Towards a causal and compositional operational semantics of programming languages

Simon Castellan\textsuperscript{1}

\textsuperscript{1}Imperial College London, UK

November 21th, 2016
LSV Seminar
Consider this program $mp$:

\[
\begin{align*}
\text{data} &= \text{flag} = 0 \\
\text{data} &:= 17; \quad r \leftarrow \text{flag}; \\
\text{flag} &:= 1 \quad \text{if}(r == 1) \{ v \leftarrow \text{data} \}
\end{align*}
\]
Message-passing on my computer

Consider this program mp:

\[
\begin{align*}
\text{data} &= \text{flag} = 0 \\
\text{data} &:= 17; \quad r \leftarrow \text{flag} \\
\text{flag} &:= 1 \quad \text{if}(r == 1) \{ v \leftarrow \text{data} \}
\end{align*}
\]

Possible execution traces on my computer:
Message-passing on my computer

Consider this program \( mp \):

\[
\begin{align*}
\text{data} &= \text{flag} = 0 \\
\text{data} &:= 17; \quad r \leftarrow \text{flag}; \\
\text{flag} &:= 1 \quad \text{if}(r == 1)\{v \leftarrow \text{data}\}
\end{align*}
\]

Possible execution traces on my computer:

- \( W_{\text{data}:=17} \)
Message-passing on my computer

Consider this program mp:

\[
\begin{align*}
\text{data} &= \text{flag} = 0 \\
\text{data} &:= 17; \quad r \leftarrow \text{flag}; \\
\text{flag} &:= 1 \quad \text{if}(r == 1)\{v \leftarrow \text{data}\}
\end{align*}
\]

Possible execution traces on my computer:

▸ \text{W}_{\text{data}}:=17 \cdot \text{W}_{\text{flag}}:=1
Message-passing on my computer

Consider this program \texttt{mp}:

\begin{verbatim}
data = flag = 0
data := 17;
r ← flag;
flag := 1
if(r == 1){v ← data}
\end{verbatim}

Possible execution traces on my computer:

\begin{itemize}
\item \texttt{Wdata:=17 · Wflag:=1 · Rflag=1}
\end{itemize}
Message-passing on my computer

Consider this program $mp$:

\[
data = \text{flag} = 0
\]
\[
data := 17; \quad r \leftarrow \text{flag}; \\
\text{flag} := 1 \quad \text{if}(r == 1)\{v \leftarrow \text{data}\}
\]

Possible execution traces on my computer:

- $W_{data:=17} \cdot W_{flag:=1} \cdot R_{flag=1} \cdot R_{data=17}$
Message-passing on my computer

Consider this program mp:

\[
\begin{align*}
\text{data} &= \text{flag} = 0 \\
\text{data} &:= 17; \quad r \leftarrow \text{flag}; \\
\text{flag} &:= 1 \quad \text{if}(r == 1)\{v \leftarrow \text{data}\}
\end{align*}
\]

Possible execution traces on my computer:

- \( W_{\text{data}} := 17 \cdot W_{\text{flag}} := 1 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 17 \)
- \( W_{\text{data}} := 17 \cdot R_{\text{flag}} = 0 \cdot W_{\text{flag}} := 1 \)
- \( R_{\text{flag}} = 0 \cdot W_{\text{data}} := 17 \cdot W_{\text{flag}} := 1 \)
**Message-passing on my phone**

```
data = flag = 0
data := 17;  r ← flag;
flag := 1  if(r == 1){v ← data}
```

Possible execution traces on my **phone**:

- \[ \text{W}_{\text{data}}:=17 \cdot \text{W}_{\text{flag}}:=1 \cdot \text{R}_{\text{flag}}=1 \cdot \text{R}_{\text{data}}=17 \]
- \[ \text{W}_{\text{data}}:=17 \cdot \text{R}_{\text{flag}}=0 \cdot \text{W}_{\text{flag}}:=1 \]
- \[ \text{R}_{\text{flag}}=0 \cdot \text{W}_{\text{data}}:=17 \cdot \text{W}_{\text{flag}}:=1 \]
Message-passing on my phone

\[
\text{data} = \text{flag} = 0
\]
\[
\text{data} := 17; \quad r \leftarrow \text{flag};
\]
\[
\text{flag} := 1 \quad \text{if}(r == 1)\{v \leftarrow \text{data}\}
\]

Possible execution traces on my phone:

- \(W_{\text{data}}:=17 \cdot W_{\text{flag}}:=1 \cdot R_{\text{flag}}=1 \cdot R_{\text{data}}=17\)
- \(W_{\text{data}}:=17 \cdot R_{\text{flag}}=0 \cdot W_{\text{flag}}:=1\)
- \(R_{\text{flag}}=0 \cdot W_{\text{data}}:=17 \cdot W_{\text{flag}}:=1\)
- \(W_{\text{flag}}:=1\)
Message-passing on my phone

\[
\begin{align*}
\text{data} &= \text{flag} = 0 \\
data &:= 17; \quad r \leftarrow \text{flag} \\
\text{flag} &:= 1 \quad \text{if}(r == 1)\{v \leftarrow \text{data}\}
\end{align*}
\]

Possible execution traces on my phone:

- \(W_{\text{data}}:=17 \cdot W_{\text{flag}}:=1 \cdot R_{\text{flag}}=1 \cdot R_{\text{data}}=17\)
- \(W_{\text{data}}:=17 \cdot R_{\text{flag}}=0 \cdot W_{\text{flag}}:=1\)
- \(R_{\text{flag}}=0 \cdot W_{\text{data}}:=17 \cdot W_{\text{flag}}:=1\)
- \(W_{\text{flag}}:=1 \cdot R_{\text{flag}}=1\)
Message-passing on my phone

\[
\begin{align*}
\text{data} &= \text{flag} = 0 \\
\text{data} &:= 17; \quad | \quad r \leftarrow \text{flag}; \\
\text{flag} &:= 1 \quad | \quad \text{if}(r == 1)\{v \leftarrow \text{data}\}
\end{align*}
\]

Possible execution traces on my phone:

- \(W_{\text{data}}:=17 \cdot W_{\text{flag}}:=1 \cdot R_{\text{flag}}=1 \cdot R_{\text{data}}=17\)
- \(W_{\text{data}}:=17 \cdot R_{\text{flag}}=0 \cdot W_{\text{flag}}:=1\)
- \(R_{\text{flag}}=0 \cdot W_{\text{data}}:=17 \cdot W_{\text{flag}}:=1\)
- \(W_{\text{flag}}:=1 \cdot R_{\text{flag}}=1 \cdot R_{\text{data}}=0\)

A different architecture, much harder to reason about...
Message-passing on my phone

\[
data = \text{flag} = 0
\]
\[
data := 17; \quad r \leftarrow \text{flag};
\]
\[
\text{flag} := 1 \quad \mathbf{\text{if}}(r == 1)\{ v \leftarrow \text{data}\}
\]

Possible execution traces on my phone:

▶ \text{W data:}=17 \cdot \text{W flag:}=1 \cdot \text{R flag}=1 \cdot \text{R data}=17
▶ \text{W data:}=17 \cdot \text{R flag}=0 \cdot \text{W flag:}=1
▶ \text{R flag}=0 \cdot \text{W data:}=17 \cdot \text{W flag:}=1
▶ \text{W flag:}=1 \cdot \text{R flag}=1 \cdot \text{R data}=0 \cdot \text{W data:}=17
Message-passing on my phone

\[
\begin{align*}
data & = \text{flag} = 0 \\
data & := 17; \quad \| \quad r \leftarrow \text{flag}; \\
\text{flag} & := 1 \quad \| \quad \text{if}(r == 1)\{v \leftarrow \text{data}\}
\end{align*}
\]

Possible execution traces on my phone:

- \(W_{\text{data}} := 17 \cdot W_{\text{flag}} := 1 \cdot R_{\text{flag}} := 1 \cdot R_{\text{data}} := 17\)
- \(W_{\text{data}} := 17 \cdot R_{\text{flag}} := 0 \cdot W_{\text{flag}} := 1\)
- \(R_{\text{flag}} := 0 \cdot W_{\text{data}} := 17 \cdot W_{\text{flag}} := 1\)
- \(W_{\text{flag}} := 1 \cdot R_{\text{flag}} := 1 \cdot R_{\text{data}} := 0 \cdot W_{\text{data}} := 17\)
- \(W_{\text{flag}} := 1 \cdot R_{\text{flag}} := 1 \cdot W_{\text{data}} := 17 \cdot R_{\text{data}} := 17\)
- \(W_{\text{flag}} := 1 \cdot W_{\text{data}} := 17 \cdot R_{\text{flag}} := 1 \cdot R_{\text{data}} := 17\)
- \(R_{\text{flag}} := 0 \cdot W_{\text{flag}} := 1 \cdot W_{\text{data}} := 17\)

A different architecture, much harder to reason about...
Structure behind traces

\[
\begin{align*}
W_{\text{flag}} &: = 1 \cdot W_{\text{data}} = 17 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 17 \\
W_{\text{flag}} &: = 1 \cdot R_{\text{flag}} = 1 \cdot W_{\text{data}} = 17 \cdot R_{\text{data}} = 17 \\
W_{\text{data}} &: = 17 \cdot W_{\text{flag}} = 1 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 17
\end{align*}
\]

\[
\begin{align*}
W_{\text{flag}} &: = 1 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 0 \cdot W_{\text{data}} = 17
\end{align*}
\]

\[
\begin{align*}
R_{\text{flag}} &: = 0 \cdot W_{\text{data}} = 17 \cdot W_{\text{flag}} = 1 \\
W_{\text{data}} &: = 17 \cdot R_{\text{flag}} = 0 \cdot W_{\text{flag}} = 1 \\
R_{\text{flag}} &: = 0 \cdot W_{\text{flag}} = 1 \cdot W_{\text{data}} = 17
\end{align*}
\]
Structure behind traces

\[
\begin{align*}
W_{\text{flag}} &= 1 \cdot W_{\text{data}} = 17 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 17 \\
W_{\text{flag}} &= 1 \cdot R_{\text{flag}} = 1 \cdot W_{\text{data}} = 17 \cdot R_{\text{data}} = 17 \\
W_{\text{data}} &= 17 \cdot W_{\text{flag}} = 1 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 17 \\
W_{\text{flag}} &= 1 \quad W_{\text{data}} = 17 \\
R_{\text{flag}} &= 1 \quad R_{\text{data}} = 17 \\
W_{\text{flag}} &= 1 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 0 \cdot W_{\text{data}} = 17 \\
R_{\text{flag}} &= 0 \cdot W_{\text{data}} = 17 \cdot W_{\text{flag}} = 1 \\
W_{\text{data}} &= 17 \cdot R_{\text{flag}} = 0 \cdot W_{\text{flag}} = 1 \\
R_{\text{flag}} &= 0 \cdot W_{\text{flag}} = 1 \cdot W_{\text{data}} = 17
\end{align*}
\]
Towards a causal and compositional operational semantics of programming languages · Simon Castellan
Structure behind traces

\[
\begin{align*}
W_{\text{flag}} &= 1 \cdot W_{\text{data}} = 17 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 17 \\
W_{\text{flag}} &= 1 \cdot R_{\text{flag}} = 1 \cdot W_{\text{data}} = 17 \cdot R_{\text{data}} = 17 \\
W_{\text{data}} &= 17 \cdot W_{\text{flag}} = 1 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 17
\end{align*}
\]

\[
\begin{align*}
W_{\text{flag}} &= 1 \cdot W_{\text{data}} = 17 \\
\downarrow \\
R_{\text{flag}} &= 1 \rightarrow R_{\text{data}} = 17
\end{align*}
\]

\[
\begin{align*}
W_{\text{flag}} &= 1 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 0 \cdot W_{\text{data}} = 17 \\
\downarrow \quad \downarrow \quad \downarrow \\
R_{\text{flag}} &= 1 \rightarrow W_{\text{data}} = 17
\end{align*}
\]

\[
\begin{align*}
R_{\text{flag}} &= 0 \cdot W_{\text{data}} = 17 \cdot W_{\text{flag}} = 1 \\
W_{\text{data}} &= 17 \cdot R_{\text{flag}} = 0 \cdot W_{\text{flag}} = 1 \\
R_{\text{flag}} &= 0 \cdot W_{\text{flag}} = 1 \cdot W_{\text{data}} = 17
\end{align*}
\]

\[
\begin{align*}
R_{\text{flag}} &= 0 \cdot W_{\text{data}} = 17 \\
\downarrow \\
W_{\text{flag}} &= 1
\end{align*}
\]

Towards a causal and compositional operational semantics of programming languages · Simon Castellan
Structure behind traces

\[
\begin{align*}
W_{\text{flag}} &= 1 \cdot W_{\text{data}} = 17 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 17 \\
W_{\text{flag}} &= 1 \cdot R_{\text{flag}} = 1 \cdot W_{\text{data}} = 17 \cdot R_{\text{data}} = 17 \\
W_{\text{data}} &= 17 \cdot W_{\text{flag}} = 1 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 17
\end{align*}
\]

\[
\begin{align*}
W_{\text{flag}} &= 1 \cdot W_{\text{data}} = 17 \\
R_{\text{flag}} &= 1 \rightarrow R_{\text{data}} = 17
\end{align*}
\]

\[
\begin{align*}
W_{\text{flag}} &= 1 \cdot R_{\text{flag}} = 1 \cdot R_{\text{data}} = 0 \cdot W_{\text{data}} = 17 \\
R_{\text{flag}} &= 1 \rightarrow R_{\text{data}} = 0 \\
W_{\text{flag}} &= 1 \cdot W_{\text{data}} = 17
\end{align*}
\]

\[
\begin{align*}
R_{\text{flag}} &= 0 \cdot W_{\text{data}} = 17 \cdot W_{\text{flag}} = 1 \\
W_{\text{data}} &= 17 \cdot R_{\text{flag}} = 0 \cdot W_{\text{flag}} = 1 \\
R_{\text{flag}} &= 0 \cdot W_{\text{flag}} = 1 \cdot W_{\text{data}} = 17
\end{align*}
\]

\[
\begin{align*}
R_{\text{flag}} &= 0 \cdot W_{\text{data}} = 17 \\
W_{\text{flag}} &= 1
\end{align*}
\]
Sets of partial orders and event structures

The set of partial orders describes the semantics of \( mp \):

\[
\begin{align*}
W_{\text{flag}} := 1 & \quad W_{\text{data}} := 17 \\
R_{\text{flag}} := 1 & \quad R_{\text{data}} := 17 \\
R_{\text{flag}} := 0 & \quad W_{\text{data}} := 17
\end{align*}
\]

Towards a causal and compositional operational semantics of programming languages · Simon Castellan
Sets of partial orders and event structures

The set of partial orders describes the semantics of mp:

\[
\begin{align*}
W_{\text{flag}} &= 1 \\
W_{\text{data}} &= 17 \\
R_{\text{flag}} &= 1 \\
R_{\text{data}} &= 17 \\
R_{\text{flag}} &= 0 \\
W_{\text{data}} &= 17
\end{align*}
\]

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

\[
\begin{align*}
W_{\text{data}} &= 17 \\
W_{\text{flag}} &= 1 \\
R_{\text{flag}} &= 0 \\
R_{\text{data}} &= 17 \\
R_{\text{flag}} &= 1 \\
W_{\text{data}} &= 17
\end{align*}
\]
The set of partial orders describes the semantics of \textit{mp}:

\[
\begin{align*}
W_{\text{flag}} & = 1 \\
W_{\text{data}} & = 17 \\
R_{\text{flag}} & = 1 \\
R_{\text{data}} & = 17 \\
W_{\text{data}} & = 17
\end{align*}
\]

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. Interpreting 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.

\[
e ::= r \mid e + e \mid \ldots
\]
\[
t ::= \text{fence}; t \mid x := e; t \mid r \leftarrow x; t
\]
\[
p ::= t \parallel \ldots \parallel t
\]

Actions. We observe the following actions from the programs:

\[
\Sigma ::= \text{w}_{x:=k} \mid \text{r}_{x:=k} \mid \text{fence}.
\]

Semantics. Described by a labeled transition system on states \(p, \mu\):

\[
\langle p \circ \mu \rangle \xrightarrow{\ell \in \Sigma} \langle p' \circ \mu' \rangle. \quad (\mu, \mu' : \text{Var} \rightarrow \mathbb{N})
\]

It is relaxed: operations on independent variables can be reordered.
A few rules

Thread rules: \( \langle t \otimes \mu \rangle \xrightarrow{\ell} \langle t' \otimes \mu' \rangle : \)

\[\langle x := k; t \otimes \mu \rangle \xrightarrow{w_{x := k}} \langle t \otimes \mu[x := k] \rangle\]
A few rules

Thread rules: \( \langle t \odot \mu \rangle \xrightarrow{\ell} \langle t' \odot \mu' \rangle : \)

\[
\frac{\langle x := k; t \odot \mu \rangle}{W_{x:=k} \quad \frac{\langle t \odot \mu[x := k] \rangle}{\langle \text{fence}; t \odot \mu \rangle \xrightarrow{\text{fence}} \langle t \odot \mu \rangle}}
\]

This generates the operational (partial) traces:

\[
T_r(p, \mu) = \{ \ell_1 \ldots \ell_n | \langle p \odot \mu \rangle \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_n} \langle p' \odot \mu' \rangle \}.
\]
A few rules

Thread rules: \( \langle t \odot \mu \rangle \xrightarrow{\ell} \langle t' \odot \mu' \rangle : \)

\[
\frac{\langle x := k; t \odot \mu \rangle \xrightarrow{\text{w}} \langle t \odot \mu[x := k] \rangle}{\langle x := k; t \odot \mu \rangle \xrightarrow{\ell} \langle t \odot \mu \rangle}
\]

\[
\frac{\langle \text{fence}; t \odot \mu \rangle \xrightarrow{\text{fence}} \langle t \odot \mu \rangle}{\langle \text{fence}; t \odot \mu \rangle \xrightarrow{\ell} \langle \text{fence}; t \odot \mu \rangle}
\]

\[
\frac{\langle t \odot \mu \rangle \xrightarrow{\ell} \langle t' \odot \mu' \rangle \quad \ell \neq \text{fence} \quad \text{var}(\ell) \neq x}{\langle x := k; t \odot \mu \rangle \xrightarrow{\ell} \langle x := k; t' \odot \mu' \rangle}
\]
A few rules

Thread rules: \(\langle t \circ \mu \rangle \xrightarrow{\ell} \langle t' \circ \mu' \rangle\):

\[
\langle x := k; t \circ \mu \rangle \xrightarrow{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{\text{
A few rules

Thread rules: \[ \langle t \otimes \mu \rangle \xrightarrow{\ell} \langle t' \otimes \mu' \rangle : \]

\[
\langle x := k; t \otimes \mu \rangle \xrightarrow{w_{x := k}} \langle t \otimes \mu[x := k] \rangle \quad \langle \text{fence}; t \otimes \mu \rangle \xrightarrow{\text{fence}} \langle t \otimes \mu \rangle
\]

\[
\langle t \otimes \mu \rangle \xrightarrow{\ell} \langle t' \otimes \mu' \rangle \quad \ell \neq \text{fence} \quad \text{var}(\ell) \neq x
\]

\[
\langle x := k; t \otimes \mu \rangle \xrightarrow{\ell} \langle x := k; t' \otimes \mu' \rangle
\]

And then:

\[
\langle t_i \otimes \mu \rangle \xrightarrow{\ell} \langle t'_i \otimes \mu' \rangle
\]

\[
\langle t_1 \parallel \ldots \parallel t_i \parallel \ldots \parallel t_n \otimes \mu \rangle \xrightarrow{\ell} \langle t_1 \parallel \ldots \parallel t'_i \parallel \ldots \parallel t_n \otimes \mu' \rangle
\]

This generates the operational (partial) traces:

\[
\text{Tr}(p, \mu) = \{ \ell_1 \ldots \ell_n \mid \langle p \otimes \mu \rangle \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_n} \langle p' \otimes \mu' \rangle \}.
\]
Labeled event structures

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

\[
\begin{array}{ccc}
a & b \\
\downarrow & \nearrow & \downarrow \\
c & d \\
\downarrow & \nearrow \\
e \\
\end{array}
\]
Labeled event structures

Definition

A (\(\Sigma\)-labeled) event structure is a tuple
\((E, \leq_E, \#_E, \ell : E \rightarrow \Sigma)\) where \((E, \leq_E)\) is a partial order and \(\#_E\) is a symmetric relation on \(E\), satisfying finite causes and conflict inheritance.

- Configurations are downclosed, conflict-free subsets of \(E\).
  \(\mathcal{C}(E)\) is the set of configurations of \(E\).
Labeled event structures

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

- Configurations are downclosed, conflict-free subsets of \(E\). 
  \(C(E)\) is the set of configurations of \(E\).

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

Our goal: a mapping \([\cdot]\) from states to event structures s.t.:
\[
\text{Tr}(p, \mu) = \text{Tr}[p, \mu].
\]
An overview of the semantics

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

   \[
   \begin{align*}
   W_{\text{flag}} &= 1 & W_{\text{data}} &= 17 & R_{\text{flag}} &= 0 & \sim & R_{\text{flag}} &= 1 & \sim & R_{\text{flag}} &= 2 & \ldots
   \\
   R_{\text{data}} &= 0 & \sim & R_{\text{data}} &= 1 & \ldots
   \end{align*}
   \]

2. **Final semantics**: context is assumed empty

   Compute interactions with memory:

   \[
   \begin{align*}
   W_{\text{data}} &= 17 & W_{\text{flag}} &= 1 & \sim & R_{\text{flag}} &= 0 \\
   R_{\text{data}} &= 17 & \leftarrow & R_{\text{flag}} &= 1 & W_{\text{flag}} &= 1 \\
   R_{\text{data}} &= 0 \\
   W_{\text{data}} &= 17
   \end{align*}
   \]
Thread semantics

**Fences.** \([\text{fence}; t] = \text{fence} \cdot [t]\)

\((\leq \ell.E = \leq E \cup \{(\ell, \ell')\})\)
Thread semantics

Fences. \([\text{fence}; t] = \text{fence} \cdot [t]\)

\((\leq \ell . E = \leq E \cup\{(\ell, \ell')\})\)

Writes. \([x := k; t] = W_{x := k}; [t]\)

\((\leq \ell; E = \leq E \cup\{(\ell, \text{fence}), (\ell, \ell') | \text{var}(\ell) = \text{var}(\ell')\})\).
**Thread semantics**

**Fences.** \([\text{fence}; t] = \text{fence} \cdot [t]\)

\((\leq \ell.E = \leq E \cup \{(\ell, \ell')\}))

**Writes.** \([x := k; t] = W_{x := k}; [t]\)

\((\leq \ell; E = \leq E \cup \{(\ell, \text{fence}), (\ell, \ell') | \var(\ell) = \var(\ell')\})\).

**Reads.** \([r \leftarrow x; t] = \sum_{n \in \mathbb{N}} R_{x = n}; [t[n/r]]\)

\[R_{x = 0} \quad R_{x = 1} \quad \ldots\]

\([t[0/r]] \quad [t[1/r]] \quad \ldots\]
Thread semantics

Fences. \([\text{fence}; t] = \text{fence} \cdot [t]\)

\((\leq \ell. E = \leq E \cup \{(\ell, \ell')\})\)

Writes. \([x := k; t] = \text{W}_{x := k}; [t]\)

\((\leq \ell; E = \leq E \cup \{(\ell, \text{fence}), (\ell, \ell') | \text{var}(\ell) = \text{var}(\ell')\})\).

Reads. \([r \leftarrow x; t] = \sum_{n \in \mathbb{N}} R_{x=n}; [t[n/r]]\)

\(R_{x=0} \quad R_{x=1} \quad \ldots\)

\(\leftarrow [t[0/r]] \quad \leftarrow [t[1/r]] \quad \ldots\)

Program. No interaction: \([t_1 \parallel \ldots \parallel t_n] = [t_1] \parallel \ldots [t_n].\)
Wiring memory behaviour

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

\[ C_\mu ::= W_{x:=k} \cdot C_{\mu[x:=k]} \mid \text{fence} \cdot C_\mu \mid R_{x=\mu(x)} \cdot C_\mu \]

**Theorem**

*For a machine state* \((p, \mu)\), \(Tr(p, \mu) = Tr[p] \cap C_\mu.\)
Wiring memory behaviour

The memory behaviour is specified through consistent traces:

\[ C_\mu ::= W_{x:=k} \cdot C_\mu[x:=k] \mid \text{fence} \cdot C_\mu \mid R_{x=\mu(x)} \cdot C_\mu \]

Theorem

For a machine state \((p, \mu)\), \(Tr(p, \mu) = Tr[p] \cap C_\mu\).

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

The memory behaviour is specified through consistent traces:

\[ C_\mu ::= \mathcal{W}_{x:=k} \cdot C_\mu[x:=k] \mid \text{fence} \cdot C_\mu \mid R_{x=\mu(x)} \cdot C_\mu \]

Theorem

For a machine state \((p, \mu)\), \(\text{Tr}(p, \mu) = \text{Tr}[p] \cap C_\mu\).

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

The causal account of memory can be defined as:

\[ C_\mu = \{ q \mid \text{Tr}(q) \in C_\mu \} \in \text{Set}(\text{PO}). \]

How to combine \([p]\) and \(C_\mu\)?
Bridding a gap: event-based and execution-based models

ES

\[ [p] \]
Bridding a gap: event-based and execution-based models

\[ \text{ES} \]

\[ \text{Set(PO)} \]

\[ [p] \]

\[ \mathcal{C}_\mu \]
Briding a gap: event-based and execution-based models

\[ \mathcal{C}(\cdot) \]

\[ \text{ES} \rightarrow \text{Set}(\text{PO}) \]

\[ \begin{align*}
W_x &= 1 \\
R_y &= 2 \\
W_x &= 1 \\
R_y &= 3
\end{align*} \]

\[ \left\{ 
\begin{array}{c}
W_x = 1 \\
W_x = 1 \\
R_y = 2 \\
R_y = 3
\end{array}
\right\} \]
Briding a gap: event-based and execution-based models

Towards a causal and compositional operational semantics of programming languages · Simon Castellan
Bringing a gap: event-based and execution-based models

Towards a causal and compositional operational semantics of programming languages - Simon Castellan
Bridding a gap: event-based and execution-based models

\[ \text{ES} \xleftarrow{\mathcal{C}(\cdot)} \top \xrightarrow{\text{Pr}(\cdot)} \text{Set}(\text{PO}) \]

\[
\begin{align*}
W_x := 1 & \\
\Downarrow & \\
R_y = 2 & \\
\sim & \\
R_y = 3 & \\
\Uparrow & \\
W_x := 1 & \\
\Downarrow & \\
R_y = 2 & \\
W_x := 1 & \\
\Downarrow & \\
R_y = 3 & \\
\end{align*}
\]

Towards a causal and compositional operational semantics of programming languages · Simon Castellan
A partial product on partial orders

Given two partial orders \( \leq_q, \leq_{q'} \) on the same carrier set, write:

\[
q \land q' = \begin{cases} 
(q \cup q')^* & \text{if a partial order} \\
\text{undefined} & \text{otherwise}
\end{cases}
\]
A partial product on partial orders

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

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

\[
\begin{pmatrix}
W_d := 17 & W_f := 1 & R_f := 1 \\
\downarrow & & \downarrow
\end{pmatrix}_{\in \mathbb{[mp]}} \land \begin{pmatrix}
W_d := 17 & W_f := 1 & R_f := 1 \\
\downarrow & & \downarrow
\end{pmatrix}_{\in \mathbb{C}_\mu} = \begin{pmatrix}
W_d := 17 & W_f := 1 & R_f := 1 \\
\downarrow & & \downarrow
\end{pmatrix}_{\in \mathbb{C}_\mu}
\]
A partial product on partial orders

Given two partial orders \( \leq_q, \leq_{q'} \) on the same carrier set, write:

\[
q \land q' = \begin{cases} 
(q \cup q')^* & \text{if a partial order} \\
\text{undefined} & \text{otherwise}
\end{cases}
\]

Towards a causal and compositional operational semantics of programming languages · Simon Castellan
For $P, Q \in \text{Sets}(\text{PO})$, let:

$$P \star Q = \{p \land q \mid p \in P, q \in Q\}.$$
... generating a product on event structures

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

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

For $E, E' \in ES$, let:

$$E \star E' = \Pr(C(E) \star C(E')).$$
... generating a product on event structures

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

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

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

$$E \ast E' = \text{Pr}(\mathcal{C}(E) \ast \mathcal{C}(E')).$$

**Theorem**

*Both operations are categorical products.*

**Note:**

$$\text{Tr}(E \ast E') = \text{Tr}(E) \cap \text{Tr}(E')$$
A final model

Define \([p, \mu] = \Pr(C([p]) \ast C_\mu)\). We have:

\[
\text{Tr}[p, \mu] = \text{Tr}[p] \cap \text{Tr}[C_\mu] = \text{Tr}[p] \cap C_\mu = \text{Tr}(p, \mu).
\]

Yields the desired result:

\[
[[mp, (x \mapsto 0)]] = \begin{cases}
W_{\text{data}}:=17 & W_{\text{flag}}:=1 \leadsto R_{\text{flag}}=0 \\
R_{\text{data}}=17 & \leftarrow R_{\text{flag}}=1 & W_{\text{flag}}:=1 \\
R_{\text{data}}=0 & \downarrow \\
W_{\text{data}}:=17 & \downarrow
\end{cases}
\]
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 $C_\mu$ 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 $C_\mu$ 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 (e.g., 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:

\[
\begin{align*}
\text{alloc}(x); \\
\text{alloc}(y); \\
r & \leftarrow x; \\
\text{if}(r = 1)\{y := 1\} \\
\text{dealloc}(x); \text{dealloc}(y)
\end{align*}
\]
Nontrivial scopes

Imagine now our threads look like:

\[
\begin{align*}
\text{alloc}(x); \\
\text{alloc}(y); \\
\text{r} &\leftarrow x; \\
\text{if}(r = 1)\{y := 1\} \\
\text{dealloc}(x); \text{dealloc}(y)
\end{align*}
\]

Labels should now be:

\[
\Sigma ::= \cdots | a_x | d_x
\]
Nontrivial scopes

Imagine now our threads look like:

\[
\begin{align*}
&\text{alloc}(x); \\
&\text{alloc}(y); \\
&r \leftarrow x; \\
&\text{if}(r = 1)\{y := 1\} \\
&\text{dealloc}(x);\text{dealloc}(y)
\end{align*}
\]

Labels should now be:

\[
\sum ::= \cdots | a_x | d_x
\]
Nontrivial scopes

Imagine now our threads look like:

\begin{align*}
\text{alloc}(x); \\
\text{alloc}(y); \\
r & \leftarrow x; \\
\text{if}(r = 1)\{y := 1\} \\
\text{dealloc}(x); \text{dealloc}(y)
\end{align*}

Labels should now be:

\[ \Sigma ::= \cdots | a_x | d_x \]

*Implicit allocation rules* give \(\Sigma\) 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:

thread \[\xrightarrow{\text{alloc}}\] memory

The protocol is described by the following partial order:

\[A = \text{alloc} \rightarrow \text{done} \rightarrow w \rightarrow r \rightarrow \text{dealloc} \rightarrow \text{done} \rightarrow 0 \rightarrow 1 \rightarrow \ldots\]

Such a partial order with priority annotations is a game.
Protocols as types

Interaction thread/memory is an interaction client/server:

thread \[\text{0xdead}\] memory
Protocols as types

Interaction thread/memory is an interaction client/server:

```
thread  \ w(0xdead, 1)  \ memory
```
Protocols as types

Interaction thread/memory is an interaction client/server:

thread \[\xrightarrow{\text{done}}\] memory
Protocols as types

Interaction thread/memory is an interaction client/server:

```
thread  r(0xdead)  memory
```
Protocols as types

Interaction thread/memory is an interaction client/server:

```
thread 1 memory
```
Protocols as types

Interaction thread/memory is an interaction client/server:

![Diagram showing a directed graph with nodes labeled 'thread' and 'memory' connected by an edge labeled 'd(0xdead)']

What is an event structure labeled by a game?

Towards a causal and compositional operational semantics of programming languages · Simon Castellan 21 / 37
Protocols as types

Interaction thread/memory is an interaction client/server:

![Diagram of thread/memory interaction]

The protocol is described by the following partial order:

\[
A = \begin{align*}
&
\downarrow_{\text{alloc}}

&
\downarrow_{\text{done}}

&
\downarrow_{r}

&
\downarrow_{\text{dealloc}}

& w_k

&\text{done}

&0

&1

&\ldots
\end{align*}
\]
Protocols as types

Interaction thread/memory is an interaction client/server:

The protocol is described by the following partial order:

\[
A = \begin{align*}
&\text{alloc} \\
&\downarrow \\
\text{done} \\
&\text{wk} \\
&\downarrow \\
\text{done} \\
&\text{r} \\
&\downarrow \\
0 & 1 & \ldots \\
&\text{dealloc}
\end{align*}
\]

Such a partial order with polarity annotations is a game.
Protocols as types

Interaction thread/memory is an interaction client/server:

The protocol is described by the following partial order:

\[ A = \]

\[
\begin{array}{c}
\text{alloc} \\
\downarrow \\
\text{done} \\
\downarrow \\
\text{done} \\
\downarrow \\
0 \\
\downarrow \\
1 \\
\downarrow \\
\ldots \\
\end{array}
\]

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 $\sigma : S \rightarrow A$ s.t.:

1. (Respects the rules) $\sigma$ maps configurations of $S$ to down-closed subsets of $A$

2. (Linearity) $\sigma$ is injective on configurations.

![Diagram](https://example.com/diagram.png)
Agent (or pre-strategy)

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

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

No explicit names: each event is below a unique $a$, in the game
Agents and π-calculus

Consider \( A = q \).
Agents and $\pi$-calculus

Consider $A = q \downarrow q \downarrow q \downarrow tt \downarrow ff \downarrow tt \downarrow ff$.

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

$$a : A \vdash a(x, rtt, rff).$$
Agents and $\pi$-calculus

Consider $A = \nu \bar{x} (tt, ff). x(tt, ff)$. Agents can be described by terms of the pi-calculus:

$$a : A \vdash a(x, rtt, rff). \bar{x}(tt, ff).$$
Agents and $\pi$-calculus

Consider $A = q_{tt, rf}$. Agents can be described by terms of the pi-calculus:

$$a : A \vdash a(x, rtt, rff). \bar{x}(tt, ff). \ tt().$$
Agents and $\pi$-calculus

Consider $A = q \quad q \quad tt \quad ff$.

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

$$a : A \vdash a(x, rtt, rff).\overline{x}(tt, ff). \; tt(). \; \overline{rff}$$
Consider $A = q \cdot tt \cdot ff$. Agents can be described by terms of the pi-calculus:

$$a : A \vdash a(x, rtt, rff). \overline{x}(tt, ff).(tt(). \overline{rff} \parallel ff(). \overline{rtt}).$$
Copycat, or the asynchronous forwarder

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

For $B = \underbrace{\q}_{\text{tt}} \underbrace{\q}_{\text{ff}}$, the **copycat** on $B$ is the agent $\mathcal{C}_B$:

\[
\begin{array}{c}
B^\perp \\
\text{tt} \\
\text{ff} \\
\hline
\end{array} ||
\begin{array}{c}
B \\
\text{tt} \\
\text{tt} \\
\end{array}
\]

(corresponding to the term:

\[
a : B^\perp, b : B \vdash b(\text{rtt, rff}). \overline{a}(\text{tt, ff}). (\text{tt}(). \text{rtt} || \text{tt}(). \text{rtt}).
\]
Restriction and composition

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

$$\sigma : S \to A \perp || B \quad \iff \quad \iota : A \perp, o : B \vdash P.$$
Restriction and composition

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

$$\sigma : S \to A \perp || B \iff \iota : A \perp, o : B \not\vdash P.$$ 

Such agents can be composed:

$$\sigma : S \to A \perp || B \quad \tau : T \to B \perp || C \quad \tau \circ \sigma : T \circ S \to A \perp || C$$

Towards a causal and compositional operational semantics of programming languages · Simon Castellan
Restriction and composition

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

$$\sigma : S \rightarrow A \perp \parallel B \quad \Leftrightarrow \quad \iota : A \perp, o : B \vdash P.$$  

Such agents can be composed:

$$\sigma : S \rightarrow A \perp \parallel B \quad \tau : T \rightarrow B \perp \parallel C \quad \Rightarrow \quad \tau \circ \sigma : T \circ S \rightarrow A \perp \parallel C$$

$$a : A \perp, b : B \vdash P \quad b : B \perp, c : C \vdash Q \quad \Rightarrow \quad a : A \perp, c : C \vdash (\nu b)(P \parallel Q)$$
Restriction and composition

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

$$\sigma : S \rightarrow A^\perp \parallel B \quad \iff \quad \iota : A^\perp, o : B \vdash P.$$ 

Such agents can be composed:

$$\sigma : S \rightarrow A^\perp \parallel B \quad \tau : T \rightarrow B^\perp \parallel C \quad \Rightarrow \quad \tau \circ \sigma : T \circ S \rightarrow A^\perp \parallel C$$

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

In two steps:

1. **Interaction** of the common parts of $\sigma$ and $\tau$

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.
   → Interaction $\sigma$ and $\tau$ gives
   $$\tau \otimes \sigma : T \otimes S \rightarrow A \parallel B \parallel C.$$ 

2. **Hiding** relies on **projection** of event structures: events in $B$ become invisible.
   $$\tau \odot \sigma : T \otimes S \downarrow V \rightarrow A \parallel B \parallel C \quad V = \tau \otimes \sigma^{-1}(A \parallel C).$$
An example

Consider:

\[ \sigma = \overset{q}{\downarrow} : \mathbf{0} \parallel B \quad \tau = \mathcal{C}_B : B \parallel B. \]
An example

Consider:

\[
\sigma = \begin{array}{c}
\text{q} \\
\text{tt} \\
\end{array} : \emptyset \perp \parallel B \\
\text{tt} \rightarrow \text{ff}
\]

\[
\tau = c_B : B \perp \parallel B.
\]

The interaction gives:

\[
B \parallel B
\]

Towards a causal and compositional operational semantics of programming languages · Simon Castellan
An example

Consider:

$$\sigma = q \triangleright \text{ : } \emptyset \perp \parallel B \quad \tau = \mathcal{c}_B : B^\perp \parallel B.$$  

The interaction gives:

$$B \parallel B$$
An example

Consider:

\[ \sigma = \text{q} \quad : \emptyset \perp \parallel B \quad \tau = \text{c}_B : B \perp \parallel B. \]

The interaction gives:

\[
\begin{pmatrix}
\left\lfloor a(tt, ff) \cdot \overline{tt} \cdot \overline{ff} \right\rfloor \\
\left\lfloor b(rtt, rff) \cdot \overline{a}(tt, ff) \cdot \left( \left\lfloor tt() \cdot \overline{rtt} \right\rfloor \right. \right\rfloor \\
\left\lfloor ff() \cdot \overline{rff} \right\rfloor
\end{pmatrix}
\]
An example
Consider:

\[ \sigma = \begin{array}{c} q \\ \texttt{tt} \rightarrow \text{ff} \end{array} : \emptyset^\perp \parallel B \quad \tau = \mathfrak{c}_B : B^\perp \parallel B. \]

The interaction gives:

\[
\begin{array}{c}
B \parallel B \\
\texttt{tt} \rightarrow \text{ff} \\
\texttt{tt} \rightarrow \text{ff} \\
\end{array}
\]

\[
(\nu a) \left( \begin{array}{c} a(\texttt{tt}, \text{ff}). \overline{\texttt{tt}}. \overline{\text{ff}} \\
\| b(\text{rtt}, \text{rff}). \overline{a}(\texttt{tt}, \text{ff}). (\texttt{tt}(). \overline{\text{rtt}}) \\
\| \text{ff}(). \overline{\text{rff}} \end{array} \right)
\]
An example

Consider:

\[
\sigma = \begin{array}{c}
\begin{array}{c}
q \\
tt \rightarrow \rightarrow ff
\end{array}
\end{array} : \emptyset \perp \parallel B \quad \tau = \mathcal{C}_B : B \perp \parallel B.
\]

The interaction gives:

\[
\begin{array}{c}
\begin{array}{c}
B \quad \parallel \quad B
\end{array}
\end{array}
\]

\[
(\nu a) \left( \begin{array}{c}
\begin{array}{c}
\overline{tt} \cdot \overline{ff} \\
\parallel b(rtt, rff). \left( \begin{array}{c}
\begin{array}{c}
\overline{tt}() \cdot \overline{rtt} \\
\parallel \overline{ff}() \cdot \overline{rff}
\end{array}
\end{array}
\right)
\end{array}
\right)
\]

Towards a causal and compositional operational semantics of programming languages · Simon Castellan
An example

Consider:

$$\sigma = q$$

$$\text{tt} \rightarrow \text{ff}$$

$$: \emptyset \perp \parallel B$$

$$\tau = c_B : B \perp \parallel B.$$ 

The interaction gives:

$$B \parallel B$$

$$(\nu a) \left( \lll b(rtt, rff). \left( \frac{rtt}{\lll ff()} \right) \right)$$
An example

Consider:

\[
\sigma = q : \emptyset \parallel B \quad \tau = \mathcal{C}_B : B^\perp \parallel B.
\]

The interaction gives:

\[
(\nu a) \left( \parallel b(rtt, rff). \left( \frac{rtt}{\parallel rff} \right) \right)
\]
An example

Consider:

\[ \sigma = q \quad : \emptyset \perp \parallel B \quad \tau = \mathfrak{c}_B : B^\perp \parallel B. \]

The interaction gives:

\[ B \parallel B \]

\[ (\nu a) \left( \begin{array}{c} \parallel b(rtt, rff) \cdot \left( \frac{rtt}{\parallel rff} \right) \end{array} \right) \]

\( \sigma \) not invariant under the asynchronous forwarder.
Asynchronous agents, or strategies

Which agents $\sigma$ satisfy $c_A \circ \sigma \simeq \sigma$?

**Definition**

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

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

**Theorem (Rideau, Winskel)**

An agent $\sigma$ is a strategy if and only if $c_A \circ \sigma \simeq \sigma$.

→ Games and strategies model linear languages (compact-closed category).
III. Interpreting functional programming languages
Local injectivity and copy indices

To represent nonlinear agents:

\[ B \Rightarrow B \]
Local injectivity and copy indices

To represent nonlinear agents:

\[ B \Rightarrow B \]

The labelling to \( B \Rightarrow B \) fails local injectivity.
Local injectivity and copy indices

To represent nonlinear agents:

\[ !^+ ( B \Rightarrow B ) \]

The labelling to \( B \Rightarrow B \) fails local injectivity.

→ We make the game bigger.
Uniformity

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

$$B$$

$$q$$

$$tt$$ $$tt$$
Uniformity

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

![Diagram](image-url)

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 $! J \Gamma K \perp \parallel J A K$,
- Composition is: interaction + hiding.
Uniformity

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

\[
\begin{align*}
!B & \\
q_0 & \downarrow \quad q_1 & \downarrow \\
tt_0 & \quad tt_1 & tt_0 & tt_1 & \ldots
\end{align*}
\]
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:

\[
\begin{array}{c}
\begin{array}{c}
!B \\
q_0 \\
\uparrow \\
tt_0
\end{array} & \begin{array}{c}
q_1 \\
\uparrow \\
tt_1
\end{array} & \ldots \\
\begin{array}{c}
q_{2i} \\
\downarrow \\
tt_0
\end{array} & \begin{array}{c}
q_{2i+1} \\
\downarrow \\
ff_0
\end{array}
\end{array}
\]

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 \( !([\Gamma] \parallel [A]) \),
- Composition is: interaction + hiding.
An example of higher-order

Consider the call-by-name program

\[
\text{strict} = \lambda f. \text{new } r \text{ in } f(r := 1; 1); !r = 1 : (\mathbb{N} \Rightarrow \mathbb{N}) \Rightarrow \mathbb{B}.
\]
An example of higher-order

Consider the call-by-name program

\[ \text{strict} = \lambda f. \text{new } r \text{ in } f(r := 1; 1); !r = 1 : (\mathbb{N} \Rightarrow \mathbb{N}) \Rightarrow \mathbb{B}. \]

If \( f \) is concurrent with control operators, strict exhibits a race.

**Theorem**

*CHO can interpret concurrent languages, adequately for may:*

\[ M \Downarrow \iff [M] \text{ contains a positive move.} \]
Hidden divergences

However, in nondeterministic languages convergence is more subtle:

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

Does \( M \) choice converge?
Hidden divergences

However, in nondeterministic languages convergence is more subtle:

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

Does \( M \) choice converge?

\[
\begin{array}{ccc}
[M] & : & B \\
& & \Rightarrow \\
q & & B
\end{array}
\]
Hidden divergences

However, in nondeterministic languages convergence is more subtle:

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

Does \( M \text{ choice} \) converge?

\[
[M] \otimes \text{choice} : \begin{array}{c}
\text{q} \\
\text{q} \\
\text{tt} \sim \sim \sim \sim \text{ff} \\
\text{tt}
\end{array}
\]
Hidden divergences

However, in nondeterministic languages convergence is more subtle:

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

Does \( M \) choice converge?

\[
\mathcal{B} = [M] \circ \text{choice}:
\]

As a result: \([M \text{ choice}] = [tt]\. 

Model inadequate for non-angelic convergences!
Essential events

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

\[ [M] \circ \text{choice} : \]

\[
\begin{array}{c}
q \\
tt \sim \sim \sim \sim \sim ff \\
\end{array}
\]

Towards a causal and compositional operational semantics of programming languages · Simon Castellan
Essential events

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

\[
[M] \circ \text{choice} : \mathbb{B}
\]

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

**Theorem (C., Clairambault, Hayman, Winskel)**

*The partial strategies* \( \tau \odot \sigma \) *and* \( \tau \boxdot \sigma \) *are weakly bisimilar.*

\( \rightarrow \) Partial hiding does not lose behaviour up to weak bisimilarity.
The category $\text{CHO}_\odot$

Despite not hiding everything, we still get a category:

**Theorem (C.)**

The following model $\text{CHO}_\odot$ is a model of higher-order computation:

- Types are interpreted as in $\text{CHO}$,
- Terms are interpreted by strategies with internal events,
- Composition is: interaction + partial hiding.

Moreover $\text{CHO}_\odot$ interprets nondeterministic languages, adequately for non-angelic convergences (must, fair), ...

In $\text{CHO}_\odot$, 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, \textit{l-nets}.
  Partial order representation for ludics.

Parallel works:

▶ Hirschowitz \textit{et. al.}: presheaves over graphs. (no hiding)
  Gives intensional models of $\pi$-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 \textit{et. al.}, Pichon \textit{et. al.}
Extensions / Other work

Axis of development:

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

2. **Adding quantitative 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 $\pi$-calculus (with Clairambault and Yoshida).