VERSATILE EVENT CORRELATION WITH ALGEBRAIC EFFECTS

Oliver Bračevac - TU Darmstadt, DE
Nada Amin - University of Cambridge, UK
Guido Salvaneschi - TU Darmstadt, DE
Sebastian Erdweg - TU Delft, NL
Mira Mezini - TU Darmstadt, DE
Patrick Eugster - USI Lugano, CH

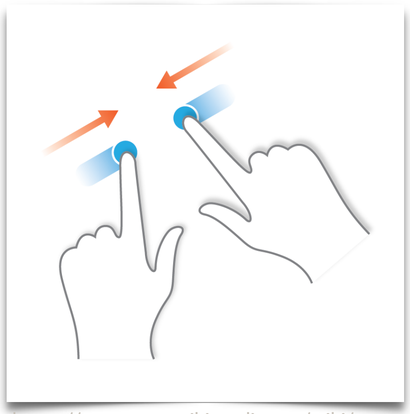
VERSATILE EVENT CORRELATION WITH ALGEBRAIC EFFECTS

Oliver Bračevac - TU Darmstadt, DE
Nada Amin - University of Cambridge, UK
Guido Salvaneschi - TU Darmstadt, DE
Sebastian Erdweg - TU Delft, NL
Mira Mezini - TU Darmstadt, DE
Patrick Eugster - USI Lugano, CH

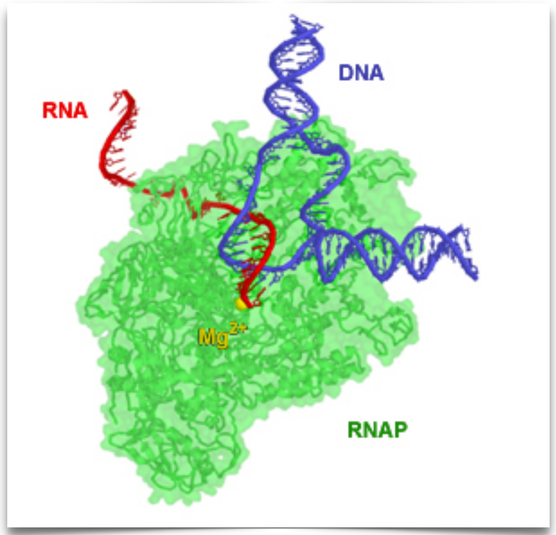
AN OPEN PROBLEM: WHAT ARE JOINS?

JOIN ABSTRACTIONS HAVE BEEN REINVENTED SEVERAL TIMES OVER

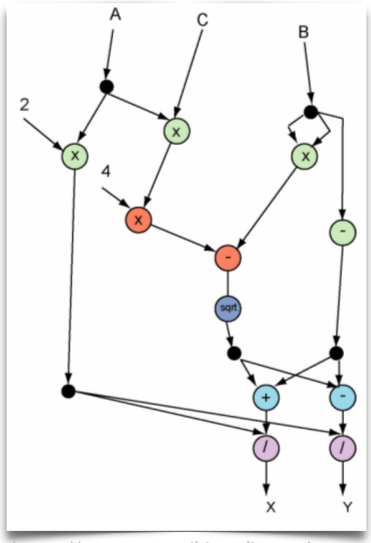
https://pxhere.com/en/photo/138038



https://commons.wikimedia.org/wiki/File:Gestures_Two_Hand_Pinch.png



https://commons.wikimedia.org/wiki/File:RNAP_TEC_small.jpg

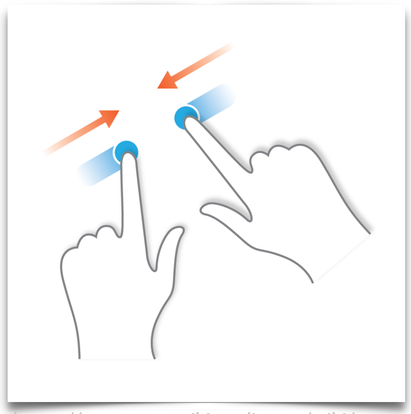


https://commons.wikimedia.org/wiki/File:Dataflow_graph.PNG

AN OPEN PROBLEM: WHAT ARE JOINS?

JOIN ABSTRACTIONS HAVE BEEN REINVENTED SEVERAL TIMES OVER

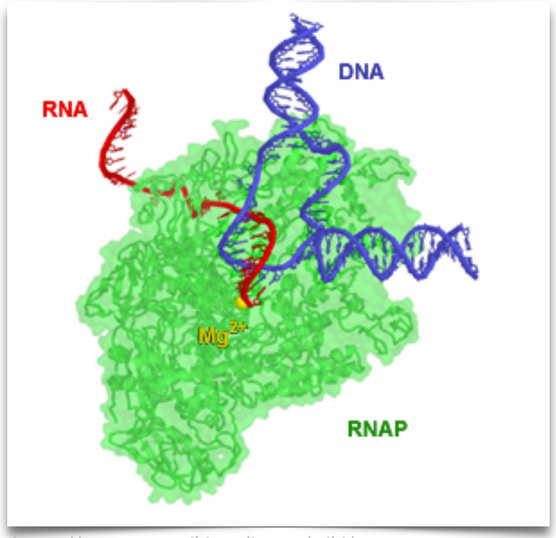
https://pxhere.com/en/photo/138038



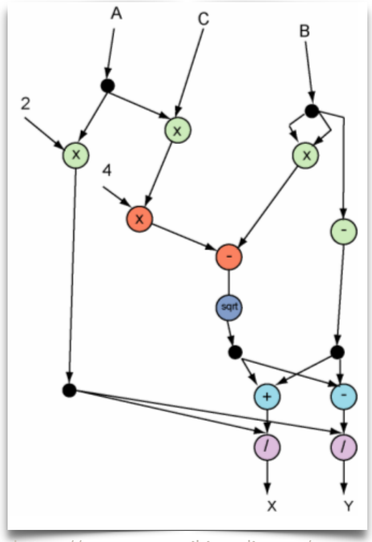
https://commons.wikimedia.org/wiki/File:Gestures_Two_Hand_Pinch.png

```
PATTERN SEQ(Stock+ a[], Stock b)
WHERE skip_till_next_match(a[], b) {
   [symbol]
   and a[1].volume > 1000
   and a[i].price > avg(a[..i-1].price)
   and b.volume < 0.8*a[a.LEN].volume }
WITHIN 1 hour
```

Stream/Complex Event Processing (SASE+ [Agrawal et al. '08])



https://commons.wikimedia.org/wiki/File:RNAP_TEC_small.jpg



https://commons.wikimedia.org/wiki/File:Dataflow_graph.PNG

```
def wait() & finished(r) =
  reply (Some r) to wait

or  wait() & timeout() =
  reply None to wait
```

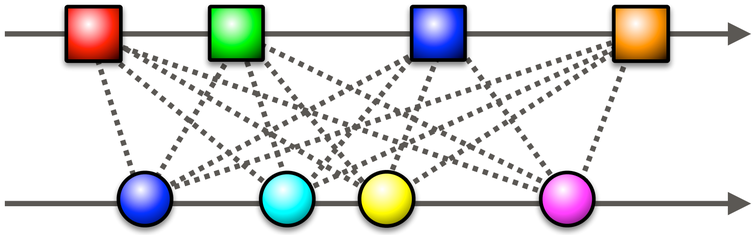
Concurrent Programming
(JoCaml [Fournet & Gonthier '96, Conchon et al. '99])

IN THIS WORK: WHAT ARE JOINS DOING?

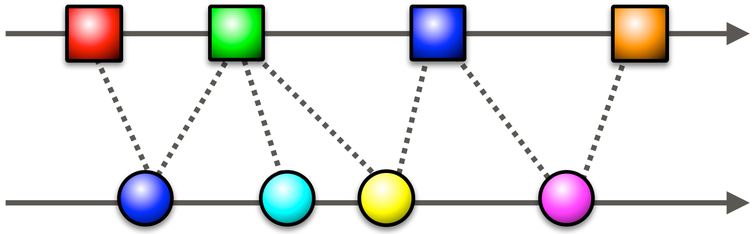
COMPUTATIONAL INTERPRETATION OF ASYNCHRONOUS JOINS

IN THIS WORK: WHAT ARE JOINS DOING?

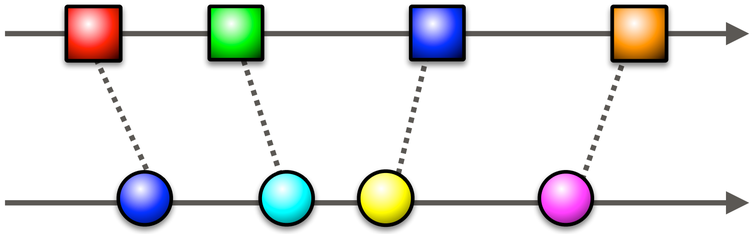
COMPUTATIONAL INTERPRETATION OF ASYNCHRONOUS JOINS



Cartesian product



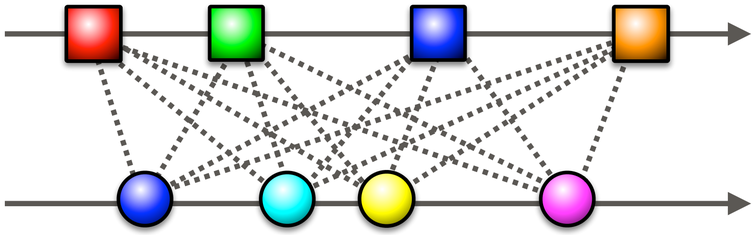
combineLatest



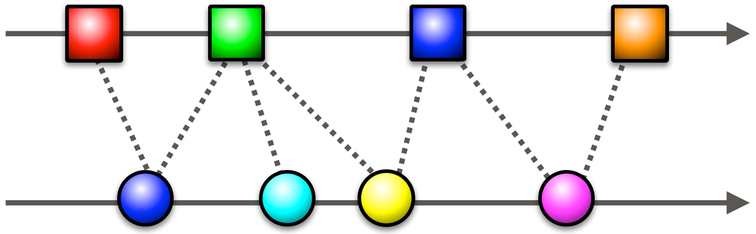
zip

IN THIS WORK: WHAT ARE JOINS DOING?

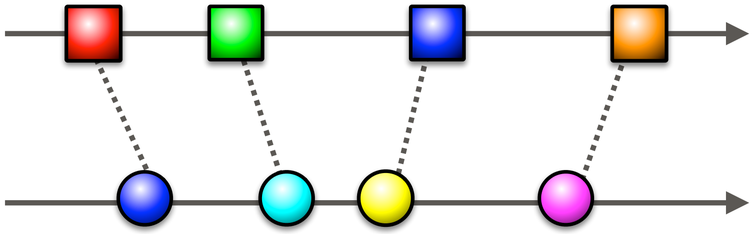
COMPUTATIONAL INTERPRETATION OF ASYNCHRONOUS JOINS



Cartesian product



combineLatest

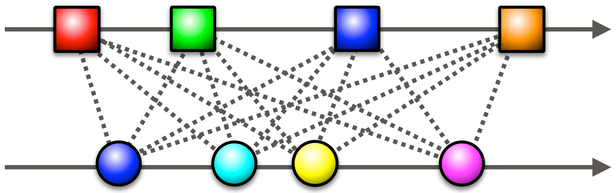


zip

Enumerate the cartesian product, where (user-defined) side effects influence how the computation proceeds.

OUR MAIN CONTRIBUTION

CARTESIUS: PURELY FUNCTIONAL, VERSATILE EVENT CORRELATION



Cartesian product

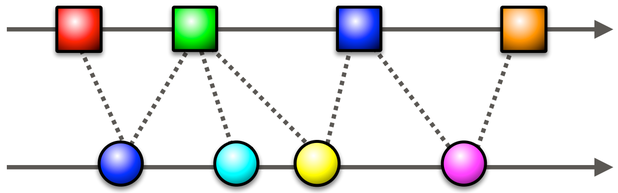
```
let cart: R[A] -> R[B] -> R[(A,B)]
  = λ a b. correlate {
             x from a
             y from b
             yield (x,y)}
```

We support

- Direct-style join specifications

OUR MAIN CONTRIBUTION

CARTESIUS: PURELY FUNCTIONAL, VERSATILE EVENT CORRELATION



combineLatest

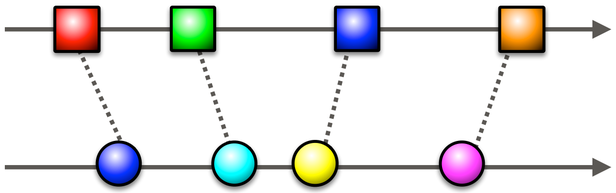
```
let implicit ?restriction =
      (mostRecently a)
  |+| (mostRecently b) in
correlate {
  x from a
  y from b
  yield (x,y)}
```

We support

- Direct-style join specifications
- Controllable matching behavior

OUR MAIN CONTRIBUTION

CARTESIUS: PURELY FUNCTIONAL, VERSATILE EVENT CORRELATION




```
let implicit ?restriction =
      (mostRecently a)
  |+| (mostRecently b)
  |+| (aligning a b) in
correlate {
  x from a
  y from b
  yield (x,y)}
```

We support

- Direct-style join specifications
- Controllable matching behavior
- Mix-and-match composition of features
- Uniform programming model & extensibility

OUR MAIN CONTRIBUTION

CARTESIUS: PURELY FUNCTIONAL, VERSATILE EVENT CORRELATION

In the Paper: Timing and Time Windows

```
correlate {
  x from a
  y from b
  where
    abs(?time_x - ?time_y) < 5s
  yield (x,y)}
```

```
correlate {
  with slidingWindow(1h)
  x from a
  y from b
  yield (x,y)}
```

ns

avior

n of features

- Uniform programming model & extensibility

correlate {
  x from a
  y from b
  yield (x,y)}

ALGEBRAIC EFFECTS AND HANDLERS

[PLOTKIN & POWER '03, PLOTKIN & PRETNAR '09]

ALGEBRAIC EFFECTS AND HANDLERS

[PLOTKIN & POWER '03, PLOTKIN & PRETNAR '09]

Effect Interfaces:

yield : Int $\rightarrow$ Unit

async$_\alpha$: $\{\alpha\} \rightarrow$ Future[α]

await$_\alpha$: Future[α] $\rightarrow \alpha$

[Dolan et al. '17, Leijen '17a]

ALGEBRAIC EFFECTS AND HANDLERS

[PLOTKIN & POWER '03, PLOTKIN & PRETNAR '09]

Effect Interfaces:

$\mathbf{yield}: \text{Int} \to \text{Unit}$

$\mathbf{async}_{\alpha}: \{\alpha\} \to \text{Future}[\alpha]$

$\mathbf{await}_{\alpha}: \text{Future}[\alpha] \to \alpha$

[Dolan et al. '17, Leijen '17a]

Effect Handlers:

```
let interact = handler
  x -> <>
  yield val resume ->
    println val;
    match readkey <>
      \cr -> resume <>
       _  -> <>
```

ALGEBRAIC EFFECTS AND HANDLERS

[PLOTKIN & POWER '03, PLOTKIN & PRETNAR '09]

Effect Interfaces:

$\mathbf{yield}: \text{Int} \to \text{Unit}$

$\mathbf{async}_{\alpha}: \{\alpha\} \to \text{Future}[\alpha]$

$\mathbf{await}_{\alpha}: \text{Future}[\alpha] \to \alpha$

[Dolan et al. '17, Leijen '17a]

Effect Handlers:

```
let interact = handler
  x -> <>
  yield val resume ->
    println val;
    match readkey <>
      \cr -> resume <>
       _  -> <>
```

$\text{interact}: \forall \alpha\, \mu.\, \{\alpha\}^{\langle \mathbf{yield}, \mu \rangle} \to^{\langle \mathbf{println}, \mathbf{readkey}, \mu \rangle} \text{Unit}$

ALGEBRAIC EFFECTS AND HANDLERS

[PLOTKIN & POWER '03, PLOTKIN & PRETNAR '09]

Effect Interfaces:

$\mathbf{yield}: \text{Int} \to \text{Unit}$

$\mathbf{async}_{\alpha}: \{\alpha\} \to \text{Future}[\alpha]$

$\mathbf{await}_{\alpha}: \text{Future}[\alpha] \to \alpha$

[Dolan et al. '17, Leijen '17a]

Effect Handlers:

```
let interact = handler
  x -> <>
  yield val resume ->
    println val;
    match readkey <>
      \cr -> resume <>
       _  -> <>
```

$\text{interact}: \forall \alpha\, \mu.\, \{\alpha\}^{\langle \mathbf{yield}, \mu \rangle} \to^{\langle \mathbf{println}, \mathbf{readkey}, \mu \rangle} \text{Unit}$

```
with interact {
  map (yield)[1,2,3,4]}
```

ALGEBRAIC EFFECTS AND HANDLERS

[PLOTKIN & POWER '03, PLOTKIN & PRETNAR '09]

Effect Interfaces:

yield: Int → Unit

$\mathbf{async}_{\alpha}$: $\{\alpha\} \to \text{Future}[\alpha]$

$\mathbf{await}_{\alpha}$: $\text{Future}[\alpha] \to \alpha$

[Dolan et al. '17, Leijen '17a]

Effect Handlers:

```
let interact = handler
  x -> <>
  yield val resume ->
    println val;
    match readkey <>
      \cr -> resume <>
       _  -> <>
```

$\text{interact}: \forall \alpha\, \mu.\, \{\alpha\}^{\langle \mathbf{yield}, \mu \rangle} \to^{\langle \mathbf{println}, \mathbf{readkey}, \mu \rangle} \text{Unit}$

```
with interact {
  map (yield)[1,2,3,4]}
```

Handler composition:

$h_1 \boxplus h_2 := \lambda tn\,.\,\mathbf{with}\ h_1\ (\mathbf{with}\ h_2\ (tn\ \langle\rangle))$

PROCESSING MODEL

INTERLEAVING CONCURRENCY

$$\text{interleave}: \ \forall \mu \,.\, \text{List}[\{\text{Unit}\}^{\langle \textbf{async}, \mu \rangle}] \to^{\langle \textbf{async}, \mu \rangle} \text{Unit}$$

[Leijen '17a]

$$e_1 \parallel \cdots \parallel e_n \quad \rightsquigarrow \quad \{\text{interleave}\ [e_1, \ldots, e_n]\}$$

PROCESSING MODEL

INTERLEAVING CONCURRENCY

$\text{interleave}: \ \forall \mu \,.\, \text{List}[\{\text{Unit}\}^{\langle \textbf{async}, \mu \rangle}] \to^{\langle \textbf{async}, \mu \rangle} \text{Unit}$

[Leijen '17a]

$e_1 \parallel \cdots \parallel e_n \quad \rightsquigarrow \quad \{\text{interleave}\ [e_1, \ldots, e_n]\}$

The calling context observes effects of the strands!

$\{\textbf{await}\ x\} \parallel \{\textbf{println}\ \text{"foo"}\} \parallel \{\textbf{yield}\ 1\}: \ \{\text{Unit}\}^{\langle \textbf{yield}, \textbf{println}, \textbf{async} \rangle}$

CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS

The calling context observes effects of the strands!

https://commons.wikimedia.org/wiki/File:Scoubidou_6_strands.jpg

CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS

The calling context observes effects of the strands!

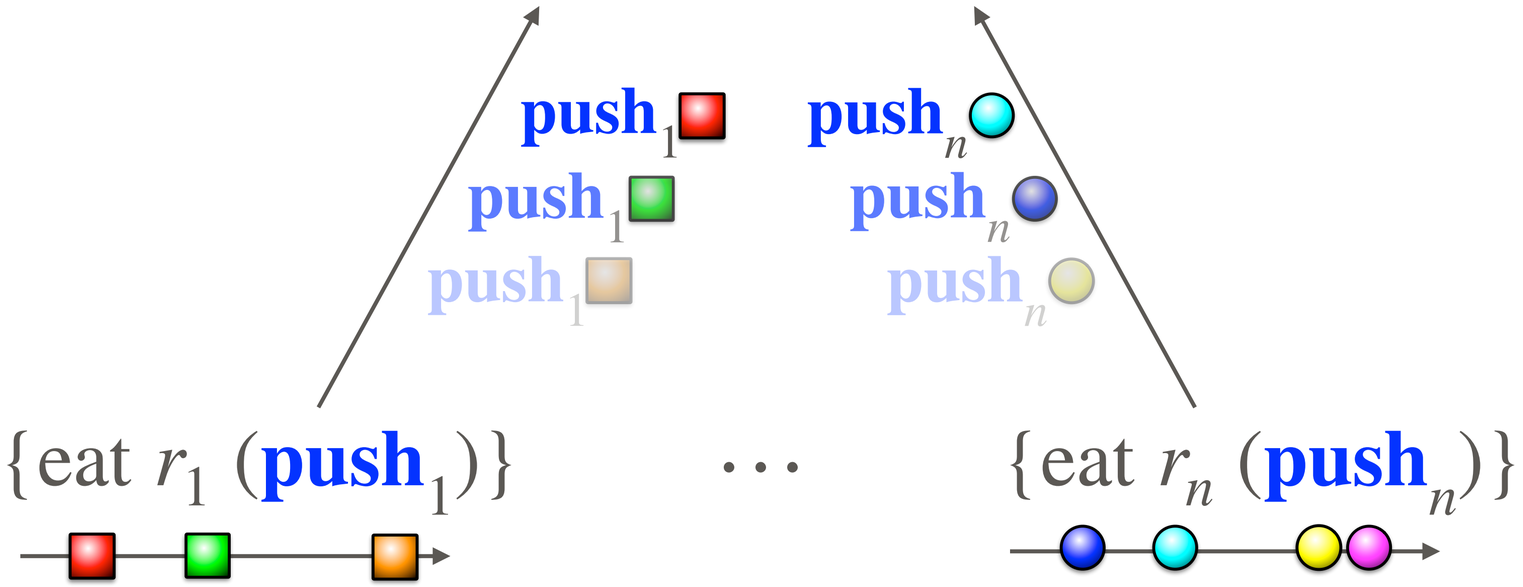
https://commons.wikimedia.org/wiki/File:Scoubidou_6_strands.jpg

$\{\text{eat } r_1\ (\textbf{push}_1)\}$ $\cdots$ $\{\text{eat } r_n\ (\textbf{push}_n)\}$

CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS

The calling context observes effects of the strands!

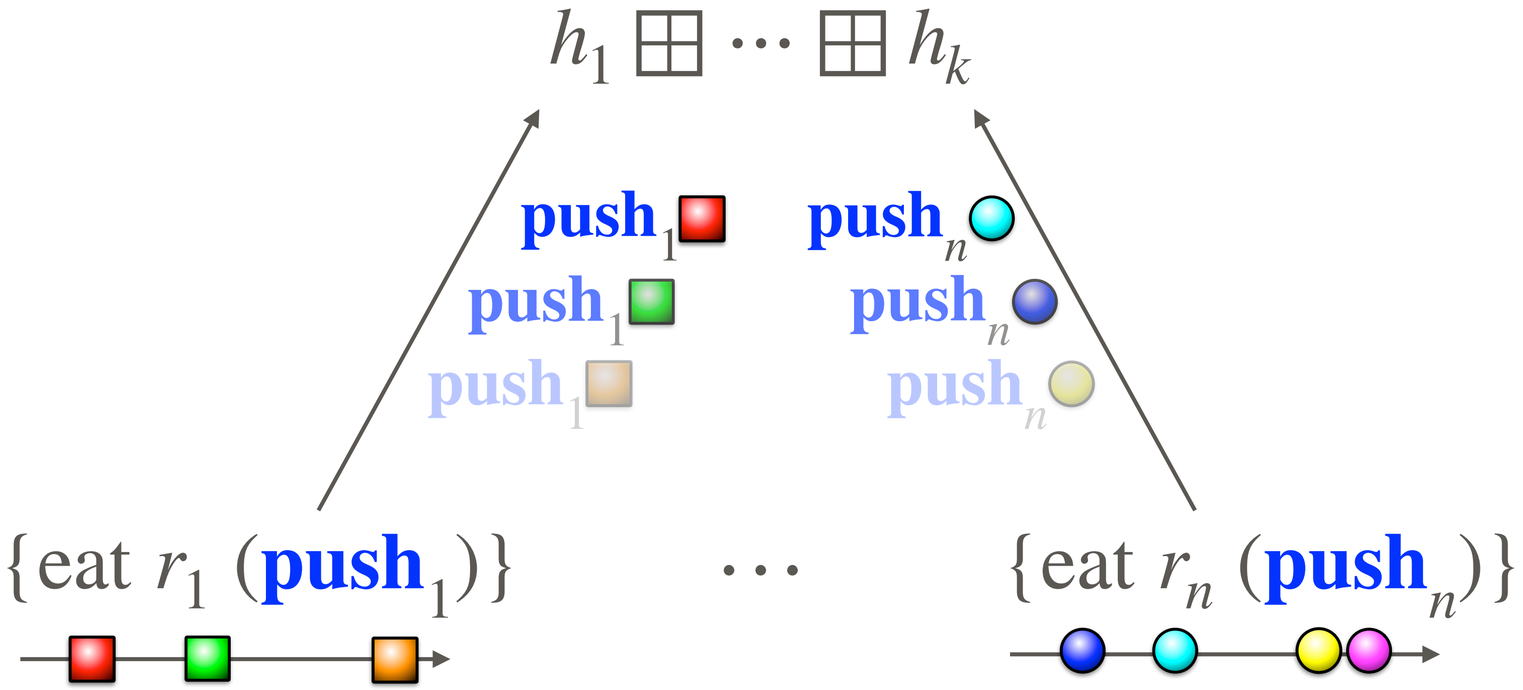


https://commons.wikimedia.org/wiki/File:Scoubidou_6_strands.jpg

CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS

The calling context observes effects of the strands!

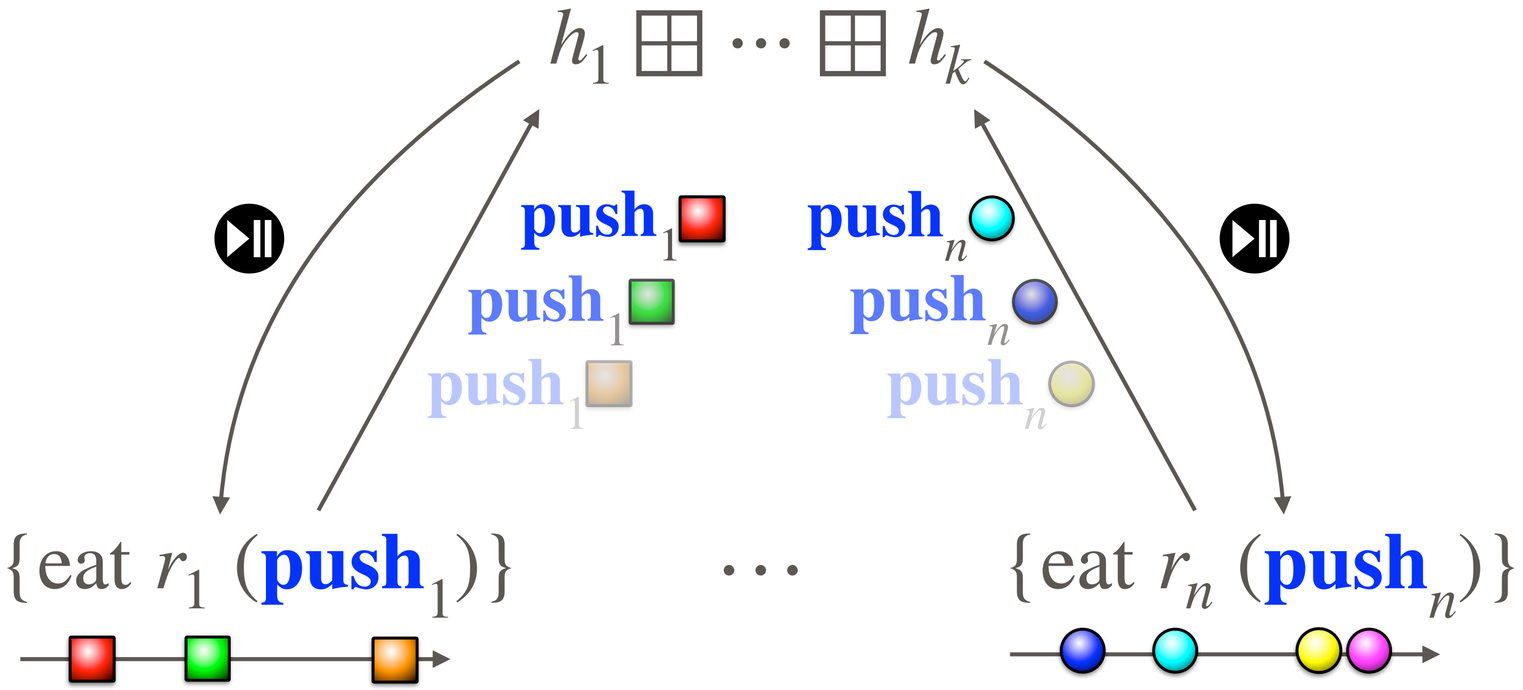


https://commons.wikimedia.org/wiki/File:Scoubidou_6_strands.jpg

CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS

The calling context observes effects of the strands!



https://commons.wikimedia.org/wiki/File:Scoubidou_6_strands.jpg

CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS

$$h_1 \boxplus \cdots \boxplus h_k$$

CORRELATE BY HANDLING

DEFINES JOINS IN TERMS OF EFFECTS AND HANDLERS

$$h_1 \boxplus \cdots \boxplus h_k : \{\mathsf{Unit}\}^{\langle \mathbf{push}_1,\ldots,\mathbf{push}_n,\, \mathbf{async},\varepsilon\rangle} \to^{\langle \mathbf{async},\varepsilon\rangle} \mathsf{Unit}$$

THE BIG PICTURE

```
let r_out =
  let implicit ?restriction = h in
  correlate {
    x_1 from r_1
        …
    x_n from r_n
    where p
    yield e }
```

THE BIG PICTURE

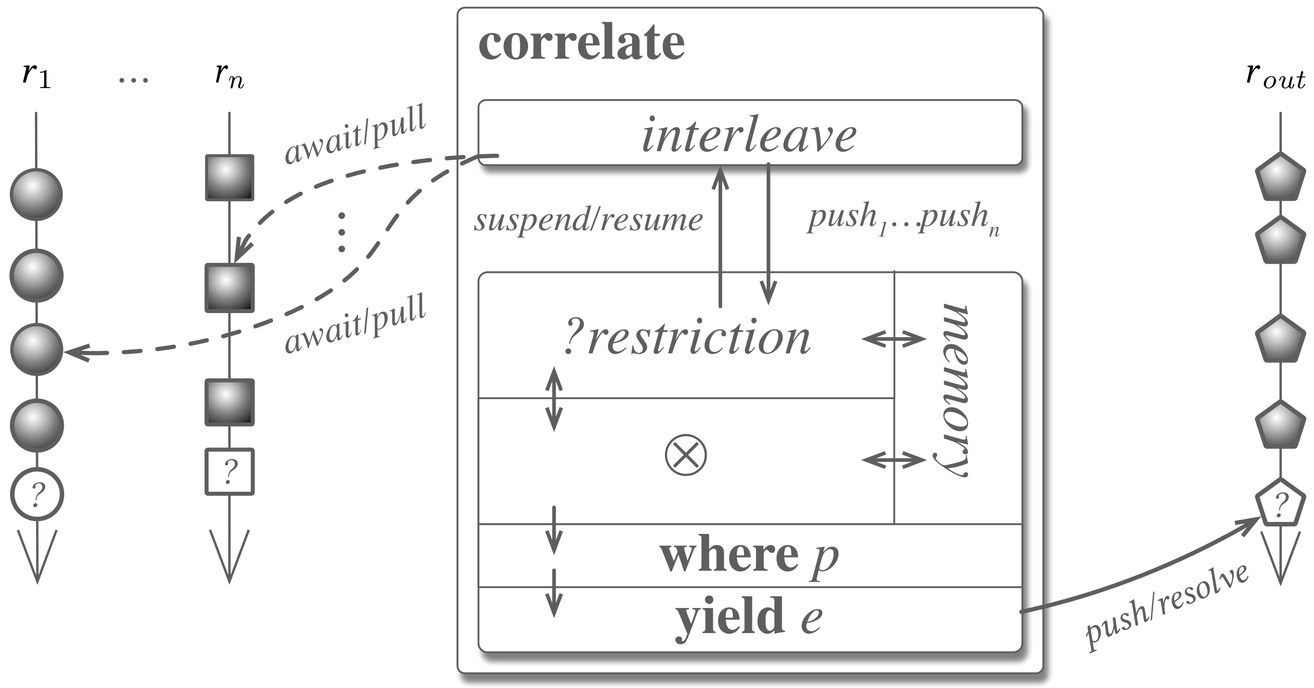
```
let r_out =
  let implicit ?restriction = h in
  correlate {
    x_1 from r_1
        …
    x_n from r_n
    where p
    yield e }
```

$$\textbf{let } r_{\text{out}} = \textbf{with } \left(h_{\text{cart}} \boxplus h\right) \; \{\text{eat } r_1 \; (\textbf{push}_1)\} \parallel \cdots \parallel \{\text{eat } r_n \; (\textbf{push}_n)\}$$

THE BIG PICTURE

```
let r_out =
  let implicit ?restriction = h in
  correlate {
    x1 from r1
       …
    xn from rn
    where p
    yield e }
```

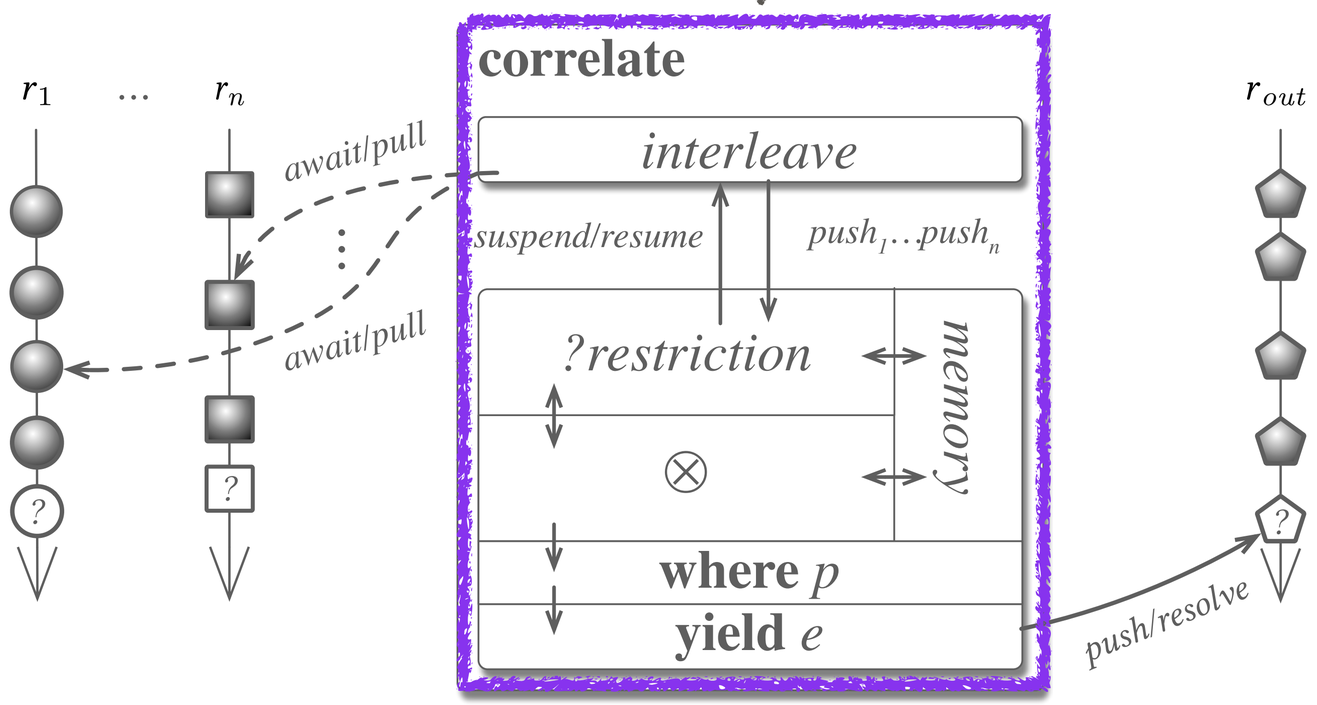
$$\textbf{let } r_{\text{out}} = \textbf{with } \left(h_{\text{cart}} \boxplus h\right) \; \{\text{eat } r_1\ (\textbf{push}_1)\} \parallel \cdots \parallel \{\text{eat } r_n\ (\textbf{push}_n)\}$$



THE BIG PICTURE

```
let r_out =
  let implicit ?restriction = h in
  correlate {
    x1 from r1
       …
    xn from rn
    where p
    yield e }
```

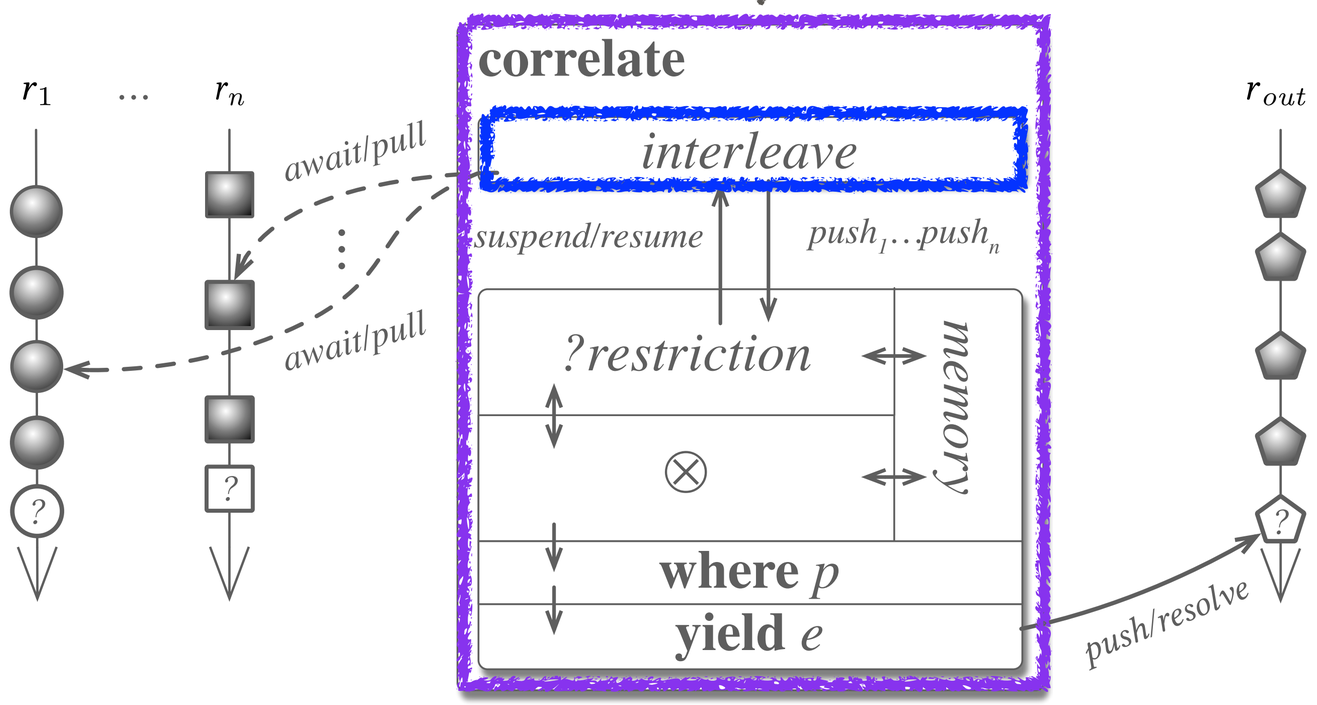
$$\textbf{let } r_{\text{out}} = \textbf{with } \left(h_{\text{cart}} \boxplus h\right) \; \{\text{eat } r_1\ (\textbf{push}_1)\} \parallel \cdots \parallel \{\text{eat } r_n\ (\textbf{push}_n)\}$$



THE BIG PICTURE

```
let r_out =
  let implicit ?restriction = h in
  correlate {
    x1 from r1
       …
    xn from rn
    where p
    yield e }
```

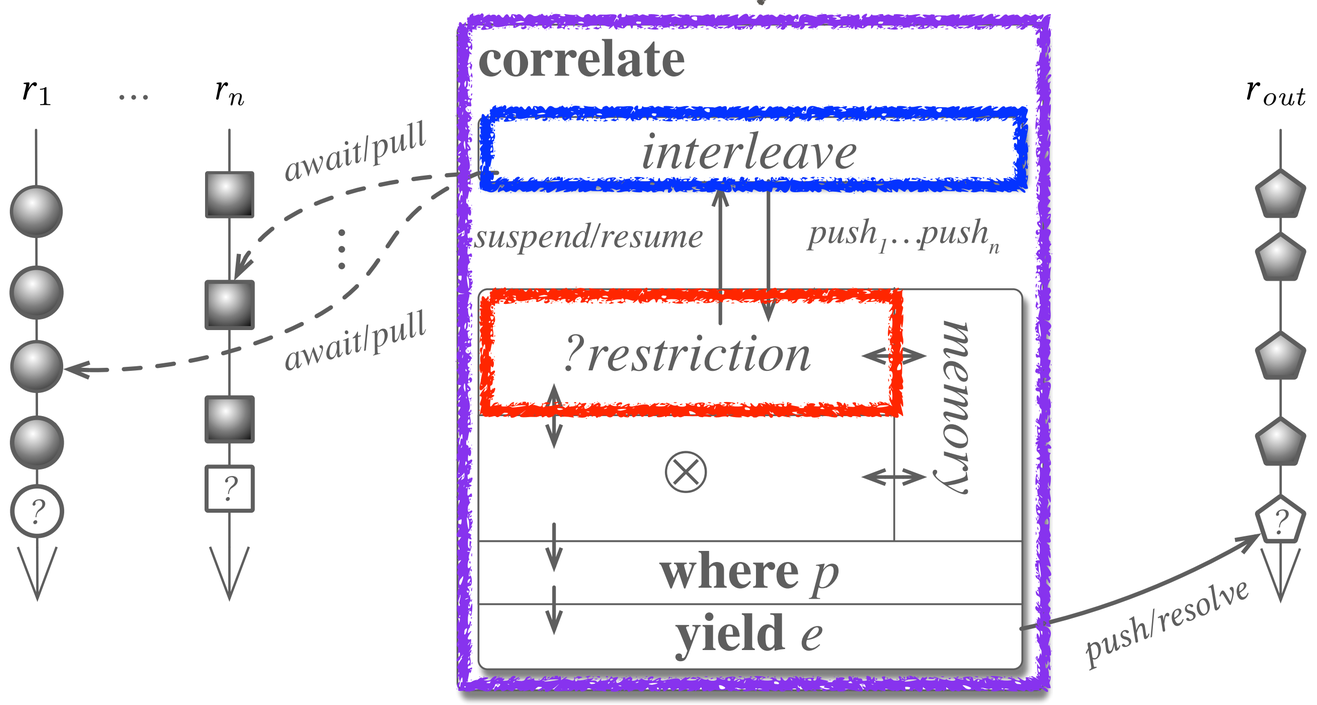
$$\textbf{let } r_{\text{out}} = \textbf{with } \left(h_{\text{cart}} \boxplus h\right) \; \{\text{eat } r_1\ (\textbf{push}_1)\} \parallel \cdots \parallel \{\text{eat } r_n\ (\textbf{push}_n)\}$$



THE BIG PICTURE

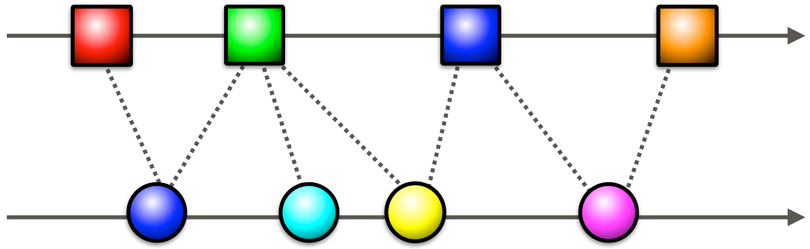
```
let r_out =
  let implicit ?restriction = h in
  correlate {
    x1 from r1
       …
    xn from rn
    where p
    yield e }
```

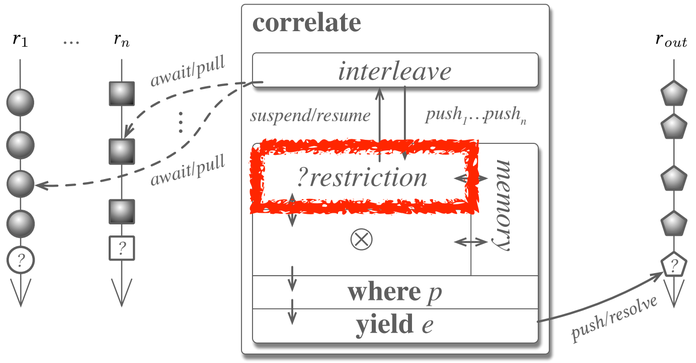
$$\textbf{let } r_{\text{out}} = \textbf{with } \left(h_{\text{cart}} \boxplus h\right) \; \{\text{eat } r_1\ (\textbf{push}_1)\} \parallel \cdots \parallel \{\text{eat } r_n\ (\textbf{push}_n)\}$$



RESTRICTION HANDLERS

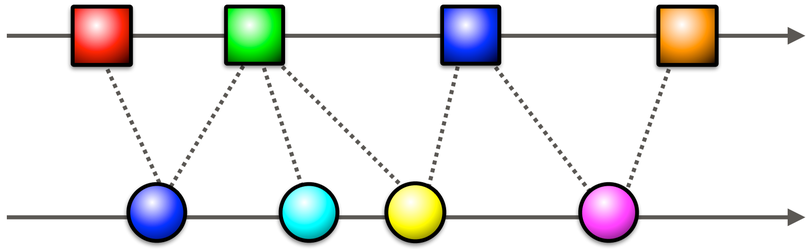
SPECIALISE THE NAIVE CARTESIAN PRODUCT

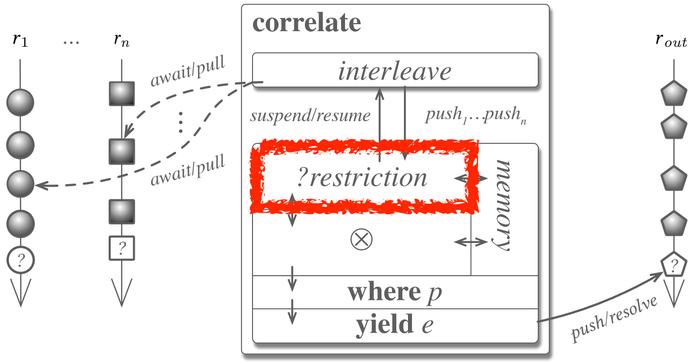




RESTRICTION HANDLERS

SPECIALISE THE NAIVE CARTESIAN PRODUCT

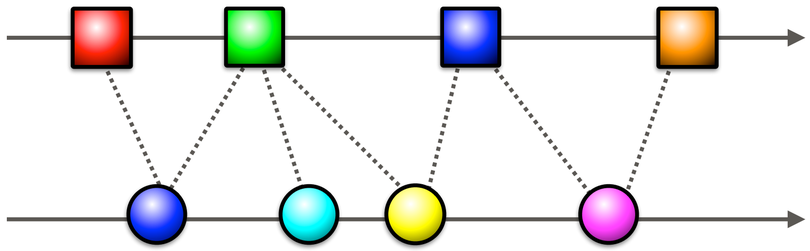


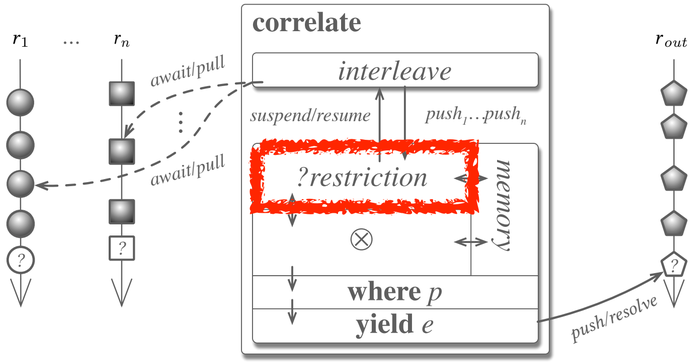



```
let mostRecently_i = handler
  push_i ev resume ->
    set_i (<ev,∞> :: nil);
    resume (push_i ev)
```

RESTRICTION HANDLERS

SPECIALISE THE NAIVE CARTESIAN PRODUCT

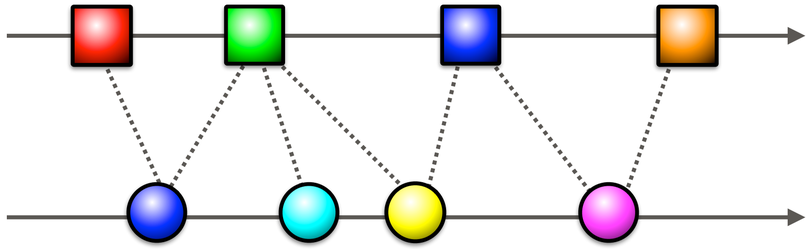


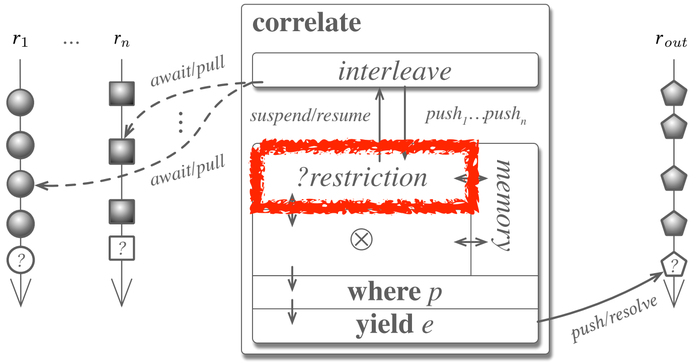



```
let mostRecently_i = handler
  push_i ev resume ->
    set_i (<ev,∞> :: nil);
    resume (push_i ev)
```

RESTRICTION HANDLERS

SPECIALISE THE NAIVE CARTESIAN PRODUCT






```
let mostRecently_i = handler
  push_i ev resume ->
    set_i (<ev,∞> :: nil);
    resume (push_i ev)
```

HOW “EFFECTIVE” ARE RESTRICTION HANDLERS?

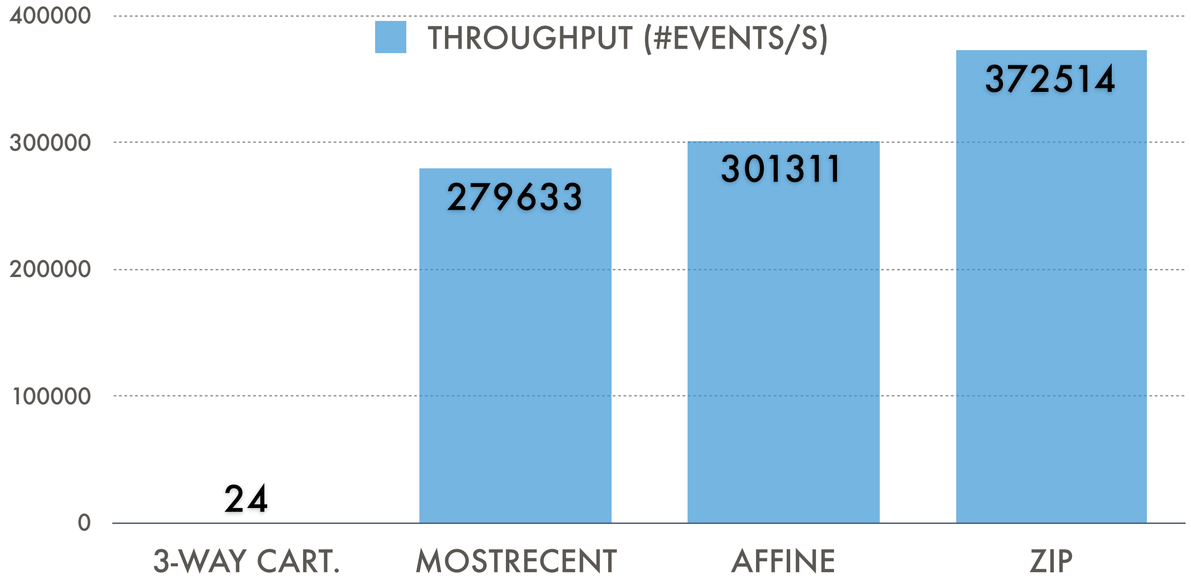
MICROBENCHMARKS

Executed 3-way join variants with ca.
10 million random events.

HOW "EFFECTIVE" ARE RESTRICTION HANDLERS?

MICROBENCHMARKS

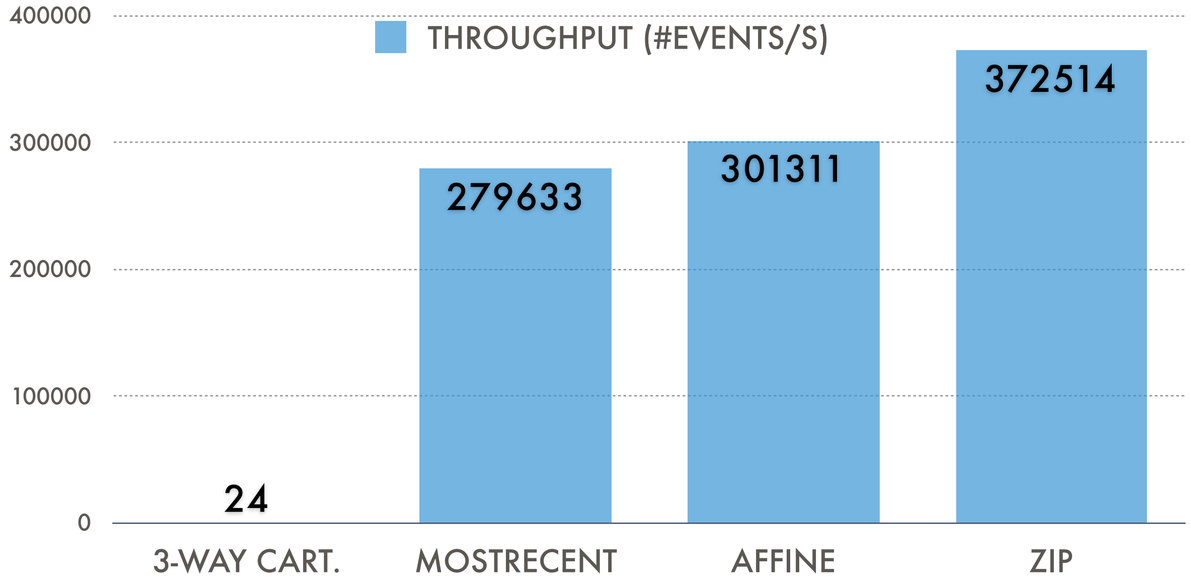
Executed 3-way join variants with ca. 10 million random events.

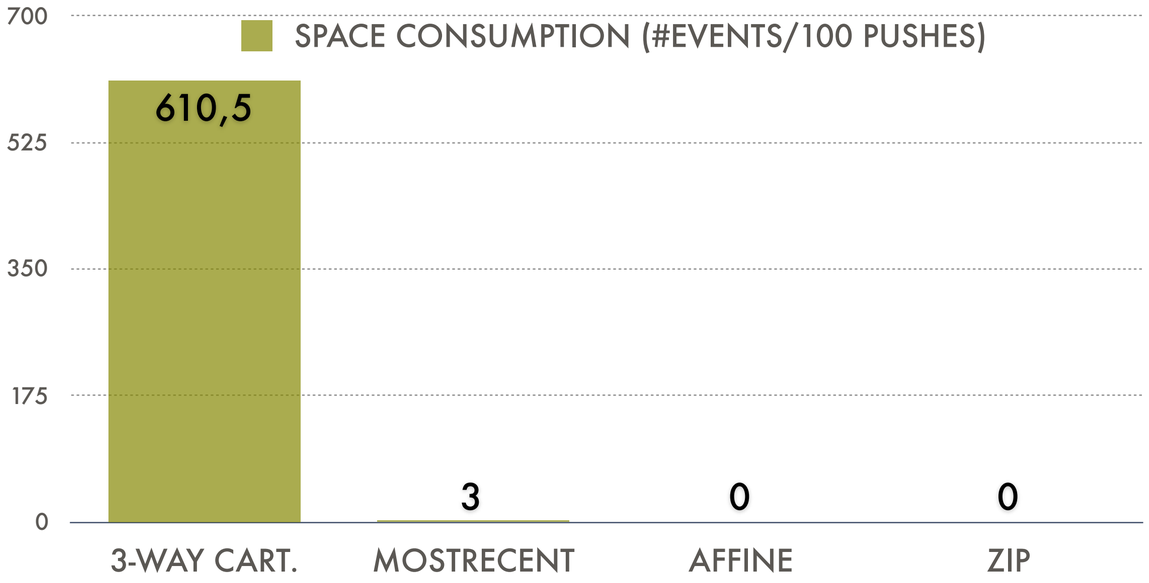


HOW "EFFECTIVE" ARE RESTRICTION HANDLERS?

MICROBENCHMARKS

Executed 3-way join variants with ca. 10 million random events.

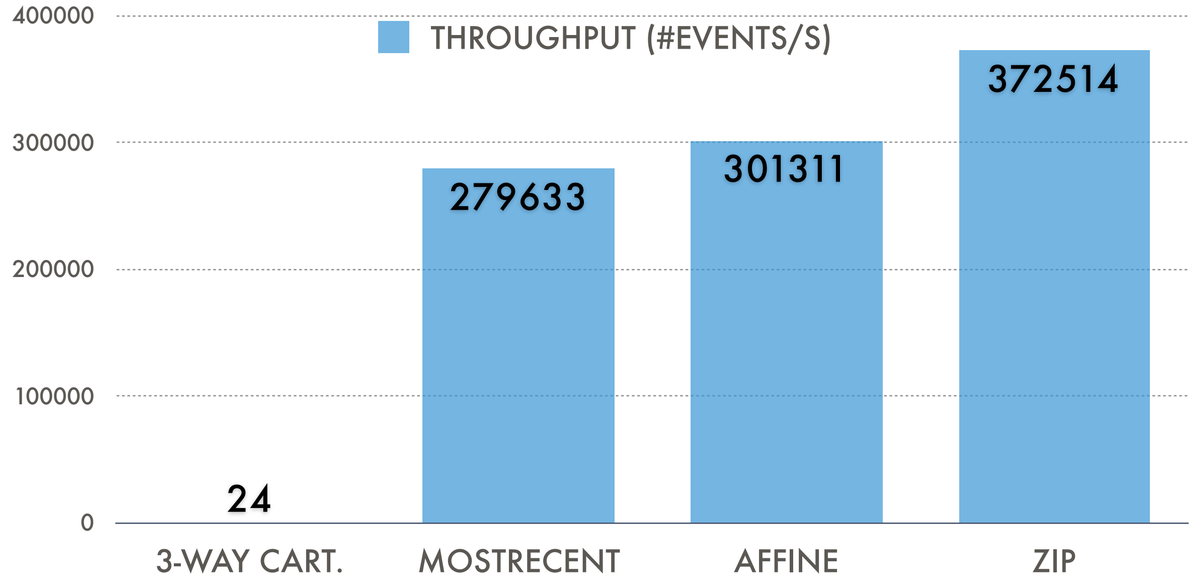


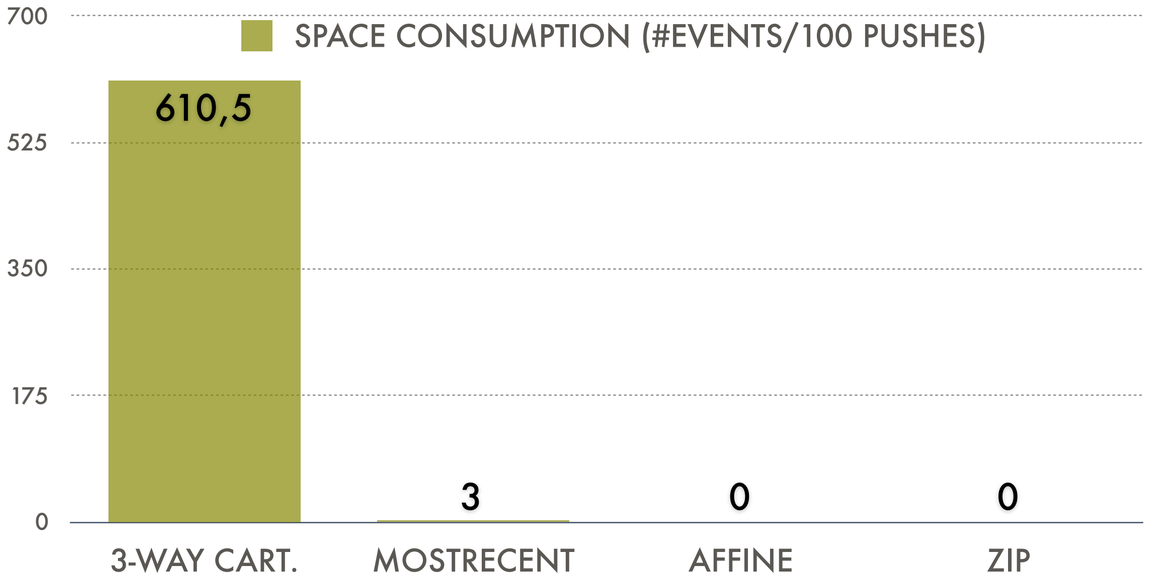


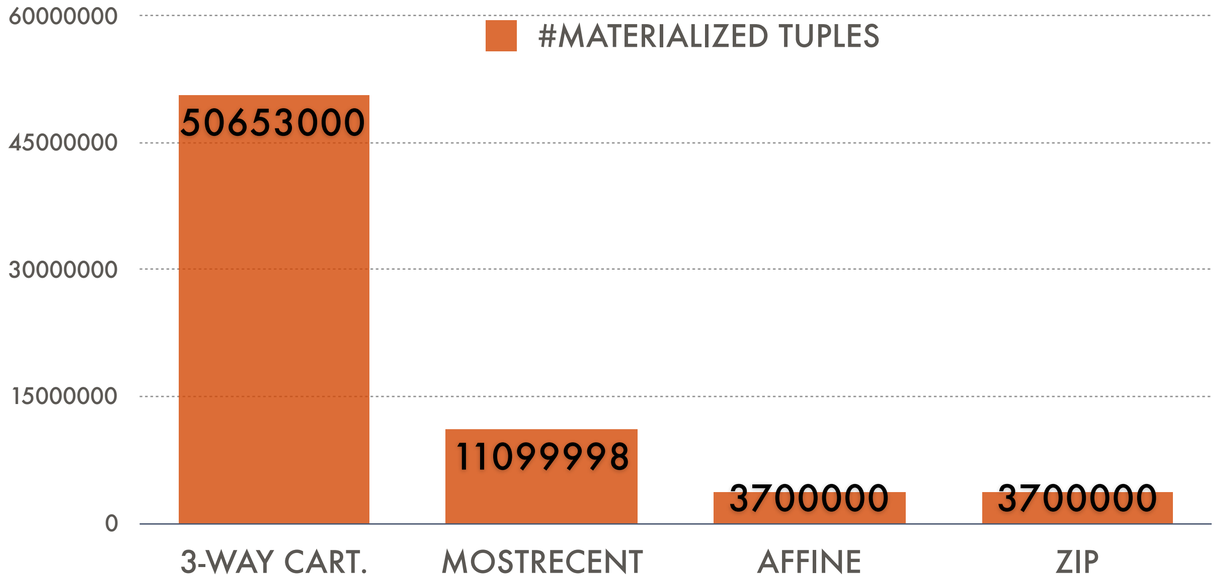
HOW "EFFECTIVE" ARE RESTRICTION HANDLERS?

MICROBENCHMARKS

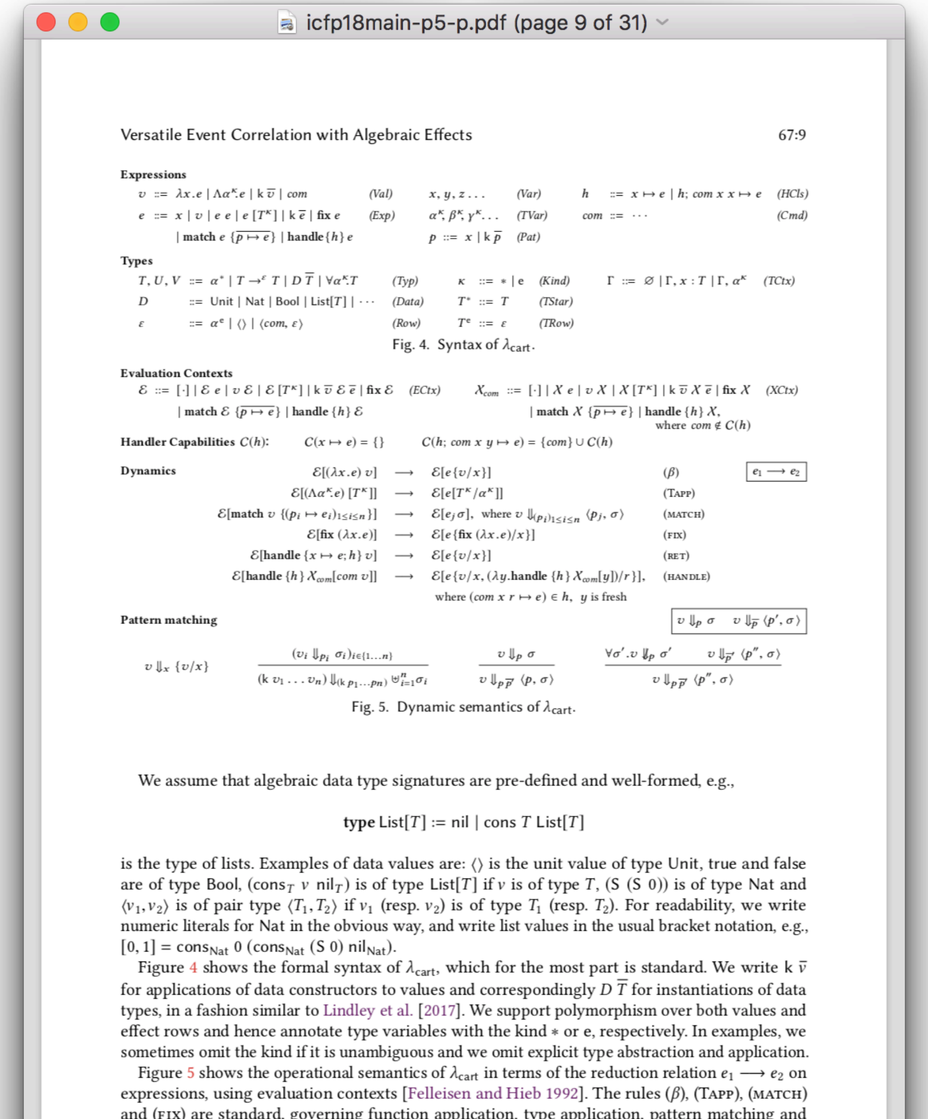
Executed 3-way join variants with ca. 10 million random events.







MORE CONTENT IN THE PAPER



Versatile Event Correlation with Algebraic Effects 67:9

Expressions

$v ::= \lambda x.e \mid \Lambda\alpha^\kappa.e \mid \mathsf{k}\,\overline{v} \mid com$ (Val) $\quad x, y, z \ldots$ (Var) $\quad h ::= x \mapsto e \mid h; com\ x\ x \mapsto e$ (HCls)

$e ::= x \mid v \mid e\ e \mid e\ [T^\kappa] \mid \mathsf{k}\,\overline{e} \mid \mathbf{fix}\ e \mid \mathbf{match}\ e\ \{\overline{p \mapsto e}\} \mid \mathbf{handle}\{h\}\,e$ (Exp) $\quad \alpha^\kappa, \beta^\kappa, \gamma^\kappa \ldots$ (TVar) $\quad com ::= \cdots$ (Cmd)

$p ::= x \mid \mathsf{k}\,\overline{p}$ (Pat)

Types

$T, U, V ::= \alpha^\kappa \mid T \to^\varepsilon T \mid D\,\overline{T} \mid \forall\alpha^\kappa.T$ (Typ) $\quad \kappa ::= * \mid \mathsf{e}$ (Kind) $\quad \Gamma ::= \varnothing \mid \Gamma, x : T \mid \Gamma, \alpha^\kappa$ (TCtx)

$D ::= \mathsf{Unit} \mid \mathsf{Nat} \mid \mathsf{Bool} \mid \mathsf{List}[T] \mid \cdots$ (Data) $\quad T^* ::= T$ (TStar)

$\varepsilon ::= \alpha^\mathsf{e} \mid \langle\rangle \mid \langle com, \varepsilon\rangle$ (Row) $\quad T^\mathsf{e} ::= \varepsilon$ (TRow)

Fig. 4. Syntax of λ_{cart}.

Evaluation Contexts

$\mathcal{E} ::= [\cdot] \mid \mathcal{E}\ e \mid v\ \mathcal{E} \mid \mathcal{E}\ [T^\kappa] \mid \mathsf{k}\,\overline{v}\ \mathcal{E}\ \overline{e} \mid \mathbf{fix}\ \mathcal{E} \mid \mathbf{match}\ \mathcal{E}\ \{\overline{p \mapsto e}\} \mid \mathbf{handle}\{h\}\,\mathcal{E}$ (ECtx)

$\mathcal{X}_{com} ::= [\cdot] \mid \mathcal{X}\ e \mid v\ \mathcal{X} \mid \mathcal{X}\ [T^\kappa] \mid \mathsf{k}\,\overline{v}\ \mathcal{X}\ \overline{e} \mid \mathbf{fix}\ \mathcal{X} \mid \mathbf{match}\ \mathcal{X}\ \{\overline{p \mapsto e}\} \mid \mathbf{handle}\{h\}\,\mathcal{X}$, where $com \notin C(h)$ (XCtx)

Handler Capabilities $C(h)$: $\quad C(x \mapsto e) = \{\} \qquad C(h; com\ x\ y \mapsto e) = \{com\} \cup C(h)$

Dynamics $\quad e_1 \longrightarrow e_2$

$$\begin{array}{rcll}
\mathcal{E}[(\lambda x.e)\ v] & \longrightarrow & \mathcal{E}[e\{v/x\}] & (\beta) \\
\mathcal{E}[(\Lambda\alpha^\kappa.e)\ [T^\kappa]] & \longrightarrow & \mathcal{E}[e[T^\kappa/\alpha^\kappa]] & (\textsc{Tapp}) \\
\mathcal{E}[\mathbf{match}\ v\ \{(p_i \mapsto e_i)_{1\le i\le n}\}] & \longrightarrow & \mathcal{E}[e_j\sigma], \text{ where } v \Downarrow_{(p_i)_{1\le i\le n}} \langle p_j, \sigma\rangle & (\textsc{match}) \\
\mathcal{E}[\mathbf{fix}\ (\lambda x.e)] & \longrightarrow & \mathcal{E}[e\{\mathbf{fix}\ (\lambda x.e)/x\}] & (\textsc{fix}) \\
\mathcal{E}[\mathbf{handle}\ \{x \mapsto e; h\}\ v] & \longrightarrow & \mathcal{E}[e\{v/x\}] & (\textsc{ret}) \\
\mathcal{E}[\mathbf{handle}\ \{h\}\ \mathcal{X}_{com}[com\ v]] & \longrightarrow & \mathcal{E}[e\{v/x, (\lambda y.\mathbf{handle}\ \{h\}\ \mathcal{X}_{com}[y])/r\}], & (\textsc{handle}) \\
 & & \text{where } (com\ x\ r \mapsto e) \in h,\ y \text{ is fresh} &
\end{array}$$

Pattern matching $\quad v \Downarrow_p \sigma \qquad v \Downarrow_{\overline{p}} \langle p', \sigma\rangle$

$$v \Downarrow_x \{v/x\} \qquad \frac{(v_i \Downarrow_{p_i} \sigma_i)_{i\in\{1\ldots n\}}}{(\mathsf{k}\ v_1 \ldots v_n) \Downarrow_{(\mathsf{k}\ p_1 \ldots p_n)} \uplus_{i=1}^n \sigma_i} \qquad \frac{v \Downarrow_p \sigma}{v \Downarrow_{p\,\overline{p}} \langle p, \sigma\rangle} \qquad \frac{\forall\sigma'.v \Downarrow_p \sigma' \qquad v \Downarrow_{\overline{p}'} \langle p'', \sigma\rangle}{v \Downarrow_{p\,\overline{p}'} \langle p'', \sigma\rangle}$$

Fig. 5. Dynamic semantics of λ_{cart}.

We assume that algebraic data type signatures are pre-defined and well-formed, e.g.,

$$\mathbf{type}\ \mathsf{List}[T] := \mathsf{nil} \mid \mathsf{cons}\ T\ \mathsf{List}[T]$$

is the type of lists. Examples of data values are: $\langle\rangle$ is the unit value of type Unit, true and false are of type Bool, $(\mathsf{cons}_T\ v\ \mathsf{nil}_T)$ is of type $\mathsf{List}[T]$ if v is of type T, $(\mathsf{S}\ (\mathsf{S}\ 0))$ is of type Nat and $\langle v_1, v_2\rangle$ is of pair type $\langle T_1, T_2\rangle$ if v_1 (resp. v_2) is of type T_1 (resp. T_2). For readability, we write numeric literals for Nat in the obvious way, and write list values in the usual bracket notation, e.g., $[0, 1] = \mathsf{cons}_{\mathsf{Nat}}\ 0\ (\mathsf{cons}_{\mathsf{Nat}}\ (\mathsf{S}\ 0)\ \mathsf{nil}_{\mathsf{Nat}})$.

Figure 4 shows the formal syntax of λ_{cart}, which for the most part is standard. We write $\mathsf{k}\ \overline{v}$ for applications of data constructors to values and correspondingly $D\ \overline{T}$ for instantiations of data types, in a fashion similar to Lindley et al. [2017]. We support polymorphism over both values and effect rows and hence annotate type variables with the kind $*$ or e, respectively. In examples, we sometimes omit the kind if it is unambiguous and we omit explicit type abstraction and application.

Figure 5 shows the operational semantics of λ_{cart} in terms of the reduction relation $e_1 \longrightarrow e_2$ on expressions, using evaluation contexts [Felleisen and Hieb 1992]. The rules (β), (Tapp), (match) and (fix) are standard, governing function application, type application, pattern matching and

- Formal semantics (SOS) & type system
- Combinator source code
- Survey of join features across the board
- Koka & multicore OCaml implementations
- Implicit variables as effects/handlers

MORE CONTENT IN THE PAPER

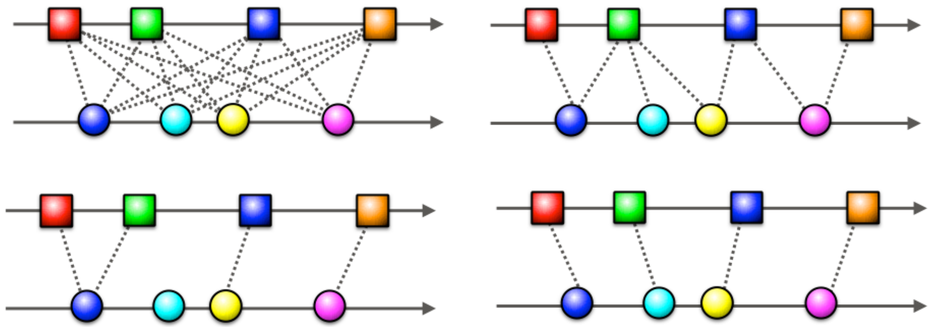
(AND TONIGHT'S POSTER SESSION!)

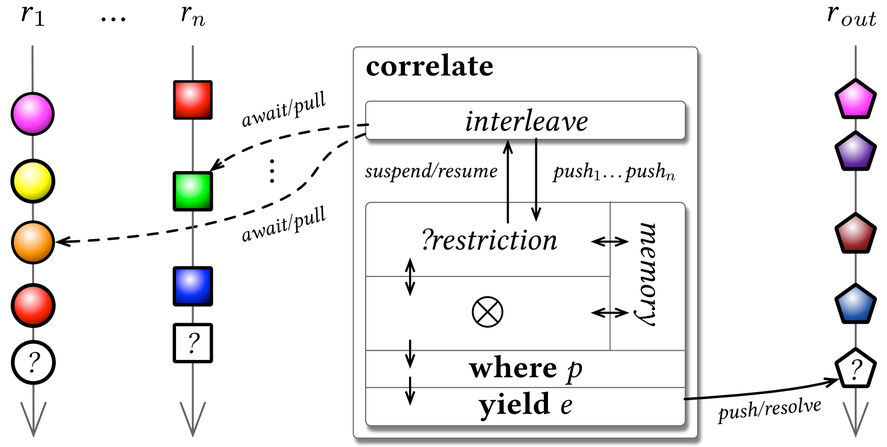
https://open.library.ubc.ca/collections/wwposters/items/1.0038235

- Formal semantics (SOS) & type system
- Combinator source code
- Survey of join features across the board
- Koka & multicore OCaml implementations
- Implicit variables as effects/handlers

CONCLUSIONS

- The **first** work that **uniformly** expresses join variants.
- Programming of **"interleaved coroutines".**
- Effects **make naïve enumerations efficient.**





BIBLIOGRAPHY

Agrawal, J., Diao, Y., Gyllstrom, D., and Immerman, N. (2008). Efficient pattern matching over event streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

Arasu, A., Babu, S., and Widom, J. (2004). CQL: A language for continuous queries over streams and relations. In *Database Programming Languages*. Springer Berlin Heidelberg.

Conchon, S. and Le Fessant, F. (1999). JoCaml: mobile agents for Objective-Caml. In *First and Third International Symposium on Agent Systems Applications, and Mobile Agents (ASAMA)*.

Cugola, G. and Margara, A. (2012). Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Computing Surveys*, 44(3):15:1-15:62.

Czaplicki, E. and Chong, S. (2013). Asynchronous Functional Reactive Programming for GUIs. In *Proceedings of Conference on Programming Language Design and Implementation (PLDI)*.

Edwards, J. (2009). Coherent reaction. In *Proceedings of Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*.

Fournet, C. and Gonthier, G. (1996). The reflexive CHAM and the Join-calculus. In *Proceedings of Symposium on Principles of Programming Languages (POPL)*.

Leijen, D. (2017a). Structured asynchrony with algebraic effects. In *Proceedings of the International Workshop on Type-Driven Development (TyDe)*.

Leijen, D. (2017b). Type directed compilation of row-typed algebraic effects. In *Proceedings of Symposium on Principles of Programming Languages (POPL)*.

Lindley, S., McBride, C., and McLaughlin, C. (2017). Do be do be do. In *Proceedings of Symposium on Principles of Programming Languages (POPL)*.

Meijer, E., Beckman, B., and Bierman, G. (2006). LINQ: Reconciling object, relations and XML in the .NET framework. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

Plotkin, G. D. and Power, J. (2003). Algebraic operations and generic effects. *Applied Categorical Structures*.

Plotkin, G. D. and Pretnar, M. (2009). Handlers of algebraic effects. In *Proceedings of the European Conference on Programming Languages and Systems (ESOP)*.

Wadler, P. (1992). Comprehending monads. *Mathematical Structures in Computer Science*, 2(4):461-493.