Modal Types for Asynchronous FRP
(Conditionally accepted for ICFP 2023)

Patrick Bahr **Rasmus Ejlers Møgelberg**

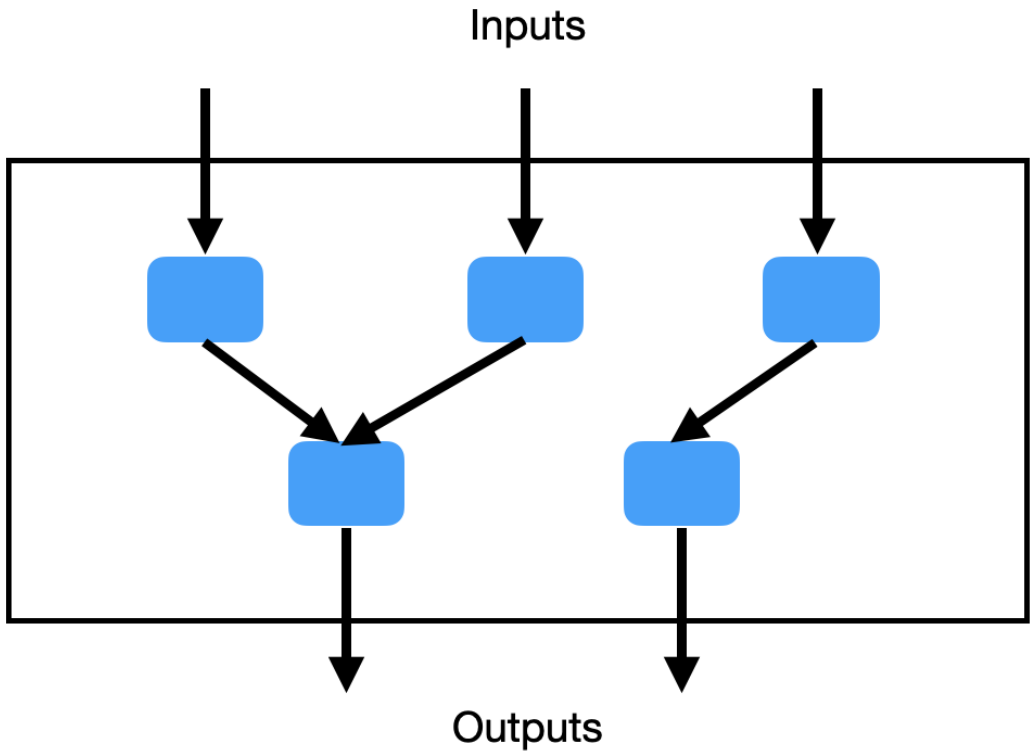
IT University of Copenhagen

Types, Valencia
June 12, 2023

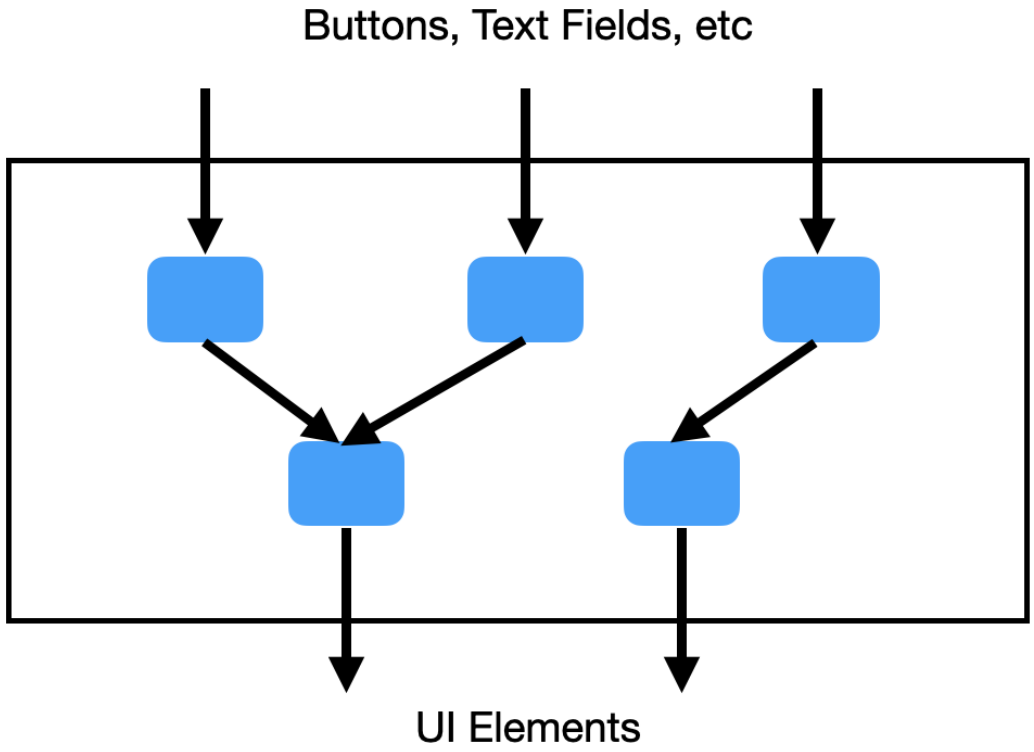
Overview

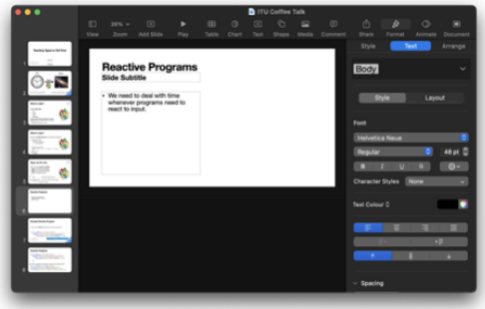
- Reactive programming
- Functional reactive programming (FRP)
- Modal FRP
- Asynchronous Modal FRP
- Async RaTT

Reactive programs



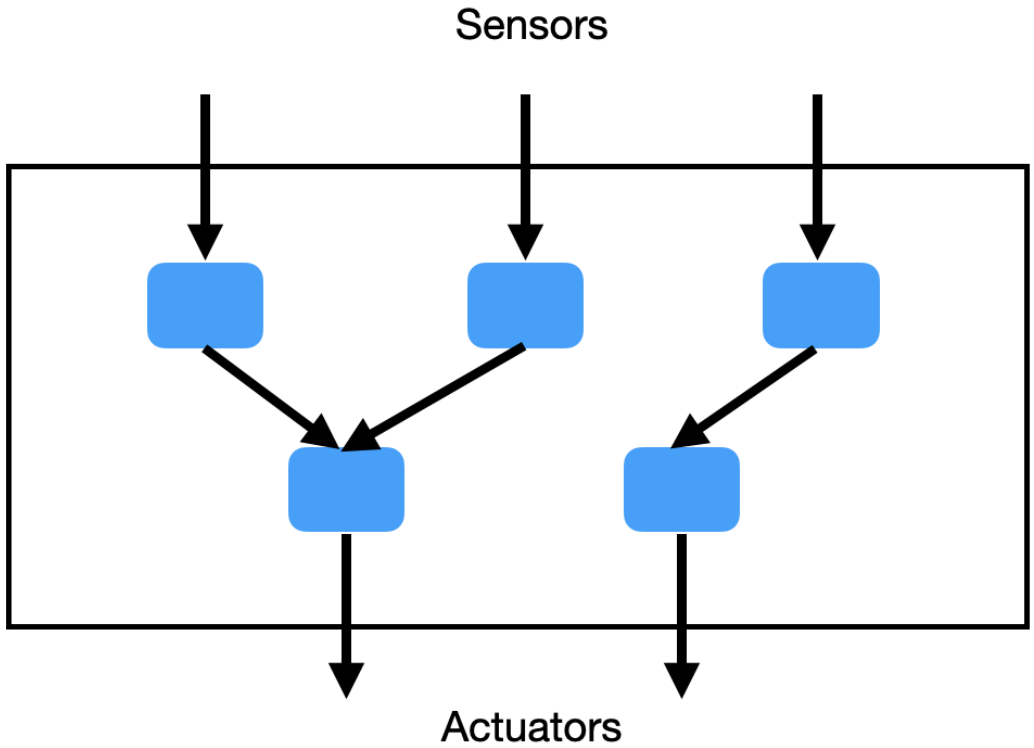
Reactive programs





GUI

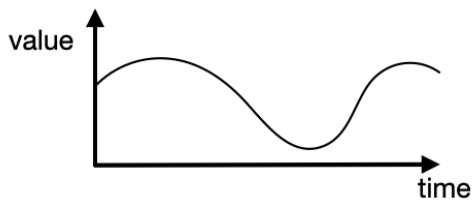
Reactive programs



Robot

Functional reactive programming (FRP)

- Signals



- Reactive programs as signal functions

$$\mathrm{Sig}(A_1) \times \cdots \times \mathrm{Sig}(A_n) \to \mathrm{Sig}(B_1) \times \cdots \times \mathrm{Sig}(B_m)$$

Functional reactive programming (FRP)

- Signals as streams?

$$\mathrm{Sig}(A) \cong A \times \mathrm{Sig}(A)$$

Functional reactive programming (FRP)

- Signals as streams?

$$\mathrm{Sig}(A) \cong A \times \mathrm{Sig}(A)$$

- Problems:
  - Causality

$$\mathit{oracle}\ (x :: y :: ys) = \text{if } x < y \text{ then } \mathit{Buy} :: (\mathit{oracle}\ (y :: ys)) \text{ else } \mathit{Sell} :: (\mathit{oracle}\ (y :: ys))$$

  - Productivity
  - Space and time leaks

Modal FRP

- Modality $\bigcirc$ represents time step

$$\mathrm{Sig}(A) \cong A \times \bigcirc \mathrm{Sig}(A)$$

- Represent stable data using $\square$ modality
- Guarded recursion

$$\mathrm{fix} : \square(\bigcirc A \to A) \to A$$

- Curry-Howard isomorphism to LTL
- Global notion of time

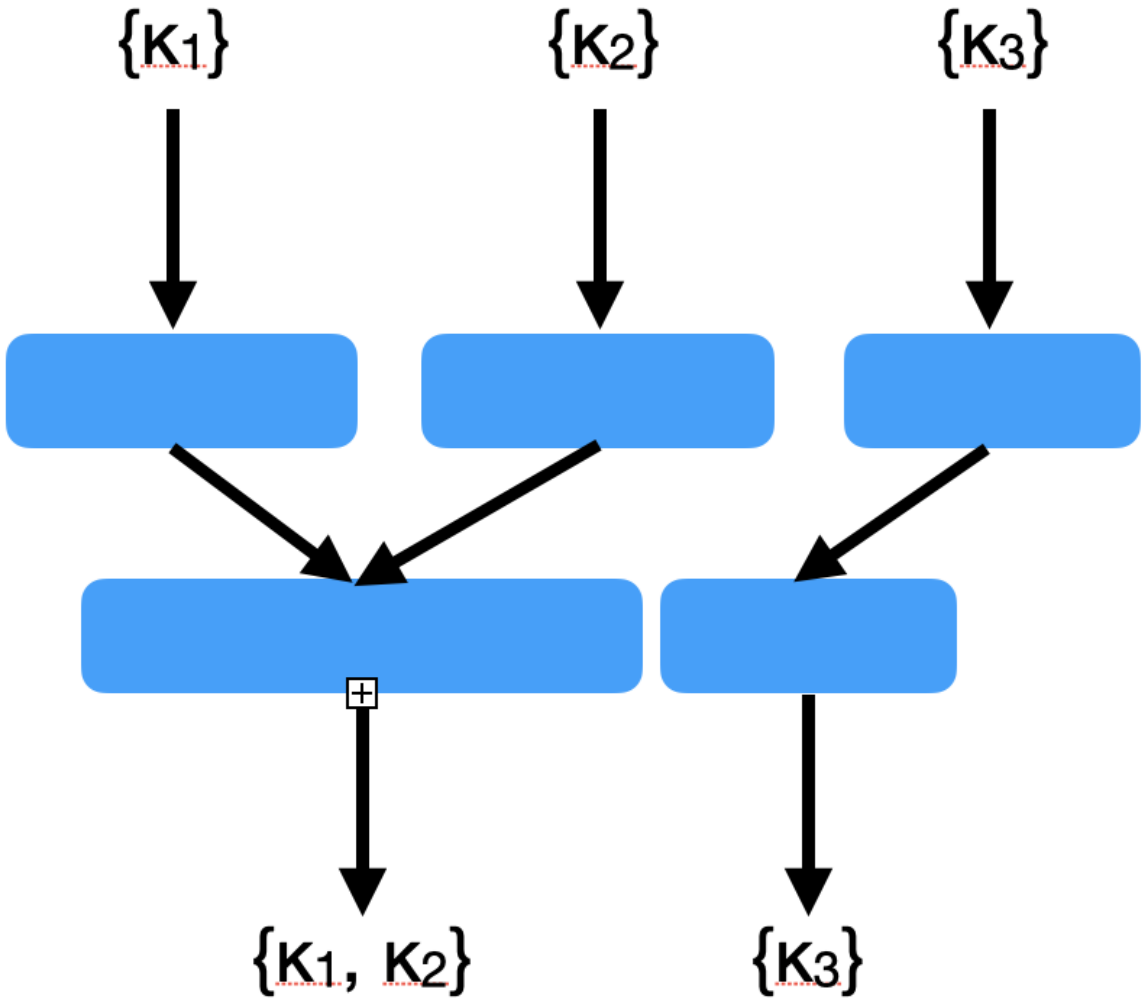
Asynchronous input/output

- Modelling asynchronous input/output

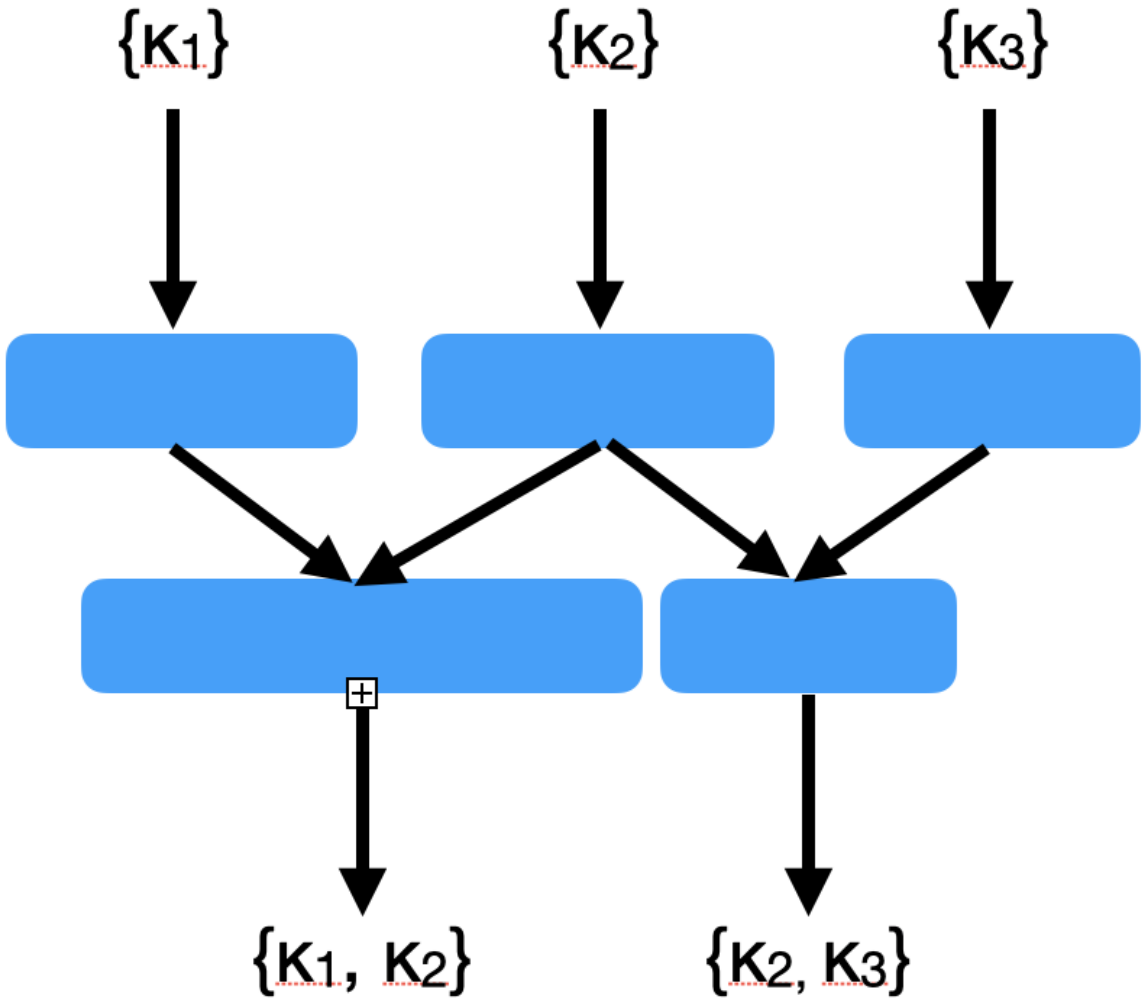
$$\Pi_i \mathrm{Sig}(\mathrm{Maybe}\, A_i) \to \Pi_j \mathrm{Sig}(\mathrm{Maybe}\, B_j)$$

- Problems
  - Efficiency
  - Abstraction barrier broken

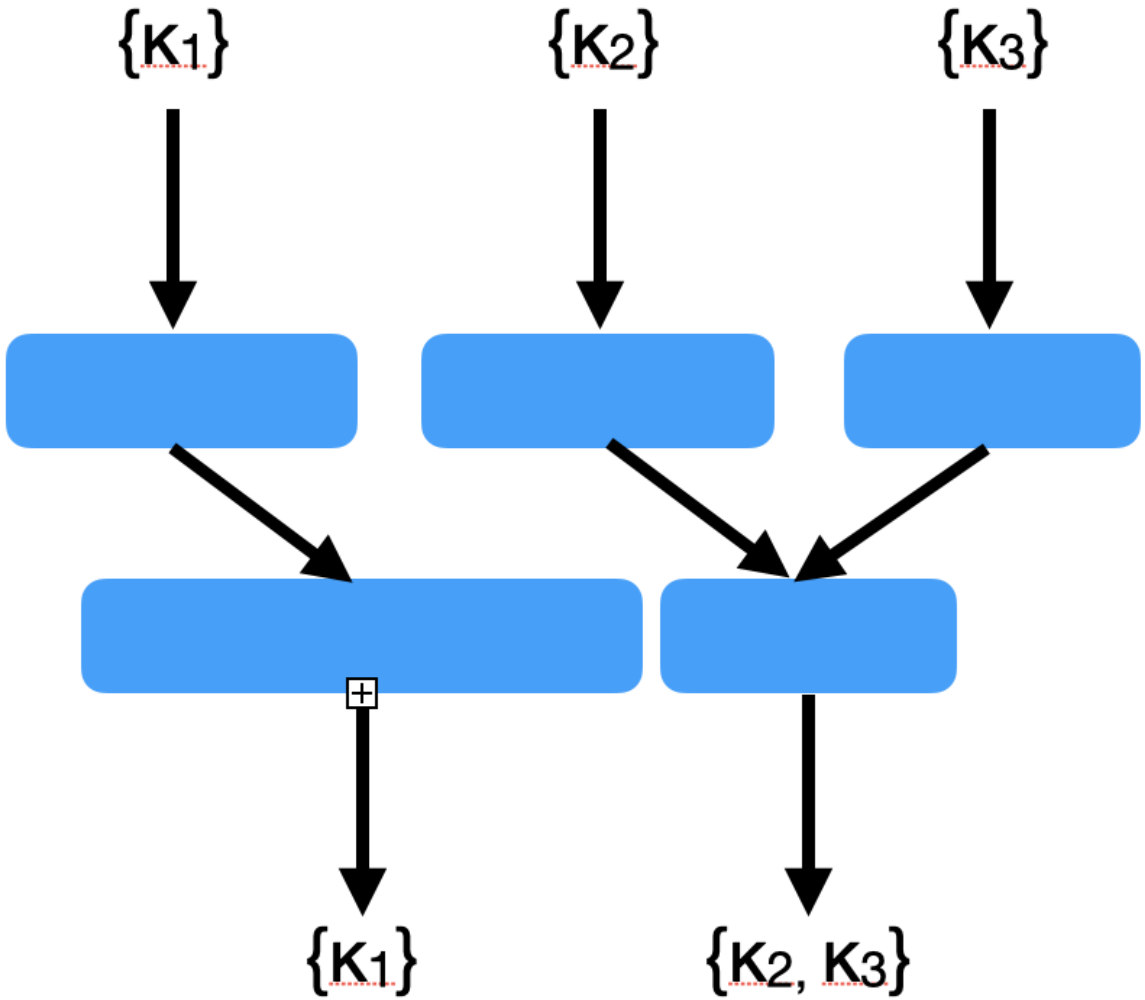
Flow graphs with clocks



Flow graphs with clocks



Flow graphs with clocks



A modality for asynchronous delay

- Asynchronous signals

$$\mathsf{Sig}(A) \cong A \times \textcircled{\exists}\mathsf{Sig}(A)$$

- Idea of type

$$\textcircled{\exists}A = \Sigma\kappa : \mathsf{Clock}.\bigcirc^{\kappa} A$$

A modality for asynchronous delay

- Asynchronous signals

$$\mathsf{Sig}(A) \cong A \times \textcircled{\exists}\mathsf{Sig}(A)$$

- Idea of type

$$\textcircled{\exists}A = \Sigma\kappa : \mathsf{Clock}.\bigcirc^{\kappa} A$$

- Input channel contexts Δ

$$\mathsf{keyPressed} :_{\mathsf{p}} \mathsf{Nat}, \mathsf{mouseCoord} :_{\mathsf{bp}} \mathsf{Nat} \times \mathsf{Nat}, \mathsf{time} :_{\mathsf{b}} \mathsf{Float}$$

- A *clock* is a set of input channels
- Complete programs

$$\vdash_{\Delta} t : \mathsf{Sig}(A_1) \times \cdots \times \mathsf{Sig}(A_n)$$

Async RaTT

Some typing rules

- Introduction and elimination for $\textcircled{\exists}$

$$\frac{\kappa :_c A \in \Delta \qquad c \in \{\mathsf{p}, \mathsf{bp}\}}{\Gamma \vdash_\Delta \mathsf{wait}_\kappa : \textcircled{\exists} A} \qquad \frac{\Gamma, \checkmark_\theta \vdash_\Delta t : A \qquad \Gamma \vdash_\Delta \theta : \mathsf{Clock}}{\Gamma \vdash_\Delta \mathsf{delay}_\theta\, t : \textcircled{\exists} A}$$

$$\frac{\Gamma \vdash_\Delta v : \textcircled{\exists} A \qquad \Gamma, \checkmark_{\mathsf{cl}(v)}, \Gamma' \vdash_\Delta}{\Gamma, \checkmark_{\mathsf{cl}(v)}, \Gamma' \vdash_\Delta \mathsf{adv}\, v : A}$$

Some typing rules

- Introduction and elimination for $\textcircled{\exists}$

$$\frac{\kappa :_c A \in \Delta \qquad c \in \{\mathsf{p}, \mathsf{bp}\}}{\Gamma \vdash_\Delta \mathsf{wait}_\kappa : \textcircled{\exists} A} \qquad \frac{\Gamma, \checkmark_\theta \vdash_\Delta t : A \qquad \Gamma \vdash_\Delta \theta : \mathsf{Clock}}{\Gamma \vdash_\Delta \mathsf{delay}_\theta\, t : \textcircled{\exists} A}$$

$$\frac{\Gamma \vdash_\Delta v : \textcircled{\exists} A \qquad \Gamma, \checkmark_{\mathsf{cl}(v)}, \Gamma' \vdash_\Delta}{\Gamma, \checkmark_{\mathsf{cl}(v)}, \Gamma' \vdash_\Delta \mathsf{adv}\, v : A}$$

- Recursion

$$\mathsf{fix} : \Box(\textcircled{\forall} A \to A) \to A$$

Examples

- Input channels as signals

$$\mathit{sigAwait}_\kappa : \ominus\,(\text{Sig } A)$$
$$\mathit{sigAwait}_\kappa = \text{delay } (\text{adv wait}_\kappa :: \mathit{sigAwait}_\kappa)$$

- Map

$$\mathit{map} : \Box\,(A \to B) \to \text{Sig } A \to \text{Sig } B$$
$$\mathit{map}\ f\ (x :: xs) = \text{unbox } f\ x :: \text{delay } (\mathit{map}\ f\ (\text{adv } xs))$$

Synchronisation

- The synchronous $\bigcirc$ is an applicative functor

$$\mathit{applicative} : \bigcirc\,(A \to B) \to \bigcirc\,A \to \bigcirc\,B$$
$$\mathit{applicative}\ f\ x = \text{delay}\ (\text{adv}\ f\ (\text{adv}\ x))$$

- The asynchronous $\ominus$ is not!

[1] Graulund, Szamozvancev and Krishnaswami: *Adjoint Reactive GUI Programming.* FoSSaCS, 2021

Synchronisation

- The synchronous $\bigcirc$ is an applicative functor

$$\mathit{applicative} : \bigcirc\,(A \to B) \to \bigcirc\,A \to \bigcirc\,B$$
$$\mathit{applicative}\ f\ x = \text{delay}\ (\text{adv}\ f\ (\text{adv}\ x))$$

- The asynchronous $\textcircled{\exists}$ is not!
- Synchronisation operator [1]

$$\text{sync} : \textcircled{\exists} A_1 \to \textcircled{\exists} A_2 \to \textcircled{\exists}((A_1 \times \textcircled{\exists} A_2) + (\textcircled{\exists} A_1 \times A_2) + (A_1 \times A_2))$$

[1] Graulund, Szamozvancev and Krishnaswami: *Adjoint Reactive GUI Programming.* FoSSaCS, 2021

Synchronisation

- The synchronous $\bigcirc$ is an applicative functor

$$\begin{aligned} &\mathit{applicative} : \bigcirc\,(A \to B) \to \bigcirc\,A \to \bigcirc\,B \\ &\mathit{applicative}\ f\ x = \mathsf{delay}\ (\mathsf{adv}\ f\ (\mathsf{adv}\ x)) \end{aligned}$$

- The asynchronous $\textcircled{\exists}$ is not!
- Synchronisation operator [1]

$$\mathsf{sync} : \textcircled{\exists} A_1 \to \textcircled{\exists} A_2 \to \textcircled{\exists}((A_1 \times \textcircled{\exists} A_2) + (\textcircled{\exists} A_1 \times A_2) + (A_1 \times A_2))$$

- Typing rule

$$\frac{\begin{array}{c}\Gamma \vdash_\Delta v_1 : \textcircled{\exists} A_1 \qquad \Gamma \vdash_\Delta v_2 : \textcircled{\exists} A_2 \\ \vdash \theta_1 \sqcup \theta_2 = \mathsf{cl}\,(v_1) \sqcup \mathsf{cl}\,(v_2) \qquad \Gamma, \checkmark_{\theta_1 \sqcup \theta_2}, \Gamma' \vdash_\Delta\end{array}}{\Gamma, \checkmark_{\theta_1 \sqcup \theta_2}, \Gamma' \vdash_\Delta \mathsf{select}\ v_1\ v_2 : ((A_1 \times \textcircled{\exists} A_2) + (\textcircled{\exists} A_1 \times A_2)) + (A_1 \times A_2)}$$

[1] Graulund, Szamozvancev and Krishnaswami: *Adjoint Reactive GUI Programming.* FoSSaCS, 2021

Examples

$$
\begin{aligned}
&\mathit{zip} : \text{stable } A, B \Rightarrow \text{Sig } A \to \text{Sig } B \to \text{Sig } (A \times B) \\
&\mathit{zip}\ (x :: xs)\ (y :: ys) = (x, y) :: \text{delay } (\text{case select } xs\ ys \text{ of} \\
&\qquad\qquad \mathit{Left}\ \ xs'\ ys'.\mathit{zip}\ xs'\ (y :: ys') \\
&\qquad\qquad \mathit{Right}\ xs'\ ys'.\mathit{zip}\ (x :: xs')\ ys' \\
&\qquad\qquad \mathit{Both}\ \ xs'\ ys'.\mathit{zip}\ xs'\ ys')
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{switch} : \text{Sig } A \to \text{Ⓔ } (\text{Sig } A) \to \text{Sig } A \\
&\mathit{switch}\ (x :: xs)\ d = x :: \text{delay } (\text{case select } xs\ d \text{ of} \\
&\qquad\qquad \mathit{Left}\ \ xs'\ d'.\mathit{switch}\ xs'\ d' \\
&\qquad\qquad \mathit{Right}\ \_\ \ d'.d' \\
&\qquad\qquad \mathit{Both}\ \ xs'\ d'.d')
\end{aligned}
$$

Operational semantics and results

Machine: Initialisation

$$\frac{\langle\langle t\rangle ; \emptyset\rangle \Downarrow^{\iota} \langle\langle v_1 :: l_1, \ldots, v_m :: l_m\rangle ; \eta\rangle}{\langle t; \iota\rangle \overset{x_1 \mapsto v_1, \ldots, x_m \mapsto v_m}{\Longrightarrow} \langle x_1 \mapsto l_1, \ldots, x_m \mapsto l_m; \eta; \iota\rangle}$$

Machine: Taking input

$$\frac{\iota' = \iota[\kappa \mapsto v] \text{ if } \kappa \in \mathsf{dom}\,(\iota) \text{ otherwise } \iota' = \iota}{\langle N; \eta; \iota \rangle \overset{\kappa \mapsto v}{\Longrightarrow} \left\langle N; [\eta]_{\kappa \in} \langle \kappa \mapsto v \rangle [\eta]_{\kappa \notin}; \iota' \right\rangle}$$

Machine: Updating output channels

$$\frac{\kappa \notin \mathsf{cl}\,(l) \qquad \langle N; \eta_N \langle \kappa \mapsto v \rangle \eta_L; \iota \rangle \overset{O}{\Longrightarrow} \langle N'; \eta; \iota \rangle}{\langle x \mapsto l, N; \eta_N \langle \kappa \mapsto v \rangle \eta_L; \iota \rangle \overset{O}{\Longrightarrow} \langle x \mapsto l, N'; \eta; \iota \rangle}$$

$$\frac{\kappa \in \mathsf{cl}\,(l) \quad \langle \mathsf{adv}\, l; \eta_N \langle \kappa \mapsto v \rangle \eta_L \rangle \Downarrow^{\iota} \langle v' :: l'; \sigma \rangle \quad \langle N; \sigma; \iota \rangle \overset{O}{\Longrightarrow} \langle N'; \eta; \iota \rangle}{\langle x \mapsto l, N; \eta_N \langle \kappa \mapsto v \rangle \eta_L; \iota \rangle \overset{x \mapsto v', O}{\Longrightarrow} \langle x \mapsto l', N'; \eta; \iota \rangle}$$

Machine: Garbage collection

$$\frac{}{\langle \cdot ; \eta_N \langle \kappa \mapsto v \rangle \eta_L ; \iota \rangle \stackrel{\cdot}{\Longrightarrow} \langle \cdot ; \eta_L ; \iota \rangle}$$

Theorem

Theorem. Given a reactive program $t : \Delta \Rightarrow \Gamma_{\text{out}}$, well-typed input values $\vdash \kappa_i \mapsto v_i : \Delta$ for all $i \in \mathbb{N}$, and a well-typed initial input buffer $\vdash \iota_0 : \Delta$, there is an infinite transition sequence

$$\langle t; \iota_0 \rangle \xRightarrow{O_0} \langle N_0; \eta_0; \iota_0 \rangle \xRightarrow{\kappa_0 \mapsto v_0} \langle N_0; \sigma_0; \iota_1 \rangle \xRightarrow{O_1} \langle N_1; \eta_1; \iota_1 \rangle \xRightarrow{\kappa_1 \mapsto v_1} \ldots$$

with $\vdash O_i : \Gamma_{\text{out}}$ for all $i \in \mathbb{N}$.

- Programs are causal and productive
- No (implicit) space leaks
- Proof: Kripke logical relation


That's all Folks!