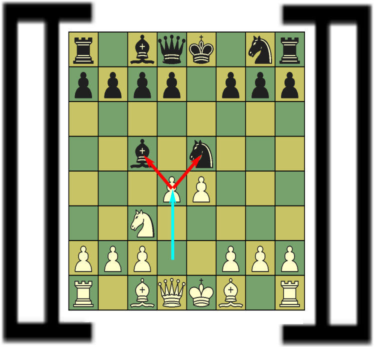
A game semantics of `fork(II)`



$= ?$

Simon Castellan(joint work with Pierre Clairambault)

GaLoP 2015

Introduction — Concurrent Games

- Rideau and Winskel developed a framework for game semantics based on event structures.
- We recently extended this to CHO, a "truly concurrent" extension of HO games.
- Two approaches to tame the broad mathematical space:
  - Cutting down the space of strategies to get definability results for increasing powerful languages.
    (Full abstraction for parallel stateless languages.)
  - Designing very expressive languages to understand the model operationally .

Introduction — Concurrent Games

- Rideau and Winskel developed a framework for game semantics based on event structures.
- We recently extended this to CHO, a "truly concurrent" extension of HO games.
- Two approaches to tame the broad mathematical space:
  - Cutting down the space of strategies to get definability results for increasing powerful languages.
    (Full abstraction for parallel stateless languages.)
  - Designing very expressive languages to understand the model operationally (this talk).

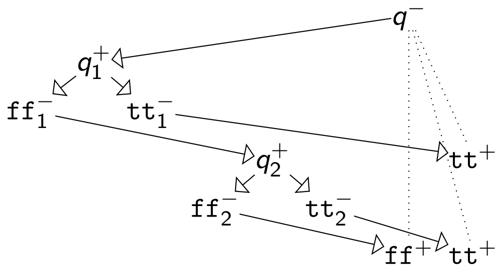
I. Presentation of CHO

Overview of concurrent games

Difference between usual game semantics and the concurrent games:

- set of plays $\to$ **one** labelled event structure
- behaviour against: all Opponents $\to$ a most general one

$\texttt{lor}: \ \mathbb{B}_1 \Longrightarrow \mathbb{B}_2 \Longrightarrow \mathbb{B}$

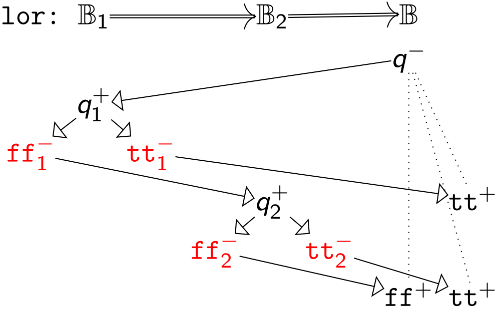


$\texttt{lor}$ is sequential innocent : the strategy is an O-branching tree.
tree = no merges = no O-merges + no P-merges

Overview of concurrent games

Difference between usual game semantics and the concurrent games:

- set of plays $\rightarrow$ **one** labelled event structure
- behaviour against: all Opponents $\rightarrow$ a most general one



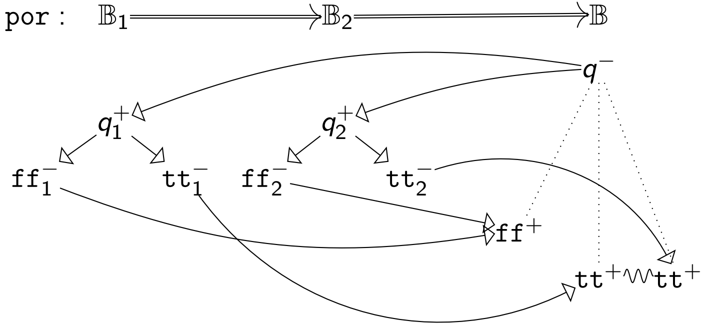
`lor` is sequential innocent : the strategy is an O-branching tree.

tree = no merges = no O-merges + no P-merges

Overview of concurrent games

Difference between usual game semantics and the concurrent games:

- set of plays → **one** labelled event structure
- behaviour against: all Opponents → a most general one
- Highlights aspects of concurrency: forks, joins, races, threads.



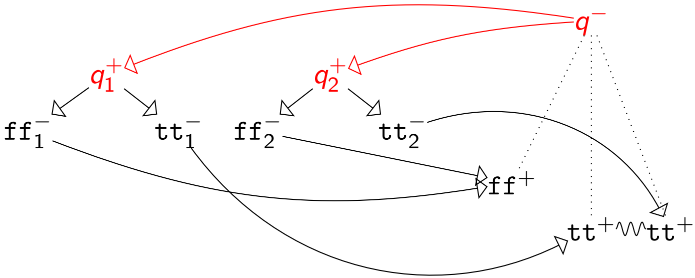
`por` is (concurrent) innocent: the strategy is only P-merging

Overview of concurrent games

Difference between usual game semantics and the concurrent games:

- set of plays $\rightarrow$ **one** labelled event structure
- behaviour against: all Opponents $\rightarrow$ a most general one
- Highlights aspects of concurrency: forks, joins, races, threads.

$\texttt{por} : \quad \mathbb{B}_1 \Longrightarrow \mathbb{B}_2 \Longrightarrow \mathbb{B}$

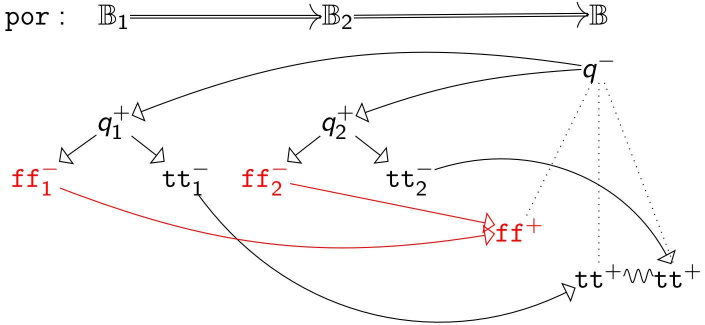


$\texttt{por}$ is (concurrent) innocent: the strategy is only P-merging

Overview of concurrent games

Difference between usual game semantics and the concurrent games:

- set of plays $\to$ **one** labelled event structure
- behaviour against: all Opponents $\to$ a most general one
- Highlights aspects of concurrency: forks, joins, races, threads.

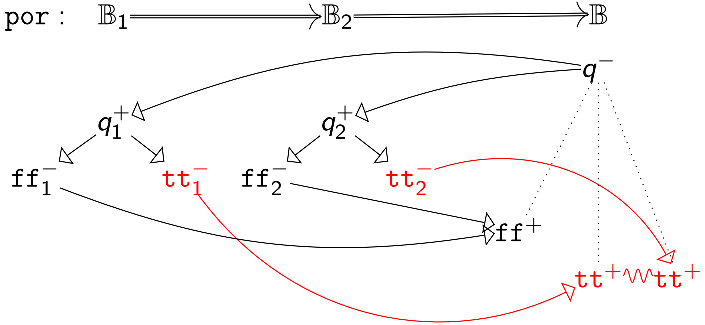


$\mathtt{por}$ is (concurrent) innocent: the strategy is only P-merging

Overview of concurrent games

Difference between usual game semantics and the concurrent games:

- set of plays → **one** labelled event structure
- behaviour against: all Opponents → a most general one
- Highlights aspects of concurrency: forks, joins, races, threads.



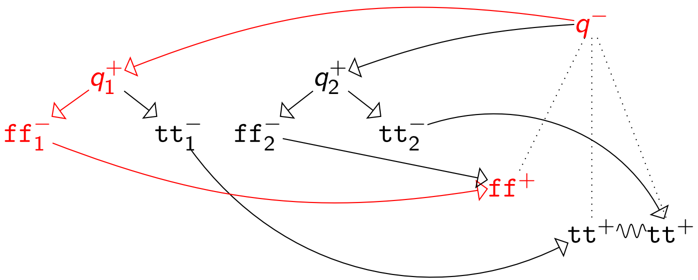
por is (concurrent) innocent: the strategy is only P-merging

Overview of concurrent games

Difference between usual game semantics and the concurrent games:

- set of plays → **one** labelled event structure
- behaviour against: all Opponents → a most general one
- Highlights aspects of concurrency: forks, joins, races, threads.

$\texttt{por}: \quad \mathbb{B}_1 \Rightarrow \mathbb{B}_2 \Rightarrow \mathbb{B}$



$\texttt{por}$ is (concurrent) innocent: the strategy is only P-merging

Event structures

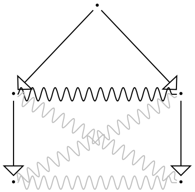
Definition (Event structures)

Event structures are tuples $(E, \leq, \sharp)$ where:

- $(E, \leq)$ is a partial order
- $\sharp \subseteq E^2$ is an irreflexive symmetric relation

satisfying *conflict inheritance*:

$$e \sharp e' \,\&\, e' \leq e'' \implies e \sharp e''$$



- **Configurations of** E ($\mathscr{C}(E)$): Finite downclosed sets of pairwise-compatible elements of E
- An **arena** $(A, \vdash, \mathrm{pol})$ (alternating forest) can be seen as an event structure $(A, (\vdash)^*, \emptyset)$ with a polarity labelling.

  It is negative when all its minimal events are.

  If A is an arena, $A^\perp$ is obtained from A by reversing the polarities.

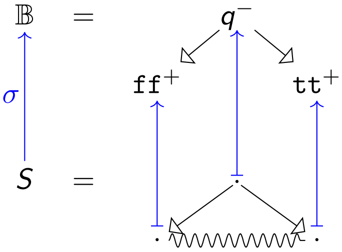
Strategies

Definition (Pre-strategies)

A **pre-strategy** on an arena A is an event structure S along with a labelling function $\sigma : S \to A$ such that

- $x \in \mathscr{C}(S) \Longrightarrow \sigma x \in \mathscr{C}(A)$
- σ is injective on configurations

This exactly means that σ is a *map of event structures*.



A strategy $\sigma : S \to A$ is a pre-strategy satisfying:

1. **courtesy**: in S the extra causal links are of the form $\ominus \rightarrowtriangle \oplus$.
2. **receptivity**: any negative extension in A of a configuration reached by σ has a unique lifting in S.

Strategies

Definition (Pre-strategies)

A **pre-strategy** on an arena A is an event structure S along with a labelling function $\sigma : S \to A$ such that

- $x \in \mathscr{C}(S) \Longrightarrow \sigma x \in \mathscr{C}(A)$
- σ is injective on configurations

This exactly means that σ is a *map of event structures*.

$$\sigma \quad : \quad \begin{tikzcd} & q^- \arrow[dl] \arrow[dr] & \\ \mathtt{ff}^+ \arrow[rr, no head, squiggly] & & \mathtt{tt}^+ \end{tikzcd}$$

A strategy $\sigma : S \to A$ is a pre-strategy satisfying:

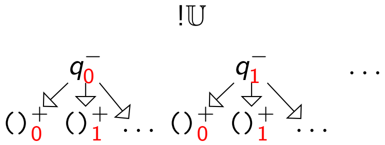
1. **courtesy**: in S the extra causal links are of the form $\ominus \rightarrowtriangle \oplus$.
2. **receptivity**: any negative extension in A of a configuration reached by σ has a unique lifting in S.

To draw a strategy we draw the corresponding event structure with labels induced by σ.

Expanded arenas

Given an arena A, form an arena $!A$:

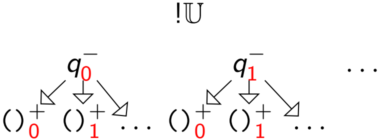
- events: $(a \in A, \alpha : [a] \to \mathbb{N})$
  α gives a copy index to dependencies of the *label* a.
- ordering: $(a, \alpha) \leq (a', \alpha')$ when $a \leq a'$ and $\alpha \subseteq \alpha'$.



Expanded arenas

Given an arena A, form an arena $!A$:

- events: $(a \in A, \alpha : [a] \to \mathbb{N})$
  α gives a copy index to dependencies of the *label* a.
- ordering: $(a, \alpha) \leq (a', \alpha')$ when $a \leq a'$ and $\alpha \subseteq \alpha'$.



A **symmetry** of $!A$ is an order-isomorphism between two configurations of $!A$ preserving labels.
Two strategies $\sigma : S \to !A$ and $\tau : T \to !A$ "are isomorphic up to copy indices" when there is an iso $\varphi : S \cong T$ such that

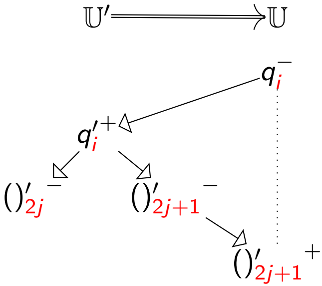
$$\theta_x = \{(\sigma s, \tau(\varphi s)) \mid s \in x\}$$

is a symmetry on $!A$ for $x \in \mathscr{C}(S)$.

Weak equivalence and uniformity

- "isomorphic up to copy indices" is **not** a congruence:

bad :



- To overcome this, we introduced:
  - $\sim$-strategies $\sigma : \mathcal{S} \to\ !A$ that are uniform wrt Opponent copy indices (method similar as that of AJM games).
  - a congruence (*weak equivalence*) of $\sim$-strategies.

The category CHO

- As usual to get a CCC one needs to ask that our strategies behave the same way no matter how many times they are called.
- In our setting, we say that $\sigma : \mathcal{S} \to\, !A$ is **single-threaded** when the subsets of $\mathcal{S}$ lying over two distinct minimal questions are *disjoint* and *compatible*.
- Then we get a cartesian closed category given by
  - **Objects**: negative arenas
  - **Morphism from** A **to** B: Negative single-threaded $\sim$-strategies playing on $!(A \Rightarrow B)$ up to weak equivalence. (where $A \Rightarrow B$ is the usual arrow construction on arenas).

What is this "most general" Opponent?

- In our setting, interaction of $\sigma : \mathcal{S} \to\, !A$ against $\tau : \mathcal{T} \to (!A)^{\perp}$ is given by **pullback of maps of event structures (without polarities)** of σ along τ:

$$\sigma \circledast \tau : S \circledast T \to\, !A$$

  (*generalized intersection*)
- The pullback of σ along the full injection $!A \hookrightarrow\, !A$ is isomorphic to σ. $\to$ It is the "most general" Opponent.
- The full injection satisfies the conditions of $\sim$-strategy...what does it mean?
  On $\mathbb{B}$ (before expansion):

$$\mathbb{B}$$

$$\begin{tikzcd} & q^- \arrow[dl] \arrow[dr] & \\ \mathtt{ff}^+ & & \mathtt{tt}^+ \end{tikzcd}$$

- Answers concurrently twice to the same question.

fork(II)

NAME fork -- spawn new process

SYNOPSIS sys fork / fork = 2.
(new process return)
(old process return)

DESCRIPTION fork is the only way new processes are created. The new process's core image is a copy of that of the caller of fork the only distinction is the return location and the fact that r0 in the old process contains the process ID of the new process. This process ID is used by wait.

FILES

SEE ALSO sys wait, sys exec

DIAGNOSTICS The error bit (c–bit) is set in the old process if a new process could not be created because of lack of swap space.

BUGS See wait for a subtle bug in process destruction.

OWNER ken, dmr

II. The fork-calculus

Syntax of the fork calculus

PCF
+ synchronous message-passing
+ a monoid structure on each base type.

$$
\begin{array}{rll}
A, B ::= & \texttt{nat} \mid \texttt{bool} \mid \texttt{unit} & \text{(base types)} \\
\mid & \text{chan} & \text{(channel types)} \\
\mid & A \Rightarrow B & \text{(arrow types)} \\
t, u ::= & x \mid \lambda x.\ t \mid t\ u \mid Y & \text{(simply typed } \lambda\text{-calculus + fixpoint)} \\
\mid & c & \text{(PCF constants)} \\
\mid & t; u \mid \text{if } t \text{ then } u \text{ else } v & \text{(destructors)} \\
\mid & \texttt{new}\ \alpha\ \texttt{in}\ t & \text{(channel creation)} \\
\mid & \text{send}\, t\, u \mid \text{recv}\, t & \text{(operations on channels)} \\
\mid & t \parallel u \mid \text{exit} & \text{(forks – only on base types)}
\end{array}
$$

Called Idealized CSP by Jim Laird.

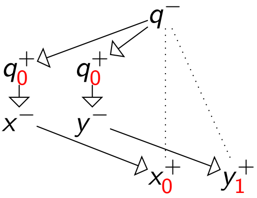
Semantics

Very similar in spirit to the model by Jim Laird based on *non-alternating HO games* (with concurrency pointers).

Interpretation of forks.

Interpretation of $\parallel$:

$\mathbb{X} \Longrightarrow \mathbb{X} \Longrightarrow \mathbb{X}$



Interpretation of channels.

- $[\![\mathrm{chan}]\!] = [\![\mathtt{nat}]\!] \times [\![\mathtt{unit}]\!]^{\mathbb{N}}$
- send and recv: usual accessors
- $\mathtt{new}\ c\ \mathtt{in}\ t$ interpreted by pre-composition with a pre-strategy $\mathtt{newchan} : [\![\mathrm{chan}]\!]$.

We have **soundness**. If $t : \mathbb{X}$ eventually evaluates to $x_1 \parallel \ldots \parallel x_n \parallel t'$ then $[\![t]\!]$ contains one positive move for each x_i.

The "control operator" flavour of fork

We can use the fork-calculus to make the previous observations formal.

- As noticed by Jim Laird, the term:

  ```
  let call/cc f =
    new α in
    Y (λp. p ∥ recv α) ∥ f (λx. send α x; exit)
  ```

  of the fork calculus has a denotation observationally equivalent to the usual strategy for `call/cc`.

- We deduce that "Every thread is well-bracketed" is not stable under composition.

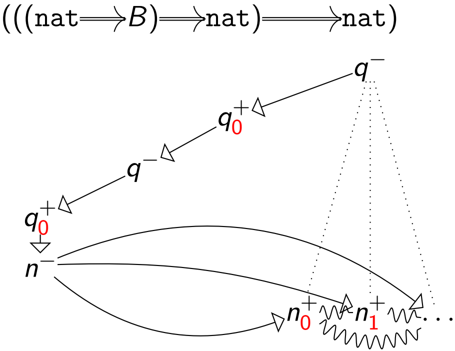
- We also have the converse direction: `fork` is definable from `call/cc` and a `join` operator:

  ```
  let fork = callcc (λk. join (k tt) (k ff))
  ```

Is call/cc really call/cc?

```
let call/cc f =
  new α in
  Y (λp. p ∥ recv α) ∥
  f (λx. send α x; exit)
```

$\rightarrow$

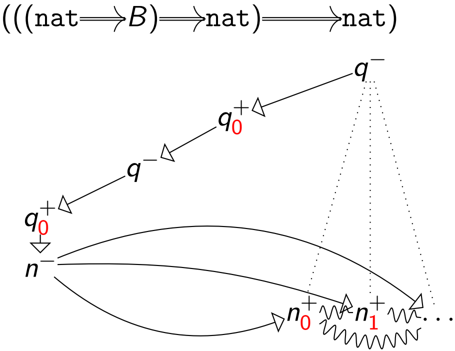


Infinitely many races.

Is call/cc really call/cc?

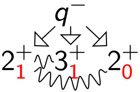
```
let call/cc f =
  new α in
  Y (λp. p ‖ recv α) ‖
  f (λx. send α x; exit)
```

$\rightarrow$

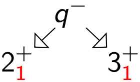


Infinitely many races. Visible divergences tracking:

`if choice then 2 else (3 || 2)` $\rightarrow$



`3 || 2` $\rightarrow$



Conclusion and perspectives

- **Conclusion**: CHO can model very complex non-deterministic and concurrent behaviour
- **Future works**:
  - Take into account hidden divergences:
    `if choice then tt else` $\Omega \not\simeq tt$
  - Factorization results (through the addition of a *program order* akin to Laird's concurrency pointers)
- **Other work**: conditions for extensional definability (parallel PCF, PCF+parallel-or)