Tail Modulo Cons, OCaml, and Relational Separation Logic

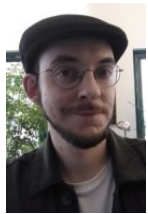
Clément Allain
Frédéric Bour
Basile Clément
François Pottier
Gabriel Scherer

January 24, 2025

Tail Modulo Cons, OCaml, and Relational Separation Logic

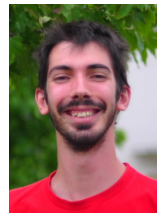
Clément Allain

Frédéric Bour



Basile Clément

François Pottier



Gabriel Scherer

map: natural implementation

```
let rec map f xs =
  match xs with
  | [] →
      []
  | x :: xs →
      let y = f x in
      y :: map f xs
```

```
# List.init 250_000 (fun _ → ())
  |> map Fun.id
  |> ignore
  ;;
Stack overflow during evaluation (looping recursion?).
```

map: accumulator-passing style

```
let rec map_aps acc f xs =
  match xs with
  | [] →
      List.rev acc
  | x :: xs →
      let y = f x in
      map_aps (y :: acc) f xs
```

```
let map xs =
  map_aps [] f xs
```

```
# List.init 250_000 (fun _ → ())
  |> map Fun.id
  |> ignore
  ;;
- : unit = ()
```

map: destination-passing style

```
let rec map_dps dst f xs =
  match xs with
  | [] →
      set_field dst 1 []
  | x :: xs →
      let y = f x in
      let dst' = y :: _ in
      set_field dst 1 dst' ;
      map_dps dst' f xs
```

```
let map f xs =
  match xs with
  | [] →
      []
  | x :: xs →
      let y = f x in
      let dst = y :: _ in
      map_dps dst f xs ;
      dst
```

```
# List.init 250_000 (fun _ → ())
  |> map Fun.id
  |> ignore
  ;;
- : unit = ()
```

map: Tail Modulo Constructor (TMC)

```
let[@tail_mod_cons] rec map f xs =
  match xs with
  | [] →
      []
  | x :: xs →
      let y = f x in
      y :: map f xs
```

```
# List.init 250_000 (fun _ → ())
  |> map Fun.id
  |> ignore
  ;;
- : unit = ()
```

TMC transformation

- **Safe:** performed by the OCaml compiler.
- **Explicit:** `[@tail_mod_cons]` annotation.

- **Generality:**
  - Works on any algebraic data type (lists, trees, *etc.*).
  - Supports mutually recursive functions.

- **Implementation details:** see the paper.
- **Performance:** see benchmarks in the paper.
- **Feature adoption:** see survey in the paper.

- **Soundness:** formally verified in Coq/Rocq in an simplified setting ...

DataLang: syntax

$$
\begin{array}{llll}
\text{Index} & \ni & i & ::= \quad 0 \mid 1 \mid 2 \\
\text{Tag} & \ni & t & \\
\mathbb{B} & \ni & b & \\
\mathbb{L} & \ni & \ell & \\
\mathbb{F} & \ni & f & \\
\mathbb{X} & \ni & x, y & \\
\text{Val} & \ni & v, w & ::= \quad () \mid i \mid t \mid b \mid \ell \mid \texttt{@}f \\
\text{Expr} & \ni & e & ::= \quad v \mid x \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \mid e_1\ \overline{e_2} \\
 & & & \quad \mid \quad e_1 = e_2 \mid \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \\
 & & & \quad \mid \quad \{\, t\,,\, e_1\,,\, e_2 \,\} \\
 & & & \quad \mid \quad e_1\,.\,(e_2) \mid e_1\,.\,(e_2) \leftarrow e_3 \\
\text{Def} & \ni & d & ::= \quad \texttt{fun}\ \overline{x} \to e \\
\text{Prog} & \ni & p & := \quad \mathbb{F} \xrightarrow{\text{fin}} \text{Def} \\
\text{State} & \ni & \sigma & := \quad \mathbb{L} \xrightarrow{\text{fin}} \text{Val} \\
\text{Config} & \ni & \rho & := \quad \text{Expr} \times \text{State}
\end{array}
$$

DataLang: map

```
map := fun f xs →
  match xs with
  | [] →
      []
  | x :: xs →
      let y = f x in
      y :: @map f xs
```

DataLang: map (transformed)

```
map_dps ≔ fun dst idx f xs →
  match xs with
  | [] →
      dst.(idx) ← []
  | x :: xs →
      let y = f x in
      let dst' = y :: ■ in
      dst.(idx) ← dst' ;
      @map_dps dst' 2 f xs
```

```
map_dir ≔ fun f xs →
  match xs with
  | [] →
      []
  | x :: xs →
      let y = f x in
      let dst = y :: ■ in
      @map_dps dst 2 f xs ;
      dst
```

TMC transformation

$$e_s \overset{\xi}{\underset{\text{dir}}{\rightsquigarrow}} e_t \qquad\qquad d_s \overset{\xi}{\underset{\text{dir}}{\rightsquigarrow}} d_t$$

$$(e_{dst}, e_{idx}, e_s) \overset{\xi}{\underset{\text{dps}}{\rightsquigarrow}} e_t \qquad\qquad d_s \overset{\xi}{\underset{\text{dps}}{\rightsquigarrow}} d_t$$

$$p_s \rightsquigarrow p_t$$

Transformation soundness

$p_s \rightsquigarrow p_t$ program p_s transforms into program p_t

$\Downarrow$

$p_s \sqsupseteq p_t$ program p_t refines program p_s
(termination-preserving refinement)

Transformation soundness

Termination-preserving behavioral refinement

$$
\begin{aligned}
p_s \sqsupseteq p_t \quad &\coloneqq \quad \forall\, f \in \mathrm{dom}(p_s), v_s, v_t. \\
&\quad\ \mathrm{wf}(v_s) \land v_s \sim v_t \implies \\
&\quad\ @f\ v_s \sqsupseteq @f\ v_t \\
\\
e_s \sqsupseteq e_t \quad &\coloneqq \quad \forall\, b_t \in \mathrm{behaviours}_{p_t}(e_t). \\
&\quad\ \exists\, b_s \in \mathrm{behaviours}_{p_s}(e_s).\ b_s \sqsupseteq b_t \\
\\
\mathrm{behaviours}_p(e) \quad &\coloneqq \quad \{\mathbf{Conv}(e') \mid \ldots\} \uplus \{\mathbf{Div} \mid (e, \emptyset) \Uparrow_p\}
\end{aligned}
$$

Transformation soundness

$p_s \rightsquigarrow p_t$ program p_s transforms into program p_t

$\Downarrow$

$p_s \sqsupseteq p_t$ program p_t refines program p_s
(termination-preserving refinement)

Transformation soundness

$p_s \rightsquigarrow p_t$ program p_s transforms into program p_t

$\Downarrow$

$p_s \gtrsim p_t$ program p_t simulates program p_s
(*relational separation logic*, SIMULIRIS)

$\Downarrow$

$p_s \sqsupseteq p_t$ program p_t refines program p_s
(termination-preserving refinement)

Relational separation logic

REL-PURE

$$\frac{e_s \xrightarrow[\text{pure}]{p_s} e_s' \qquad e_t \xrightarrow[\text{pure}]{p_t} e_t' \qquad e_s' \gtrsim e_t' \ [\Phi]}{e_s \gtrsim e_t \ [\Phi]}$$

REL-SOURCE-LOAD

$$\frac{(\ell + i) \mapsto_s v_s \qquad (\ell + i) \mapsto_s v_s \twoheadrightarrow v_s \gtrsim e_t \ [\Phi]}{\ell.(i) \gtrsim e_t \ [\Phi]}$$

REL-TARGET-LOAD

$$\frac{(\ell + i) \mapsto_t v_t \qquad (\ell + i) \mapsto_t v_t \twoheadrightarrow e_s \gtrsim v_t \ [\Phi]}{e_s \gtrsim \ell.(i) \ [\Phi]}$$

Transformation soundness

$p_s \rightsquigarrow p_t$ program p_s transforms into program p_t

Abstract protocols (calling conventions)

$$\textsc{rel-protocol}\quad \frac{\mathrm{X}(e_s, e_t, \Psi) \qquad \forall\, e_s', e_t'.\ \Psi(e_s', e_t') \twoheadrightarrow e_s' \gtrsim e_t' \,\langle \mathrm{X} \rangle\ [\Phi]}{e_s \gtrsim e_t \,\langle \mathrm{X} \rangle\ [\Phi]}$$

$p_s \sqsupseteq p_t$ program p_t refines program p_s
(termination-preserving refinement)

Transformation soundness

$p_s \rightsquigarrow p_t$ program p_s transforms into program p_t

$\Downarrow$

$p_s \gtrsim p_t$ program p_t simulates program p_s
(*relational separation logic*, SIMULIRIS)

$\Downarrow$

$p_s \sqsupseteq p_t$ program p_t refines program p_s
(termination-preserving refinement)

Specification in separation logic

$$\frac{\{???\}}{\texttt{@map}\ v_s \gtrsim \texttt{@map\_dir}\ v_t}\ \{???\}$$

$$\frac{\{???\}}{\texttt{@map}\ v_s \gtrsim \texttt{@map\_dps}\ \ell\ i\ v_t}\ \{???\}$$

Direct transformation

$$\frac{\{v_s \approx v_t\}}{\dfrac{\texttt{@map}\ v_s \ \gtrsim\ \texttt{@map\_dir}\ v_t}{\{w_s, w_t.\ w_s \approx w_t\}}}$$

REL-DIR (SIMULIRIS)

$$\frac{\begin{array}{c} f \in \mathrm{dom}(p_s) \\ v_s \approx v_t \\ \forall\, w_s, w_t.\ w_s \approx w_t \twoheadrightarrow \Phi(w_s, w_t) \end{array}}{\texttt{@}f\ v_s \gtrsim \texttt{@}f\ v_t\ [\Phi]}$$

DPS transformation

$$\frac{\{v_s \approx v_t * (\ell + i) \mapsto_t \blacksquare\}}{\texttt{@map}\ v_s \ \gtrsim\ \texttt{@map\_dps}\ \ell\ i\ v_t}$$
$$\{w_s, ().\, \exists\, w_t.\, w_s \approx w_t * (\ell + i) \mapsto_t w_t\}$$

REL-DPS

$$\frac{\xi[f] = f_{dps} \qquad \overline{v_s} \approx \overline{v_t} \qquad \ell \mapsto_t \overline{v} \qquad \forall\, w_s, w_t.\, w_s \approx w_t \mathrel{-\!\!*} \ell \mapsto_t \overline{v}[i \mapsto w_t] \mathrel{-\!\!*} \Phi(w_s, ())}{\texttt{@}f\ \overline{v_s} \gtrsim \texttt{@}f_{dps}\ \ell\ i\ \overline{v_t}\ [\Phi]}$$

REL-PROTOCOL

$$\frac{\mathrm{X}(e_s, e_t, \Psi) \qquad \forall\, e_s', e_t'.\, \Psi(e_s', e_t') \mathrel{-\!\!*} e_s' \gtrsim e_t' \langle \mathrm{X} \rangle\ [\Phi]}{e_s \gtrsim e_t \langle \mathrm{X} \rangle\ [\Phi]}$$

Conclusion

- Implementation of the TMC transformation in the OCaml compiler.
- Mechanized soundness proof using *relational separation logic*.
- *Abstract protocols* to support different calling conventions: APS, inlining.

Thank you for your attention!

Simulation

$$\lambda\, \mathit{sim}.\, \lambda\, \mathit{sim\text{-}inner}.\, \lambda\, (\Phi, e_s, e_t).\, \forall\, \sigma_s, \sigma_s.\, \mathrm{I}(\sigma_s, \sigma_t) \twoheadrightarrow \Rightarrow$$

$$\text{sim-body}_{\mathrm{X}} \coloneqq \bigvee \left[\begin{array}{ll}
① & \mathrm{I}(\sigma_s, \sigma_t) * \Phi(e_s, e_t) \\
② & \mathrm{I}(\sigma_s, \sigma_t) * \text{strongly-stuck}_{p_s}(e_s) * \text{strongly-stuck}_{p_t}(e_s) \\
③ & \exists\, e_s', \sigma_s'.\, (e_s, \sigma_s) \xrightarrow{p_s}{}^{+} (e_s', \sigma_s') * \mathrm{I}(\sigma_s', \sigma_t) * \mathit{sim\text{-}inner}(\Phi, e_s', e_t) \\
④ & \text{reducible}_{p_t}(e_t, \sigma_t) * \forall\, e_t', \sigma_t'.\, (e_t, \sigma_t) \xrightarrow{p_t} (e_t', \sigma_t') \twoheadrightarrow \Rightarrow \\
 & \quad \bigvee \left[\begin{array}{ll}
 Ⓐ & \mathrm{I}(\sigma_s, \sigma_t') * \mathit{sim\text{-}inner}(\Phi, e_s, e_t') \\
 Ⓑ & \exists\, e_s', \sigma_s'.\, (e_s, \sigma_s) \xrightarrow{p_s}{}^{+} (e_s', \sigma_s') * \\
 & \quad \mathrm{I}(\sigma_s', \sigma_t') * \mathit{sim}(\Phi, e_s', e_t')
\end{array}\right. \\
⑤ & \exists\, K_s, e_s', K_t, e_t', \Psi. \\
 & \quad e_s = K_s[e_s'] * e_t = K_t[e_t'] * \mathrm{X}(\Psi, e_s', e_t') * \mathrm{I}(\sigma_s, \sigma_t) * \\
 & \quad \forall\, e_s'', e_t''.\, \Psi(e_s'', e_t'') \twoheadrightarrow \mathit{sim\text{-}inner}(\Phi, K_s[e_s''], K_t[e_t''])
\end{array}\right.$$

$$\text{sim-inner}_{\mathrm{X}} \coloneqq \lambda\, \mathit{sim}.\, \mu\, \mathit{sim\text{-}inner}.\, \text{sim-body}_{\mathrm{X}}(\mathit{sim}, \mathit{sim\text{-}inner})$$

$$\text{sim}_{\mathrm{X}} \coloneqq \nu\, \mathit{sim}.\, \text{sim-inner}_{\mathrm{X}}(\mathit{sim})$$

$$e_s \gtrsim e_t\ \langle \mathrm{X} \rangle\ [\Phi] \coloneqq \text{sim}_{\mathrm{X}}(\Phi, e_s, e_t)$$

$$e_s \gtrsim e_t\ \langle \mathrm{X} \rangle\ \{\Phi\} \coloneqq e_s \gtrsim e_t\ \langle \mathrm{X} \rangle\ \left[\lambda(e_s', e_t').\, \exists\, v_s, v_t.\, e_s' = v_s * e_t' = v_t * \Phi(v_s, v_t)\right]$$

TMC protocol

$$
\begin{aligned}
\mathrm{X}_{\mathrm{dir}}(\Psi, e_s, e_t) \quad &:= \quad \exists\, f, v_s, v_t. \\
&\quad f \in \mathrm{dom}(p_s) * \\
&\quad e_s = @f\ v_s * e_t = @f\ v_t * v_s \approx v_t * \\
&\quad \forall\, v_s', v_t'.\, v_s' \approx v_t' \mathrel{-\!\!*} \Psi(v_s', v_t')
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{X}_{\mathrm{DPS}}(\Psi, e_s, e_t) \quad &:= \quad \exists\, f, f_{dps}, v_s, \ell, i, v_t. \\
&\quad f \in \mathrm{dom}(p_s) * \xi[f] = f_{dps} * \\
&\quad e_s = @f\ v_s * e_t = @f_{dps}\ ((\ell\,,\, i)\,,\, v_t) * v_s \approx v_t * \\
&\quad (\ell + i) \mapsto \blacksquare * \\
&\quad \forall\, v_s', v_t'.\, (\ell + i) \mapsto v_t' * v_s' \approx v_t' \mathrel{-\!\!*} \Psi(v_s', ())
\end{aligned}
$$

$$
\mathrm{X}_{\mathrm{TMC}} \quad := \quad \mathrm{X}_{\mathrm{dir}} \sqcup \mathrm{X}_{\mathrm{DPS}}
$$