Towards a causal and compositional operational semantics of programming languages

Simon Castellan[1]

[1]Imperial College London, UK

November 21th, 2016
LSV Seminar

Message-passing on my computer

Consider this program `mp`:

$$\texttt{data} = \texttt{flag} = 0$$
$$\begin{array}{l} \texttt{data} := 17; \\ \texttt{flag} := 1 \end{array} \;\Bigg\|\; \begin{array}{l} r \leftarrow \texttt{flag}; \\ \texttt{if}(r == 1)\{v \leftarrow \texttt{data}\} \end{array}$$

Message-passing on my computer

Consider this program `mp`:

$$\begin{array}{c} \texttt{data} = \texttt{flag} = 0 \\ \begin{array}{l} \texttt{data} := 17; \\ \texttt{flag} := 1 \end{array} \;\Bigg\|\; \begin{array}{l} r \leftarrow \texttt{flag}; \\ \texttt{if}(r == 1)\{v \leftarrow \texttt{data}\} \end{array} \end{array}$$

Possible execution traces on my computer:

Message-passing on my computer

Consider this program mp:

$$\begin{array}{c} \texttt{data} = \texttt{flag} = 0 \\ \begin{array}{l} \texttt{data} := 17; \\ \texttt{flag} := 1 \end{array} \;\Big\|\; \begin{array}{l} r \leftarrow \texttt{flag}; \\ \texttt{if}(r == 1)\{v \leftarrow \texttt{data}\} \end{array} \end{array}$$

Possible execution traces on my computer:

- $\mathtt{W}_{\texttt{data}:=17}$

Message-passing on my computer

Consider this program mp:

$$\begin{array}{c} \texttt{data} = \texttt{flag} = 0 \\ \begin{array}{l} \texttt{data} := 17; \\ \texttt{flag} := 1 \end{array} \;\Big\|\; \begin{array}{l} r \leftarrow \texttt{flag}; \\ \texttt{if}(r == 1)\{v \leftarrow \texttt{data}\} \end{array} \end{array}$$

Possible execution traces on my computer:

- $\texttt{W}_{\texttt{data}:=17} \cdot \texttt{W}_{\texttt{flag}:=1}$

Message-passing on my computer

Consider this program mp:

$$
\begin{array}{c}
\texttt{data} = \texttt{flag} = 0 \\
\begin{array}{l||l}
\texttt{data} := 17; & r \leftarrow \texttt{flag}; \\
\texttt{flag} := 1 & \texttt{if}(r == 1)\{v \leftarrow \texttt{data}\}
\end{array}
\end{array}
$$

Possible execution traces on my computer:

- $\texttt{W}_{\texttt{data}:=17} \cdot \texttt{W}_{\texttt{flag}:=1} \cdot \texttt{R}_{\texttt{flag}=1}$

Message-passing on my computer

Consider this program mp:

$$\texttt{data} = \texttt{flag} = 0$$
$$\begin{array}{l} \texttt{data} := 17; \\ \texttt{flag} := 1 \end{array} \;\Bigg\|\; \begin{array}{l} r \leftarrow \texttt{flag}; \\ \texttt{if}(r == 1)\{v \leftarrow \texttt{data}\} \end{array}$$

Possible execution traces on my computer:

- $\mathtt{W}_{\texttt{data}:=17} \cdot \mathtt{W}_{\texttt{flag}:=1} \cdot \mathtt{R}_{\texttt{flag}=1} \cdot \mathtt{R}_{\texttt{data}=17}$

Message-passing on my computer

Consider this program mp:

$$\mathtt{data} = \mathtt{flag} = 0$$
$$\begin{array}{l} \mathtt{data} := 17; \\ \mathtt{flag} := 1 \end{array} \;\Big\|\; \begin{array}{l} r \leftarrow \mathtt{flag}; \\ \mathtt{if}(r == 1)\{v \leftarrow \mathtt{data}\} \end{array}$$

Possible execution traces on my computer:

- $\mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17}$
- $\mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1}$
- $\mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1}$

Message-passing on my phone

$$\texttt{data} = \texttt{flag} = 0$$
$$\begin{array}{l} \texttt{data} := 17; \\ \texttt{flag} := 1 \end{array} \;\Big\|\; \begin{array}{l} r \leftarrow \texttt{flag}; \\ \texttt{if}(r == 1)\{v \leftarrow \texttt{data}\} \end{array}$$

Possible execution traces on my **phone**:

- $\mathtt{W}_{\texttt{data}:=17} \cdot \mathtt{W}_{\texttt{flag}:=1} \cdot \mathtt{R}_{\texttt{flag}=1} \cdot \mathtt{R}_{\texttt{data}=17}$
- $\mathtt{W}_{\texttt{data}:=17} \cdot \mathtt{R}_{\texttt{flag}=0} \cdot \mathtt{W}_{\texttt{flag}:=1}$
- $\mathtt{R}_{\texttt{flag}=0} \cdot \mathtt{W}_{\texttt{data}:=17} \cdot \mathtt{W}_{\texttt{flag}:=1}$

Message-passing on my phone

$$\begin{array}{c} \texttt{data} = \texttt{flag} = 0 \\ \begin{array}{l} \texttt{data} := 17; \\ \texttt{flag} := 1 \end{array} \;\Bigg\|\; \begin{array}{l} r \leftarrow \texttt{flag}; \\ \texttt{if}(r == 1)\{v \leftarrow \texttt{data}\} \end{array} \end{array}$$

Possible execution traces on my **phone**:

- $\mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17}$
- $\mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1}$
- $\mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1}$
- $\mathtt{W}_{\mathtt{flag}:=1}$

Message-passing on my phone

$$\texttt{data} = \texttt{flag} = 0$$

$$\begin{array}{l} \texttt{data} := 17; \\ \texttt{flag} := 1 \end{array} \;\Big\|\; \begin{array}{l} r \leftarrow \texttt{flag}; \\ \texttt{if}(r == 1)\{v \leftarrow \texttt{data}\} \end{array}$$

Possible execution traces on my **phone**:

- $\mathtt{W}_{\texttt{data}:=17} \cdot \mathtt{W}_{\texttt{flag}:=1} \cdot \mathtt{R}_{\texttt{flag}=1} \cdot \mathtt{R}_{\texttt{data}=17}$
- $\mathtt{W}_{\texttt{data}:=17} \cdot \mathtt{R}_{\texttt{flag}=0} \cdot \mathtt{W}_{\texttt{flag}:=1}$
- $\mathtt{R}_{\texttt{flag}=0} \cdot \mathtt{W}_{\texttt{data}:=17} \cdot \mathtt{W}_{\texttt{flag}:=1}$
- $\mathtt{W}_{\texttt{flag}:=1} \cdot \mathtt{R}_{\texttt{flag}=1}$

Message-passing on my phone

$$\texttt{data} = \texttt{flag} = 0$$
$$\begin{array}{l} \texttt{data} := 17; \\ \texttt{flag} := 1 \end{array} \;\Bigg\|\; \begin{array}{l} r \leftarrow \texttt{flag}; \\ \texttt{if}(r == 1)\{v \leftarrow \texttt{data}\} \end{array}$$

Possible execution traces on my **phone**:

- $\mathtt{W}_{\texttt{data}:=17} \cdot \mathtt{W}_{\texttt{flag}:=1} \cdot \mathtt{R}_{\texttt{flag}=1} \cdot \mathtt{R}_{\texttt{data}=17}$
- $\mathtt{W}_{\texttt{data}:=17} \cdot \mathtt{R}_{\texttt{flag}=0} \cdot \mathtt{W}_{\texttt{flag}:=1}$
- $\mathtt{R}_{\texttt{flag}=0} \cdot \mathtt{W}_{\texttt{data}:=17} \cdot \mathtt{W}_{\texttt{flag}:=1}$
- $\mathtt{W}_{\texttt{flag}:=1} \cdot \mathtt{R}_{\texttt{flag}=1} \cdot \mathtt{R}_{\texttt{data}=0}$

Message-passing on my phone

$$\mathtt{data} = \mathtt{flag} = 0$$

$$\begin{array}{l} \mathtt{data} := 17; \\ \mathtt{flag} := 1 \end{array} \;\Bigg\|\; \begin{array}{l} r \leftarrow \mathtt{flag}; \\ \mathtt{if}(r == 1)\{v \leftarrow \mathtt{data}\} \end{array}$$

Possible execution traces on my **phone**:

- $\mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17}$
- $\mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1}$
- $\mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1}$
- $\mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=0} \cdot \mathtt{W}_{\mathtt{data}:=17}$

Message-passing on my phone

$$\texttt{data} = \texttt{flag} = 0$$
$$\begin{array}{l} \texttt{data} := 17; \\ \texttt{flag} := 1 \end{array} \;\Big\|\; \begin{array}{l} r \leftarrow \texttt{flag}; \\ \texttt{if}(r == 1)\{v \leftarrow \texttt{data}\} \end{array}$$

Possible execution traces on my **phone**:

- $\mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17}$
- $\mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1}$
- $\mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1}$
- $\mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=0} \cdot \mathtt{W}_{\mathtt{data}:=17}$
- $\mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{data}=17}$
- $\mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17}$
- $\mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{W}_{\mathtt{data}:=17}$

A different **architecture**, much harder to reason about...

Structure behind traces

$$\begin{cases} \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17} \\ \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{data}=17} \\ \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17} \end{cases}$$

$$\left\{ \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \right.$$

$$\begin{cases} \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \\ \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1} \\ \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \end{cases}$$

Structure behind traces

$$\begin{cases} \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17} \\ \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{data}=17} \\ \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17} \end{cases}$$

$$\begin{tikzcd} \mathtt{W}_{\mathtt{flag}:=1} \arrow[d] & \mathtt{W}_{\mathtt{data}:=17} \arrow[d] \\ \mathtt{R}_{\mathtt{flag}=1} \arrow[r] & \mathtt{R}_{\mathtt{data}=17} \end{tikzcd}$$

$$\left\{ \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \right.$$

$$\begin{cases} \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \\ \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1} \\ \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \end{cases}$$

Structure behind traces

$$\begin{cases} \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17} \\ \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{data}=17} \\ \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17} \end{cases}$$

$$\begin{tikzcd} \mathtt{W}_{\mathtt{flag}:=1} \arrow[d] & \mathtt{W}_{\mathtt{data}:=17} \arrow[d] \\ \mathtt{R}_{\mathtt{flag}=1} \arrow[r] & \mathtt{R}_{\mathtt{data}=17} \end{tikzcd}$$

$$\left\{ \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \right.$$

$$\begin{tikzcd} \mathtt{W}_{\mathtt{flag}:=1} \arrow[d] & \mathtt{R}_{\mathtt{data}=0} \arrow[d] \\ \mathtt{R}_{\mathtt{flag}=1} \arrow[ur] & \mathtt{W}_{\mathtt{data}:=17} \end{tikzcd}$$

$$\begin{cases} \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \\ \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1} \\ \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \end{cases}$$

Structure behind traces

$$\begin{cases} \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17} \\ \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{data}=17} \\ \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17} \end{cases}$$

$$\begin{tikzcd} \mathtt{W}_{\mathtt{flag}:=1} \arrow[d] & \mathtt{W}_{\mathtt{data}:=17} \arrow[d] \\ \mathtt{R}_{\mathtt{flag}=1} \arrow[r] & \mathtt{R}_{\mathtt{data}=17} \end{tikzcd}$$

$$\left\{ \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \right.$$

$$\begin{tikzcd} \mathtt{W}_{\mathtt{flag}:=1} \arrow[d] & \mathtt{R}_{\mathtt{data}=0} \arrow[d] \\ \mathtt{R}_{\mathtt{flag}=1} \arrow[ur] & \mathtt{W}_{\mathtt{data}:=17} \end{tikzcd}$$

$$\begin{cases} \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \\ \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1} \\ \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \end{cases}$$

$$\begin{tikzcd} \mathtt{R}_{\mathtt{flag}=0} \arrow[d] & \mathtt{W}_{\mathtt{data}:=17} \\ \mathtt{W}_{\mathtt{flag}:=1} & \end{tikzcd}$$

Structure behind traces

$$\begin{cases} \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17} \\ \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{data}=17} \\ \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=17} \end{cases}$$

$$\begin{tikzcd} \mathtt{W}_{\mathtt{flag}:=1} \arrow[d] & \mathtt{W}_{\mathtt{data}:=17} \arrow[d] \\ \mathtt{R}_{\mathtt{flag}=1} \arrow[r] & \mathtt{R}_{\mathtt{data}=17} \end{tikzcd}$$

$$\left\{ \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{R}_{\mathtt{flag}=1} \cdot \mathtt{R}_{\mathtt{data}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \right.$$

$$\begin{tikzcd} \mathtt{W}_{\mathtt{flag}:=1} \arrow[d] & \mathtt{R}_{\mathtt{data}=0} \arrow[d] \\ \mathtt{R}_{\mathtt{flag}=1} \arrow[ur] & \mathtt{W}_{\mathtt{data}:=17} \end{tikzcd}$$

$$\begin{cases} \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{W}_{\mathtt{flag}:=1} \\ \mathtt{W}_{\mathtt{data}:=17} \cdot \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1} \\ \mathtt{R}_{\mathtt{flag}=0} \cdot \mathtt{W}_{\mathtt{flag}:=1} \cdot \mathtt{W}_{\mathtt{data}:=17} \end{cases}$$

$$\begin{tikzcd} \mathtt{R}_{\mathtt{flag}=0} \arrow[d] & \mathtt{W}_{\mathtt{data}:=17} \\ \mathtt{W}_{\mathtt{flag}:=1} & \end{tikzcd}$$

Sets of partial orders and event structures

The **set of partial orders** describes the semantics of `mp`:

$$\left\{ \begin{tikzcd} \mathtt{W}_{\mathtt{flag}:=1} \arrow[d] & \mathtt{W}_{\mathtt{data}:=17} \arrow[d] \\ \mathtt{R}_{\mathtt{flag}=1} \arrow[r] & \mathtt{R}_{\mathtt{data}=17} \end{tikzcd} \;,\; \begin{tikzcd} \mathtt{W}_{\mathtt{flag}:=1} \arrow[d] \\ \mathtt{R}_{\mathtt{flag}=1} \arrow[d] \\ \mathtt{R}_{\mathtt{data}=0} \arrow[d] \\ \mathtt{W}_{\mathtt{data}:=17} \end{tikzcd} \;,\; \begin{tikzcd} \mathtt{R}_{\mathtt{flag}=0} \arrow[d] & \mathtt{W}_{\mathtt{data}:=17} \\ \mathtt{W}_{\mathtt{data}:=1} & \end{tikzcd} \right\}$$

Sets of partial orders and event structures

The **set of partial orders** describes the semantics of `mp`:

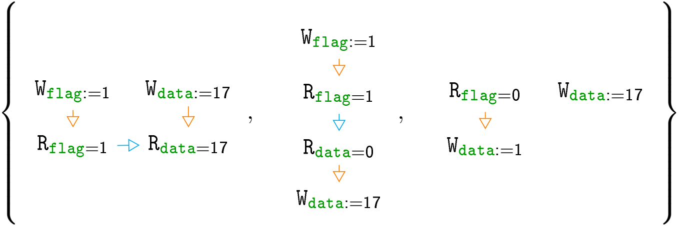
$$\left\{
\begin{tikzcd}[column sep=small]
\mathtt{W}_{\mathtt{flag}}{:=}1 \arrow[d] & \mathtt{W}_{\mathtt{data}}{:=}17 \arrow[d] \\
\mathtt{R}_{\mathtt{flag}}{=}1 \arrow[r] & \mathtt{R}_{\mathtt{data}}{=}17
\end{tikzcd}
\; , \;
\begin{tikzcd}
\mathtt{W}_{\mathtt{flag}}{:=}1 \arrow[d] \\
\mathtt{R}_{\mathtt{flag}}{=}1 \arrow[d] \\
\mathtt{R}_{\mathtt{data}}{=}0 \arrow[d] \\
\mathtt{W}_{\mathtt{data}}{:=}17
\end{tikzcd}
\; , \;
\begin{tikzcd}[column sep=small]
\mathtt{R}_{\mathtt{flag}}{=}0 \arrow[d] & \mathtt{W}_{\mathtt{data}}{:=}17 \\
\mathtt{W}_{\mathtt{data}}{:=}1 &
\end{tikzcd}
\right\}$$

This set of partial orders can be summed by an **event structure**:

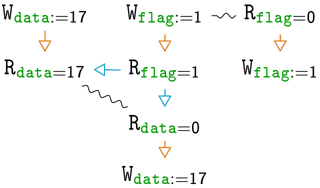
$$\begin{tikzcd}
\mathtt{W}_{\mathtt{data}}{:=}17 \arrow[d] & \mathtt{W}_{\mathtt{flag}}{:=}1 \arrow[d] \arrow[r, no head, squiggly] & \mathtt{R}_{\mathtt{flag}}{=}0 \arrow[d] \\
\mathtt{R}_{\mathtt{data}}{=}17 \arrow[dr, no head, squiggly] & \mathtt{R}_{\mathtt{flag}}{=}1 \arrow[l] \arrow[d] & \mathtt{W}_{\mathtt{flag}}{:=}1 \\
& \mathtt{R}_{\mathtt{data}}{=}0 \arrow[d] & \\
& \mathtt{W}_{\mathtt{data}}{:=}17 &
\end{tikzcd}$$

Sets of partial orders and event structures

The **set of partial orders** describes the semantics of `mp`:



This set of partial orders can be summed by an **event structure**:



This talk

1. **Modelling a first-order programming language.**
   With relaxed shared memory.

2. **When actions become contextual.**
   An introduction to game semantics for higher-order languages.

3. **Interpretating a higher-order language, adequately.**
   With concurrency & non-determinism.

I. Modelling a first-order programming language

An ARM-like semantics for a toy language

An assembly language with relaxed semantics

Syntax. Idents split in thread-local registers and global `variables`.

$$e ::= r \mid e + e \mid \dots$$

$$t ::= \texttt{fence}; t \mid \texttt{x} := e; t \mid r \leftarrow \texttt{x}; t$$

$$p ::= t \parallel \dots \parallel t$$

Actions. We observe the following actions from the programs:

$$\Sigma ::= \texttt{W}_{\texttt{x}:=k} \mid \texttt{R}_{\texttt{x}=k} \mid \texttt{fence}.$$

Semantics. Described by a labeled transition system on states p, μ:

$$\langle p \mathbin{@} \mu \rangle \xrightarrow{\ell \in \Sigma} \langle p' \mathbin{@} \mu' \rangle. \qquad (\mu, \mu' : \texttt{Var} \to \mathbb{N})$$

It is **relaxed**: operations on independent variables can be reordered.

A few rules

Thread rules: $\langle t \mathbin{@} \mu\rangle \xrightarrow{\ell} \langle t' \mathbin{@} \mu'\rangle$:

$$\frac{}{\langle \mathtt{x} := k; t@\mu\rangle \xrightarrow{\mathtt{W}_{\mathtt{x}:=k}} \langle t@\mu[\mathtt{x} := k]\rangle}$$

A few rules

Thread rules: $\langle t \mathbin{@} \mu \rangle \xrightarrow{\ell} \langle t' \mathbin{@} \mu' \rangle$:

$$\frac{}{\langle \mathtt{x} := k; t @ \mu \rangle \xrightarrow{\mathtt{W}_{\mathtt{x}:=k}} \langle t @ \mu[\mathtt{x} := k] \rangle} \qquad \frac{}{\langle \mathtt{fence}; t @ \mu \rangle \xrightarrow{\mathtt{fence}} \langle t @ \mu \rangle}$$

A few rules

Thread rules: $\langle t \,@\, \mu\rangle \xrightarrow{\ell} \langle t' \,@\, \mu'\rangle$:

$$\frac{}{\langle \mathtt{x} := k; t@\mu\rangle \xrightarrow{\mathtt{W}_{\mathtt{x}:=k}} \langle t@\mu[\mathtt{x} := k]\rangle} \qquad \frac{}{\langle \mathtt{fence}; t@\mu\rangle \xrightarrow{\mathtt{fence}} \langle t@\mu\rangle}$$

$$\frac{\langle t@\mu\rangle \xrightarrow{\ell} \langle t'@\mu'\rangle \qquad \ell \neq \mathtt{fence} \qquad \mathtt{var}(\ell) \neq \mathtt{x}}{\langle \mathtt{x} := k; t@\mu\rangle \xrightarrow{\ell} \langle \mathtt{x} := k; t'@\mu'\rangle}$$

A few rules

Thread rules: $\langle t \,@\, \mu\rangle \xrightarrow{\ell} \langle t' \,@\, \mu'\rangle$:

$$\frac{}{\langle \mathtt{x} := k; t@\mu\rangle \xrightarrow{\mathtt{W}_{\mathtt{x}:=k}} \langle t@\mu[\mathtt{x} := k]\rangle} \qquad \frac{}{\langle \mathtt{fence}; t@\mu\rangle \xrightarrow{\mathtt{fence}} \langle t@\mu\rangle}$$

$$\frac{\langle t@\mu\rangle \xrightarrow{\ell} \langle t'@\mu'\rangle \qquad \ell \neq \mathtt{fence} \qquad \mathtt{var}(\ell) \neq \mathtt{x}}{\langle \mathtt{x} := k; t@\mu\rangle \xrightarrow{\ell} \langle \mathtt{x} := k; t'@\mu'\rangle}$$

And then:

$$\frac{\langle t_i \,@\, \mu\rangle \xrightarrow{\ell} \langle t_i' \,@\, \mu'\rangle}{\langle t_1 \parallel \ldots \parallel t_i \parallel \ldots \parallel t_n \,@\, \mu\rangle \xrightarrow{\ell} \langle t_1 \parallel \ldots \parallel t_i' \parallel \ldots \parallel t_n \,@\, \mu'\rangle}$$

A few rules

Thread rules: $\langle t \mathbin{@} \mu\rangle \xrightarrow{\ell} \langle t' \mathbin{@} \mu'\rangle$:

$$\frac{}{\langle \mathtt{x} := k; t@\mu\rangle \xrightarrow{\mathtt{W}_{\mathtt{x}:=k}} \langle t@\mu[\mathtt{x} := k]\rangle} \qquad \frac{}{\langle \mathtt{fence}; t@\mu\rangle \xrightarrow{\mathtt{fence}} \langle t@\mu\rangle}$$

$$\frac{\langle t@\mu\rangle \xrightarrow{\ell} \langle t'@\mu'\rangle \qquad \ell \neq \mathtt{fence} \qquad \mathtt{var}(\ell) \neq \mathtt{x}}{\langle \mathtt{x} := k; t@\mu\rangle \xrightarrow{\ell} \langle \mathtt{x} := k; t'@\mu'\rangle}$$

And then:

$$\frac{\langle t_i \mathbin{@} \mu\rangle \xrightarrow{\ell} \langle t_i' \mathbin{@} \mu'\rangle}{\langle t_1 \parallel \dots \parallel t_i \parallel \dots \parallel t_n \mathbin{@} \mu\rangle \xrightarrow{\ell} \langle t_1 \parallel \dots \parallel t_i' \parallel \dots \parallel t_n \mathbin{@} \mu'\rangle}$$

This generates the **operational (partial) traces**:

$$\mathrm{Tr}(p, \mu) = \{\ell_1 \dots \ell_n \mid \langle p \mathbin{@} \mu\rangle \xrightarrow{\ell_1} \dots \xrightarrow{\ell_n} \langle p' \mathbin{@} \mu'\rangle\}.$$

Labeled event structures

Definition

A (Σ-labeled) **event structure** is a tuple $(E, \leq_E, \sharp_E, \ell : E \to \Sigma)$ where $(E, \leq_E)$ is a partial order and $\sharp_E$ is a symmetric relation on E, satisfying *finite causes* and *conflict inheritance*.

$$\begin{tikzcd} a \arrow[d, -{Triangle[open]}] \arrow[dr, phantom, "\sim"] & b \arrow[d, -{Triangle[open]}] \\ c \arrow[d, -{Triangle[open]}] & d \\ e & \end{tikzcd}$$

Labeled event structures

Definition

A (Σ-labeled) **event structure** is a tuple $(E, \leq_E, \sharp_E, \ell : E \to \Sigma)$ where $(E, \leq_E)$ is a partial order and $\sharp_E$ is a symmetric relation on E, satisfying *finite causes* and *conflict inheritance*.

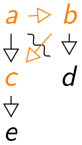
$$\begin{tikzcd} a \arrow[d] & b \arrow[d] \\ c \arrow[d] & d \\ e & \end{tikzcd}$$

- **Configurations** are downclosed, conflict-free subsets of E. $\mathscr{C}(E)$ is the set of configurations of E.

Labeled event structures

Definition

A (Σ-labeled) **event structure** is a tuple $(E, \leq_E, \sharp_E, \ell : E \to \Sigma)$ where $(E, \leq_E)$ is a partial order and $\sharp_E$ is a symmetric relation on E, satisfying *finite causes* and *conflict inheritance*.



- **Configurations** are downclosed, conflict-free subsets of E.
  $\mathscr{C}(E)$ is the set of configurations of E.

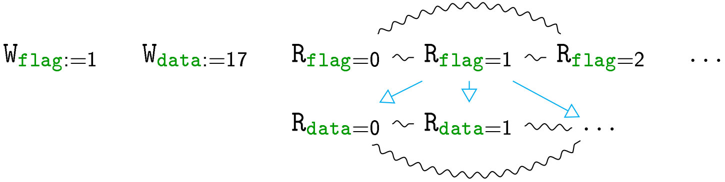
- A **trace** of E is a linearisation of a configuration of E.
  $\mathrm{Tr}(E)$ is the set of traces of E (can be seen as a subset of Σ^*).

Our goal: a mapping $[\![\cdot]\!]$ from states to event structures s.t.:

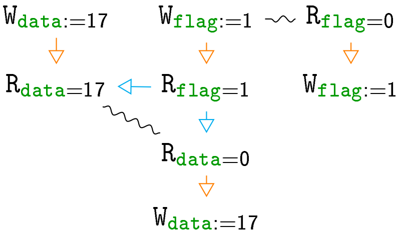
$$\mathrm{Tr}(p, \mu) = \mathrm{Tr}[\![p, \mu]\!].$$

An overview of the semantics

1. **Thread semantics**: context is left open (and unknown)



2. **Final semantics**: context is assumed empty
   Compute interactions with memory:



Thread semantics

Fences. $[\![\texttt{fence}; t]\!] = \texttt{fence} \cdot [\![t]\!]$

$$\begin{array}{c} \texttt{fence} \\ \triangledown \\ [\![t]\!] \end{array}$$

$(\leq_{\ell \cdot E} = \leq_E \cup \{(\ell, \ell')\})$

Thread semantics

Fences. $[\![\texttt{fence}; t]\!] = \texttt{fence} \cdot [\![t]\!]$

$(\leq_{\ell \cdot E} = \leq_E \cup \{(\ell, \ell')\})$

$$\begin{tikzcd} \texttt{fence} \arrow[d] \\ [\![t]\!] \end{tikzcd}$$

Writes. $[\![\texttt{x} := k; t]\!] = \texttt{W}_{\texttt{x}:=k}; [\![t]\!]$

$(\leq_{\ell ; E} = \leq_E \cup \{(\ell, \texttt{fence}), (\ell, \ell') \mid \texttt{var}(\ell) = \texttt{var}(\ell')\})$.

$$\begin{tikzcd} \texttt{W}_{\texttt{x}:=k} \arrow[d, bend right] \arrow[d, bend left] \\ [\![t]\!] \end{tikzcd}$$

Thread semantics

Fences. $[\![\mathtt{fence}; t]\!] = \mathtt{fence} \cdot [\![t]\!]$

$(\leq_{\ell \cdot E} = \leq_E \cup \{(\ell, \ell')\})$

$$\begin{tikzcd} \mathtt{fence} \arrow[d] \\ [\![t]\!] \end{tikzcd}$$

Writes. $[\![\mathtt{x} := k; t]\!] = \mathtt{W}_{\mathtt{x}:=k}; [\![t]\!]$

$(\leq_{\ell ; E} = \leq_E \cup \{(\ell, \mathtt{fence}), (\ell, \ell') \mid \mathtt{var}(\ell) = \mathtt{var}(\ell')\})$.

$$\begin{tikzcd} \mathtt{W}_{\mathtt{x}:=k} \arrow[d, bend right] \arrow[d, bend left] \\ [\![t]\!] \end{tikzcd}$$

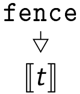
Reads. $[\![r \leftarrow \mathtt{x}; t]\!] = \sum_{n \in \mathbb{N}} \mathtt{R}_{\mathtt{x}=n}; [\![t[n/r]]\!]$

$$\begin{tikzcd} \mathtt{R}_{\mathtt{x}=0} \arrow[d, bend right] \arrow[d, bend left] & \mathtt{R}_{\mathtt{x}=1} \arrow[d, bend right] \arrow[d, bend left] & \dots \\ [\![t[0/r]]\!] & [\![t[1/r]]\!] & \dots \end{tikzcd}$$

Thread semantics

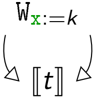
Fences. $[\![\texttt{fence}; t]\!] = \texttt{fence} \cdot [\![t]\!]$

$(\leq_{\ell \cdot E} = \leq_E \cup \{(\ell, \ell')\})$

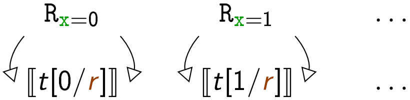


Writes. $[\![\texttt{x} := k; t]\!] = \texttt{W}_{\texttt{x}:=k}; [\![t]\!]$

$(\leq_{\ell ; E} = \leq_E \cup \{(\ell, \texttt{fence}), (\ell, \ell') \mid \texttt{var}(\ell) = \texttt{var}(\ell')\})$.



Reads. $[\![r \leftarrow \texttt{x}; t]\!] = \sum_{n \in \mathbb{N}} \texttt{R}_{\texttt{x}=n}; [\![t[n/r]]\!]$



Program. No interaction: $[\![t_1 \parallel \dots \parallel t_n]\!] = [\![t_1]\!] \parallel \dots [\![t_n]\!]$.

Wiring memory behaviour

The memory behaviour is specified through **consistent traces**:

$$C_\mu ::= \mathtt{W}_{\mathtt{x}:=k} \cdot C_{\mu[\mathtt{x}:=k]} \mid \mathtt{fence} \cdot C_\mu \mid \mathtt{R}_{\mathtt{x}=\mu(\mathtt{x})} \cdot C_\mu$$

Theorem

For a machine state (p, μ)*,* $\mathit{Tr}(p, \mu) = \mathit{Tr}[\![p]\!] \cap C_\mu$.

Wiring memory behaviour

The memory behaviour is specified through **consistent traces**:

$$C_\mu ::= \mathtt{W}_{\mathtt{x}:=k} \cdot C_{\mu[\mathtt{x}:=k]} \mid \mathtt{fence} \cdot C_\mu \mid \mathtt{R}_{\mathtt{x}=\mu(\mathtt{x})} \cdot C_\mu$$

Theorem

For a machine state (p, μ)*,* $\mathit{Tr}(p, \mu) = \mathit{Tr}[\![p]\!] \cap C_\mu$*.*

But I promised an e.s. $[\![p, \mu]\!]$! (**causally** account for memory)

Wiring memory behaviour

The memory behaviour is specified through **consistent traces**:

$$C_\mu ::= \mathtt{W}_{\mathtt{x}:=k} \cdot C_{\mu[\mathtt{x}:=k]} \mid \mathtt{fence} \cdot C_\mu \mid \mathtt{R}_{\mathtt{x}=\mu(\mathtt{x})} \cdot C_\mu$$

Theorem

For a machine state (p, μ), $\mathit{Tr}(p, \mu) = \mathit{Tr}[\![p]\!] \cap C_\mu$.

But I promised an e.s. $[\![p, \mu]\!]$! (**causally** account for memory)

The causal account of memory can be defined as:

$$\mathscr{C}_\mu = \{\mathbf{q} \mid \mathrm{Tr}(\mathbf{q}) \in C_\mu\} \in \mathrm{Set}(\mathbf{PO}).$$

How to combine $[\![p]\!]$ and $\mathscr{C}_\mu$?

Briding a gap: event-based and execution-based models

ES

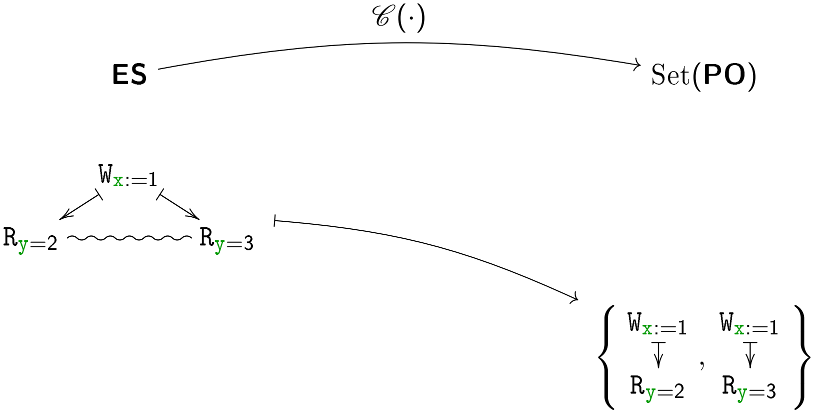
$[\![p]\!]$

Briding a gap: event-based and execution-based models

$\mathbf{ES}$ $\qquad\qquad$ $\mathrm{Set}(\mathbf{PO})$

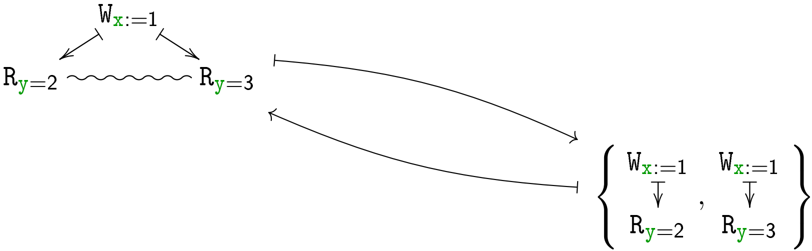
$[\![p]\!]$ $\qquad\qquad$ $\mathscr{C}_\mu$

Briding a gap: event-based and execution-based models



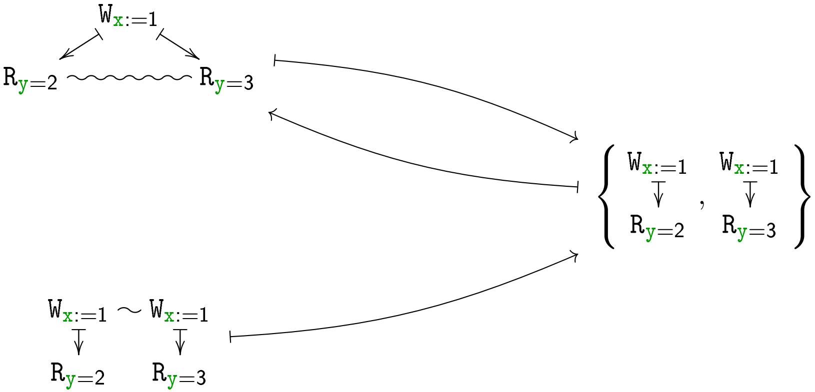
Briding a gap: event-based and execution-based models

$$\begin{tikzcd} \mathbf{ES} \arrow[rr, bend left, "\mathscr{C}(\cdot)"] & & \mathrm{Set}(\mathbf{PO}) \arrow[ll, bend left, "\mathrm{Pr}(\cdot)"] \end{tikzcd}$$



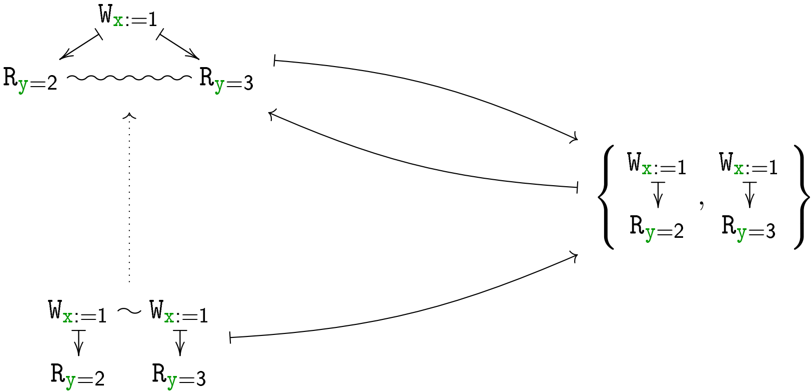
Briding a gap: event-based and execution-based models

$$
\begin{tikzcd}
\mathbf{ES} \arrow[rr, bend left, "\mathscr{C}(\cdot)"] & & \mathrm{Set}(\mathbf{PO}) \arrow[ll, bend left, "\mathrm{Pr}(\cdot)"]
\end{tikzcd}
$$



Briding a gap: event-based and execution-based models

$$
\begin{tikzcd}
\mathbf{ES} \arrow[rr, bend left, "\mathscr{C}(\cdot)"] \arrow[rr, phantom, "\top"] & & \mathrm{Set}(\mathbf{PO}) \arrow[ll, bend left, "\mathrm{Pr}(\cdot)"]
\end{tikzcd}
$$



A partial product on partial orders

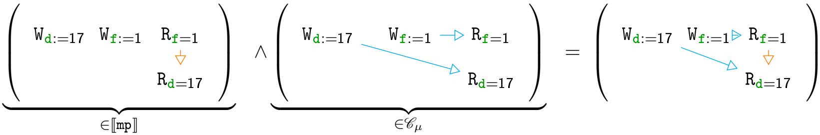
Given two partial orders $\leq_{\mathbf{q}}, \leq_{\mathbf{q}'}$ on the same carrier set, write:

$$\mathbf{q} \wedge \mathbf{q}' = \begin{cases} (\mathbf{q} \cup \mathbf{q}')^* & \text{if a partial order} \\ \text{undefined} & \text{otherwise} \end{cases}.$$

A partial product on partial orders

Given two partial orders $\leq_{\mathbf{q}}, \leq_{\mathbf{q}'}$ on the same carrier set, write:

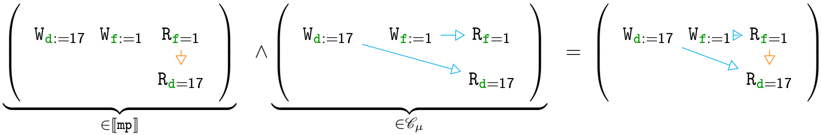
$$\mathbf{q} \wedge \mathbf{q}' = \begin{cases} (\mathbf{q} \cup \mathbf{q}')^* & \text{if a partial order} \\ \text{undefined} & \text{otherwise} \end{cases} .$$

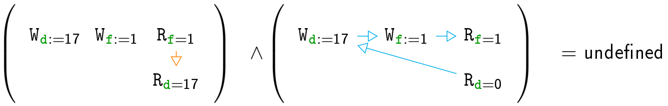


A partial product on partial orders

Given two partial orders $\leq_{\mathbf{q}}, \leq_{\mathbf{q}'}$ on the same carrier set, write:

$$\mathbf{q} \wedge \mathbf{q}' = \begin{cases} (\mathbf{q} \cup \mathbf{q}')^* & \text{if a partial order} \\ \text{undefined} & \text{otherwise} \end{cases}.$$





. . . generating a product on event structures

For $P, Q \in \mathrm{Sets}(\mathbf{PO})$, let:

$$P \star Q = \{p \wedge q \mid p \in P, q \in Q\}.$$

... generating a product on event structures

For $P, Q \in \mathrm{Sets}(\mathbf{PO})$, let:

$$P \star Q = \{p \wedge q \mid p \in P, q \in Q\}.$$

For $E, E' \in \mathbf{ES}$, let:

$$E \star E' = \mathrm{Pr}(\mathscr{C}(E) \star \mathscr{C}(E')).$$

. . . generating a product on event structures

For $P, Q \in \mathrm{Sets}(\mathbf{PO})$, let:

$$P \star Q = \{p \wedge q \mid p \in P, q \in Q\}.$$

For $E, E' \in \mathbf{ES}$, let:

$$E \star E' = \mathrm{Pr}(\mathscr{C}(E) \star \mathscr{C}(E')).$$

Theorem

Both operations are categorical products.

Note:

$$\mathrm{Tr}(E \star E') = \mathrm{Tr}(E) \cap \mathrm{Tr}(E')$$

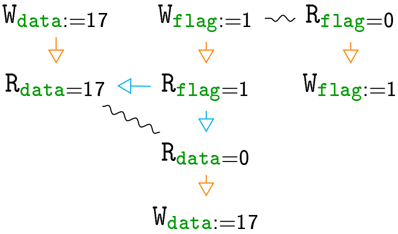
A final model

Define $[\![p, \mu]\!] = \mathrm{Pr}(\mathscr{C}([\![p]\!]) \star \mathscr{C}_\mu)$. We have:

$$\mathrm{Tr}[\![p, \mu]\!] = \mathrm{Tr}[\![p]\!] \cap \mathrm{Tr}[\![\mathscr{C}_\mu]\!] = \mathrm{Tr}[\![p]\!] \cap C_\mu = \mathrm{Tr}(p, \mu).$$

Yields the desired result:

$[\![\mathtt{mp}, (\mathtt{x} \mapsto 0)]\!] =$



Wrapping up

- This architecture is a (huge) simplification of ARM v8.0.
  We can also model SC, TSO, ...

Wrapping up

- This architecture is a (huge) simplification of ARM v8.0.
  We can also model SC, TSO, ...
- By changing $\mathscr{C}_\mu$ we get more or less compact event structures that can be useful for verification.
  (Implementation in Herd in progress)

Wrapping up

- This architecture is a (huge) simplification of ARM v8.0.
  We can also model SC, TSO, ...
- By changing $\mathscr{C}_\mu$ we get more or less compact event structures that can be useful for verification.
  (Implementation in Herd in progress)
- The treatment of reorderings should make the model useful to prove properties of architectures (eg. Data-Race-Freedom theorems.)

II. What about non-first order languages?

5mins of game semantics a day keeps the syntax away

Nontrivial scopes

Imagine now our threads look like:

$\texttt{alloc}(x);$

$\texttt{alloc}(y);$

$r \leftarrow \texttt{x};$

$\texttt{if}(r = 1)\{\texttt{y} := 1\}$

$\texttt{dealloc}(\texttt{x}); \texttt{dealloc}(\texttt{y})$

Nontrivial scopes

Imagine now our threads look like:

$$\texttt{alloc}(x);$$
$$\texttt{alloc}(y);$$
$$r \leftarrow \texttt{x};$$
$$\texttt{if}(r = 1)\{\texttt{y} := 1\}$$
$$\texttt{dealloc}(\texttt{x}); \texttt{dealloc}(\texttt{y})$$

Labels should now be:

$$\Sigma ::= \cdots \mid \mathtt{a}_{\mathtt{x}} \mid \mathtt{d}_{\mathtt{x}}$$

Nontrivial scopes

Imagine now our threads look like:

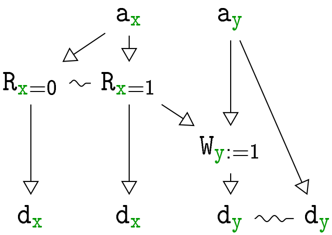
$\texttt{alloc}(x);$

$\texttt{alloc}(y);$

$r \leftarrow \texttt{x};$

$\texttt{if}(r = 1)\{\texttt{y} := 1\}$

$\texttt{dealloc}(\texttt{x}); \texttt{dealloc}(\texttt{y})$



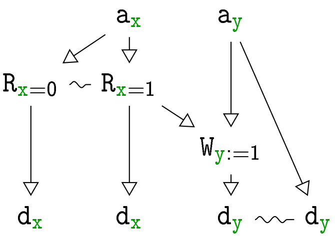
Labels should now be:

$$\Sigma ::= \cdots \mid \mathtt{a}_{\mathtt{x}} \mid \mathtt{d}_{\mathtt{x}}$$

Nontrivial scopes

Imagine now our threads look like:

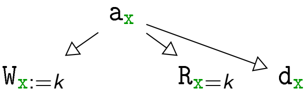
$\texttt{alloc}(x);$
$\texttt{alloc}(y);$
$r \leftarrow \texttt{x};$
$\texttt{if}(r = 1)\{\texttt{y} := 1\}$
$\texttt{dealloc}(\texttt{x}); \texttt{dealloc}(\texttt{y})$



Labels should now be:

$$\Sigma ::= \cdots \mid \mathtt{a}_{\mathtt{x}} \mid \mathtt{d}_{\mathtt{x}}$$

Implicit allocation rules give Σ some structure:



Protocols as types

Interaction thread/memory is an interaction client/server:

thread | memory

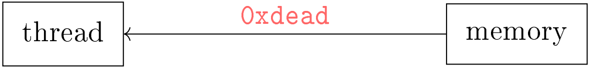
Protocols as types

Interaction thread/memory is an interaction client/server:

$$\boxed{\text{thread}} \xrightarrow{\texttt{alloc}} \boxed{\text{memory}}$$

Protocols as types

Interaction thread/memory is an interaction client/server:



Protocols as types

Interaction thread/memory is an interaction client/server:

$$\boxed{\text{thread}} \xrightarrow{\texttt{w(0xdead, 1)}} \boxed{\text{memory}}$$

Protocols as types

Interaction thread/memory is an interaction client/server:

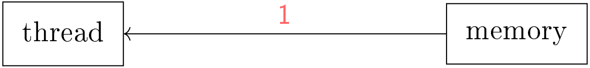
Protocols as types

Interaction thread/memory is an interaction client/server:

$$\boxed{\text{thread}} \xrightarrow{\texttt{r(0xdead)}} \boxed{\text{memory}}$$

Protocols as types

Interaction thread/memory is an interaction client/server:



Protocols as types

Interaction thread/memory is an interaction client/server:

$$\begin{tikzcd} \boxed{\text{thread}} \arrow[r, "\texttt{d(0xdead)}"] & \boxed{\text{memory}} \end{tikzcd}$$

Protocols as types

Interaction thread/memory is an interaction client/server:

thread		memory

The protocol is described by the following partial order:

$$
A = \begin{tikzcd}
 & \mathtt{alloc} \arrow[d] & & & \\
 & \mathtt{done} \arrow[dl] \arrow[dr] \arrow[drrr] & & & \\
\mathtt{w}_k \arrow[d] & & \mathtt{r} \arrow[dl] \arrow[d] & & \mathtt{dealloc} \\
\mathtt{done} & 0 & 1 & \dots &
\end{tikzcd}
$$

Protocols as types

Interaction thread/memory is an interaction client/server:

thread	memory

The protocol is described by the following partial order:

$$A = \begin{tikzcd} & & \texttt{alloc} \arrow[d] & & \\ & & \texttt{done} \arrow[dll] \arrow[d] \arrow[drr] & & \\ \texttt{w}_k \arrow[d] & & \texttt{r} \arrow[dl] \arrow[d] & & \texttt{dealloc} \\ \texttt{done} & \texttt{0} & \texttt{1} & \dots & \end{tikzcd}$$

Such a partial order with polarity annotations is a **game**.

Protocols as types

Interaction thread/memory is an interaction client/server:

$\boxed{\text{thread}}$ $\qquad\qquad\qquad$ $\boxed{\text{memory}}$

The protocol is described by the following partial order:

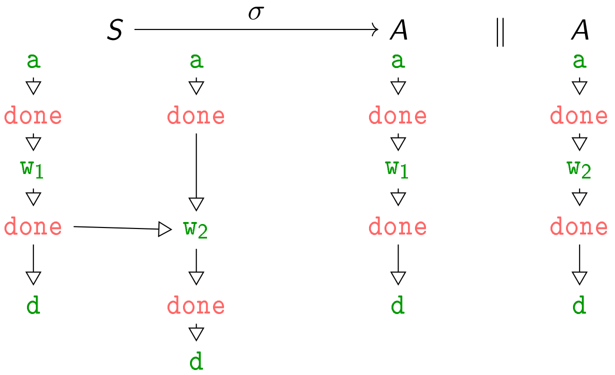
$$A = \begin{tikzcd} & \texttt{alloc} \arrow[d] & & \\ & \texttt{done} \arrow[dl] \arrow[dr] \arrow[drr] & & \\ \texttt{w}_k \arrow[d] & & \texttt{r} \arrow[dl] \arrow[d] & \texttt{dealloc} \\ \texttt{done} & \texttt{0} & \texttt{1} & \dots \end{tikzcd}$$

Such a partial order with polarity annotations is a **game**.
What is an event structure labelled by a game?

Agent (or pre-strategy)

A **agent** on a A is an e.s. S and a labelling $\sigma : S \to A$ s.t.:

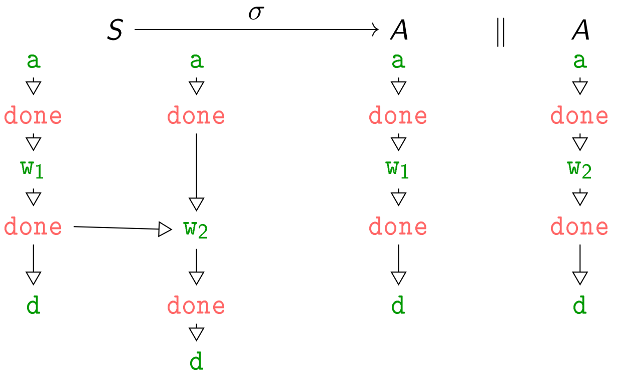
1. (Respects the rules) σ maps configurations of S to down-closed subsets of A
2. (Linearity) σ is injective on configurations.



Agent (or pre-strategy)

A **agent** on a A is an e.s. S and a labelling $\sigma : S \to A$ s.t.:

1. (Respects the rules) σ maps configurations of S to down-closed subsets of A
2. (Linearity) σ is injective on configurations.



No explicit names: each event is below a unique a, **in the game**

Agents and π-calculus

Consider $A =$
$$
\begin{tikzcd}
 & & \mathrm{q} \arrow[dl] \arrow[d] \arrow[dr] & \\
 & \mathrm{q} \arrow[dl] \arrow[dr] & \mathsf{tt} & \mathsf{ff} \\
\mathsf{tt} & & \mathsf{ff} &
\end{tikzcd}
$$
.

Agents and π-calculus

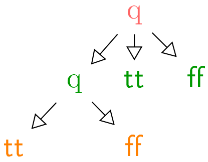
Consider $A =$
$$\begin{tikzcd}
 & & \mathrm{q} \arrow[dl] \arrow[d] \arrow[dr] & \\
 & \mathrm{q} \arrow[dl] \arrow[dr] & \mathsf{tt} & \mathsf{ff} \\
\mathsf{tt} & & \mathsf{ff} &
\end{tikzcd}.$$

Agents can be described by terms of the pi-calculus:

$$a : A \vdash a(x, rtt, rff).$$

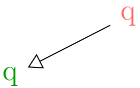
q

Agents and π-calculus

Consider $A =$  .

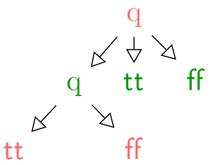
Agents can be described by terms of the pi-calculus:

$$a : A \vdash a(x, rtt, rff).\, \bar{x}(tt, ff).$$



Agents and π-calculus

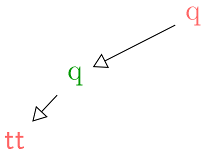
Consider $A =$



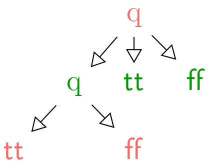
.

Agents can be described by terms of the pi-calculus:

$$a : A \vdash a(x, rtt, rff).\, \bar{x}(tt, ff).\ \ tt().$$

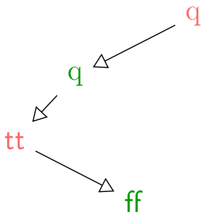


Agents and π-calculus

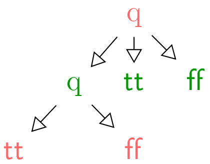
Consider $A =$  .

Agents can be described by terms of the pi-calculus:

$$a : A \vdash a(x, rtt, rff).\, \bar{x}(tt, ff).\;\; tt().\, \overline{rff}$$

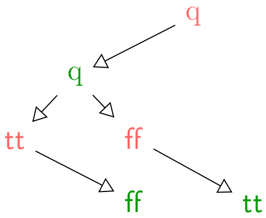


Agents and π-calculus

Consider $A =$  .

Agents can be described by terms of the pi-calculus:

$$a : A \vdash a(x, rtt, rff).\, \bar{x}(tt, ff).\, (tt().\, \overline{rff} \parallel ff().\, \overline{rtt}).$$



Copycat, or the asynchronous forwarder

Given a game A, we write $A^\perp$ for its **dual**. (polarity reversed)

For $B = \begin{tikzcd}[ampersand replacement=\&] \& q \arrow[dl] \arrow[dr] \& \\ tt \& \& ff \end{tikzcd}$, the **copycat** on B is the agent $\mathbb{c}_B$:

$$\begin{tikzcd}
 & B^\perp & \parallel & & B \\
 & & & & q \arrow[dlll] \\
 & q \arrow[dl] \arrow[dr] & & & \\
tt \arrow[drrr] & & ff \arrow[drr] & & \\
 & & & tt & tt
\end{tikzcd}$$

(corresponding to the term:

$$a : B^\perp, b : B \vdash b(rtt, rff).\, \bar{a}(tt, ff).\, (tt().\, \overline{rtt} \parallel tt().\, \overline{rtt}).$$

Restriction and composition

An agent on $A^\perp \parallel B$ can be viewed as an agent **from** A **to** B:

$$\sigma : S \to A^\perp \parallel B \qquad \Leftrightarrow \qquad \iota : A^\perp, o : B \vdash P.$$

Restriction and composition

An agent on $A^\perp \parallel B$ can be viewed as an agent **from** A **to** B:

$$\sigma : S \to A^\perp \parallel B \qquad \Leftrightarrow \qquad \iota : A^\perp, o : B \vdash P.$$

Such agents can be **composed**:

$$\sigma : S \to A^\perp \parallel B \quad \tau : T \to B^\perp \parallel C \Longrightarrow \tau \odot \sigma : T \odot S \to A^\perp \parallel C$$

Restriction and composition

An agent on $A^\perp \parallel B$ can be viewed as an agent **from** A **to** B:

$$\sigma : S \to A^\perp \parallel B \qquad \Leftrightarrow \qquad \iota : A^\perp, o : B \vdash P.$$

Such agents can be **composed**:

$$\sigma : S \to A^\perp \parallel B \quad \tau : T \to B^\perp \parallel C \Longrightarrow \tau \odot \sigma : T \odot S \to A^\perp \parallel C$$
$$a : A^\perp, b : B \vdash P \quad b : B^\perp, c : C \vdash Q \Longrightarrow a : A^\perp, c : C \vdash (\nu b)(P \parallel Q)$$

Restriction and composition

An agent on $A^\perp \parallel B$ can be viewed as an agent **from** A **to** B:

$$\sigma : S \to A^\perp \parallel B \qquad \Leftrightarrow \qquad \iota : A^\perp, o : B \vdash P.$$

Such agents can be **composed**:

$$\sigma : S \to A^\perp \parallel B \quad \tau : T \to B^\perp \parallel C \Longrightarrow \tau \odot \sigma : T \odot S \to A^\perp \parallel C$$

$$a : A^\perp, b : B \vdash P \quad b : B^\perp, c : C \vdash Q \Longrightarrow a : A^\perp, c : C \vdash (\nu b)(P \parallel Q)$$

In two steps:

1. **Interaction** of the common parts of σ and τ
2. **Hiding** of the events on B, invisible after composition.

Composition: a bird's eye view

1. **Interaction** relies on the **product** of agents, generalizing the product of labelled e.s.
   $\rightarrow$ Interaction σ and τ gives

$$\tau \circledast \sigma : T \circledast S \rightarrow A \parallel B \parallel C.$$

2. **Hiding** relies on **projejection** of event structures: events in B become invisible.

$$\tau \odot \sigma : T \circledast S \downarrow V \rightarrow A \parallel B \parallel C \qquad V = \tau \circledast \sigma^{-1}(A \parallel C).$$

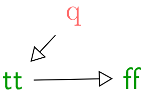
An example

Consider:

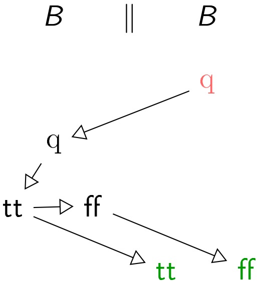
$$\sigma = \begin{tikzcd} & \mathrm{q} \arrow[dl] & \\ \mathsf{tt} \arrow[rr] & & \mathsf{ff} \end{tikzcd} : \emptyset^\perp \parallel B \qquad \tau = \mathbb{c}_B : B^\perp \parallel B.$$

An example

Consider:

$\sigma =$  $: \emptyset^\perp \parallel B \qquad \tau = \mathrm{c\!\!c}_B : B^\perp \parallel B.$

The interaction gives:



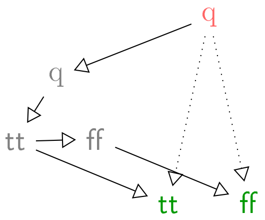
An example

Consider:

$$\sigma = \begin{tikzcd} & \mathrm{q} \arrow[dl] & \\ \mathsf{tt} \arrow[rr] & & \mathsf{ff} \end{tikzcd} : \emptyset^{\perp} \parallel B \qquad \tau = \text{\textcent}_B : B^{\perp} \parallel B.$$

The interaction gives:

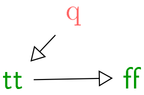
$B \quad \parallel \quad B$



An example

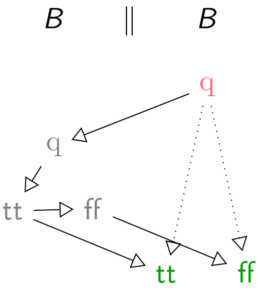
Consider:

$$\sigma = $$



$$: \emptyset^{\perp} \parallel B \qquad \tau = \mathbb{c}_B : B^{\perp} \parallel B.$$

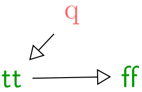
The interaction gives:



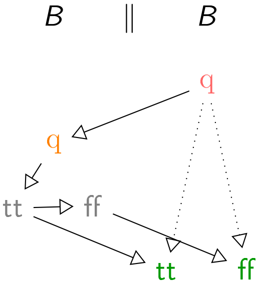
$$(\nu a)\left(\begin{array}{l} a(tt, ff).\,\overline{tt}.\,\overline{ff} \\ \parallel \color{red}{b}\color{black}(rtt, rff).\,\bar{a}(tt, ff).\left(\begin{array}{l} tt().\,\overline{rtt} \\ \parallel ff().\,\overline{rff} \end{array}\right) \end{array}\right)$$

An example

Consider:

$\sigma =$  $: \emptyset^{\perp} \parallel B \qquad \tau = ȼ_B : B^{\perp} \parallel B.$

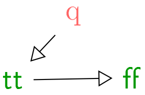
The interaction gives:



$$(\nu a)\left(\begin{array}{l} a(tt, ff).\, \overline{tt}.\, \overline{ff} \\ \parallel b(rtt, rff).\, \bar{a}(tt, ff).\, \left(\begin{array}{l} tt().\, \overline{rtt} \\ \parallel ff().\, \overline{rff} \end{array} \right) \end{array} \right)$$

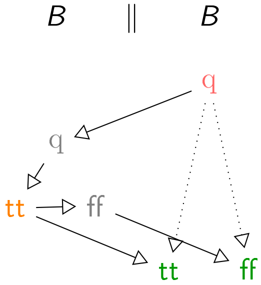
An example

Consider:

$\sigma =$ 

$: \emptyset^\perp \parallel B \qquad \tau = \mathrm{c\!\!c}_B : B^\perp \parallel B.$

The interaction gives:



$$(\nu a)\left(\begin{array}{l}\overline{tt}.\overline{ff} \\ \parallel b(rtt, rff).\left(\begin{array}{r} tt().\overline{rtt} \\ \parallel ff().\overline{rff}\end{array}\right)\end{array}\right)$$

An example

Consider:

$$\sigma = \begin{tikzcd} & \mathrm{q} \arrow[dl] & \\ \mathsf{tt} \arrow[rr] & & \mathsf{ff} \end{tikzcd} \quad : \emptyset^\perp \parallel B \qquad \tau = \mathbb{c}_B : B^\perp \parallel B.$$

The interaction gives:

$$B \qquad \parallel \qquad B$$

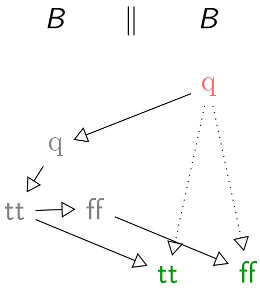
$$\begin{tikzcd} & & & \mathrm{q} \arrow[dll] \arrow[dddl, dotted] \arrow[dddr, dotted] & \\ & \mathrm{q} \arrow[dl] & & & \\ \mathsf{tt} \arrow[r] \arrow[drr] & \mathsf{ff} \arrow[drrr] & & & \\ & & \mathsf{tt} & & \mathsf{ff} \end{tikzcd} \qquad (\nu a)\left(\begin{array}{l} \overline{ff} \\ \parallel b(rtt, rff).\left(\begin{array}{l} \overline{rtt} \\ \parallel ff().\,\overline{rff} \end{array} \right) \end{array} \right)$$

An example

Consider:

$$\sigma = \begin{tikzcd}[column sep=small] & \color{red}{\mathrm{q}} \arrow[dl] & \\ \color{green}{\mathrm{tt}} \arrow[rr] & & \color{green}{\mathrm{ff}} \end{tikzcd} : \emptyset^{\perp} \parallel B \qquad \tau = \mathrm{c\!\!c}_B : B^{\perp} \parallel B.$$

The interaction gives:



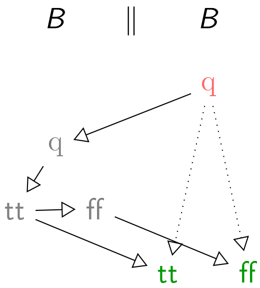
$$(\nu a) \left(\begin{array}{l} \\ \parallel \color{red}{b}(rtt, rff).\ \left(\begin{array}{l} \overline{rtt} \\ \parallel \overline{rff} \end{array} \right) \end{array} \right)$$

An example

Consider:

$$\sigma = \begin{tikzcd} & \mathrm{q} \arrow[dl] & \\ \mathsf{tt} \arrow[rr] & & \mathsf{ff} \end{tikzcd} : \emptyset^{\perp} \parallel B \qquad \tau = \mathbb{c}_B : B^{\perp} \parallel B.$$

The interaction gives:



$$(\nu a)\left(\parallel b(rtt, rff).\ \left(\begin{array}{c} \overline{rtt} \\ \parallel \overline{rff} \end{array} \right) \right)$$

σ *not* invariant under the asynchronous forwarder.

Asynchrous agents, or *strategies*

Which agents σ satsify $\mathbb{c}_A \odot \sigma \cong \sigma$?

Definition

A **strategy** is an agent $\sigma : S \to A$ such that

1. S only adds immediate causal links $\ominus \rightarrowtriangle \oplus$
2. S is cannot ignore (or duplicate) negative events.

Theorem (Rideau, Winskel)

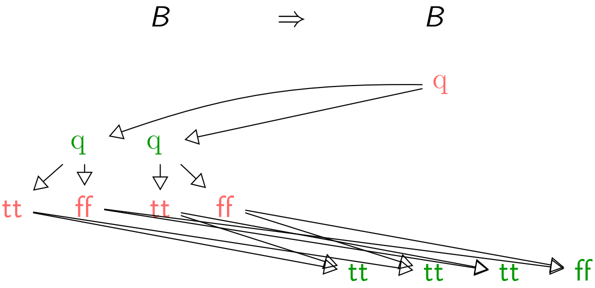
An agent σ is a strategy if and only if $\mathbb{c}_A \odot \sigma \cong \sigma$.

$\to$ Games and strategies model *linear* languages (compact-closed category).

III. Interpreting functional programming languages

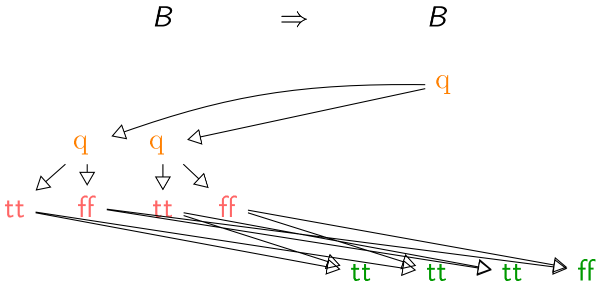
Local injectivity and copy indices

To represent nonlinear agents:



Local injectivity and copy indices

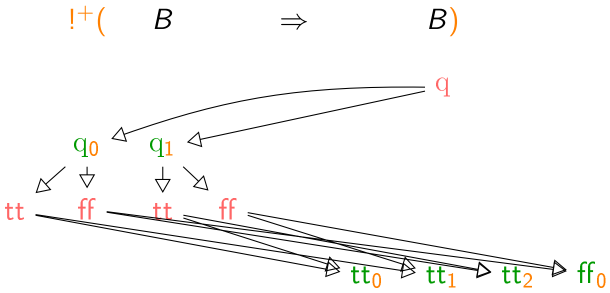
To represent nonlinear agents:



The labelling to $B \Rightarrow B$ fails local injectivity.

Local injectivity and copy indices

To represent nonlinear agents:

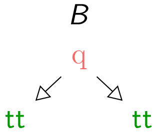


The labelling to $B \Rightarrow B$ fails local injectivity.

$\rightarrow$ We make the game bigger.

Uniformity

To compose, Opponent must be allowed to be nonlinear as well:



Uniformity

To compose, Opponent must be allowed to be nonlinear as well:

$$!^{+}B$$

$$\begin{tikzcd} & \mathrm{q} \arrow[dl] \arrow[dr] & \\ \mathtt{tt}_0 & & \mathtt{tt}_1 \end{tikzcd}$$

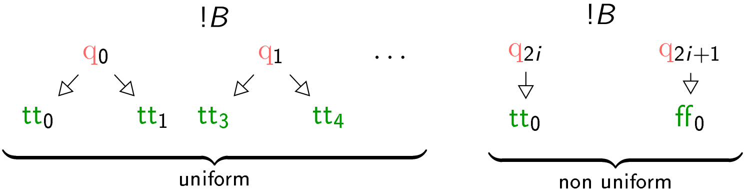
Uniformity

To compose, Opponent must be allowed to be nonlinear as well:

$$\begin{tikzcd} & & & !B \\ & q_0 \arrow[dl] \arrow[dr] & & & q_1 \arrow[dl] \arrow[dr] & & \cdots \\ \mathtt{tt}_0 & & \mathtt{tt}_1 & \mathtt{tt}_0 & & \mathtt{tt}_1 \end{tikzcd}$$

Uniformity

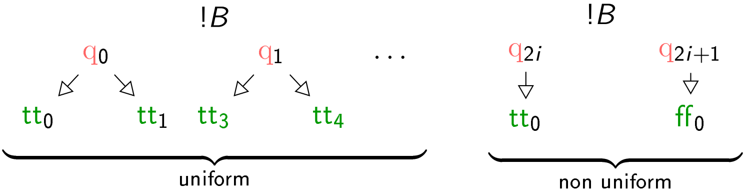
To compose, Opponent must be allowed to be nonlinear as well:



But strategies should be **uniform**.
(Uniformity is defined by using *event structures with symmetry*.)

Uniformity

To compose, Opponent must be allowed to be nonlinear as well:



But strategies should be **uniform**.
(Uniformity is defined by using *event structures with symmetry*.)

Theorem (C., Clairambault, Winskel)

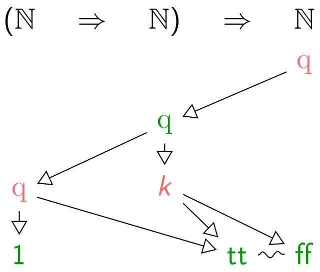
The following structure CHO is a model of higher-order computation:

- *Types are interpreted by games,*
- *Terms $\Gamma \vdash M : A$ are interpreted by uniform strategies $!([\![\Gamma]\!]^\perp \parallel [\![A]\!])$,*
- *Composition is: interaction + hiding.*

An example of higher-order

Consider the *call-by-name* program

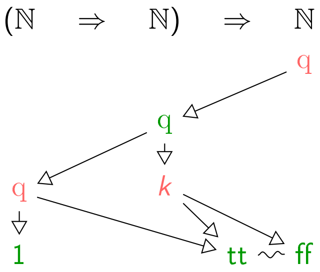
$$\texttt{strict} = \lambda f.\,\mathsf{new}\ r\ \mathsf{in}\ f(r := 1; 1); !r = 1 : (\mathbb{N} \Rightarrow \mathbb{N}) \Rightarrow \mathbb{B}.$$



An example of higher-order

Consider the *call-by-name* program

$$\texttt{strict} = \lambda f.\, \mathsf{new}\ r\ \mathsf{in}\ f(r := 1; 1); !r = 1 : (\mathbb{N} \Rightarrow \mathbb{N}) \Rightarrow \mathbb{B}.$$



If f is concurrent with control operators, $\texttt{strict}$ exhibits a race.

Theorem

CHO can interpret concurrent languages, adequately for may:

$$M \Downarrow \Leftrightarrow [\![M]\!] \text{ contains a positive move.}$$

Hidden divergences

However, in nondetermistic languages convergence is more subtle:

$$M = \lambda b. (\text{if } b \text{ then } \texttt{loop} \text{ else tt}).$$

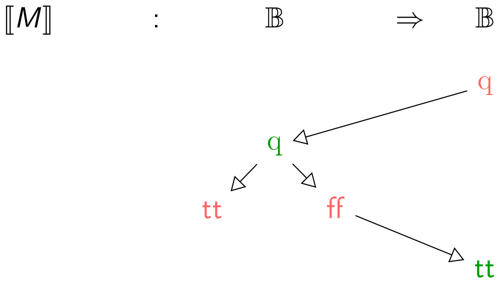
Does M `choice` converge?

Hidden divergences

However, in nondetermistic languages convergence is more subtle:

$$M = \lambda b.\,(\text{if } b \text{ then } \texttt{loop} \text{ else tt}).$$

Does M `choice` converge?



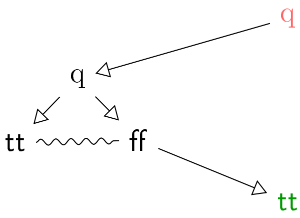
Hidden divergences

However, in nondetermistic languages convergence is more subtle:

$$M = \lambda b.\,(\text{if } b \text{ then } \texttt{loop} \text{ else tt}).$$

Does $M\,\texttt{choice}$ converge?

$[\![M]\!] \circledast \texttt{choice} :$ $\mathbb{B}$



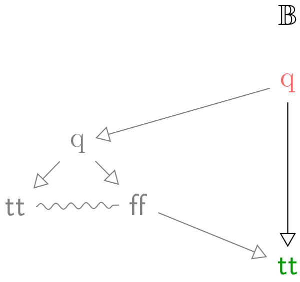
Hidden divergences

However, in nondeterministic languages convergence is more subtle:

$$M = \lambda b.\,(\text{if } b \text{ then } \texttt{loop} \text{ else tt}).$$

Does M `choice` converge?

$[\![M]\!] \odot \texttt{choice}$:



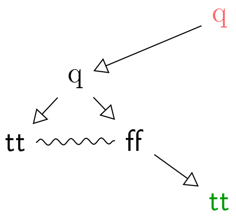
As a result: $[\![M\ choice]\!] = [\![\mathrm{tt}]\!]$.

Model inadequate for *non-angelic* convergences!

Essential events

Idea: never hide *essential events* appearing in a conflict:

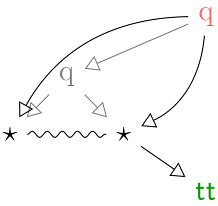
$$[\![M]\!] \circledast \texttt{choice} : \qquad \mathbb{B}$$



Essential events

Idea: never hide *essential events* appearing in a conflict:

$[\![M]\!] \circledcirc \texttt{choice} :$ $\mathbb{B}$



Strategies become **partial maps** $S \rightharpoonup A$ (with internal events).

Theorem (C., Clairambault, Hayman, Winskel)

The partial strategies $\tau \circledast \sigma$ and $\tau \circledcirc \sigma$ are weakly bisimilar.

→ Partial hiding does not lose behaviour up to weak bisimilarity.

The category $\mathrm{CHO}_{\circledcirc}$

Despite not hiding *everything*, we still get a category:

Theorem (C.)

The following model $CHO_{\circledcirc}$ is a model of higher-order computation:

- *Types are interpreted as in CHO,*
- *Terms are interpreted by strategies with internal events,*
- *Composition is: interaction + partial hiding.*

Moreover $CHO_{\circledcirc}$ interprets nondeterministic languages, adequately for non-angelic convergences (must, fair), ...

In $\mathrm{CHO}_{\circledcirc}$, one can define *intensional, causal, compositional* semantics for a wide variety of languages.

Related work

Earlier work / inspirations:

- Melliès's **asynchronous games**.
  Traces augmented with 2-dimensional tiles representing independence.
- Curien, Faggian, Piccolo, **I-nets**.
  Partial order representation for ludics.

Parallel works:

- Hirschowitz *et. al.*: **preseheaves over graphs**. (no hiding)
  Gives intensional models of π-calculus fully abstract for fair convergence.
- Ong, Tsukada: **presheaves over plays**.
  Models of nondeterministic, concurrent languages.

On causal models for **weak memory models**:

- Jeffrey & Riley, Brookes *et. al.*, Pichon *et. al.*

Extensions / Other work

Axis of development:

1. **Understanding the structure of strategies.**
   Which strategies are expressible using which effects?
   $\rightarrow$ Fully abstract models of extensions of PCF. (With Clairambault, and Winskel)

2. **Adding quantitive information.**
   - probabilities (full abstraction for probabilistic PCF) (With Clairambault, Paquet and Winskel)
   - quantum (WIP by Clairambault, De Visme, Winskel)

3. **Modelling complex languages.** Work in progress:
   - Complex memory models (with Alglave and Madiot),
   - Session π-calculus (with Clairambault and Yoshida).