SPLS, 7 Jun 2023, University of the West of Scotland

ASYNCHRONOUS **OPERATIONS**

Danel Ahman **Matija Pretnar**

University of Ljubljana, Slovenia

THE **IDEA**

M_1

$$M_1 \rightsquigarrow M_2$$

$$M_1 \rightsquigarrow M_2 \rightsquigarrow M_3$$

$$M_1 \rightsquigarrow M_2 \rightsquigarrow M_3 \rightsquigarrow M_4$$

$$M_1 \rightsquigarrow M_2 \rightsquigarrow M_3 \rightsquigarrow M_4 \rightsquigarrow M_5$$

M_1

$$M_1 \rightsquigarrow M_2$$

$$\uparrow_{\mathrm{op}}$$

$$M_1 \rightsquigarrow M_2$$

$$\begin{array}{ccccc} & & \uparrow_{\mathsf{op}} & & \downarrow_{\mathsf{res}} \\ \boldsymbol{M}_1 & \rightsquigarrow & \boldsymbol{M}_2 & & \boldsymbol{M}_3 \end{array}$$

$$\uparrow_{\mathsf{op}} \qquad\qquad \downarrow_{\mathsf{res}}$$

$$M_1 \rightsquigarrow M_2 \qquad\qquad M_3 \rightsquigarrow M_4$$

$$M_1 \rightsquigarrow M_2 \qquad M_3 \rightsquigarrow M_4 \rightsquigarrow M_5$$

$$\uparrow_{\mathsf{op}} \qquad \downarrow_{\mathsf{res}}$$

M_1

$$M_1 \rightsquigarrow M_s$$

$$M_1 \rightsquigarrow \overset{\uparrow_{\mathsf{req}}}{M_s}$$

$$M_1 \rightsquigarrow M_s \rightsquigarrow M_2$$

$$\uparrow_{\mathsf{req}}$$

$$\begin{array}{c} \quad\ \uparrow_{\mathsf{req}} \\ M_1 \rightsquigarrow M_s \rightsquigarrow M_2 \rightsquigarrow M_3 \end{array}$$

$$M_1 \rightsquigarrow \overset{\uparrow_{\mathsf{req}}}{M_s} \rightsquigarrow M_2 \rightsquigarrow M_3 \rightsquigarrow M_4$$

$$M_1 \rightsquigarrow \overset{\uparrow_{\mathsf{req}}}{M_s} \rightsquigarrow M_2 \rightsquigarrow M_3 \rightsquigarrow M_4$$

$$M_1 \rightsquigarrow \underset{\uparrow_{\mathsf{req}}}{M_s} \rightsquigarrow M_2 \rightsquigarrow \underset{\downarrow_{\mathsf{resp}}}{M_3} \rightsquigarrow M_4$$

$$
\begin{tikzcd}
 & \uparrow_{\mathsf{req}} & & \downarrow_{\mathsf{resp}} & \\
M_1 \arrow[r, rightsquigarrow] & M_s \arrow[r, rightsquigarrow] & M_2 \arrow[r, rightsquigarrow] & M_3 \arrow[r, rightsquigarrow] \arrow[dr, rightsquigarrow] & M_4 \\
 & & & & M_p
\end{tikzcd}
$$

M_1

$$M_1 \rightsquigarrow M_w$$

$$\boldsymbol{M}_1 \rightsquigarrow \overset{\downarrow_{\text{req}}}{\boldsymbol{M}_w}$$

$$\begin{array}{c} \qquad\downarrow_{\mathsf{req}} \qquad\qquad \\ \boldsymbol{M}_1 \rightsquigarrow \boldsymbol{M}_w \rightsquigarrow \boldsymbol{M}_p \end{array}$$

$$\boldsymbol{M}_1 \rightsquigarrow \overset{\downarrow_{\mathsf{req}}}{\boldsymbol{M}_w} \rightsquigarrow \boldsymbol{M}_p \rightsquigarrow \boldsymbol{M}_s$$

$$
\begin{array}{cccc}
 & \downarrow_{\mathsf{req}} & & \uparrow_{\mathsf{resp}} \\
\boldsymbol{M}_1 \rightsquigarrow & \boldsymbol{M}_w \rightsquigarrow & \boldsymbol{M}_p \rightsquigarrow & \boldsymbol{M}_s
\end{array}
$$

$$M_1 \rightsquigarrow \underset{}{\overset{\downarrow_{\mathsf{req}}}{M_w}} \rightsquigarrow M_p \rightsquigarrow \overset{\uparrow_{\mathsf{resp}}}{M_s} \rightsquigarrow M_w$$

M_1

$$M_1 \rightsquigarrow M_2$$

$$\downarrow_{\mathsf{stop}}$$

$$M_1 \rightsquigarrow M_2$$

$$\begin{array}{c} \downarrow_{\text{stop}} \\ M_1 \rightsquigarrow M_2 \rightsquigarrow M_{w_2} \end{array}$$

$$M_1 \rightsquigarrow \underset{\downarrow \text{stop}}{M_2} \rightsquigarrow \underset{\downarrow \text{go}}{M_{w_2}}$$

$$M_1 \rightsquigarrow \underset{}{\overset{\downarrow \mathsf{stop}}{M_2}} \rightsquigarrow \overset{\downarrow \mathsf{go}}{M_{w_2}} \rightsquigarrow M_2$$

$$M_1 \rightsquigarrow \overset{\downarrow \mathsf{stop}}{M_2} \rightsquigarrow \overset{\downarrow \mathsf{go}}{M_{w_2}} \rightsquigarrow M_2 \rightsquigarrow M_3$$

M_1

$$\uparrow_{\text{tick}}$$
$$M_1$$

$$\overset{\uparrow_{\mathsf{tick}}}{M_1} \rightsquigarrow M_2$$

$$\overset{\uparrow_{\mathsf{tick}}}{M_1} \rightsquigarrow \overset{\uparrow_{\mathsf{tock}}}{M_2}$$

$$\underset{\uparrow_{\text{tick}}}{M_1} \rightsquigarrow \underset{\uparrow_{\text{tock}}}{M_2} \rightsquigarrow M_1$$

$$\underset{\uparrow_{\mathsf{tick}}}{M_1} \rightsquigarrow \underset{\uparrow_{\mathsf{tock}}}{M_2} \rightsquigarrow \underset{\uparrow_{\mathsf{tick}}}{M_1}$$

$$\overset{\uparrow_{\text{tick}}}{M_1} \rightsquigarrow \overset{\uparrow_{\text{tock}}}{M_2} \rightsquigarrow \overset{\uparrow_{\text{tick}}}{M_1} \rightsquigarrow M_2$$

$$\overset{\uparrow_{\text{tick}}}{M_1} \rightsquigarrow \overset{\uparrow_{\text{tock}}}{M_2} \rightsquigarrow \overset{\uparrow_{\text{tick}}}{M_1} \rightsquigarrow \overset{\uparrow_{\text{tock}}}{M_2}$$

$$\overset{\uparrow_{\mathsf{tick}}}{M_1} \rightsquigarrow \overset{\uparrow_{\mathsf{tock}}}{M_2} \rightsquigarrow \overset{\uparrow_{\mathsf{tick}}}{M_1} \rightsquigarrow \overset{\uparrow_{\mathsf{tock}}}{M_2} \rightsquigarrow M_1$$

THE **IDEA**

CORE

CALCULUS

$\text{fun } (x : X) \mapsto M \qquad V\, W$

$\text{return } V \qquad \text{let } x = M \text{ in } N$

$$(\textsf{fun}\ (x : X) \mapsto M)\ V \rightsquigarrow M[V/x]$$

$$\textsf{let}\ x = (\textsf{return}\ V)\ \textsf{in}\ N \rightsquigarrow N[V/x]$$

$$\frac{M \rightsquigarrow N}{\mathcal{E}[M] \rightsquigarrow \mathcal{E}[N]}$$

$$\mathcal{E} \ ::= \ [\ ] \ \Big| \ \textsf{let}\ x = \mathcal{E}\ \textsf{in}\ N \ \Big| \ \cdots$$

demo

CORE
CALCULUS

$\text{fun}\ (x : X) \mapsto M \qquad V\ W$

$\text{return}\ V \qquad \text{let}\ x = M\ \text{in}\ N$

OUTGOING

SIGNALS

$$\uparrow \mathrm{op}\,(V, M)$$

$$\mathsf{let}\ x = (\,\uparrow \mathsf{op}\,(V, M))\ \mathsf{in}\ N$$

$$\rightsquigarrow\ \uparrow \mathsf{op}\,(V, \mathsf{let}\ x = M\ \mathsf{in}\ N)$$

$$\mathscr{E} \;::=\; \cdots \;\Big|\; \uparrow \mathsf{op}\,(V, \mathscr{E})$$

$$\mathsf{let}\ x = (\uparrow \mathsf{op}(V, M))\ \mathsf{in}\ N$$

$$\rightsquigarrow \uparrow \mathsf{op}(V, \mathsf{let}\ x = M\ \mathsf{in}\ N)$$

$$\mathcal{E} \ ::= \ \cdots \ \big| \ \uparrow \mathsf{op}(V, \mathcal{E})$$

no bound variable

demo

OUTGOING

SIGNALS

$$\uparrow \mathrm{op}\,(V, M)$$

INCOMING

INTERRUPTS

$\downarrow \mathrm{op}\,(V, M)$

$$\downarrow \mathsf{op}\,(V, \mathsf{return}\ W) \rightsquigarrow \mathsf{return}\ W$$

$$\downarrow \mathsf{op}\,(V,\ \uparrow \mathsf{op}'\,(W, M)) \rightsquigarrow\ \uparrow \mathsf{op}'\,(W,\ \downarrow \mathsf{op}\,(V, M))$$

$$\mathcal{E} \ ::= \ \cdots \ \Big| \ \downarrow \mathsf{op}\,(V, \mathcal{E})$$

demo

INCOMING

INTERRUPTS

$\downarrow \mathrm{op}\,(V, M)$

INTERRUPT

HANDLERS

promise (op $x \mapsto M$) as p in N

$$\mathsf{let}\ x = (\mathsf{promise}\ (\mathsf{op}\ y \mapsto M_1)\ \mathsf{as}\ p\ \mathsf{in}\ M_2)\ \mathsf{in}\ N$$
$$\rightsquigarrow \mathsf{promise}\ (\mathsf{op}\ y \mapsto M_1)\ \mathsf{as}\ p\ \mathsf{in}\ (\mathsf{let}\ x = M_2\ \mathsf{in}\ N)$$

$$\mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ {\uparrow}\,\mathsf{op}'(V, N)$$
$$\rightsquigarrow {\uparrow}\,\mathsf{op}'(V, \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N)$$

$$\mathcal{E} \ ::= \ \cdots \ \Big| \ \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \mathcal{E}$$

$$\mathsf{let}\ x = (\mathsf{promise}\ (\mathsf{op}\ y \mapsto M_1)\ \mathsf{as}\ p\ \mathsf{in}\ M_2)\ \mathsf{in}\ N$$
$$\rightsquigarrow \mathsf{promise}\ (\mathsf{op}\ y \mapsto M_1)\ \mathsf{as}\ p\ \mathsf{in}\ (\mathsf{let}\ x = M_2\ \mathsf{in}\ N)$$

$$\mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \uparrow \mathsf{op}'(V, N)$$
$$\rightsquigarrow\ \uparrow \mathsf{op}'(V, \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N)$$

$$\mathscr{E} \ ::=\ \cdots \ \mid\ \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \mathscr{E}$$

bound variable

$$\mathsf{let}\ x = (\mathsf{promise}\ (\mathsf{op}\ y \mapsto M_1)\ \mathsf{as}\ p\ \mathsf{in}\ M_2)\ \mathsf{in}\ N$$

$$\rightsquigarrow \mathsf{promise}\ (\mathsf{op}\ y \mapsto M_1)\ \mathsf{as}\ p\ \mathsf{in}\ (\mathsf{let}\ x = M_2\ \mathsf{in}\ N)$$

algebraicity

$$\mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \uparrow \mathsf{op}'(V, N)$$

$$\rightsquigarrow \uparrow \mathsf{op}'(V, \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N)$$

$$\mathcal{E} \ ::= \ \cdots \ \Big| \ \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \mathcal{E}$$

bound variable

$$\mathsf{let}\ x = (\mathsf{promise}\ (\mathsf{op}\ y \mapsto M_1)\ \mathsf{as}\ p\ \mathsf{in}\ M_2)\ \mathsf{in}\ N$$

$$\rightsquigarrow \mathsf{promise}\ (\mathsf{op}\ y \mapsto M_1)\ \mathsf{as}\ p\ \mathsf{in}\ (\mathsf{let}\ x = M_2\ \mathsf{in}\ N)$$

algebraicity

$$\mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \uparrow \mathsf{op}'(V, N)$$

$$\rightsquigarrow \uparrow \mathsf{op}'(V, \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N)$$

commutativity

$$\mathcal{E} ::= \cdots \mid \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \mathcal{E}$$

bound variable

$$\downarrow \mathsf{op}\,(V, \mathsf{promise}\;(\mathsf{op}\; x \mapsto M)\;\mathsf{as}\; p\;\mathsf{in}\; N)$$
$$\rightsquigarrow \mathsf{let}\; p = M[V/x]\;\mathsf{in}\;\downarrow \mathsf{op}\,(V, N)$$

$$\downarrow \mathsf{op}'\,(V, \mathsf{promise}\;(\mathsf{op}\; x \mapsto M)\;\mathsf{as}\; p\;\mathsf{in}\; N)$$
$$\rightsquigarrow \mathsf{promise}\;(\mathsf{op}\; x \mapsto M)\;\mathsf{as}\; p\;\mathsf{in}\;\downarrow \mathsf{op}'\,(V, N)$$

$$\downarrow \mathsf{op}\,(V, \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N)$$

$$\rightsquigarrow\ \mathsf{let}\ p = M[V/x]\ \mathsf{in}\ \downarrow \mathsf{op}\,(V, N)$$

"handling"

$$\downarrow \mathsf{op}'\,(V, \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N)$$

$$\rightsquigarrow\ \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \downarrow \mathsf{op}'\,(V, N)$$

INTERRUPT

HANDLERS

promise (op $x \mapsto M$) as p in N

AWAITING

PROMISES

$$\langle V \rangle$$

await V until $\langle x \rangle$ in M

$$\mathsf{await}\ \langle V \rangle\ \mathsf{until}\ \langle x \rangle\ \mathsf{in}\ M \rightsquigarrow M[V/x]$$

$$\begin{aligned}&\mathsf{let}\ x = (\mathsf{await}\ V\ \mathsf{until}\ \langle y \rangle\ \mathsf{in}\ M)\ \mathsf{in}\ N\\ \rightsquigarrow\ &\mathsf{await}\ V\ \mathsf{until}\ \langle y \rangle\ \mathsf{in}\ (\mathsf{let}\ x = M\ \mathsf{in}\ N)\end{aligned}$$

$$\begin{aligned}&{\downarrow}\mathsf{op}\,(V, \mathsf{await}\ W\ \mathsf{until}\ \langle x \rangle\ \mathsf{in}\ M)\\ \rightsquigarrow\ &\mathsf{await}\ W\ \mathsf{until}\ \langle x \rangle\ \mathsf{in}\ {\downarrow}\mathsf{op}\,(V, M)\end{aligned}$$

$$\mathsf{await}\ \langle V \rangle\ \mathsf{until}\ \langle x \rangle\ \mathsf{in}\ M \leadsto M[V/x]$$

$$\mathsf{let}\ x = (\mathsf{await}\ V\ \mathsf{until}\ \langle y \rangle\ \mathsf{in}\ M)\ \mathsf{in}\ N$$
$$\leadsto \mathsf{await}\ V\ \mathsf{until}\ \langle y \rangle\ \mathsf{in}\ (\mathsf{let}\ x = M\ \mathsf{in}\ N)$$

algebraicity

$$\downarrow \mathsf{op}\,(V, \mathsf{await}\ W\ \mathsf{until}\ \langle x \rangle\ \mathsf{in}\ M)$$
$$\leadsto \mathsf{await}\ W\ \mathsf{until}\ \langle x \rangle\ \mathsf{in}\ \downarrow \mathsf{op}\,(V, M)$$

"handling"

demo

AWAITING

PROMISES

$$\langle V \rangle$$

$$\text{await } V \text{ until } \langle x \rangle \text{ in } M$$

TYPES

$$\Gamma \vdash V : X$$

$$\Gamma \vdash M : X ! \mathscr{C}$$

$$\frac{}{\Gamma, x : X, \Gamma' \vdash x : X}$$

$$\frac{}{\Gamma \vdash () : 1}$$

$$\frac{\Gamma, x : X \vdash M : Y ! \mathscr{C}}{\Gamma \vdash \textsf{fun}\ (x : X) \mapsto M : X \to Y ! \mathscr{C}}$$

$$\frac{\Gamma \vdash V : X \to Y ! \mathscr{C} \qquad \Gamma \vdash W : X}{\Gamma \vdash V\,W : Y ! \mathscr{C}}$$

$$\frac{\Gamma \vdash V : X}{\Gamma \vdash \textsf{return}\ V : X ! \mathscr{C}}$$

$$\frac{\Gamma \vdash M : X ! \mathscr{C} \qquad \Gamma, x : X \vdash N : Y ! \mathscr{C}}{\Gamma \vdash \textsf{let}\ x = M\ \textsf{in}\ N : Y ! \mathscr{C}}$$

$$\frac{\Gamma \vdash V : X}{\Gamma \vdash \langle V \rangle : \langle X \rangle}$$

$$\frac{\Gamma \vdash V : \langle X \rangle \qquad \Gamma, x : X \vdash M : Y ! \mathscr{C}}{\Gamma \vdash \textsf{await}\ V\ \textsf{until}\ \langle x \rangle\ \textsf{in}\ M : Y ! \mathscr{C}}$$

$$\mathscr{C} = (o, \iota)$$

$$o = \{\mathsf{op}_1, \ldots, \mathsf{op}_m\}$$

$$\iota = \{\mathsf{op}'_1 \mapsto \mathscr{C}_1, \ldots, \mathsf{op}'_n \mapsto \mathscr{C}_n\}$$

$$\iota_{p_1} = \{\mathsf{ping} \mapsto (\{\mathsf{pong}\}, \varnothing)\}$$
$$\iota_p = \{\mathsf{ping} \mapsto (\{\mathsf{pong}\}, \iota_p)\}$$

$$\iota_m = \left\{\mathsf{stop} \mapsto \left(\varnothing, \{\mathsf{go} \mapsto (\varnothing, \iota_m)\}\right)\right\}$$

$$\frac{\mathsf{op} \in o \qquad \Gamma \vdash V : A_{\mathsf{op}} \qquad \Gamma \vdash M : X!(o, \iota)}{\Gamma \vdash \uparrow \mathsf{op}\,(V, M) : X!(o, \iota)}$$

$$\dfrac{\iota(\mathsf{op}) = \mathscr{C} \qquad \Gamma, x : A_{\mathsf{op}} \vdash M : \langle X \rangle \,!\, \mathscr{C} \qquad \Gamma, p : \langle X \rangle \vdash N : Y \,!\, (o, \iota)}{\Gamma \vdash \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N : Y \,!\, (o, \iota)}$$

$$\frac{\Gamma \vdash V : A_{\mathsf{op}} \qquad \Gamma \vdash M : X ! \mathscr{C}}{\Gamma \vdash \ \downarrow \mathsf{op}\,(V, M) : X!(\mathsf{op} \downarrow \mathscr{C})}$$

$$\mathsf{op} \downarrow (o, \iota) = \begin{cases} (o \cup o', \iota\,|_{\mathsf{op}' \neq \mathsf{op}} \cup \iota') & \iota(\mathsf{op}) = (o', \iota') \\ (o, \iota) & \text{otherwise} \end{cases}$$

progress

$$\vdash M : X!\mathscr{C}$$

$$\implies M \rightsquigarrow M' \quad \vee \quad M \text{ is final}$$

$$\Gamma \vdash M : X!\mathscr{C} \quad \wedge \quad M \rightsquigarrow M'$$

$$\implies \Gamma \vdash M' : X!\mathscr{C}$$

preservation

progress

$$\vdash M : X!\mathscr{C}$$

$$\implies M \rightsquigarrow M' \quad \vee \quad M \text{ is final}$$

✔

$$\Gamma \vdash M : X!\mathscr{C} \quad \wedge \quad M \rightsquigarrow M'$$

$$\implies \Gamma \vdash M' : X!\mathscr{C}$$

preservation

progress

$$\vdash M : X!\mathscr{C} \implies M \rightsquigarrow M' \quad \vee \quad M \text{ is final}$$

$$\Gamma \vdash M : X!\mathscr{C} \quad \wedge \quad M \rightsquigarrow M' \implies \Gamma \vdash M' : X!\mathscr{C}$$

preservation

TYPES

$$\Gamma \vdash V : X$$

$$\Gamma \vdash M : X!(o, \iota)$$

PROCESSES

$$\mathsf{run}\, M \qquad P \parallel Q$$

$$\uparrow \mathsf{op}\,(V, P) \qquad \downarrow \mathsf{op}\,(V, P)$$

$$\frac{M \rightsquigarrow N}{\mathsf{run}\ M \rightsquigarrow \mathsf{run}\ N} \qquad \frac{P \rightsquigarrow Q}{\mathcal{F}[P] \rightsquigarrow \mathcal{F}[Q]}$$

$$\begin{aligned} \mathcal{F} \ ::= \ & [\ ] \ \mid \ \mathcal{F} \parallel Q \ \mid \ P \parallel \mathcal{F} \\ \mid \ & {\uparrow}\,\mathsf{op}\,(V, \mathcal{F}) \ \mid \ {\downarrow}\,\mathsf{op}\,(V, \mathcal{F}) \end{aligned}$$

$$\mathsf{run}\ (\uparrow \mathsf{op}\,(V, M)) \rightsquigarrow \uparrow \mathsf{op}\,(V, \mathsf{run}\ M)$$

$$\uparrow \mathsf{op}\,(V, P) \parallel Q \rightsquigarrow \uparrow \mathsf{op}\,(V, P \parallel \downarrow \mathsf{op}\,(V, Q))$$

$$P \parallel \uparrow \mathsf{op}\,(V, Q) \rightsquigarrow \uparrow \mathsf{op}\,(V, \downarrow \mathsf{op}\,(V, P) \parallel Q)$$

$$\downarrow \mathsf{op}\,(V, \mathsf{run}\ M) \rightsquigarrow \mathsf{run}\,(\downarrow \mathsf{op}\,(V, M))$$

$$\downarrow \mathsf{op}\,(V, P \parallel Q) \rightsquigarrow \downarrow \mathsf{op}\,(V, P) \parallel \downarrow \mathsf{op}\,(V, Q)$$

$$\downarrow \mathsf{op}\,(V, \uparrow \mathsf{op}'\,(W, P)) \rightsquigarrow \uparrow \mathsf{op}'\,(W, \downarrow \mathsf{op}\,(V, P))$$

demo

demo

$$\frac{M_i \rightsquigarrow M_i'}{M_1 \parallel \cdots \parallel M_i \parallel \cdots \parallel M_n \rightsquigarrow M_1 \parallel \cdots \parallel M_i' \parallel \cdots \parallel M_n}$$

demo

$$\frac{M_i \rightsquigarrow M_i'}{M_1 \parallel \cdots \parallel M_i \parallel \cdots \parallel M_n \rightsquigarrow M_1 \parallel \cdots \parallel M_i' \parallel \cdots \parallel M_n}$$

$$M_1 \parallel \cdots \parallel \uparrow \mathsf{op}\,(V, M_i) \parallel \cdots \parallel M_n$$
$$\rightsquigarrow \downarrow \mathsf{op}\,(V, M_1) \parallel \cdots \ M_i' \cdots \parallel \downarrow \mathsf{op}\,(V, M_n)$$

$$\frac{\Gamma \vdash M : X!(o, \iota)}{\Gamma \vdash \mathsf{run}\ M : X!!(o, \iota)}$$

$$\frac{\Gamma \vdash P : C \qquad \Gamma \vdash Q : D}{\Gamma \vdash P \parallel Q : C \parallel D}$$

$$\frac{\mathsf{op} \in \mathsf{signals}(C) \qquad \Gamma \vdash V : A_{\mathsf{op}} \qquad \Gamma \vdash P : C}{\Gamma \vdash \ \uparrow \mathsf{op}\,(V, P) : C}$$

$$\frac{\Gamma \vdash V : A_{\mathsf{op}} \qquad \Gamma \vdash P : C}{\Gamma \vdash \ \downarrow \mathsf{op}\,(V, P) : \mathsf{op} \downarrow C}$$

$$\mathsf{op} \downarrow (X!!(o, \iota)) = X!!(\mathsf{op} \downarrow (o, \iota))$$

$$\mathsf{op} \downarrow (C \parallel D) = (\mathsf{op} \downarrow C) \parallel (\mathsf{op} \downarrow D)$$

progress

$$\vdash P : C$$

$$\implies P \rightsquigarrow P' \quad \vee \quad P \text{ is final}$$

$$\Gamma \vdash P : C \quad \wedge \quad P \rightsquigarrow P'$$

$$\implies \Gamma \vdash P' : C$$

preservation

progress

$$\vdash P : C$$

$$\implies P \rightsquigarrow P' \quad \vee \quad P \text{ is final}$$ ✔

$$\Gamma \vdash P : C \quad \wedge \quad P \rightsquigarrow P'$$

$$\implies \Gamma \vdash P' : C$$

preservation

progress

$$\vdash P : C \implies P \rightsquigarrow P' \quad \lor \quad P \text{ is final}$$

$$\Gamma \vdash P : C \quad \land \quad P \rightsquigarrow P' \implies \Gamma \vdash P' : C$$

preservation

$$\uparrow \mathsf{op}\,(V, P) \parallel Q \rightsquigarrow \uparrow \mathsf{op}\,(V, P \parallel \downarrow \mathsf{op}\,(V, Q))$$

additional effects of
triggered handlers

$$\uparrow \mathsf{op}\,(V, P) \parallel Q \rightsquigarrow \uparrow \mathsf{op}\,(V, P \parallel \downarrow \mathsf{op}\,(V, Q))$$

$$\frac{}{X!!(o,\iota) \rightsquigarrow X!!(o,\iota)}$$

$$\frac{}{X!!\mathsf{ops}\downarrow(o,\iota) \rightsquigarrow X!!\mathsf{ops}\downarrow(\mathsf{op}\downarrow(o,\iota))}$$

$$\frac{C \rightsquigarrow C' \qquad D \rightsquigarrow D'}{C \parallel D \rightsquigarrow C' \parallel D'}$$

progress

$$\vdash P : C$$

$$\implies P \rightsquigarrow P' \quad \vee \quad P \text{ is final}$$

$$\Gamma \vdash P : C \quad \wedge \quad P \rightsquigarrow P'$$

$$\implies \exists C' \,.\, C \leadsto C' \quad \wedge \quad \Gamma \vdash P' : C'$$

preservation

progress

$$\vdash P : C$$

$$\Longrightarrow \quad P \rightsquigarrow P' \quad \vee \quad P \text{ is final}$$

$$\Gamma \vdash P : C \quad \wedge \quad P \rightsquigarrow P'$$

$$\Longrightarrow \quad \exists C' \,.\, C \leadsto C' \quad \wedge \quad \Gamma \vdash P' : C'$$

preservation

progress

$$\vdash P : C$$

$$\implies \quad P \rightsquigarrow P' \quad \lor \quad P \text{ is final}$$

$$\Gamma \vdash P : C \quad \land \quad P \rightsquigarrow P'$$

$$\implies \exists C' \,.\, C \leadsto C' \quad \land \quad \Gamma \vdash P' : C'$$

preservation

PROCESSES

$$\mathsf{run}\ M \qquad P \parallel Q$$

$$\uparrow \mathsf{op}\,(V, P) \qquad \downarrow \mathsf{op}\,(V, P)$$

EXTENSIONS

$$\frac{\mathscr{C} \sqsubseteq \iota\,(\mathsf{op}) \qquad \Gamma, p : \langle X \rangle \vdash N : Y\,!\,(o, \iota) \qquad \Gamma, x : A_{\mathsf{op}}, r : 1 \to \langle X \rangle\,!\big(\emptyset, \{\mathsf{op} \mapsto \mathscr{C}\}\big) \vdash M : \langle X \rangle\,!\,\mathscr{C}}{\Gamma \vdash \mathsf{promise}\ (\mathsf{op}\ x\ r \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N : Y\,!\,(o, \iota)}$$

$$\begin{aligned} &\downarrow \mathsf{op}\,(V, \mathsf{promise}\ (\mathsf{op}\ x\ r \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N) \\ &\rightsquigarrow \mathsf{let}\ p = M[V/x, R/r]\ \mathsf{in}\ \downarrow \mathsf{op}\,(V, N) \end{aligned}$$

where $R = \mathsf{fun}\ () \mapsto \mathsf{promise}\ (\mathsf{op}\ x\ r \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \mathsf{return}\ p$

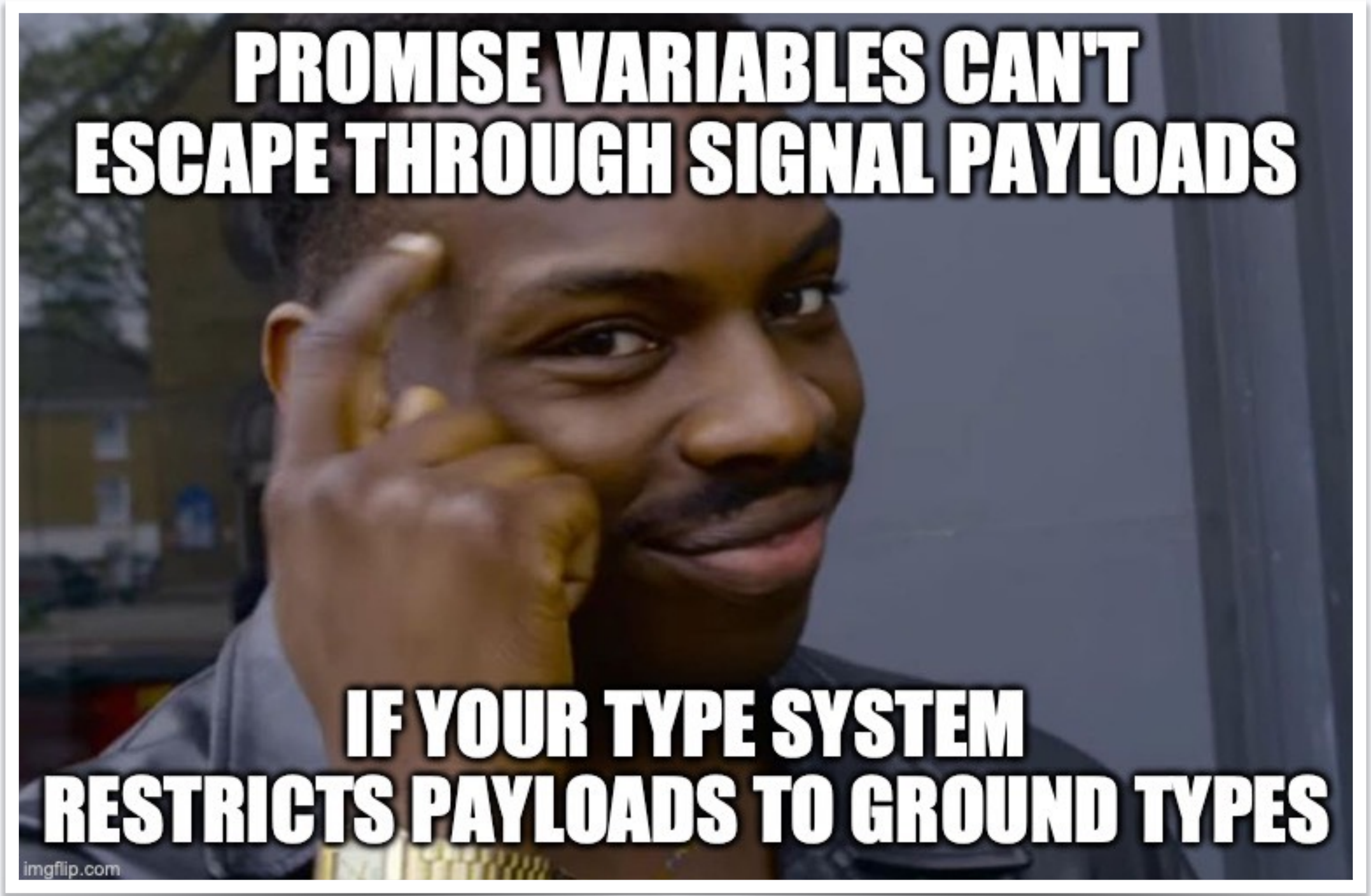
$$\mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \uparrow \mathsf{op}'(V, N)$$
$$\rightsquigarrow\ \uparrow \mathsf{op}'(V, \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N)$$

$$\mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \uparrow \mathsf{op}'(V, N)$$

$$\rightsquigarrow\ \uparrow \mathsf{op}'(V, \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N)$$

😬

$$\mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \uparrow \mathsf{op}'(V, N)$$
$$\rightsquigarrow\ \uparrow \mathsf{op}'(V, \mathsf{promise}\ (\mathsf{op}\ x \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N)$$


PROMISE VARIABLES CAN'T
ESCAPE THROUGH SIGNAL PAYLOADS
IF YOUR TYPE SYSTEM
RESTRICTS PAYLOADS TO GROUND TYPES
imgflip.com

$$
\begin{aligned}
A, B \; &::= \mathsf{b} \;\Big|\; 1 \;\Big|\; 0 \;\Big|\; A \times B \;\Big|\; A + B \\
X, Y \; &::= A \;\Big|\; X \times Y \;\Big|\; X + Y \;\Big| \\
&\qquad X \to Y!(o, \iota) \;\Big|\; \langle X \rangle
\end{aligned}
$$

$$
\frac{\mathsf{op} \in o \qquad \Gamma \vdash V : A_{\mathsf{op}} \qquad \Gamma \vdash M : X!(o, \iota)}{\Gamma \vdash \;\uparrow \mathsf{op}\,(V, M) : X!(o, \iota)}
$$

$$
\begin{aligned}
A, B \ &::= \mathsf{b} \mid \mathsf{1} \mid \mathsf{0} \mid A \times B \mid A + B \mid [X] \\
X, Y \ &::= A \mid X \times Y \mid X + Y \mid \\
&\qquad X \to Y!(o, \iota) \mid \langle X \rangle \mid [X]
\end{aligned}
$$

$$
\frac{\mathsf{op} \in o \qquad \Gamma \vdash V : A_{\mathsf{op}} \qquad \Gamma \vdash M : X!(o, \iota)}{\Gamma \vdash \ \uparrow \mathsf{op}\,(V, M) : X!(o, \iota)}
$$

$$\frac{X \text{ is mobile} \quad \text{or} \quad \blacksquare \notin \Gamma'}{\Gamma, x : X, \Gamma' \vdash x : X} \qquad \frac{\Gamma, \blacksquare \vdash V : X}{\Gamma \vdash [V] : [X]}$$

$$\frac{\Gamma \vdash V : [X] \qquad \Gamma, x : X \vdash M : Y!\mathscr{C}}{\Gamma \vdash \textbf{unbox}\ V\ \textbf{as}\ [x]\ \textbf{in}\ M : Y!\mathscr{C}}$$

$$\textbf{unbox}\ [V]\ \textbf{as}\ [x]\ \textbf{in}\ M \rightsquigarrow M[V/x]$$

$$\mathsf{run}\ (\mathsf{spawn}\ (M, N)) \rightsquigarrow \mathsf{run}\ M \parallel \mathsf{run}\ N$$

$$\frac{\Gamma, \blacksquare \vdash M : X ! \mathscr{C} \qquad \Gamma \vdash N : Y ! \mathscr{C}'}{\Gamma \vdash \mathsf{spawn}\ (M, N) : Y ! \mathscr{C}'}$$

$$\mathsf{let}\ x = (\mathsf{spawn}\ (M_1, M_2))\ \mathsf{in}\ N$$
$$\rightsquigarrow \mathsf{spawn}\ (M_1, \mathsf{let}\ x = M_2\ \mathsf{in}\ N)$$

$$\mathsf{promise}\ (\mathsf{op}\ x\, r \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \mathsf{spawn}\ (N_1, N_2)$$
$$\rightsquigarrow \mathsf{spawn}\ (N_1, (\mathsf{promise}\ (\mathsf{op}\ x\, r \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N_2))$$

$$\downarrow \mathsf{op}\, (V, \mathsf{spawn}\ (M, N))$$
$$\rightsquigarrow \mathsf{spawn}\ (M, \downarrow \mathsf{op}\, (V, N))$$

$$\mathsf{let}\ x = (\mathsf{spawn}\ (M_1, M_2))\ \mathsf{in}\ N$$

$$\rightsquigarrow \mathsf{spawn}\ (M_1, \mathsf{let}\ x = M_2\ \mathsf{in}\ N)$$

algebraicity

$$\mathsf{promise}\ (\mathsf{op}\ x\ r \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ \mathsf{spawn}\ (N_1, N_2)$$

$$\rightsquigarrow \mathsf{spawn}\ (N_1, (\mathsf{promise}\ (\mathsf{op}\ x\ r \mapsto M)\ \mathsf{as}\ p\ \mathsf{in}\ N_2))$$

commutativity

$$\downarrow \mathsf{op}\ (V, \mathsf{spawn}\ (M, N))$$

$$\rightsquigarrow \mathsf{spawn}\ (M, \downarrow \mathsf{op}\ (V, N))$$

"handling"

demo

EXTENSIONS

INTERESTED?

Asynchronous Effects

DANEL AHMAN and MATIJA PRETNAR, University of Ljubljana, Slovenia

We explore asynchronous programming with algebraic effects. We complement their conventional synchronous treatment by showing how to naturally also accommodate asynchrony within them, namely, by decoupling the execution of operation calls into signalling that an operation's implementation needs to be executed, and interrupting a running computation with the operation's result, to which the computation can react by installing interrupt handlers. We formalise these ideas in a small core calculus, called $\lambda_{æ}$. We demonstrate the flexibility of $\lambda_{æ}$ using examples ranging from a multi-party web application, to preemptive multi-threading, to remote function calls, to a parallel variant of runners of algebraic effects. In addition, the paper is accompanied by a formalisation of $\lambda_{æ}$'s type safety proofs in Agda, and a prototype implementation of $\lambda_{æ}$ in OCaml.

CCS Concepts: • **Theory of computation** → **Concurrency**; **Program constructs**; **Program semantics**.

Additional Key Words and Phrases: algebraic effects, asynchrony, concurrency, interrupt handling, signals.

ACM Reference Format:
Danel Ahman and Matija Pretnar. 2021. Asynchronous Effects. *Proc. ACM Program. Lang.* 5, POPL, Article 24 (January 2021), 28 pages. https://doi.org/10.1145/3434305

1 INTRODUCTION

Effectful programming abstractions are at the heart of many modern general-purpose programming languages. They can increase expressiveness by giving access to first-class continuations, but often simply help users to write cleaner code, e.g., by avoiding having to manage a program's memory explicitly in state-passing style, or getting lost in callback hell while programming asynchronously.

An increasing number of language designers and programmers are starting to embrace *algebraic effects*, where one uses algebraic operations [Plotkin and Power 2002] and effect handlers [Plotkin and Pretnar 2013] to uniformly and user-definably express a wide range of effectful behaviour, ranging from basic examples such as state, rollbacks, exceptions, and nondeterminism [Bauer and Pretnar 2015], to advanced applications in concurrency [Dolan et al. 2018] and statistical probabilistic programming [Bingham et al. 2019], and even quantum computation [Staton 2015].

While covering many examples, the conventional treatment of algebraic effects is *synchronous* by nature. In it effects are invoked by placing operation calls in one's code, which then propagate outwards until they trigger the actual effect, finally yielding a result to the rest of the computation that has been *waiting* the whole time. While blocking the computation is indeed sometimes needed, e.g., in the presence of general effect handlers that can execute their continuation any number of times, it forces all uses of algebraic effects to be synchronous, even when this is not necessary, e.g., when the effect involves executing a remote query to which a response is not needed (immediately).

Motivated by the recent interest in the combination of asynchrony and algebraic effects [Dolan et al. 2018; Leijen 2017], we explore what it takes (in terms of language design, safe programming abstractions, and a self-contained core calculus) to accompany the synchronous treatment of

Authors' address: Danel Ahman, danel.ahman@fmf.uni-lj.si; Matija Pretnar, matija.pretnar@fmf.uni-lj.si, University of Ljubljana, Faculty of Mathematics and Physics, Jadranska 21, Ljubljana, SI-1000, Slovenia.

2475-1421/2021/1-ART24
https://doi.org/10.1145/3434305

HIGHER-ORDER ASYNCHRONOUS EFFECTS *

DANEL AHMAN [a] AND MATIJA PRETNAR [a,b]

[a] University of Ljubljana, Faculty of Mathematics and Physics, Jadranska 21, SI-1000 Ljubljana, Slovenia
e-mail address: danel.ahman@fmf.uni-lj.si, matija.pretnar@fmf.uni-lj.si

[b] Institute of Mathematics, Physics and Mechanics, Jadranska 21, SI-1000 Ljubljana, Slovenia

Abstract. We explore asynchronous programming with algebraic effects. We complement their conventional synchronous treatment by showing how to naturally also accommodate asynchrony within them, namely, by decoupling the execution of operation calls into signalling that an operation's implementation needs to be executed, and interrupting a running computation with the operation's result, to which the computation can react by installing interrupt handlers. We formalise these ideas in a small core calculus, called $\lambda_{æ}$. We demonstrate the flexibility of $\lambda_{æ}$ using examples ranging from a multi-party web application, to preemptive multi-threading, to remote function calls, to a parallel variant of runners of algebraic effects. In addition, the paper is accompanied by a formalisation of $\lambda_{æ}$'s type safety proofs in Agda, and a prototype implementation of $\lambda_{æ}$ in OCaml.

Before sumbitting go through LMCS check list

Create LMCS Git tags for A and Agda

1. Introduction

Effectful programming abstractions are at the heart of many modern general-purpose programming languages. They can increase expressiveness by giving access to first-class continuations, but often simply help users to write cleaner code, e.g., by avoiding having to manage a program's memory explicitly in state-passing style, or getting lost in callback hell while programming asynchronously.

An increasing number of language designers and programmers are starting to embrace *algebraic effects*, where one uses algebraic operations [PP02] and effect handlers [PP13] to uniformly and user-definably express a wide range of effectful behaviour, ranging from basic examples such as state, rollbacks, exceptions, and nondeterminism [BP15], to advanced applications in concurrency [DEH+18] and statistical probabilistic programming [BCJ+19], and even quantum computation [Sta15].

While covering many examples, the conventional treatment of algebraic effects is *synchronous* by nature. In it effects are invoked by placing operation calls in one's code, which

Key words and phrases: algebraic effects, asynchrony, concurrency, interrupt handling, signals.

* This paper is an extended version of our previous work [AP21], which simplifies the meta-theory, removes the reliance on general recursion for reinstallable interrupt handlers, extends the calculus with higher-order interrupt payloads and dynamic process creation, and strengthens the examples of application.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 834146. This material is based upon work supported by the Air Force Office of Scientific Research under awards number FA9550-17-1-0326 and FA9550-21-1-0024.

Preprint submitted to
Logical Methods in Computer Science

Product Solutions Open Source Pricing

Search / Sign in Sign up

danelahman / **aeff-agda** Public

Notifications | Fork 0 | Star 6

Code | Issues | Pull requests | Actions | Projects | Security | Insights

master | 1 branch | 1 tag | Go to file | Code

danelahman license — 71ebed9 on Oct 7, 2021 — **162** commits

AEff.agda	merging value, computation, and process type modules	3 years ago
AwaitingComputations.agda	removing unused lemmas and better naming conventions	3 years ago
EffectAnnotations.agda	Proof-irrelevance of subtyping relations	3 years ago
Finality.agda	Finality of result forms	3 years ago
LICENSE.md	license	2 years ago
Preservation.agda	Syncing a lemma annotation with the paper. Removing a spurious mu...	3 years ago
ProcessFinality.agda	Tweaking the notation	3 years ago
ProcessPreservation.agda	Tweaking the notation	3 years ago
ProcessProgress.agda	Finality of process result forms	3 years ago
Progress.agda	Syncing names with the paper	3 years ago
README.md	Update README.md	2 years ago
Renamings.agda	actions of renamings and substitutions for processes	3 years ago
Substitutions.agda	actions of renamings and substitutions for processes	3 years ago
Types.agda	Tweaking the notation	3 years ago

README.md

Agda formalisation of the AEff language

Note: For the Agda formalisation of a newer version of AEff (extended with reinstallable interrupt handlers, higher-order payloads for signals and interrupts, and dynamic process creation), see here.

- The formalisation has been tested with Agda version 2.6.1 and standard library version 1.3.
- The unicode symbols used in the source code have tested to display correctly with the DejaVu Sans Mono

About

Agda formalisation of the AEff language

Readme

MIT license

6 stars

3 watching

0 forks

Report repository

Releases 1

POPL 2021 Latest

on Nov 10, 2020

Packages

No packages published

Languages

Agda 100.0%

danelahman / **higher-order-aeff-agda** Public

main | 1 branch | 0 tags | Go to file | Code

danelahman license b41df71 on Oct 7, 2021 **36** commits

AEff.agda	removing let-rec from the core calculus	2 years ago
EffectAnnotations.agda	Removing a redundant mutual block	2 years ago
Finality.agda	removing the separate judgement of awaiting computations (not nee...	2 years ago
LICENSE.md	license	2 years ago
Preservation.agda	removing let-rec from the core calculus	2 years ago
ProcessFinality.agda	for symmetry, allow type-level spawning both in left and right	2 years ago
ProcessPreservation.agda	for symmetry, allow type-level spawning both in left and right	2 years ago
ProcessProgress.agda	removing the separate judgement of awaiting computations (not nee...	2 years ago
Progress.agda	removing the separate judgement of awaiting computations (not nee...	2 years ago
README.md	Update README.md	2 years ago
Renamings.agda	removing let-rec from the core calculus	2 years ago
Substitutions.agda	removing let-rec from the core calculus	2 years ago
Types.agda	wip: reinstallable interrupt handlers (up to coercion rules)	2 years ago

README.md

Agda formalisation of the AEff language for higher-order asynchronous effects

- The core language formalised here differs from the original language as follows:
  - interrupt handlers are now able to reinstall themselves (without resorting to general let-rec);
  - payloads of signals/interrupts have been generalised to allow higher-order values (by the means of modal types);

About

No description, website, or topics provided.

Readme
MIT license
1 star
3 watching
0 forks
Report repository

Releases

No releases published

Packages

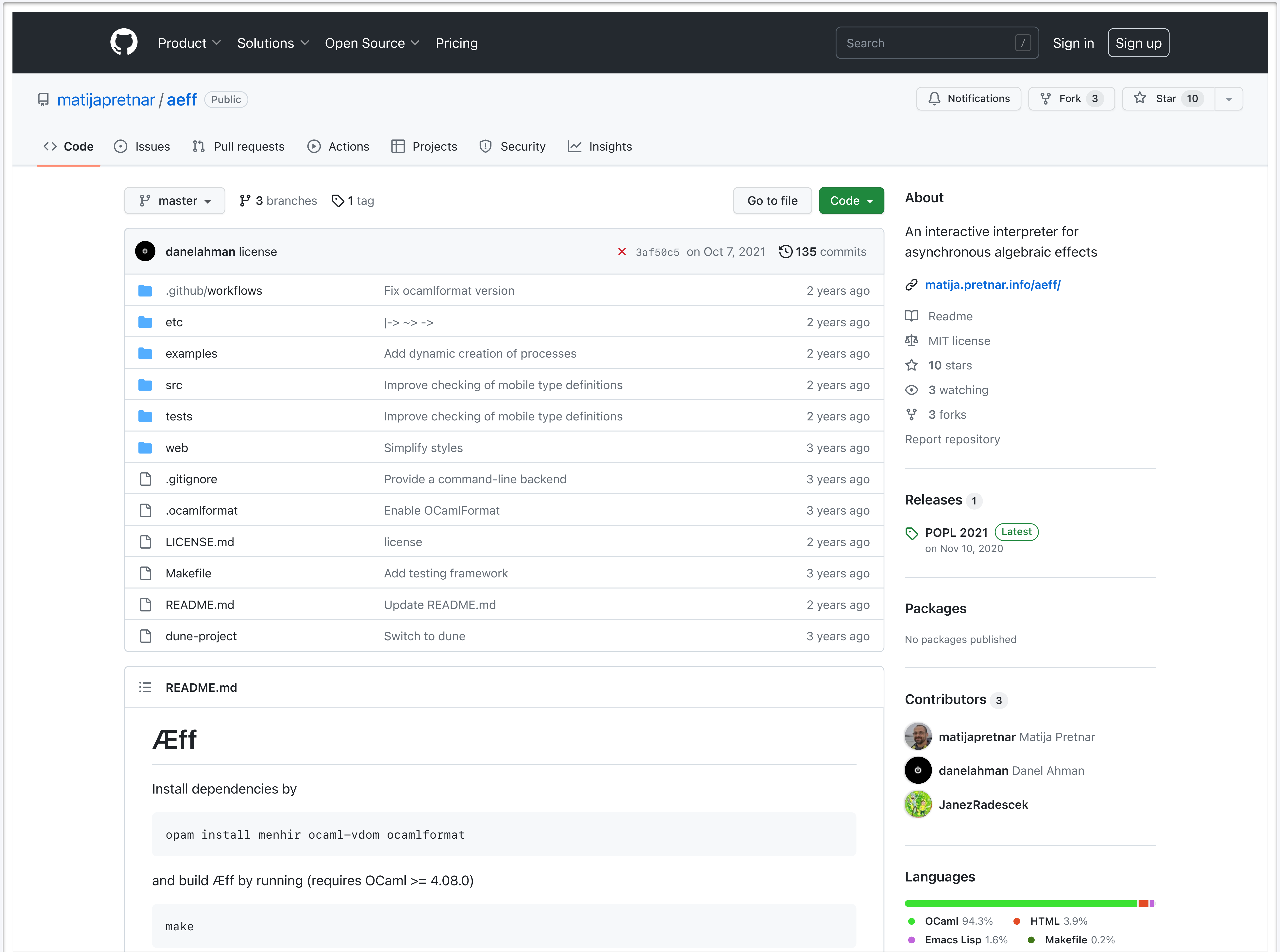
No packages published

Contributors 2

danelahman Danel Ahman
matijapretnar Matija Pretnar

Languages

Agda 100.0%


Product
Solutions
Open Source
Pricing
Search
Sign in
Sign up
matijapretnar / aeff
Public
Notifications
Fork 3
Star 10
Code
Issues
Pull requests
Actions
Projects
Security
Insights
master
3 branches
1 tag
Go to file
Code
danelahman license
3af50c5 on Oct 7, 2021
135 commits
.github/workflows
Fix ocamlformat version
2 years ago
etc
|-> ~> ->
2 years ago
examples
Add dynamic creation of processes
2 years ago
src
Improve checking of mobile type definitions
2 years ago
tests
Improve checking of mobile type definitions
2 years ago
web
Simplify styles
3 years ago
.gitignore
Provide a command-line backend
3 years ago
.ocamlformat
Enable OCamlFormat
3 years ago
LICENSE.md
license
2 years ago
Makefile
Add testing framework
3 years ago
README.md
Update README.md
2 years ago
dune-project
Switch to dune
3 years ago
README.md
Æff
Install dependencies by
opam install menhir ocaml-vdom ocamlformat
and build Æff by running (requires OCaml >= 4.08.0)
make
About
An interactive interpreter for asynchronous algebraic effects
matija.pretnar.info/aeff/
Readme
MIT license
10 stars
3 watching
3 forks
Report repository
Releases 1
POPL 2021
Latest
on Nov 10, 2020
Packages
No packages published
Contributors 3
matijapretnar Matija Pretnar
danelahman Danel Ahman
JanezRadescek
Languages
OCaml 94.3%
HTML 3.9%
Emacs Lisp 1.6%
Makefile 0.2%

Product Solutions Open Source Pricing Search / Sign in Sign up

matijapretnar / **millet** Public

Notifications | Fork 0 | Star 19

Code | Issues | Pull requests | Actions | Security | Insights

main | 4 branches | 0 tags | Go to file | Code

matijapretnar Bump OCaml & ocamlformat version | f28a4ba on Nov 16, 2022 | 43 commits

.github/workflows	Bump OCaml & ocamlformat version	7 months ago
examples	Initial commit	3 years ago
src	Bump OCaml & ocamlformat version	7 months ago
tests	Check for well-formed type definitions	2 years ago
web	Sort out names	2 years ago
.gitignore	Sort out names	2 years ago
.ocamlformat	Bump OCaml & ocamlformat version	7 months ago
Makefile	Simplify test folder	2 years ago
README.md	Bump OCaml & ocamlformat version	7 months ago
dune-project	Add initial cram tests setup	2 years ago

README.md

Millet

Do you, like me, test theoretical programming language concepts by building your own programming language? Do you, like me, do it by copying and modifying your most recent language because you are too lazy to build everything from scratch? Do you, like me, end up with a mess? Then Millet is for you. It is a pure ML-like language with simple and modular codebase that you can use as a template for your next language.

How to install and run Millet?

Install dependencies by

```
opam install menhir ocaml-vdom ocamlformat
```

About

A ML-like pure functional language that can be used as a template for creating your own language

matija.pretnar.info/millet/

Readme

19 stars

3 watching

0 forks

Report repository

Releases

No releases published

Packages

No packages published

Contributors 2

matijapretnar Matija Pretnar

zputrle Žiga Putrle

Languages

OCaml 90.8% **Perl** 8.4%

Other 0.8%

FUTURE WORK

EFFICIENT

INTERPRETER

EFFECT-AWARE

OPTIMISATIONS

DENOTATIONAL

SEMANTICS

SCOPED?

HANDLERS

QUESTIONS?