# Simply `RaTT`

A Fitch Style Modal Calculus for Reactive Programming Without Space Leaks.

Patrick Bahr, **Christian Graulund**, Rasmus Møgelberg (The `RaTT` pack)

*IT University of Copenhagen*

Types 2019

# Simply RaTT: An overview

**Simply RaTT:**

- A simply typed calcus for reactive programming.
- Novel Fitch style approach to modal FRP.
- Uses guarded recursion to define reactive programs.

**Contributions:**

- Heap-based operational semantics for streams and stream transducers.
- Disallow (implicit) space leaks *by construction.*
- Type system that ensures safety, causality and productivity.

**Goal:**

- Full dependent type theory for reactive programming (`RaTT`).

# Functional Reactive Programming[1]

- In general, programming with streams:

$$\mathsf{Stream}\ \mathsf{A} \cong \mathsf{A} \times (\mathsf{Stream}\ \mathsf{A})$$

- Programs are *stream transducers*:

$$\mathsf{prog} : \mathsf{Stream}\ \mathsf{A} \to \mathsf{Stream}\ \mathsf{B}$$

- Many application: Control systems, GUIs, music, games . . . .
- Has an explicit model of time.
- Problems: causality, productivity, time and space leaks . . . .

[1] Elliott & Hudak, ICFP'97

# Examples

$$\text{noncausal} : \text{Str Nat} \to \text{Str Nat}$$

$$\text{noncausal } ns = \text{head}(\text{tail } ns) :: \text{noncausal } (\text{tail } ns)$$

$$\text{unprod} : \text{Str Nat} \to \text{Str Nat}$$

$$\text{unprod } ns = \text{tail } (\text{unprod } (\text{tail } ns))$$

$$\text{leaky} : \text{Str Nat} \to \text{Str } (\text{Str Nat})$$

$$\text{leaky } ns = ns :: \text{leaky } (\text{tail } ns)$$

Known solution: Restrict direct use of streams (e.g. arrows)

# One solution using streams: Modal FRP[2]

- Modal FRP = FRP + modal types + guarded fixed point
- Add modal type former $\bigcirc$ pronounced *"Later"* or *"Delay"*.
- $\bigcirc A$ denotes *"A one time step from now"*.
- Uses Nakano style *guarded recursion*:

$$\frac{\Gamma, x : \bigcirc A \vdash t : A}{\Gamma \vdash \mathrm{fix}\, x.t : A}$$

- The use of $\bigcirc$ ensures productivity and causality.
- We now work with *guarded streams*:

$$\text{Stream A} \cong \text{A} \times \bigcirc(\text{Stream A})$$

[2] Jeffrey, PLPV'12. Jeltsch 2011.

# Let considered harmful

Traditional: Dual contexts

$$\frac{\overbrace{\Theta}^{\textit{now}} \mid \overbrace{\emptyset}^{\textit{later}} \vdash t : A}{\underbrace{\Gamma}_{\textit{now}} \mid \underbrace{\Theta}_{\textit{later}} \vdash \mathrm{delay}(t) : \bigcirc A}$$

$$\frac{\begin{array}{c}\Gamma \mid \Theta \vdash t : \bigcirc A \\ \Gamma \mid \Theta, x : A \vdash t' : C\end{array}}{\Gamma \mid \Theta \vdash \mathrm{let}\ x = t\ \mathrm{in}\ t' : C}$$

$$\frac{}{\Gamma, x : A, \Gamma' \mid \Theta \vdash x : A}$$

# Let considered harmful

Traditional: Dual contexts

$$\frac{\overbrace{\Theta}^{\textit{now}} \mid \overbrace{\emptyset}^{\textit{later}} \vdash t : A}{\underbrace{\Gamma}_{\textit{now}} \mid \underbrace{\Theta}_{\textit{later}} \vdash \mathsf{delay}(t) : \bigcirc A}$$

$$\frac{\Gamma \mid \Theta \vdash t : \bigcirc A \qquad \Gamma \mid \Theta, x : A \vdash t' : C}{\Gamma \mid \Theta \vdash \mathsf{let}\ x = t\ \mathsf{in}\ t' : C}$$

$$\frac{}{\Gamma, x : A, \Gamma' \mid \Theta \vdash x : A}$$

Modern: Fitch Style

$$\frac{\Gamma, \checkmark \vdash t : A}{\Gamma \vdash \mathsf{delay}(t) : \bigcirc A}$$

$$\frac{\overbrace{\Gamma}^{\textit{now}} \vdash t : \bigcirc A}{\underbrace{\Gamma}_{\textit{earlier}}, \checkmark, \underbrace{\Gamma'}_{\textit{now}} \vdash \mathsf{adv}(t) : A}$$

$$\frac{\checkmark\text{-free}(\Gamma')}{\Gamma, x : A, \Gamma' \vdash x : A}$$

# Guarded *Reactive* Fixed Points

- Standard fixed point typing:

$$\frac{\Gamma, x : \bigcirc A \vdash t : A}{\Gamma \vdash \mathrm{fix}\ x.t : A}$$

- Problem: Unfolding is not well-typed.

$$\mathrm{fix}\ x.t \rightsquigarrow t(\mathrm{delay}(\mathrm{fix}\ x.t))$$

- In general we don't have $A \to \bigcirc A$
- Recursive programs need to call themselves *in the future*.
- Hence, need types that are available at any time step.

# Stable types

- Stable types have the typing rules:

$$\frac{\Gamma, \sharp \vdash t : A}{\Gamma \vdash \mathrm{box}(t) : \Box A} \qquad \frac{\Gamma \vdash t : \Box A}{\Gamma, \sharp, \Gamma' \vdash \mathrm{unbox}(t) : A}$$

- In general, we do have $\Box A \to \Box \bigcirc A$.
- New fixed point combinator:

$$\frac{\Gamma, \sharp, x : \bigcirc A \vdash t : A}{\Gamma \vdash \mathrm{fix}\ x.t : \Box A}$$

- Crucially, the recursive variable is *after the lock*.

# Operational Semantics

- Work with heap based[3] operational semantics:

$$\langle t; \underbrace{\eta_N}_{\textit{now}} \checkmark \underbrace{\eta_L}_{\textit{later}} \rangle \Downarrow \langle v; \eta'_N \checkmark \eta'_L \rangle$$

- After evaluation, we garbage collect *the entire now heap:*

$$\frac{\langle t : \mathsf{Str}\ A; \eta\checkmark \rangle \Downarrow \langle v :: l; \eta_N \checkmark \eta_L \rangle}{\langle t; \eta \rangle \stackrel{v}{\Longrightarrow} \langle \mathsf{adv}(l); \eta_L \rangle}$$

- For stream transducers ($\mathsf{f} : \mathsf{Str}\ A \to \mathsf{Str}\ B$), the input stream is allocated on the heap, hence, garbage collected.
- Theorem: Evaluation of streams and transducers is safe.

[3] Following Krishnaswami ICFP'13

# Time Leaks

- Computation on tail of stream will never evaluate fully.

$$\begin{aligned}&\mathsf{leakyNats} : \mathsf{Str\ Nat}\\&\mathsf{leakyNats} = \mathsf{fix}\ ns.\ 0 :: \underbrace{\mathsf{delay}\ (\mathsf{map}\ {+1})\ ns}_{\bigcirc \mathsf{Str\ Nat}}\end{aligned}$$

- We (roughly) have the unfolding:

$$\begin{aligned}&\quad 0 :: (\mathsf{map}\ {+1})\ ns\\&\rightsquigarrow 0 :: (\mathsf{map}\ {+1})\ (0 :: (\mathsf{map}\ {+1})\ ns)\\&\rightsquigarrow 0 :: 1 :: (\mathsf{map}\ {+1})\ (\mathsf{map}\ {+1})\ (0 :: (\mathsf{map}\ {+1})\ ns)\\&\rightsquigarrow 0 :: \cdots :: n-1 :: (\mathsf{map}\ {+1})^n\ (0 :: (\mathsf{map}\ {+1})\ ns)\end{aligned}$$

- To compute $n$ we need to recompute $(n-1)$ elements!

# Time Leaks II

- Solution: Disallow fixed points under delay.

$$\frac{\Gamma, \sharp, x : \bigcirc A \vdash t : A}{\Gamma \vdash \mathrm{fix}\ x.t : \Box A} \qquad \frac{\Gamma, \checkmark \vdash t : A}{\Gamma \vdash \mathrm{delay}\ t : \bigcirc A}$$

$$\frac{\Gamma \vdash t : \Box A \qquad \checkmark\text{-free}(\Gamma')}{\Gamma, \sharp, \Gamma' \vdash \mathrm{unbox}\ t : A}$$

- Solution is *unique to Fitch style approach*.
- We now use explicit buffering:

$$\begin{array}{ll} \mathrm{nats} : \mathrm{Str\ Nat} & \mathrm{from} : \mathrm{Nat} \to \mathrm{Str}\ Nat \\ \mathrm{nats} = \mathrm{from}\ 0 & \mathrm{from} = \mathrm{fix}\ f.\ \lambda n.\ n :: \mathrm{f} \circledast \mathrm{delay}(n+1) \end{array}$$

# Conclusion

- Fitch style approach to Modal FRP.
- Heap-based operational semantics that rules out space leaks.
- Type system that ensures memory safety, causality and productivity.
- Rule out a common source of time leaks.
- Preprint on ArXiv (1903.05879).
- Formalized meta-theory in Coq.

# Thank you

Thank you! Questions?

# Stream Transducer:

- For $t : \mathsf{Str}\ A \to \mathsf{Str}\ B$:

$$\frac{\left\langle t; \sharp\eta, l^* \mapsto v :: l^* \checkmark l^* \mapsto \langle\rangle \right\rangle \Downarrow \left\langle v' :: l; \sharp\eta_N \checkmark \eta_L, l^* \mapsto \langle\rangle \right\rangle}{\langle t; \eta \rangle \overset{v/v'}{\Longrightarrow} \langle \mathsf{adv}\ l; \eta_L \rangle}$$

- Head of input stream is stored on heap and deleted after each iteration

# Typing Rules:

$$\frac{\Gamma, x : A, \Gamma' \vdash \qquad \textbf{token-free}(\Gamma')}{\Gamma, x : A, \Gamma' \vdash x : A}
\qquad
\frac{\Gamma \vdash}{\Gamma \vdash \langle\rangle : \mathbf{1}}
\qquad
\frac{n \in \mathbb{N}}{\Gamma \vdash \bar{n} : \mathbf{Nat}}$$

$$\frac{\Gamma \vdash s : \mathbf{Nat} \qquad \Gamma \vdash t : \mathbf{Nat}}{\Gamma \vdash s + t : \mathbf{Nat}}
\qquad
\frac{\Gamma, x : A \vdash t : B \qquad \textbf{tick-free}(\Gamma)}{\Gamma \vdash \lambda x.t : A \to B}
\qquad
\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash t' : A}{\Gamma \vdash t\,t' : B}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash t' : B}{\Gamma \vdash \left\langle t, t' \right\rangle : A \times B}
\qquad
\frac{\Gamma \vdash t : A_{\mathbf{1}} \times A_{\mathbf{2}} \qquad i \in \{1, 2\}}{\Gamma \vdash \pi_i\, t : A_i}
\qquad
\frac{\Gamma \vdash t : A_i \qquad i \in \{1, 2\}}{\Gamma \vdash \mathbf{in}_i\, t : A_{\mathbf{1}} + A_{\mathbf{2}}}$$

$$\frac{\Gamma, x : A_i \vdash t_i : B \qquad \Gamma \vdash t : A_{\mathbf{1}} + A_{\mathbf{2}} \qquad i \in \{1, 2\}}{\Gamma \vdash \mathbf{case}\ t\ \mathbf{of}\ \mathbf{in}_{\mathbf{1}}\, x.t_{\mathbf{1}} ; \mathbf{in}_{\mathbf{2}}\, x.t_{\mathbf{2}} : B}
\qquad
\frac{\Gamma, \checkmark \vdash t : A}{\Gamma \vdash \mathbf{delay}\ t : \bigcirc A}$$

$$\frac{\Gamma \vdash t : \bigcirc A \qquad \Gamma, \checkmark, \Gamma' \vdash}{\Gamma, \checkmark, \Gamma' \vdash \mathbf{adv}\ t : A}
\qquad
\frac{\Gamma \vdash t : \Box A \qquad \textbf{token-free}(\Gamma')}{\Gamma, \sharp, \Gamma' \vdash \mathbf{unbox}\ t : A}
\qquad
\frac{\Gamma, \sharp \vdash t : A}{\Gamma \vdash \mathbf{box}\ t : \Box A}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma, \checkmark, \Gamma' \vdash \qquad A\ \textbf{stable}}{\Gamma, \checkmark, \Gamma' \vdash \mathbf{progress}\ t : A}
\qquad
\frac{\Gamma \vdash t : A \qquad \Gamma, \sharp, \Gamma' \vdash \qquad A\ \textbf{stable}}{\Gamma, \sharp, \Gamma' \vdash \mathbf{promote}\ t : A}$$

$$\frac{\Gamma \vdash t : A[\bigcirc(\mu\alpha.A)/\alpha]}{\Gamma \vdash \mathbf{into}\ t : \mu\alpha.A}
\qquad
\frac{\Gamma \vdash t : \mu\alpha.A}{\Gamma \vdash \mathbf{out}\ t : A[\bigcirc(\mu\alpha.A)/\alpha]}
\qquad
\frac{\Gamma, \sharp, x : \bigcirc A \vdash t : A}{\Gamma \vdash \mathbf{fix}\ x.t : \Box A}$$

# Operational Semantics (selected rules):

$$\frac{}{\langle v; \sigma \rangle \Downarrow \langle v; \sigma \rangle} \qquad \frac{\langle t; \sigma \rangle \Downarrow \left\langle \lambda x.s; \sigma' \right\rangle \qquad \left\langle t'; \sigma' \right\rangle \Downarrow \left\langle v; \sigma'' \right\rangle \qquad \left\langle s[v/x]; \sigma'' \right\rangle \Downarrow \left\langle v'; \sigma''' \right\rangle}{\left\langle t\, t'; \sigma \right\rangle \Downarrow \left\langle v'; \sigma''' \right\rangle}$$

$$\frac{\sigma \neq \bot \qquad l = \mathbf{alloc}\,(\sigma)}{\langle \mathbf{delay}\; t; \sigma \rangle \Downarrow \langle l; \sigma, l \mapsto t \rangle} \qquad \frac{\langle t; \sharp\eta_N \rangle \Downarrow \left\langle l; \sharp\eta'_N \right\rangle \qquad \left\langle \eta'_N(l); \sharp\eta'_N \checkmark \eta_L \right\rangle \Downarrow \left\langle v; \sigma' \right\rangle}{\langle \mathbf{adv}\; t; \sharp\eta_N \checkmark \eta_L \rangle \Downarrow \left\langle v; \sigma' \right\rangle}$$

$$\frac{\langle t; \bot \rangle \Downarrow \langle v; \bot \rangle \qquad \sigma \neq \bot}{\langle \mathbf{promote}\; t; \sigma \rangle \Downarrow \langle v; \sigma \rangle} \qquad \frac{\langle t; \sharp\eta_N \rangle \Downarrow \left\langle v; \sharp\eta'_N \right\rangle}{\langle \mathbf{progress}\; t; \sharp\eta_N \checkmark \eta_L \rangle \Downarrow \left\langle v; \sharp\eta'_N \checkmark \eta_L \right\rangle}$$

$$\frac{\langle t; \bot \rangle \Downarrow \left\langle \mathbf{box}\; t'; \bot \right\rangle \qquad \left\langle t'; \sigma \right\rangle \Downarrow \left\langle v; \sigma' \right\rangle \qquad \sigma \neq \bot}{\langle \mathbf{unbox}\; t; \sigma \rangle \Downarrow \left\langle v; \sigma' \right\rangle} \qquad \frac{\langle t; \sigma \rangle \Downarrow \left\langle v; \sigma' \right\rangle}{\langle \mathbf{into}\; t; \sigma \rangle \Downarrow \left\langle \mathbf{into}\; v; \sigma' \right\rangle}$$

$$\frac{\langle t; \sigma \rangle \Downarrow \left\langle \mathbf{into}\; v; \sigma' \right\rangle}{\langle \mathbf{out}\; t; \sigma \rangle \Downarrow \left\langle v; \sigma' \right\rangle}$$

$$\frac{\langle t; \bot \rangle \Downarrow \left\langle \mathbf{fix}\; x.t'; \bot \right\rangle \qquad \left\langle t'[l/x]; \sigma, l \mapsto \mathbf{unbox}(\mathbf{fix}\; x.t') \right\rangle \Downarrow \left\langle v; \sigma' \right\rangle \qquad \sigma \neq \bot \qquad l = \mathbf{alloc}\,(\sigma)}{\langle \mathbf{unbox}\; t; \sigma \rangle \Downarrow \left\langle v; \sigma' \right\rangle}$$

# SafeNats

$$\mathit{from} : \Box(\mathsf{Nat} \to \mathsf{Str\ Nat})$$

$$\mathit{from} = \mathsf{fix}\ f.\lambda(n : \mathsf{Nat}).n :: \mathsf{delay}((\mathsf{adv}\ f)\ (\mathsf{progress}\ n))$$

$$\mathit{nats} : \Box(\mathsf{Str\ Nat})$$

$$\mathit{nats} = \mathsf{box}(\mathsf{unbox}(\mathit{from})\ \mathsf{promote}(0))$$

# LeakyNats

$$
\begin{aligned}
& \langle \mathsf{unbox\ leakyNats}; \emptyset \rangle \\
\overset{\bar{0}}{\Longrightarrow}\quad & \Big\langle \mathsf{adv}\ l_1';\ l_1 \mapsto \mathsf{unbox\ leakyNats},\ l_1' \mapsto \mathsf{unbox\ map}\,(\mathsf{box}\,\lambda x.x+\bar{1})\,(\mathsf{adv}\ l_1) \Big\rangle \\
\overset{\bar{1}}{\Longrightarrow}\quad & \left\langle \mathsf{adv}\ l_2^3;\ \begin{array}{ll} l_2^0 \mapsto \mathsf{unbox\ leakyNats}, & l_2^1 \mapsto \mathsf{unbox\ map}\,(\mathsf{box}\,\lambda x.x+\bar{1})\,(\mathsf{adv}\ l_2^0), \\ l_2^2 \mapsto \mathsf{unbox\ step}, & l_2^3 \mapsto \mathsf{adv}\ l_2^2\,(\mathsf{adv}\,(\mathsf{tail}\,(\bar{0} :: l_2^1))) \end{array} \right\rangle \\
\overset{\bar{2}}{\Longrightarrow}\quad & \left\langle \mathsf{adv}\ l_3^5;\ \begin{array}{ll} l_3^0 \mapsto \mathsf{unbox\ leakyNats}, & l_3^1 \mapsto \mathsf{unbox\ map}\,(\mathsf{box}\,\lambda x.x+\bar{1})\,(\mathsf{adv}\ l_3^0), \\ l_3^2 \mapsto \mathsf{unbox\ step}, & l_3^3 \mapsto \mathsf{adv}\ l_3^2\,(\mathsf{adv}\,(\mathsf{tail}\,(\bar{0} :: l_3^1))) \\ l_3^4 \mapsto \mathsf{unbox\ step}, & l_3^5 \mapsto \mathsf{adv}\ l_3^4\,(\mathsf{adv}\,(\mathsf{tail}\,(\bar{1} :: l_3^3))) \end{array} \right\rangle \\
\vdots\quad &
\end{aligned}
$$

where $\mathsf{step} = \mathsf{fix}\, f.\lambda\, s.\mathsf{unbox}\,(\mathsf{box}\,\lambda\, n.n+\bar{1})\,(\mathsf{head}\ s) :: (f \circledast \mathsf{tail}\ s)$.