt{;}\ \mathsf{e}) \triangleq$$

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

Modal VCgen

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

$$\mathcal{A}(\textbf{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

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

$$\mathcal{A}(\textbf{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \textbf{in}\ \mathsf{e}) \triangleq$$

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

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

verify above barrier

---

verify below barrier

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

$$\mathcal{B}(\textbf{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

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

$$\mathcal{B}(\textbf{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \textbf{in}\ \mathsf{e}) \triangleq$$

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

$$\mathcal{B}(\textbf{hide}\ \mathsf{e}) \triangleq \mathcal{A}(\mathsf{e}) \wedge \mathcal{B}(\mathsf{e})$$

Modal VCgen

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

$$\mathcal{A}(\textbf{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

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

$$\mathcal{A}(\textbf{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \textbf{in}\ \mathsf{e}) \triangleq$$

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

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

verify above barrier

---

verify below barrier

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

$$\mathcal{B}(\textbf{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

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

$$\mathcal{B}(\textbf{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \textbf{in}\ \mathsf{e}) \triangleq$$

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

$$\mathcal{B}(\textbf{hide}\ \mathsf{e}) \triangleq \mathcal{A}(\mathsf{e}) \wedge \mathcal{B}(\mathsf{e})$$

Modal VCgen

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

$$\mathcal{A}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

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

$$\mathcal{A}(\texttt{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \texttt{in}\ \mathsf{e}) \triangleq \texttt{let}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathcal{A}(\mathsf{d})\ \texttt{in}\ \mathcal{A}(\mathsf{e}) \wedge \forall \bar{\mathsf{x}}\bar{\mathsf{g}}.\ \mathcal{B}(\mathsf{d})$$

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

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

verify above barrier

---

verify below barrier

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

$$\mathcal{B}(\texttt{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

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

$$\mathcal{B}(\texttt{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \texttt{in}\ \mathsf{e}) \triangleq \texttt{let}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathcal{A}(\mathsf{d})\ \texttt{in}\ \mathcal{B}(\mathsf{e})$$

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

$$\mathcal{B}(\texttt{hide}\ \mathsf{e}) \triangleq \mathcal{A}(\mathsf{e}) \wedge \mathcal{B}(\mathsf{e})$$

Modal VCgen

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

$$\mathcal{A}(\textbf{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

$$\mathcal{A}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{A}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{A}(\mathsf{o}_1) \cdots \mathcal{A}(\mathsf{o}_n)$$

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

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

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

verify above barrier

---

verify below barrier

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

$$\mathcal{B}(\textbf{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

$$\mathcal{B}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{B}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{B}(\mathsf{o}_1) \cdots \mathcal{B}(\mathsf{o}_n)$$

$$\mathcal{B}(\textbf{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \textbf{in}\ \mathsf{e}) \triangleq \textbf{let}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathcal{A}(\mathsf{d})\ \textbf{in}\ \mathcal{B}(\mathsf{e})$$

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

$$\mathcal{B}(\textbf{hide}\ \mathsf{e}) \triangleq \mathcal{A}(\mathsf{e}) \wedge \mathcal{B}(\mathsf{e})$$

Modal VCgen

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

$$\mathcal{A}(\textbf{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

$$\mathcal{A}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{A}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{A}(\mathsf{o}_1) \cdots \mathcal{A}(\mathsf{o}_n)$$

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

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

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

verify above barrier

---

verify below barrier

$$\mathcal{B}(\mathsf{h}) \triangleq \natural\mathsf{h}$$ where $\natural$ turns every & into $\longrightarrow$

$$\mathcal{B}(\textbf{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq$$

$$\mathcal{B}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{B}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{B}(\mathsf{o}_1) \cdots \mathcal{B}(\mathsf{o}_n)$$

$$\mathcal{B}(\textbf{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \textbf{in}\ \mathsf{e}) \triangleq \textbf{let}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathcal{A}(\mathsf{d})\ \textbf{in}\ \mathcal{B}(\mathsf{e})$$

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

$$\mathcal{B}(\textbf{hide}\ \mathsf{e}) \triangleq \mathcal{A}(\mathsf{e}) \wedge \mathcal{B}(\mathsf{e})$$

Modal VCgen

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

$$\mathcal{A}(\textbf{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq \lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{A}(\mathsf{e})$$

$$\mathcal{A}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{A}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{A}(\mathsf{o}_1) \cdots \mathcal{A}(\mathsf{o}_n)$$

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

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

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

verify above barrier

---

verify below barrier

$$\mathcal{B}(\mathsf{h}) \triangleq \natural\mathsf{h}$$ where $\natural$ turns every & into $\longrightarrow$

$$\mathcal{B}(\textbf{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq \lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{B}(\mathsf{e})$$

$$\mathcal{B}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{B}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{B}(\mathsf{o}_1) \cdots \mathcal{B}(\mathsf{o}_n)$$

$$\mathcal{B}(\textbf{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \textbf{in}\ \mathsf{e}) \triangleq \textbf{let}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathcal{A}(\mathsf{d})\ \textbf{in}\ \mathcal{B}(\mathsf{e})$$

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

$$\mathcal{B}(\textbf{hide}\ \mathsf{e}) \triangleq \mathcal{A}(\mathsf{e}) \wedge \mathcal{B}(\mathsf{e})$$

Modal VCgen

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

$$\mathcal{A}(\textbf{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq (\lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{A}(\mathsf{e})) \wedge \natural(\lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{B}(\mathsf{e}))$$

$$\mathcal{A}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{A}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{A}(\mathsf{o}_1) \cdots \mathcal{A}(\mathsf{o}_n)$$

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

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

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

verify above barrier

---

verify below barrier

$$\mathcal{B}(\mathsf{h}) \triangleq \natural\mathsf{h}$$ where $\natural$ turns every & into $\longrightarrow$

$$\mathcal{B}(\textbf{fun}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} \longrightarrow \mathsf{e}) \triangleq (\lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{B}(\mathsf{e})) \wedge \natural(\lambda \bar{\mathsf{x}} \bar{\mathsf{g}}.\ \mathcal{A}(\mathsf{e}))$$

$$\mathcal{B}(\mathsf{k}\ \bar{\mathsf{s}}\ \bar{\mathsf{o}}) \triangleq \mathcal{B}(\mathsf{k})\ \bar{\mathsf{s}}\ \mathcal{B}(\mathsf{o}_1) \cdots \mathcal{B}(\mathsf{o}_n)$$

$$\mathcal{B}(\textbf{let rec}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathsf{d}\ \textbf{in}\ \mathsf{e}) \triangleq \textbf{let}\ \mathsf{f}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \mathcal{A}(\mathsf{d})\ \textbf{in}\ \mathcal{B}(\mathsf{e})$$

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

$$\mathcal{B}(\textbf{hide}\ \mathsf{e}) \triangleq \mathcal{A}(\mathsf{e}) \wedge \mathcal{B}(\mathsf{e})$$

Computation rules

Fully applied VCs reduce to **first-order formulas**

- $\forall \mathsf{h}.\Phi \triangleq \texttt{let}\ \mathsf{h}\ \bar{\mathsf{x}}\ \bar{\mathsf{g}} = \bot\ \&\ \bigwedge_{\mathsf{g}} \forall \bar{\mathsf{z}}\bar{\mathsf{o}}.\ \mathsf{g}\ \bar{\mathsf{z}}\ \bar{\mathsf{o}}\ \texttt{in}\ \Phi$
- β-reduction eliminates bound predicate variables
- non-neutralized $\Phi\ \&\ \Psi$ becomes $\Phi \wedge \Psi$

Comaleer

Motivation

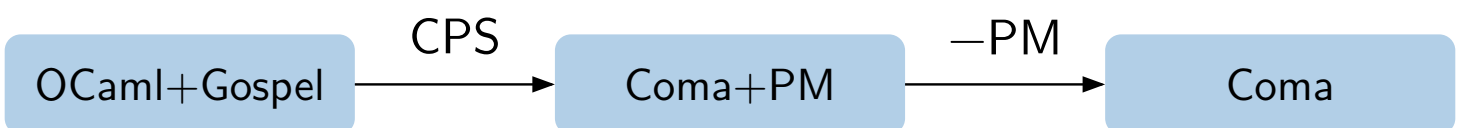
```
type tree = Node of tree * elt * tree
          | Empty

let remove_root (t: tree): tree =
  match t with
  | Node (l, _, r) ⟶ merge_tree l r
  | Empty          ⟶ assert false
(*@ result = remove_root t
             requires t ≠ Empty
             ensures  match t with
                      | Node (l, _, r) ⟶ ∀e. e ∈ result ⟷ e ∈ l ∨ e ∈ r
                      | Empty ⟶ false *)
```

Motivation

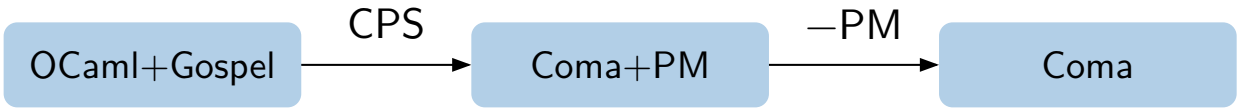
```
type tree = Node of tree * elt * tree
          | Empty

let remove_root (t: tree): tree =
  match t with
  | Node (l, _, r)
    (*@ ensures ∀e. e ∈ result ⟷ e ∈ l ∨ e ∈ r *)
          ⟶ merge_tree l r
  | Empty  ⟶ assert false
```

Comaleer

- Why? to take advantage of the flexible barrier: **in-depth specifications**
- How?



OCaml+Gospel subset

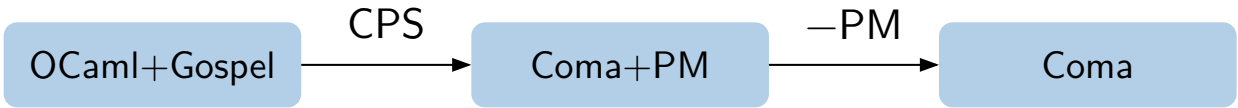


$$
\begin{array}{rcl}
e & ::= & a \mid e\ \bar{a} \\
& \mid & \texttt{let}\ x = e\ \texttt{in}\ e \\
& \mid & \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \\
& \mid & \texttt{match}\ e\ \texttt{with}\ \overline{p\ S \to e} \\
& \mid & \texttt{try}\ e\ \texttt{with}\ \overline{E\ \bar{a}\ S \to e} \\
& \mid & \texttt{raise}\ E\ \bar{a} \\
& \mid & \texttt{assert false}
\end{array}
$$

$$
\begin{array}{rcl}
p & ::= & \_ \mid x \mid (p, \ldots, p) \mid C(p, \ldots, p) \\
a & ::= & cst \mid x \mid (a, \ldots, a) \mid C(a, \ldots, a) \\
& \mid & a \diamond a \mid \circ a \\
S & ::= & (*@\ \texttt{requires}\ \varphi[\bar{x}] \\
& & \quad \texttt{ensures}\ \varphi[\texttt{result}, \bar{x}]\ *)
\end{array}
$$

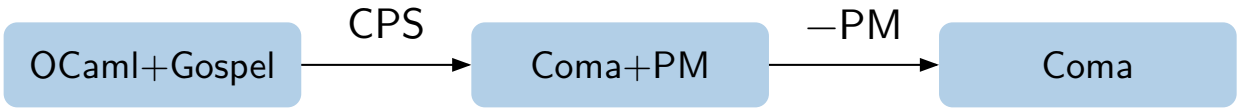
$$
\begin{array}{rcl}
def & ::= & \texttt{let rec}\ f\ \bar{x} = e \\
& & (*@\ r = f\ \bar{x} \\
& & \quad \texttt{requires}\ \varphi[\bar{x}] \\
& & \quad \texttt{ensures}\ \varphi[r, \bar{x}]\ *)
\end{array}
$$

Coma+PM



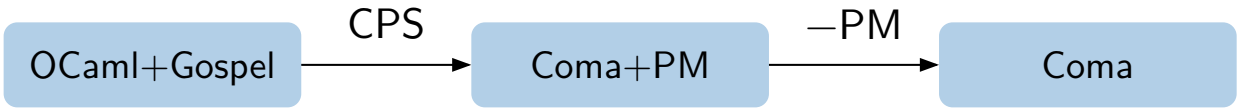
x, y, z			variable
s, t	$::=$	$x \mid 0 \ldots \mid s + t \ldots$	term
φ, ψ	$::=$	$s > t \ldots \mid \varphi \wedge \psi \ldots$	formula
h, g, f			handler symbol
k, o	$::=$	$h \mid \texttt{fun}\ \bar{x}\ \bar{g} \longrightarrow e$	(un)named handler
e, d	$::=$	$k\ \bar{s}\ \bar{o}$	handler call
	$\mid$	$\texttt{let rec}\ f\ \bar{s}\ \bar{o} = d\ \texttt{in}\ e$	handler definition
	$\mid$	$\texttt{assert}\ \varphi\ \texttt{;}\ e$	blocking assertion
	$\mid$	$\texttt{hide}\ e$	abstraction barrier

Coma+PM



x, y, z			variable
s, t	$::=$	$x \mid 0 \ldots \mid s + t \ldots$	term
φ, ψ	$::=$	$s > t \ldots \mid \varphi \wedge \psi \ldots$	formula
h, g, f			handler symbol
k, o	$::=$	$h \mid \texttt{fun}\ \bar{x}\ \bar{g} \longrightarrow e$	(un)named handler
e, d	$::=$	$k\ \bar{s}\ \bar{o}$	handler call
	$\mid$	$\texttt{let rec}\ f\ \bar{s}\ \bar{o} = d\ \texttt{in}\ e$	handler definition
	$\mid$	$\texttt{assert}\ \varphi\ ;\ e$	blocking assertion
	$\mid$	$\texttt{hide}\ e$	abstraction barrier
	$\mid$	$\texttt{match}\ x\ \texttt{with}\ \overline{p\ S \rightarrow e}$	

CPS translation (1/2)



[Compiling with continuations, continued - Kennedy 2007]

$$\langle\ \cdot\ |\ \cdot\ \rangle\ :\ \alpha \to (\alpha \to \bot) \to \bot$$

$$\langle\ \mathtt{a}\ |\ \kappa\ \rangle = \kappa\ \mathtt{a}$$

$$\langle\ \mathtt{e}\ \bar{\mathtt{a}}\ |\ \kappa\ \rangle = \langle\ \mathtt{e}\ |\ \mathbf{fun}\ \mathtt{f} \to \mathtt{f}\ \bar{\mathtt{a}}\ \kappa\ \rangle$$

$$\langle\ \mathbf{let}\ \mathtt{x} = \mathtt{e}_1\ \mathbf{in}\ \mathtt{e}_2\ |\ \kappa\ \rangle = \mathbf{let}\ \mathtt{j}\ \mathtt{x} = \langle\ \mathtt{e}_2\ |\ \kappa\ \rangle\ \mathbf{in}\ \langle\!\langle\ \mathtt{e}_1\ |\ \mathtt{j}\ \rangle\!\rangle$$

$$\langle\ \mathbf{if}\ \mathtt{e}_1\ \mathbf{then}\ \mathtt{e}_2\ \mathbf{else}\ \mathtt{e}_3\ |\ \kappa\ \rangle = \langle\ \mathtt{e}_1\ |\ \mathbf{fun}\ \mathtt{z} \to \mathbf{let}\ \mathtt{j}\ \mathtt{x} = \kappa\ \mathtt{x}\ \mathbf{in}\ \mathbf{if}\ \mathtt{z}\ \mathbf{then}\ \langle\!\langle\ \mathtt{e}_2\ |\ \mathtt{j}\ \rangle\!\rangle\ \mathbf{else}\ \langle\!\langle\ \mathtt{e}_3\ |\ \mathtt{j}\ \rangle\!\rangle\ \rangle$$

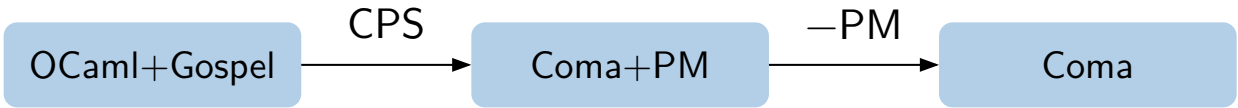
$$\langle\ \mathbf{match}\ \mathtt{e}_1\ \mathbf{with}\ \overline{\mathtt{p}\ \mathtt{S} \to \mathtt{e}}\ |\ \kappa\ \rangle = \langle\ \mathtt{e}_1\ |\ \mathbf{fun}\ \mathtt{z} \to \mathbf{let}\ \mathtt{j}\ \mathtt{x} = \kappa\ \mathtt{x}\ \mathbf{in}\ \mathbf{match}\ \mathtt{z}\ \mathbf{with}\ |_i\ \mathtt{p}_i\ \mathtt{S}_i \to \langle\!\langle\ \mathtt{e}_i\ |\ \mathtt{j}\ \rangle\!\rangle\ \rangle$$

$$\langle\ \mathbf{assert}\ \mathbf{false}\ |\ \kappa\ \rangle = \mathbf{fail}$$

$$\langle\ \mathbf{try}\ \mathtt{e}\ \mathbf{with}\ \overline{\mathtt{E}\ \bar{\mathtt{a}}\ \mathtt{S} \to \mathtt{e}}\ |\ \kappa\ \rangle = \ldots$$

$$\langle\ \mathbf{raise}\ \mathtt{E}\ \bar{\mathtt{a}}\ |\ \kappa\ \rangle = \ldots$$

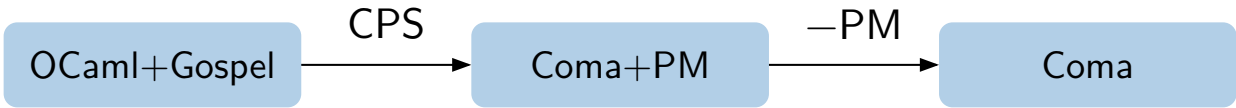
CPS translation (2/2)



[Compiling with continuations, continued - Kennedy 2007]

$$
\begin{aligned}
\langle\!\langle\ \cdot \mid \cdot\ \rangle\!\rangle &: \alpha \to \mathrm{ident} \to \bot \\
\langle\!\langle\ \mathtt{a} \mid \mathtt{k}\ \rangle\!\rangle &= \mathtt{k\ a} \\
\langle\!\langle\ \mathtt{e\ \bar{a}} \mid \mathtt{k}\ \rangle\!\rangle &= \langle\ \mathtt{e} \mid \mathtt{fun\ f} \to \mathtt{f\ \bar{a}\ k}\ \rangle \\
\langle\!\langle\ \mathtt{let\ x = e_1\ in\ e_2} \mid \mathtt{k}\ \rangle\!\rangle &= \mathtt{let\ j\ x} = \langle\!\langle\ \mathtt{e_2} \mid \mathtt{k}\ \rangle\!\rangle\ \mathtt{in}\ \langle\!\langle\ \mathtt{e_1} \mid \mathtt{j}\ \rangle\!\rangle \\
\langle\!\langle\ \mathtt{if\ e\ then\ e\ else\ e} \mid \mathtt{k}\ \rangle\!\rangle &= \langle\ \mathtt{e_1} \mid \mathtt{fun\ z} \to \mathtt{if\ z\ then}\ \langle\!\langle\ \mathtt{e_2} \mid \mathtt{k}\ \rangle\!\rangle\ \mathtt{else}\ \langle\!\langle\ \mathtt{e_3} \mid \mathtt{k}\ \rangle\!\rangle\ \rangle \\
\langle\!\langle\ \mathtt{match\ e\ with}\ \overline{\mathtt{p\ S} \to \mathtt{e}} \mid \mathtt{k}\ \rangle\!\rangle &= \langle\ \mathtt{e} \mid \mathtt{fun\ z} \to \mathtt{match\ z\ with}\ |_i\ \mathtt{p}_i\ \mathtt{S}_i \to \langle\!\langle\ \mathtt{e}_i \mid \mathtt{k}\ \rangle\!\rangle\ \rangle \\
\langle\!\langle\ \mathtt{assert\ false} \mid \mathtt{k}\ \rangle\!\rangle &= \mathtt{fail} \\
\langle\!\langle\ \mathtt{try\ e\ with}\ \overline{\mathtt{E\ \bar{a}\ S} \to \mathtt{e}} \mid \mathtt{k}\ \rangle\!\rangle &= \ldots \\
\langle\!\langle\ \mathtt{raise\ E\ \bar{a}} \mid \mathtt{k}\ \rangle\!\rangle &= \ldots
\end{aligned}
$$

PM compilation (1/3)



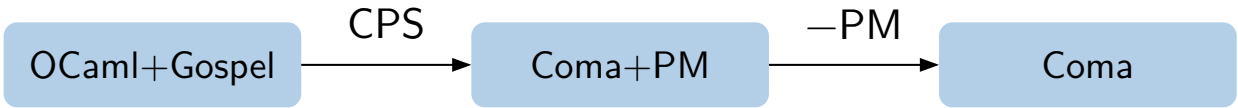
replace nested patterns with nested match statements

use only simple patterns

$$v ::= \_ \mid x$$

$$p ::= v \mid C(v, \ldots, v)$$

PM compilation (1/3)



replace nested patterns with nested match statements

use only simple patterns

$$v ::= \_ \mid x$$

$$p ::= v \mid C(v, \ldots, v)$$

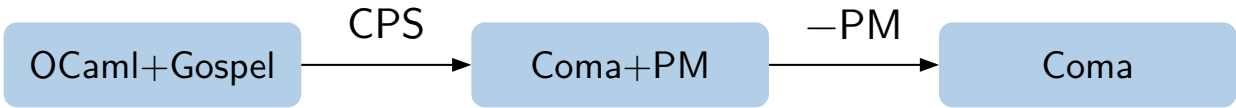
allow on the fly generation of destructors

```
match x with
| A (y, z) ⟶ e1
| B t      ⟶ e2
| _        ⟶ e3
```

$\longrightarrow$

```
case_x1 x
  (fun y z ⟶ e1)
  (fun t   ⟶ e2)
  (fun _   ⟶ e3)

val case_x1 x
  (case_A  y z {x = A (y, z)})
  (case_B  t   {(x = B t) ∧ (∀yz. x ≠ A (y, z))})
  (default _   {(∀t. x ≠ B t) ∧ (∀yz. x ≠ A (y, z))})
```

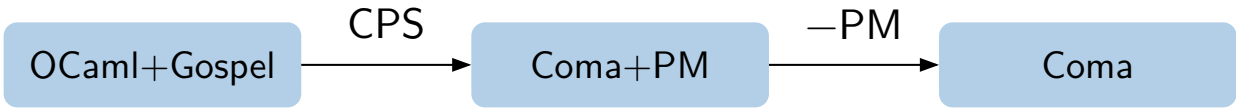
PM compilation (2/3)



OCaml

```
let rec merge t1 t2 =
  match t1, t2 with
  | Empty, _ ⟶ t2
  | _, Empty ⟶ t1
  | Node (l1, x1, r1),
    Node (l2, x2, r2) ⟶ ...
```

PM compilation (2/3)



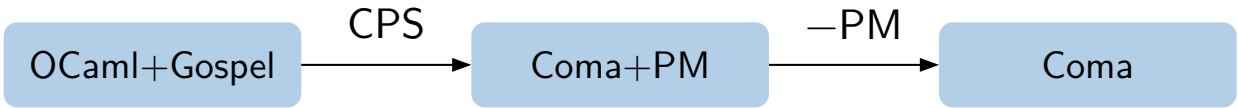
OCaml

```
let rec merge t1 t2 =
  match t1, t2 with
  | Empty, _ ⟶ t2
  | _, Empty ⟶ t1
  | Node (l1, x1, r1),
    Node (l2, x2, r2) ⟶ ...
```

Coma+PM

```
let rec merge t1 t2 k =
  match t1, t2 with
  | Empty, _ ⟶ k t2
  | _, Empty ⟶ k t1
  | Node (l1, x1, r1),
    Node (l2, x2, r2) ⟶ « ... | k »
```

PM compilation (2/3)



OCaml

```
let rec merge t1 t2 =
  match t1, t2 with
  | Empty, _ ⟶ t2
  | _, Empty ⟶ t1
  | Node (l1, x1, r1),
    Node (l2, x2, r2) ⟶ ...
```

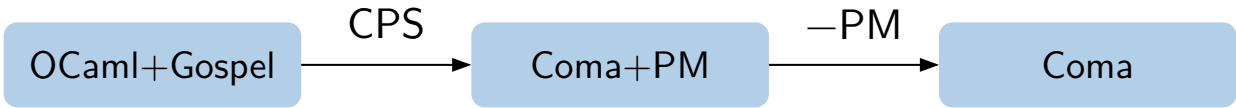
Coma+PM'

```
let rec merge t1 t2 k =
  match t1 with
  | Node (l1, x1, r1) ⟶
    (match t2 with
     | Node (l2, x2, r2) ⟶ « ... | k »
     | Empty ⟶ k t1)
  | Empty ⟶
    (match t2 with
     | Empty ⟶ k t2
     | _ ⟶ k t2)
  | _ ⟶ match t2 with Empty ⟶ k t1
```

Coma+PM

```
let rec merge t1 t2 k =
  match t1, t2 with
  | Empty, _ ⟶ k t2
  | _, Empty ⟶ k t1
  | Node (l1, x1, r1),
    Node (l2, x2, r2) ⟶ « ... | k »
```

PM compilation (2/3)



OCaml

```
let rec merge t1 t2 =
  match t1, t2 with
  | Empty, _ ⟶ t2
  | _, Empty ⟶ t1
  | Node (l1, x1, r1),
    Node (l2, x2, r2) ⟶ ...
```

Coma+PM'

```
let rec merge t1 t2 k =
  match t1 with
  | Node (l1, x1, r1) ⟶
    (match t2 with
     | Node (l2, x2, r2) ⟶ « ... | k »
     | Empty ⟶ k t1)
  | Empty ⟶
    (match t2 with
     | Empty ⟶ k t2
     | _ ⟶ k t2)
  | _ ⟶ match t2 with Empty ⟶ k t1
```

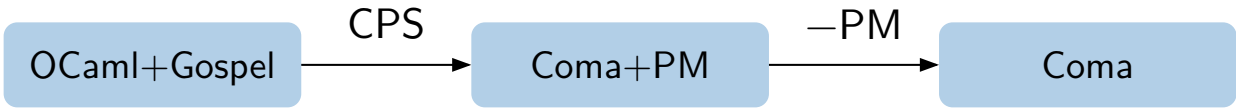
Coma+PM

```
let rec merge t1 t2 k =
  match t1, t2 with
  | Empty, _ ⟶ k t2
  | _, Empty ⟶ k t1
  | Node (l1, x1, r1),
    Node (l2, x2, r2) ⟶ « ... | k »
```

Coma

```
let rec merge t1 t2 k =
  case_merge1 t1
    (fun l1 x1 r1 ⟶
       case_merge2 t2
        (fun l2 x2 r2 ⟶ « ... | k »)
        (⟶ k t2))
    (⟶
      case_merge3 t2
       (⟶ k t2)
       (fun _ ⟶ k t2))
    (fun _ ⟶ case_merge4 t2 (⟶ k t1))
```

PM compilation (3/3)




```
let rec merge t1 t2 =
  match t1, t2 with
  | Empty, _ ⟶ t2
  | _, Empty ⟶ t1
  | Node (l1, x1, r1),
    Node (l2, x2, r2)
    (*@ requires P ensures Q *) ⟶ ...
```

```
let rec merge t1 t2 k =
  match t1, t2 with
  | Empty, _ ⟶ k t2
  | _, Empty ⟶ k t1
  | Node (l1, x1, r1),
    Node (l2, x2, r2)
    (*@ requires P ensures Q *) ⟶ « ... | k »
```

```
let rec merge t1 t2 k =
  match t1 with
  | Node (l1, x1, r1) ⟶
    (match t2 with
     | Node (l2, x2, r2)
       (*@ requires P ensures Q *)
       ⟶  « ... | k »
     | Empty ⟶ k t1)
  | Empty ⟶
    (match t2 with
     | Empty ⟶ k t2
     | _ ⟶ k t2)
  | _ ⟶ match t2 with Empty ⟶ k t1
```

```
let rec merge t1 t2 k =
  case_merge1 t1
    (fun l1 x1 r1 ⟶
       case_merge2 t2
        (fun l2 x2 r2 ⟶
           let k' r = assert Q ; hide k r in
           assert P ; hide « ... | k' »)
        (⟶ k t2))
      (⟶
        case_merge3 t2
        (⟶ k t2)
        (fun _ ⟶ k t2))
      (fun _ ⟶ case_merge4 t2 (⟶ k t1))
```

In-depth specification

```
let f t =
  match t with
  | Node (Empty, _, _)
         (*@ requires P1
             ensures  Q1 *) ⟶ 1
  | Empty (*@ requires P2
              ensures  Q2 *) ⟶ 2
  | Node (_, _, _)          ⟶ 3
(*@ r = f t
      requires Pre
      ensures  Post *)
```

```
let f t k =
  let k1 r = assert Post ; hide k r in
  assert Pre ;
  hide case_f1 t
    (fun l v r ⟶
      case_f2 l
        (⟶ let k2 r = assert Q1 ;
                      hide k1 r in
           assert P1 ;
           hide k2 1)
        (fun _ ⟶ k1 3))
    (⟶ let k2 r = assert Q2 ;
                  hide k1 r in
       assert P2 ;
       hide k3 2)
```

In-depth specification

```
let f t =
  match t with
  | Node (Empty, _, _)
         (*@ requires P1
             ensures  Q1 *) ⟶ 1
  | Empty (*@ requires P2
             ensures  Q2 *) ⟶ 2
  | Node (_, _, _)          ⟶ 3
(*@ r = f t
      requires Pre
      ensures  Post *)
```

```
∀t. Pre ⟶
    (∀lvr. t = Node l v r ⟶
      (l = Empty ⟶ (P1 && Q1) ∧ (Q1 ⟶ Post)) ∧
      (l ≠ Empty ⟶ Post)) ∧
    ((t = Empty && ∀lvr. t ≠ Node l v r) ⟶
     (P2 && Q2) ∧ (Q2 ⟶ Post)) ∧
    ((∀lvr. t ≠ Node l v r) && (t ≠ Empty) ⟶
     Post)
```

```
let f t k =
  let k1 r = assert Post ; hide k r in
  assert Pre ;
  hide case_f1 t
    (fun l v r ⟶
      case_f2 l
        (⟶ let k2 r = assert Q1 ;
                      hide k1 r in
           assert P1 ;
           hide k2 1)
        (fun _ ⟶ k1 3))
      (⟶ let k2 r = assert Q2 ;
                    hide k1 r in
         assert P2 ;
         hide k3 2)
```

In-depth opened specification

```
let f t =
  match t with
  | Node (Empty, _, _)
         (*@ requires P1
             ensures  Q1 *) ⟶ 1
  | Empty (*@ requires P2
              ensures  Q2 *) ⟶ 2
  | Node (_, _, _)          ⟶ 3
(*@ r = f t
      requires Pre
      requires !!
      ensures  Post *)
```

```
∀t. Pre ⟶
 (∀vr. t = Node Empty v r ⟶ P1 ⟶ Q1) ∧
 ((t = Empty && ∀lvr. t ≠ Node l v r) ⟶ P2 ⟶ Q2)
```

↳ nothing about `Post`,
delagated to the client of `f`

```
let f t k =
  let k1 r = assert Post ; hide k r in
  assert Pre ;
  case_f1 t
    (fun l v r ⟶
      case_f2 l
        (⟶ let k2 r = assert Q1 ;
                      hide k1 r in
           assert P1 ;
           hide k2 1)
        (fun _ ⟶ k1 3))
    (⟶ let k2 r = assert Q2 ;
                  hide k1 r in
       assert P2 ;
       hide k3 2)
```

Results

File	LoC	LoS[1]	LoS⋆[2]	reduction %[3]
binary_search_tree.ml	27	45	42	6
leftist_heap.ml	38	73	41	43
paring_heap.ml	36	78	54	30
red_black_tree.ml	93	131	111	15
same_fring.ml	22	17	14	17
skew_heap.ml	25	48	27	43

[1]: standard Cameleer

[2]: Comaleer

[3]: $\frac{\text{LoS} - \text{LoS}\star}{\text{LoS}} \times 100$

Take away

- Coma
  - new IVL with strong pen-and-paper theory
  - features an explicit abstraction barrier
  - makes specification more concise and natural
  - used by Creusot and Cameleer
- Related publications
  - Comaleer: INForum *(wip)*
  - Coma: ESOP 2025, Dafny Workshop 2026, TOPLAS 2026
  - Creusot/Coma: JFLA 2025

Appendix

Neutralisation

Removes the proper goal of the underlying formula.

$$\texttt{let f x g} = \texttt{let out z} = \texttt{assert Q x z ; hide g z in}\ \texttt{assert P x ; hide} \ldots \qquad \text{Coma}$$

$$\texttt{let f x g} = (\texttt{P x} \,\&\, \top) \wedge (\forall \texttt{z}.\ \texttt{Q x z} \to \texttt{g z})\ \texttt{in} \ldots \qquad \text{VC}$$

$$\natural\texttt{f} = \lambda \texttt{x g}.(\texttt{P x} \to \top) \wedge (\forall \texttt{z}.\ \texttt{Q x z} \to \texttt{g z}) \qquad \natural\text{VC}$$

$$= \lambda \texttt{x g}.\forall \texttt{z}.\ \texttt{Q x z} \to \texttt{g z}$$

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

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

$$\Upsilon(\Phi \wedge \Psi) \equiv \Upsilon\Phi \wedge \Upsilon\Psi \quad \text{when} \quad \natural\Psi_1 = \natural\Psi_2$$

$$\mathcal{C}(e) \equiv \mathcal{A}(e) \wedge \mathcal{B}(e)$$

Crash

$$
\begin{aligned}
&\mathcal{C}(\texttt{let crash = (fun f → hide f) fail in crash}) \\
=\ &\texttt{let crash} = \mathcal{A}(\texttt{(fun f → hide f) fail})\ \texttt{in}\ \mathcal{C}(\texttt{crash}) \wedge \mathcal{B}(\texttt{(fun f → hide f) fail}) \\
=\ &\texttt{let crash} = \mathcal{A}(\texttt{fun f → hide f})\ \mathcal{A}(\texttt{fail})\ \texttt{in crash} \wedge \mathcal{B}(\texttt{fun f → hide f})\ \mathcal{B}(\texttt{fail}) \\
=\ &\texttt{let crash} = ((\lambda \texttt{f}.\ \mathcal{A}(\texttt{hide f})) \wedge \natural(\lambda \texttt{f}.\ \mathcal{B}(\texttt{hide f})))\ \texttt{fail in crash} \wedge \mathcal{B}(\texttt{fun f → hide f})\ \mathcal{B}(\texttt{fail}) \\
=\ &\texttt{let crash} = ((\lambda \texttt{f}.\top) \wedge \natural(\lambda \texttt{f}.\ \texttt{f}))\ \texttt{fail in crash} \wedge \mathcal{B}(\texttt{fun f → hide f})\ \mathcal{B}(\texttt{fail}) \\
=\ &\texttt{let crash} = ((\lambda \texttt{f}.\top) \wedge \natural(\lambda \texttt{f}.\ \texttt{f}))\ \texttt{fail in crash} \wedge ((\lambda \texttt{f}.\ \mathcal{B}(\texttt{hide f})) \wedge \natural(\lambda \texttt{f}.\ \mathcal{A}(\texttt{hide f})))\ \mathcal{B}(\texttt{fail}) \\
=\ &\texttt{let crash} = ((\lambda \texttt{f}.\top) \wedge \natural(\lambda \texttt{f}.\ \texttt{f}))\ \texttt{fail in crash} \wedge ((\lambda \texttt{f}.\ \texttt{f}) \wedge \natural(\lambda \texttt{f}.\top))\ \natural\texttt{fail} \\
=\ &(\lambda \texttt{f}.\top)\ \texttt{fail} \wedge \natural(\lambda \texttt{f}.\ \texttt{f})\ \texttt{fail} \wedge (\lambda \texttt{f}.\ \texttt{f})\ \natural\texttt{fail} \wedge \natural(\lambda \texttt{f}.\top)\ \natural\texttt{fail} \\
=\ &\top \wedge \texttt{fail} \wedge \natural\texttt{fail} \wedge \top \\
=\ &\top \wedge (\bot\ \&\ \top) \wedge \natural(\bot\ \&\ \top) \wedge \top \\
=\ &\top \wedge (\bot \wedge \top) \wedge (\bot \to \top) \wedge \top \\
=\ &\top \wedge \bot \wedge \top \wedge \top \\
=\ &\bot
\end{aligned}
$$