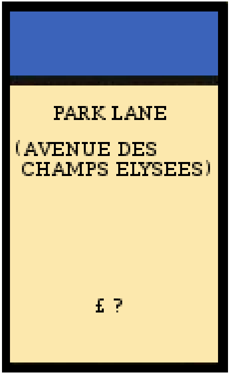
Structures concurrentes en sémantique des jeux

Simon Castellan, LIP
Soutenance de thèse

13 juillet 2017

A Monopoly problem: the theory

Property of Albert.



Property of Barnabé.

To buy both:

$$\text{price} = \text{price Albert} + \text{price Barnabé} \stackrel{?}{=} \text{price Barnabé} + \text{price Albert}$$

A Monopoly problem: the practice

— (to B) How much for park lane?
— (B) **£400**.
— (to A) How much for mayfair?
— (A) Never mind the monopoly, I need money: **£300**

Total price: £700.

A Monopoly problem: the practice

— (to B) How much for park lane?
— (B) **£400**.
— (to A) How much for mayfair?
— (A) Never mind the monopoly, I need money: **£300**

Total price: £700.

— (to A) How much for mayfair?
— (A) I need money: **£300**
— (to B) How much for park lane?
— (B) Eh! This monopoly will cost you: **£600**

Total price: £900.

Beyond formulae : strategies

$$\text{price} = \text{price Albert} + \text{price Barnabé} \stackrel{?}{=} \text{price Barnabé} + \text{price Albert}$$

Albert *then* Barnabé

(to *A*) How much for mayfair?	q_A
(*A*) I need money: **£300**	↓ 300
(to *B*) How much for park lane?	↓ q_B
(*B*) Eh! **£600**	↓ 600
Total price: £900	↓ 900

Beyond formulae : strategies

$$\text{price} = \text{price Albert} + \text{price Barnabé} \stackrel{?}{=} \text{price Barnabé} + \text{price Albert}$$

Albert *then* Barnabé

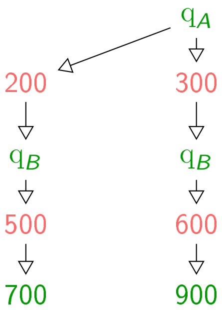
(to A) How much for mayfair?

(A) I need money: **£200**

(to B) How much for park lane?

(B) Eh! **£500**

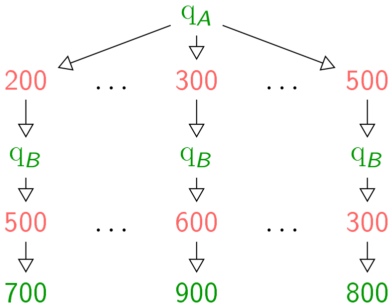
Total price: £700



Beyond formulae : strategies

$$\text{price} = \text{price Albert} + \text{price Barnabé} \stackrel{?}{=} \text{price Barnabé} + \text{price Albert}$$

Albert *then* Barnabé

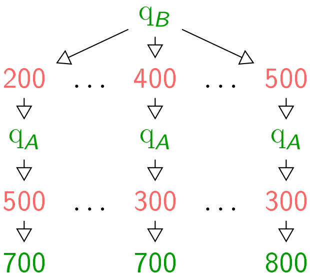


Beyond formulae : strategies

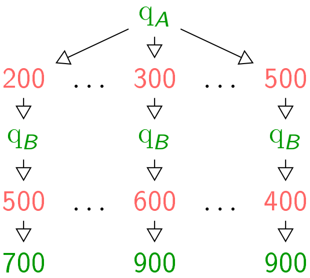
$$\text{price} = \text{price Albert} + \text{price Barnabé} \stackrel{?}{=} \text{price Barnabé} + \text{price Albert}$$

Same *formula*, two **different strategies:**

Barnabé *then* Albert



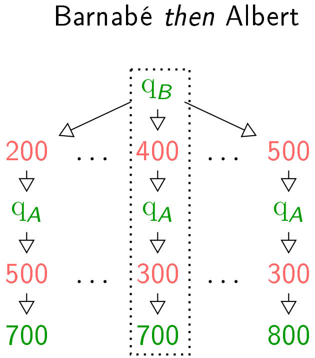
Albert *then* Barnabé



Beyond formulae : strategies

$$\text{price} = \text{price Albert} + \text{price Barnabé} \stackrel{?}{=} \text{price Barnabé} + \text{price Albert}$$

Same *formula*, two **different strategies:**



A **branch** is a play of the strategy against a particular environment.

Two kinds of interpretations

Those strategies correspond to different programs:

$x = \texttt{askAlbert}()$	$x = \texttt{askBarnabé}()$
$y = \texttt{askBarnabé}()$	$y = \texttt{askAlbert}()$
$\text{tot} = x + y$	$\text{tot} = x + y$

Two kinds of interpretations

Those strategies correspond to different programs:

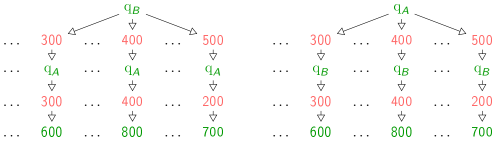
x = `askAlbert()`	x = `askBarnabé()`
y = `askBarnabé()`	y = `askAlbert()`
tot $= x + y$	tot $= x + y$

These programs have two different interpretations (or semantics):

- *Static* interpretation via **formulae** (Input / Output)

$$\text{price} = \text{price Albert} + \text{price Barnabé}$$

- *Dynamic* interpretation via **strategies** (Interactive process)



Game semantics

Game semantics interpretation of more complicated formulae:

$$\texttt{average}(f) = \frac{f(\,0\,) + f(1\,)}{2}$$

Game semantics

Game semantics interpretation of more complicated formulae:

$$\mathtt{average}(f) = \frac{f(0) + f(1)}{2} \qquad \text{against} \qquad f(x) = 2 \times x + 3$$

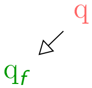
?

q

Game semantics

Game semantics interpretation of more complicated formulae:

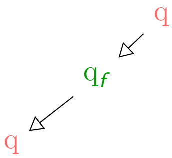
$$\texttt{average}(f) = \frac{f(\,0\,) + f(1\,)}{2} \qquad \text{against} \qquad f(\overset{?}{x}) = 2 \times x + 3$$



Game semantics

Game semantics interpretation of more complicated formulae:

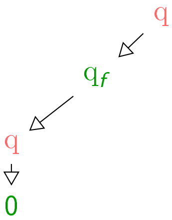
$$\texttt{average}(f) = \frac{f(\overset{?}{0}) + f(1)}{2} \qquad \text{against} \qquad f(x) = 2 \times x + 3$$



Game semantics

Game semantics interpretation of more complicated formulae:

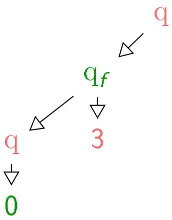
$$\texttt{average}(f) = \frac{f(0) + f(1\,)}{2} \qquad \text{against} \qquad f(x) = 2 \times 0 + 3$$



Game semantics

Game semantics interpretation of more complicated formulae:

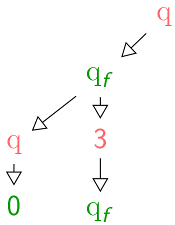
$$\texttt{average}(f) = \frac{3 + f(1)}{2} \qquad \text{against} \qquad f(x) = 3$$



Game semantics

Game semantics interpretation of more complicated formulae:

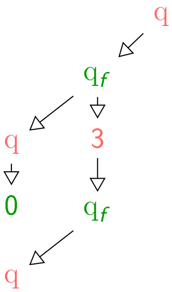
$$\texttt{average}(f) = \frac{3 + f(1)}{2} \qquad \text{against} \qquad f(\overset{?}{x}) = 2 \times x + 3$$



Game semantics

Game semantics interpretation of more complicated formulae:

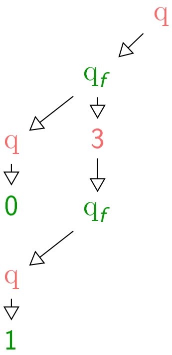
$$\texttt{average}(f) = \frac{3 + f(\overset{?}{1})}{2} \qquad \text{against} \qquad f(x) = 2 \times x + 3$$



Game semantics

Game semantics interpretation of more complicated formulae:

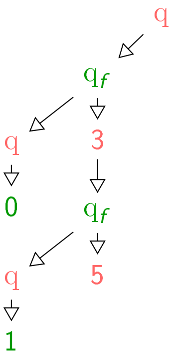
$$\texttt{average}(f) = \frac{3 + f(1)}{2} \qquad \text{against} \qquad f(x) = 2 \times 1 + 3$$



Game semantics

Game semantics interpretation of more complicated formulae:

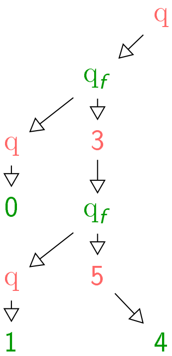
$$\texttt{average}(f) = \frac{3+5}{2} \qquad \text{against} \qquad f(x) = 5$$



Game semantics

Game semantics interpretation of more complicated formulae:

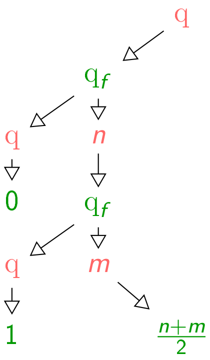
$$\texttt{average}(f) = 4 \qquad \text{against} \qquad f(x) = 2 \times x + 3$$



Game semantics

Game semantics interpretation of more complicated formulae:

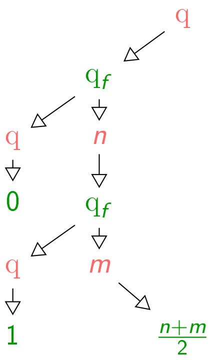
$$\texttt{average}(f) = \frac{f\,(0) + f\,(1)}{2}$$



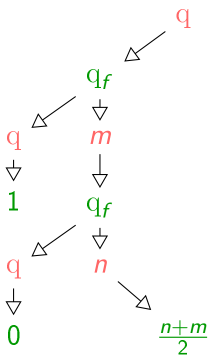
Game semantics

Game semantics interpretation of more complicated formulae:

$$\texttt{average}(f) = \frac{f(0) + f(1)}{2}$$



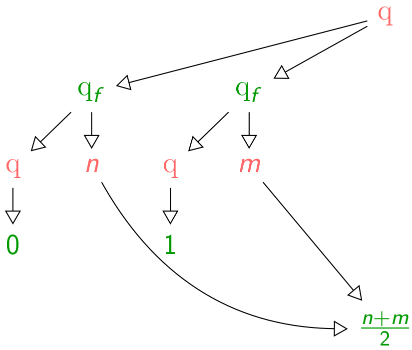
left *then* right



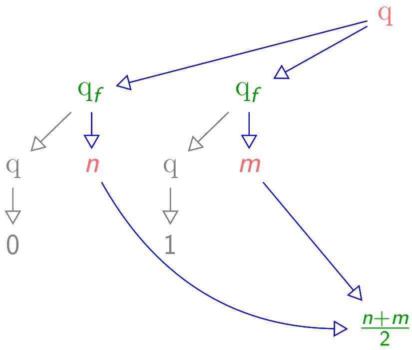
right *then* left

What if we want to compute in parallel?

What if we want to compute in parallel?



What if we want to compute in parallel?



Problem

Usual game semantics only manipulates sequential plays.

Is this sound?

For which strategies is this optimization correct?

Code	
`func` $f_1(x)$: `return` $2x + 3$	

`func` $f_2(x)$:
  `if(rand()) return` $2 \times x$
  `else return 0`	

`func` $f_3(x)$:
  increment counter
  `return counter`	

Is this sound?

For which strategies is this optimization correct?

Code	Average
`func` $f_1(x)$: `return` $2x + 3$	4
`func` $f_2(x)$: `if(rand()) return` $2 \times x$ `else return 0`	
`func` $f_3(x)$: increment counter `return counter`	

Is this sound?

For which strategies is this optimization correct?

Code	Average
`func` $f_1(x)$: `return` $2x + 3$	4
`func` $f_2(x)$: `if(rand()) return` $2 \times x$ `else return 0`	0, 1, or 2
`func` $f_3(x)$: increment counter `return counter`	

Is this sound?

For which strategies is this optimization correct?

Code	Average
`func` $f_1(x)$: `return` $2x + 3$	4
`func` $f_2(x)$: `if(rand()) return` $2 \times x$ `else return 0`	0, 1, or 2
`func` $f_3(x)$: increment counter `return counter`	depends on the impl.

Correct **sequential** strategies are **innocent** and **well-bracketed**.

Is this sound?

For which strategies is this optimization correct?

Code	Average
`func` $f_1(x)$: `return` $2x + 3$	4
`func` $f_2(x)$: `if(rand()) return` $2 \times x$ `else return 0`	0, 1, or 2
`func` $f_3(x)$: increment counter `return counter`	depends on the impl.

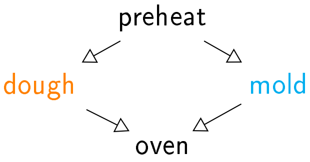
Correct **sequential** strategies are **innocent** and **well-bracketed**.

Problem

And what about *concurrent* strategies?

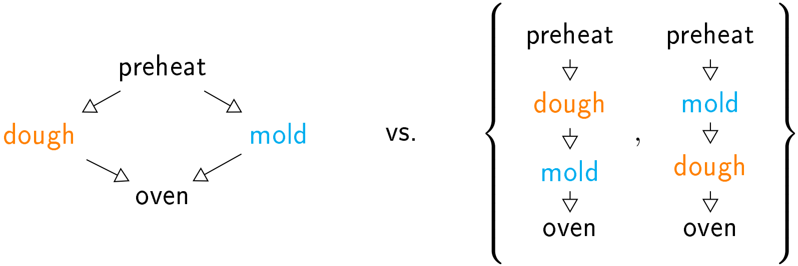
Partial orders or interleavings?

To represent concurrency:



Partial orders or interleavings?

To represent concurrency:



partial order

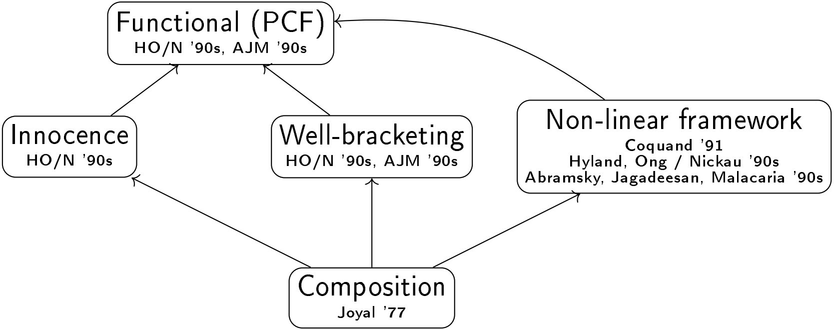
interleavings

(true concurrency)

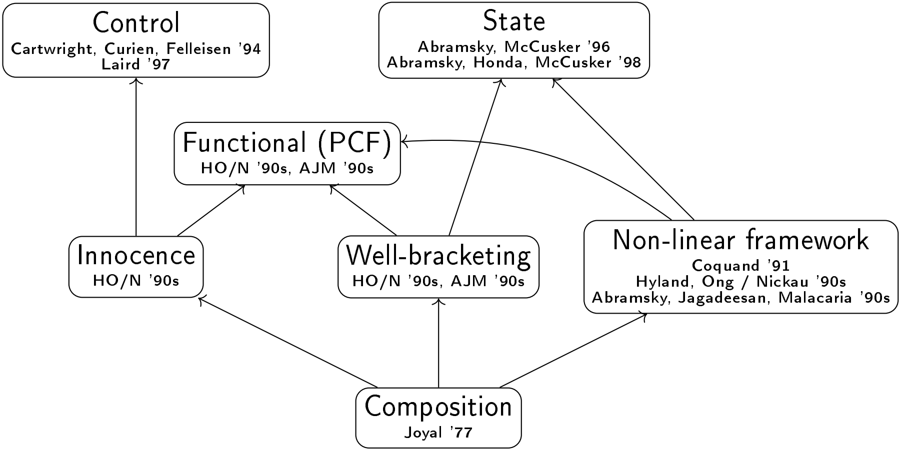
Game semantics: sequential strategies

Composition
Joyal '77

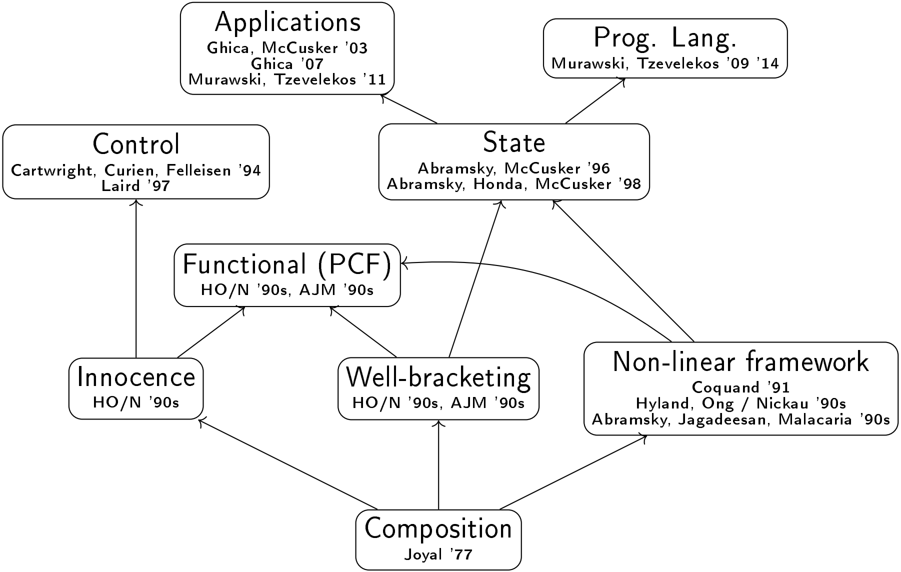
Game semantics: sequential strategies



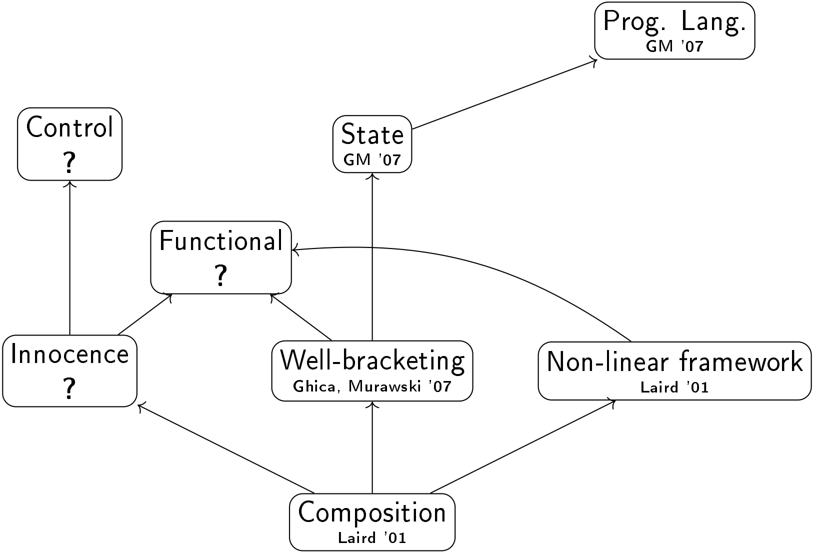
Game semantics: sequential strategies



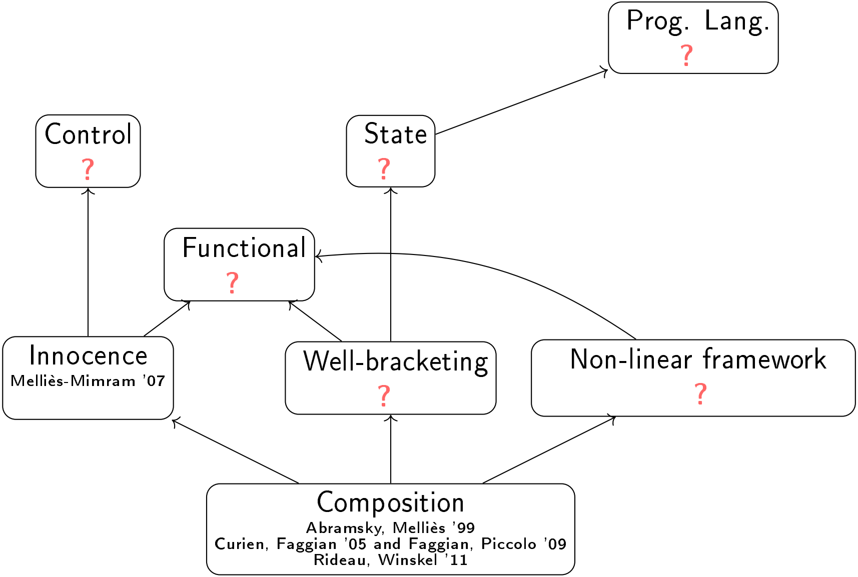
Game semantics: sequential strategies



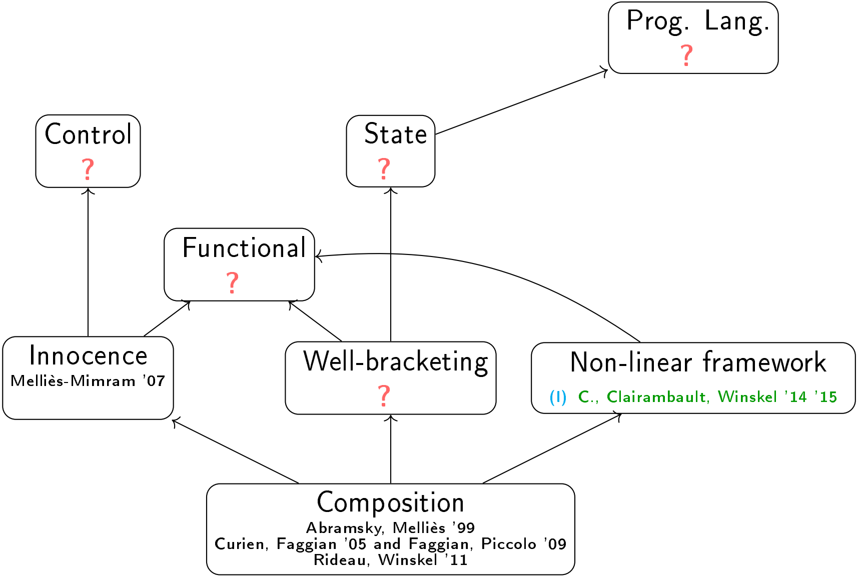
Game semantics: concurrent strategies via interleavings



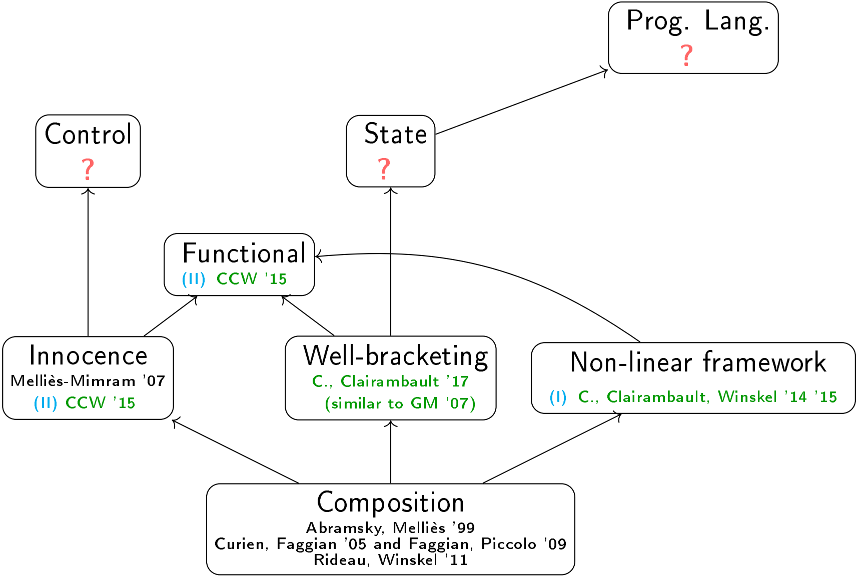
Game semantics: truly concurrent strategies



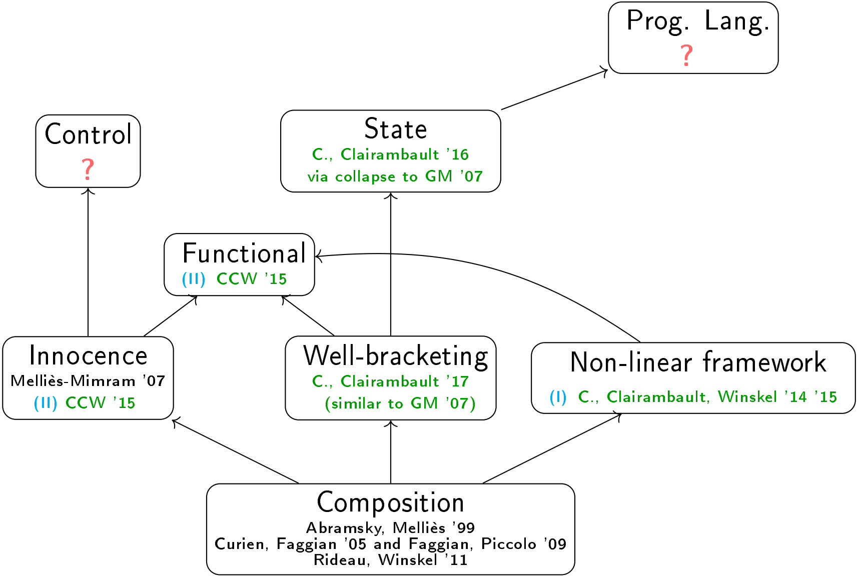
Game semantics: truly concurrent strategies



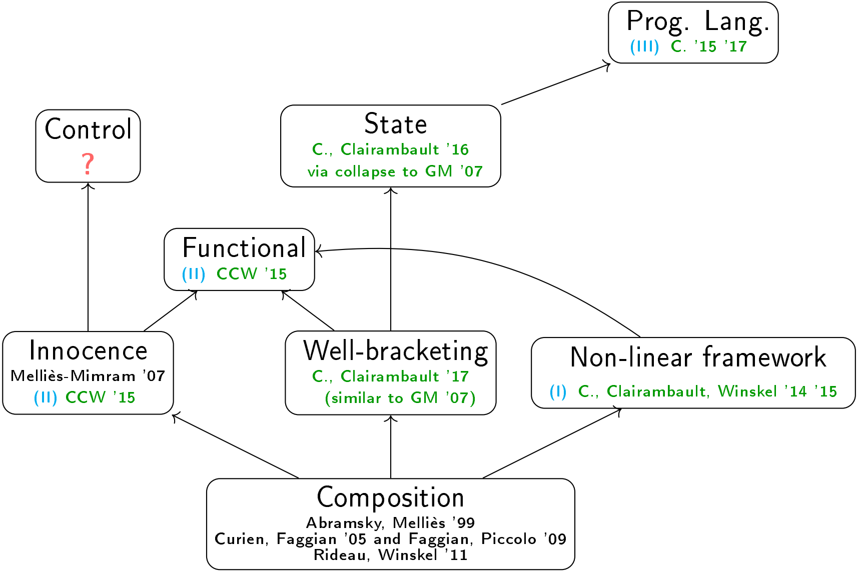
Game semantics: truly concurrent strategies



Game semantics: truly concurrent strategies



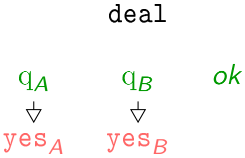
Game semantics: truly concurrent strategies



I. A framework for partial-order strategies

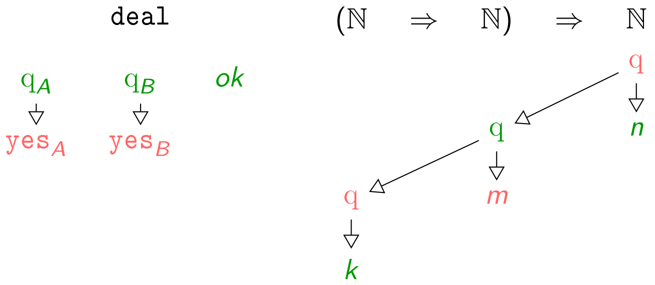
Games

Game: partial order where each *move* has a polarity (*Opponent*, *Player*).



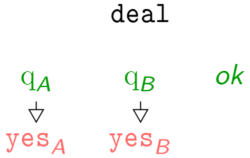
Games

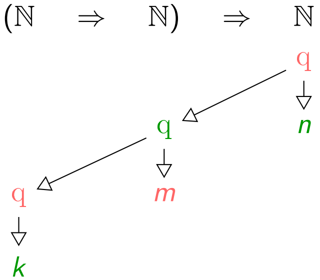
Game: partial order where each *move* has a polarity (*Opponent*, *Player*).



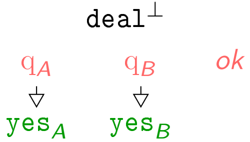
Games

Game: partial order where each *move* has a polarity (*Opponent*, *Player*).

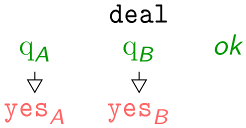




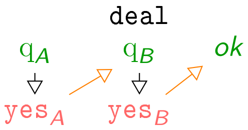
From a game A we build its dual $A^\perp$ by reversing polarities:



(Deterministic) Strategies

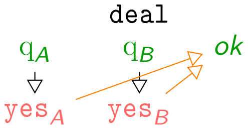


(Deterministic) Strategies



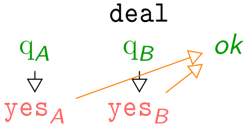
$\sigma_{A;B}$: Albert *then* Barnabé

(Deterministic) Strategies



$\sigma_{A\|B}$: Albert *and* Barnabé *in parallel*

(Deterministic) Strategies



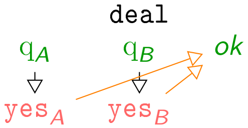
$\sigma_{A\|B}$: Albert *and* Barnabé *in parallel*

Definition

A **strategy** on $(A, \leq_A)$ is a partial order $\sigma = (S, \leq_S)$ such that

- $S \subseteq A$, and if $s \leq_A s'$ then $s \leq_S s'$ (rule-respecting)
- S only adds immediate links $\ominus \rightarrow \oplus$ (courteous)

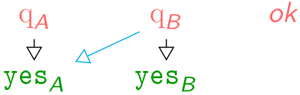
(Deterministic) Strategies



$\sigma_{A\|B}$: Albert *and* Barnabé *in parallel*

Strategies on $A^\perp$ represent **counter-strategies**:

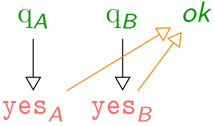
τ : $\mathtt{deal}^\perp$

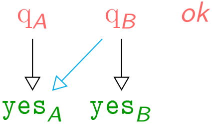


Interaction of strategies

Given a strategy on A and one on $A^\perp$, how do they interact?

$\sigma_{A\|B} : \texttt{deal}$ $\qquad$ $\sigma_{A\|B} \circledast \tau$ $\qquad$ $\tau : \texttt{deal}^\perp$

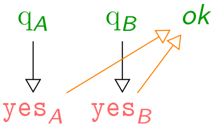




Interaction of strategies

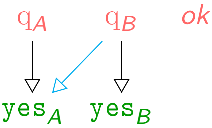
Given a strategy on A and one on $A^\perp$, how do they interact?

$\sigma_{A\|B}$: deal



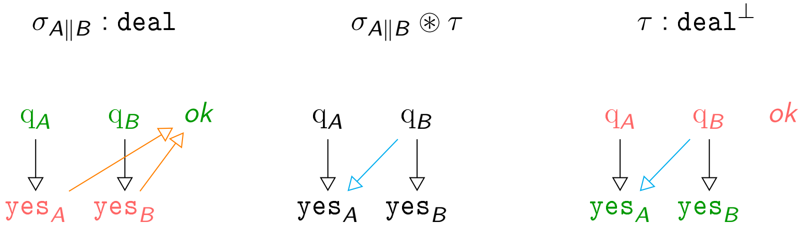
$\sigma_{A\|B} \circledast \tau$

τ : deal$^\perp$



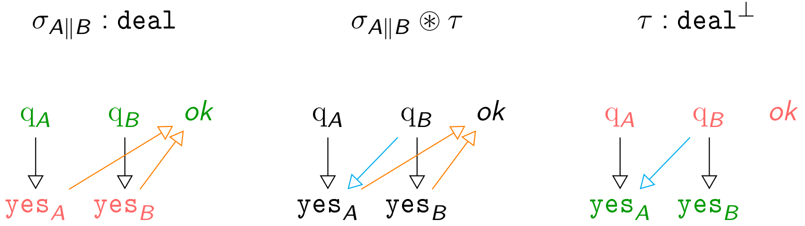
Interaction of strategies

Given a strategy on A and one on $A^\perp$, how do they interact?



Interaction of strategies

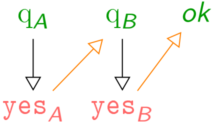
Given a strategy on A and one on $A^\perp$, how do they interact?

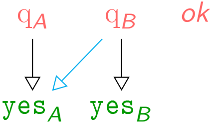


Interaction of strategies

Given a strategy on A and one on $A^\perp$, how do they interact?

$\sigma_{A;B}$: deal $\qquad\qquad \sigma_{A;B} \circledast \tau \qquad\qquad \tau$: $\mathtt{deal}^\perp$

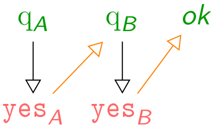


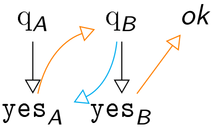


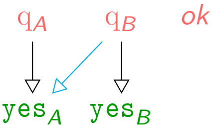
Interaction of strategies

Given a strategy on A and one on $A^\perp$, how do they interact?

$\sigma_{A;B} : \texttt{deal}$ $\quad$ $\sigma_{A;B} \circledast \tau$ $\quad$ $\tau : \texttt{deal}^\perp$



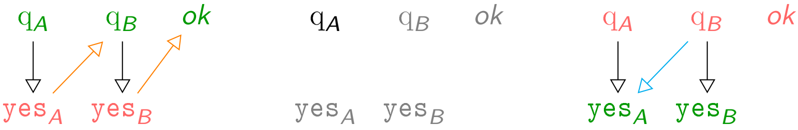




Interaction of strategies

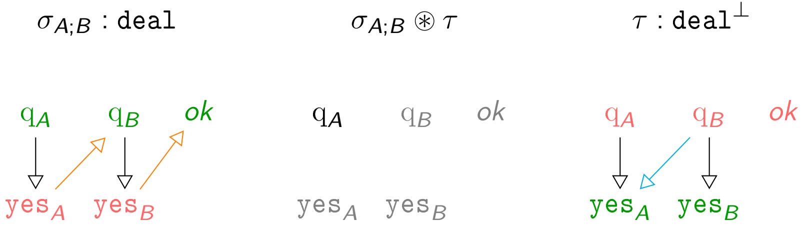
Given a strategy on A and one on $A^{\perp}$, how do they interact?

$\sigma_{A;B}$: deal $\qquad \sigma_{A;B} \circledast \tau \qquad \tau$: $\mathtt{deal}^{\perp}$



Interaction of strategies

Given a strategy on A and one on $A^\perp$, how do they interact?



Definition

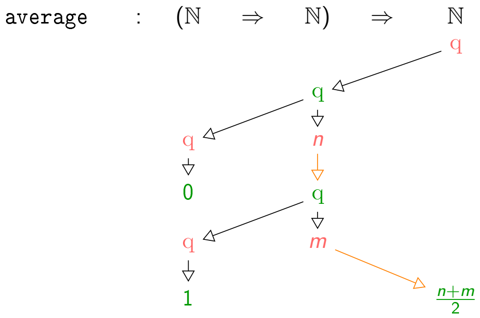
The interaction of $(S, \leq_S)$ and $(T, \leq_T)$ is obtained from

$$(S \cap T, (\leq_S \cup \leq_T)^*)$$

by removing events occurring in a causal loop.

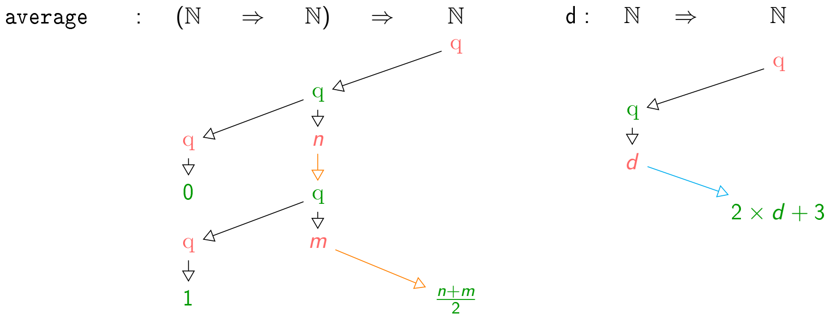
Composition

Interaction is used to compute application and composition of strat.



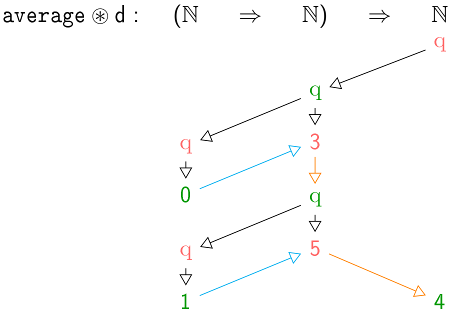
Composition

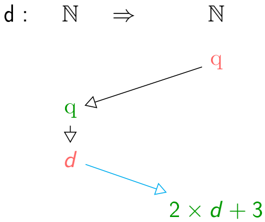
Interaction is used to compute application and composition of strat.



Composition

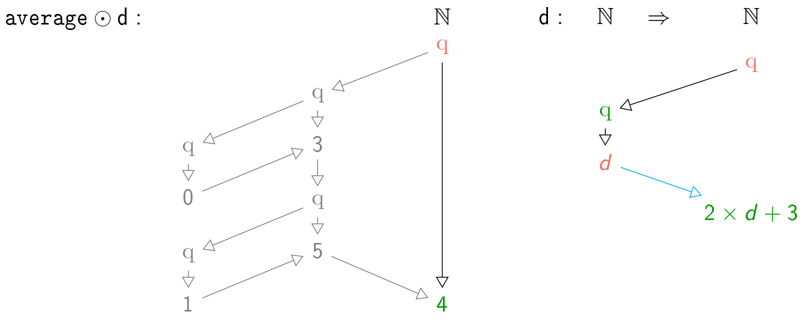
Interaction is used to compute application and composition of strat.





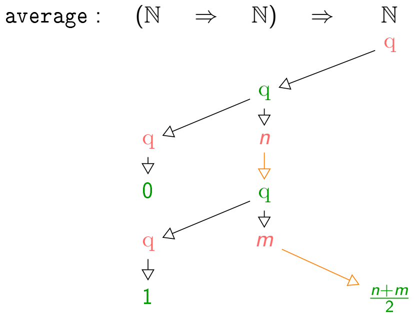
Composition

Interaction is used to compute application and composition of strat.



Linearity & duplication

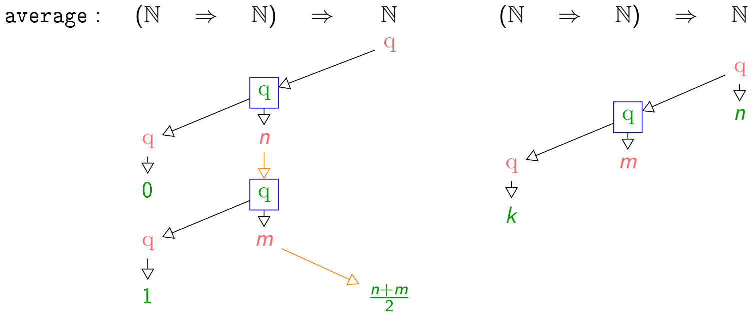
Our average strategy is **not** a strategy!



Linearity & duplication

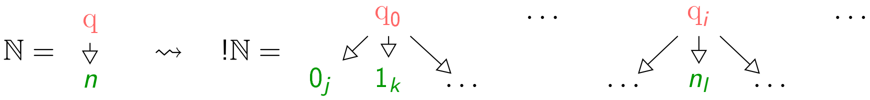
Our average strategy is **not** a strategy! It is not linear:

$$\texttt{average}(f) = \frac{f(0) + f(1)}{2}$$

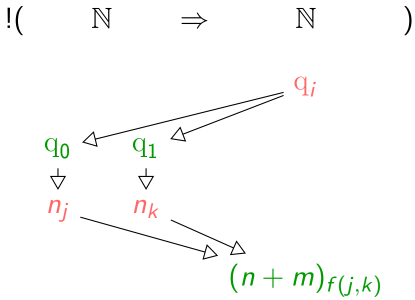


Copy indices & the game $!A$

To solve this problem, we play on expanded arenas.

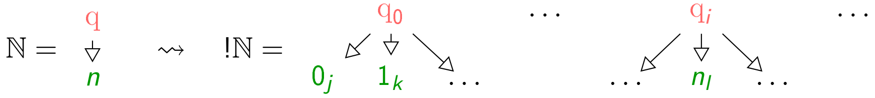


A (parallel) strategy implementing $d(x) := x + x$ becomes:

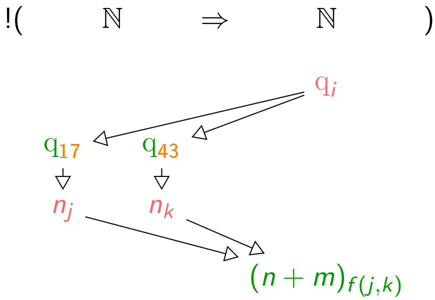


Copy indices & the game $!A$

To solve this problem, we play on expanded arenas.



A (parallel) strategy implementing $d(x) := x + x$ becomes:



The cartesian-closed category CHO

We get a cartesian closed category (CCC):

Theorem (C., Clairambault, Winskel)

The following structure CHO is a CCC:

- Objects *Games A (which are alternating forests)*
- Morphisms *Strategies uniform (and single-threaded) on* $!(A \Rightarrow B)$.

(Usual sequential innocent HO games form a subcategory of CHO)

$\rightsquigarrow$ Rich semantic framework to interpret concurrent higher-order programs.

What was swept under the rug

- Manage to identify strategies up to **choice of copy indices**.

  $\rightsquigarrow$ A notion of **weak isomorphism**

- Define a notion of **uniformity** (when a strategy is blind to Opponent indices).

  $\rightsquigarrow$ Strategies become equipped with **symmetry**.

- Show that, on uniform strategies, weak isomorphism is a **congruence**.

  $\rightsquigarrow$ Proof of a **bipullback property** of interaction.

What was swept under the rug

- Manage to identify strategies up to **choice of copy indices**.

  $\rightsquigarrow$ A notion of **weak isomorphism**

- Define a notion of **uniformity** (when a strategy is blind to Opponent indices).

  $\rightsquigarrow$ Strategies become equipped with **symmetry**.

- Show that, on uniform strategies, weak isomorphism is a **congruence**.

  $\rightsquigarrow$ Proof of a **bipullback property** of interaction.

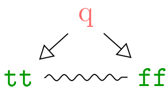
- Representation of **nondeterminism** in strategies

  $\rightsquigarrow$ Addition of **essential events**.

Nondeterminism via event structures

Nondeterminism can be represented via a conflict relation:

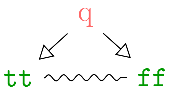
`coin` : $\mathbb{B}$



Nondeterminism via event structures

Nondeterminism can be represented via a conflict relation:

coin : $\mathbb{B}$

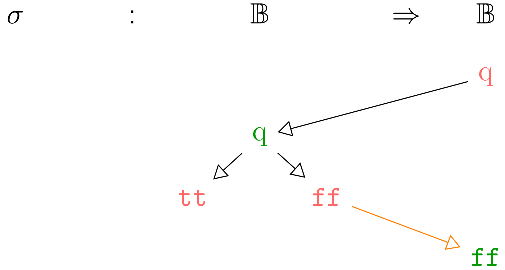


Definition

An **event structure** is a partial order E equipped with a binary relation (representing conflict) satisfying some axioms.

Hidden divergences via essential events

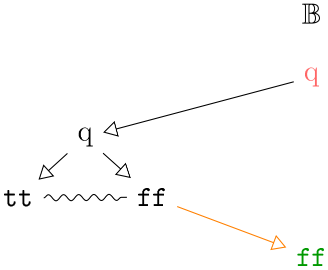
But, nondeterminism and composition require some care:



Hidden divergences via essential events

But, nondeterminism and composition require some care:

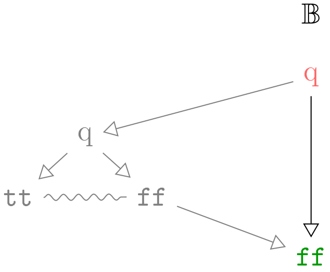
$\sigma \circledast \texttt{coin}$:



Hidden divergences via essential events

But, nondeterminism and composition require some care:

$\sigma \odot \texttt{coin}$:

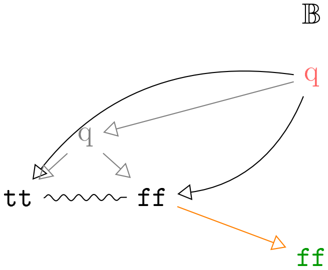


$\sigma \odot \texttt{coin}$ coincides with $\texttt{ff}$!

Hidden divergences via essential events

But, nondeterminism and composition require some care:

$\sigma \circledcirc$ `coin` :



$\sigma \odot$ `coin` coincides with `ff` !

$\tau \circledcirc \sigma$ retains more information than $\tau \odot \sigma$ (must adequacy)

II. Order of evaluation and innocence

PCF and its interpretations

We can interpret PCF in CHO:

$$A, B ::= \mathbb{N} \mid \mathbb{B} \mid \texttt{proc} \mid A \Rightarrow B \qquad \text{(PCF types)}$$

$$M, N ::= \texttt{tt} \mid \texttt{ff} \mid \texttt{if}\ M\ N_1\ N_2 \mid () \mid M; N$$

$$\mid x \mid \lambda x.\, M \mid M\, N \mid Y \mid \dots \qquad \text{(PCF terms)}$$

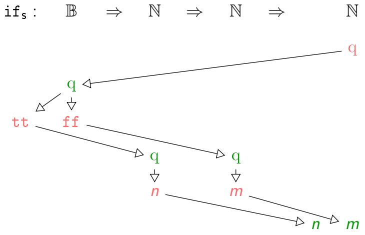
PCF and its interpretations

We can interpret PCF in CHO:

$$A, B ::= \mathbb{N} \mid \mathbb{B} \mid \texttt{proc} \mid A \Rightarrow B \qquad \text{(PCF types)}$$

$$M, N ::= \texttt{tt} \mid \texttt{ff} \mid \texttt{if}\ M\ N_1\ N_2 \mid () \mid M; N$$
$$\mid x \mid \lambda x.\, M \mid M\ N \mid Y \mid \ldots \qquad \text{(PCF terms)}$$

Two interpretations of `if` in CHO:



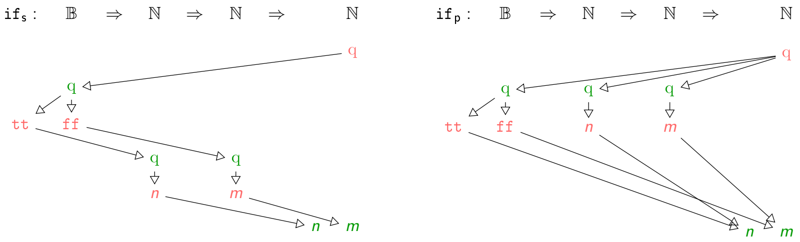
PCF and its interpretations

We can interpret PCF in CHO:

$$A, B ::= \mathbb{N} \mid \mathbb{B} \mid \texttt{proc} \mid A \Rightarrow B \qquad \text{(PCF types)}$$

$$M, N ::= \texttt{tt} \mid \texttt{ff} \mid \texttt{if}\ M\ N_1\ N_2 \mid () \mid M; N$$
$$\mid x \mid \lambda x.\, M \mid M\, N \mid Y \mid \dots \qquad \text{(PCF terms)}$$

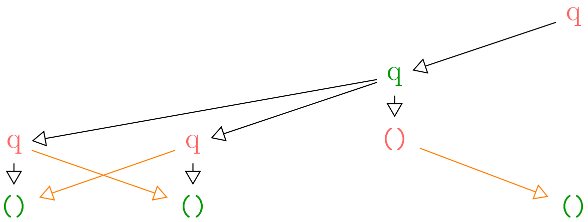
Two interpretations of `if` in CHO:



These two interpretations are indistinguishable by terms of PCF.

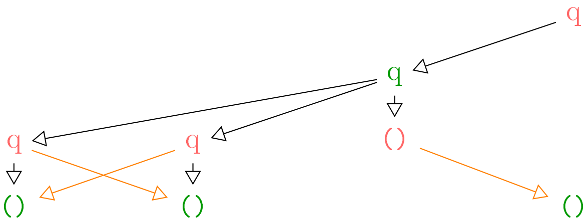
Sometimes `parallel` if just is not the same

$\texttt{sync}: \quad (\texttt{proc} \Rightarrow \texttt{proc} \Rightarrow \texttt{proc}) \Rightarrow \texttt{proc}$



Sometimes parallel `if` just is not the same

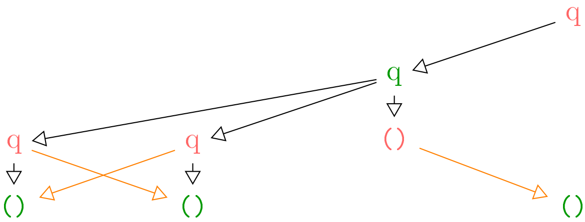
sync : (proc ⇒ proc ⇒ proc) ⇒ proc



sync can distinguish both implementations of `if`.

Sometimes parallel if just is not the same

sync : (proc ⇒ proc ⇒ proc) ⇒ proc



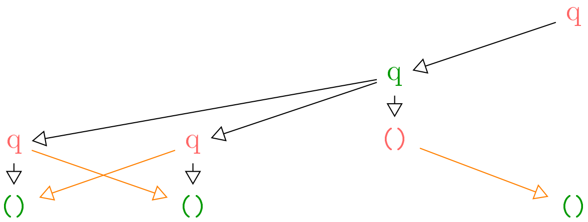
sync can distinguish both implementations of if.

(For $M_\epsilon = \lambda x \lambda y.\mathrm{if}_\epsilon\,(x;\mathtt{tt})\,(y;\mathtt{tt})\,\mathtt{tt}$:

sync M_p converges but not sync M_s.)

Sometimes parallel if just is not the same

sync : (proc ⇒ proc ⇒ proc) ⇒ proc



sync can distinguish both implementations of if.

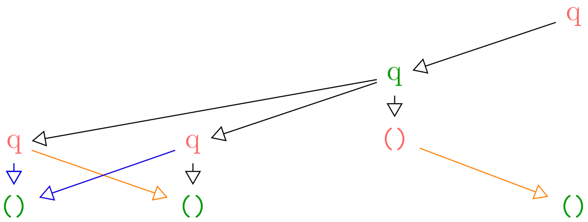
(For $M_\epsilon = \lambda x \lambda y.\mathrm{if}_\epsilon\,(x; \mathtt{tt})\,(y; \mathtt{tt})\,\mathtt{tt}$:

sync M_p converges but not sync M_s.)

What is wrong?

Sometimes parallel `if` just is not the same

$\texttt{sync} : \quad (\texttt{proc} \quad \Rightarrow \quad \texttt{proc} \quad \Rightarrow \quad \texttt{proc}) \quad \Rightarrow \quad \texttt{proc}$



`sync` can distinguish both implementations of `if`.

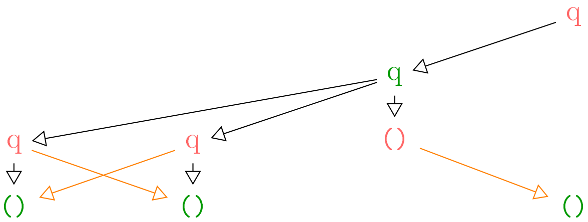
(For $M_\epsilon = \lambda x \lambda y.\mathrm{if}_\epsilon\,(x;\texttt{tt})\,(y;\texttt{tt})\,\texttt{tt}$:

$\texttt{sync}\ M_p$ converges but not $\texttt{sync}\ M_s$.)

What is wrong? Player merges two threads started by Opponent.

Sometimes parallel `if` just is not the same

$$\texttt{sync}: \quad (\texttt{proc} \quad \Rightarrow \quad \texttt{proc} \quad \Rightarrow \quad \texttt{proc}) \quad \Rightarrow \quad \texttt{proc}$$



`sync` can distinguish both implementations of `if`.

(For $M_\epsilon = \lambda x \lambda y.\mathrm{if}_\epsilon\,(x; \texttt{tt})\,(y; \texttt{tt})\,\texttt{tt}$:

$\texttt{sync}\; M_p$ converges but not $\texttt{sync}\; M_s$.)

What is wrong? Player merges two threads started by Opponent.

Banning such patterns gives a notion of **concurrent innocence**.

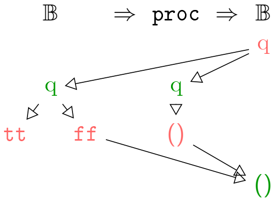
Finite definability

A **PCF strategy** is an innocent and well-bracketed strategy.

Theorem (Finite definability)

If σ is PCF strategy, there exists a term M of PCF such that

$$[\![M]\!] \text{ and } \sigma \text{ are indistinguishable by PCF strategies.}$$



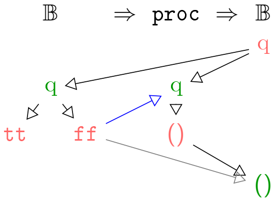
Finite definability

A **PCF strategy** is an innocent and well-bracketed strategy.

Theorem (Finite definability)

If σ is PCF strategy, there exists a term M of PCF such that

$$[\![M]\!] \text{ and } \sigma \text{ are indistinguishable by PCF strategies.}$$



$\rightsquigarrow \qquad \lambda b \lambda x.\, \mathrm{if}\, b \perp x$

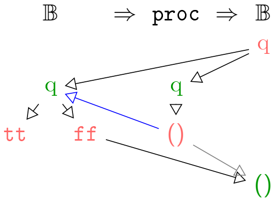
Finite definability

A **PCF strategy** is an innocent and well-bracketed strategy.

Theorem (Finite definability)

If σ is PCF strategy, there exists a term M of PCF such that

$$[\![M]\!] \text{ and } \sigma \text{ are indistinguishable by PCF strategies.}$$



$\rightsquigarrow \quad \lambda b \lambda x.\, x;\ \text{if}\ b \perp ()$

What was swept under the rug 2

- Innocence and well-bracketing:
  - Stability under **composition**.
    $\rightsquigarrow$ **Forking** lemma.
  - Requires the addition of **visibility** (and **locality**).
    $\rightsquigarrow$ Control **interferences** between Player threads

What was swept under the rug 2

- Innocence and well-bracketing:
  - Stability under **composition**.
    $\rightsquigarrow$ **Forking** lemma.
  - Requires the addition of **visibility** (and **locality**).
    $\rightsquigarrow$ Control **interferences** between Player threads
- Finite definability:
  - Define a notion of **finite** strategies.
    $\rightsquigarrow$ **Reduced form** (P-view "dag")
  - **Factorisation** theorem for higher-order strategies.
    $\rightsquigarrow$ First-order / λ-calculus

III. What lies beyond PCF?

What real concurrent programs are made of

```
while(1) {                                  ||  while(1) {
  while (canWrite == 0) ;                   ||    while (canRead == 0) ;
  x = produce();                            ||    x = value;
  value := x;                               ||    consume(x);
  canRead := 1; canWrite := 0;              ||    canRead := 0; canWrite := 1;
}                                           ||  }
```

$\rightsquigarrow$ loops, conditionals, function calls, shared memory.

What real concurrent programs are made of

```
while(1) {                                  ||  while(1) {
  while (canWrite == 0) ;                   ||    while (canRead == 0) ;
  x = produce();                            ||    x = value;
  value := x;                               ||    consume(x);
  canRead := 1; canWrite := 0;              ||    canRead := 0; canWrite := 1;
}                                           ||  }
```

$\rightsquigarrow$ $\underbrace{\text{loops, conditionals, function calls}}_{\text{in PCF}}$, shared memory.

What real concurrent programs are made of

$$\begin{array}{l} \texttt{value} := 1; \\ \texttt{canRead} := 1; \end{array} \Bigg\| \begin{array}{l} f \leftarrow \texttt{canRead}; \\ x \leftarrow \texttt{value}\ ; \end{array}$$

$\rightsquigarrow$ loops, conditionals, function calls, shared memory

What real concurrent programs are made of

$$\begin{array}{l} \texttt{value} := 1; \\ \texttt{canRead} := 1; \end{array} \;\Big\|\; \begin{array}{l} f \leftarrow \texttt{canRead}; \\ x \leftarrow \texttt{value}\ ; \end{array}$$

Expectation: $f = 1$ implies $x = 1$

$\rightsquigarrow$ loops, conditionals, function calls, shared memory

Running concurrent programs on a processor

```
value := 1;
canRead := 1;
```

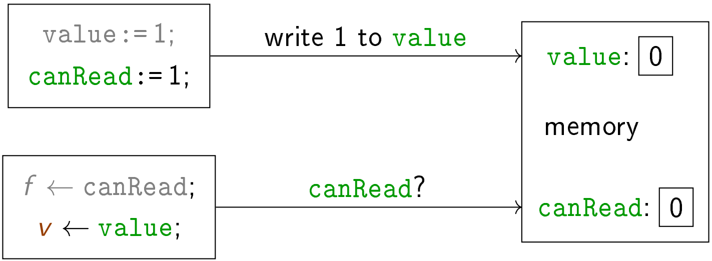
$f \leftarrow$ `canRead`;
$v \leftarrow$ `value`;

`value`: 0

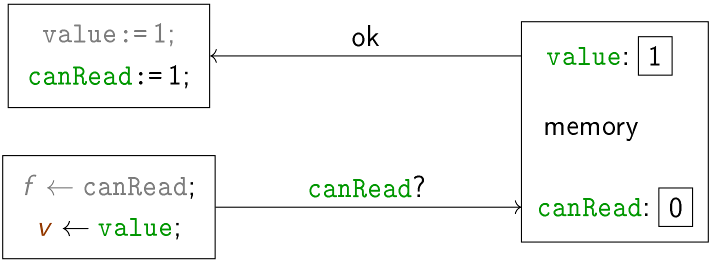
memory

`canRead`: 0

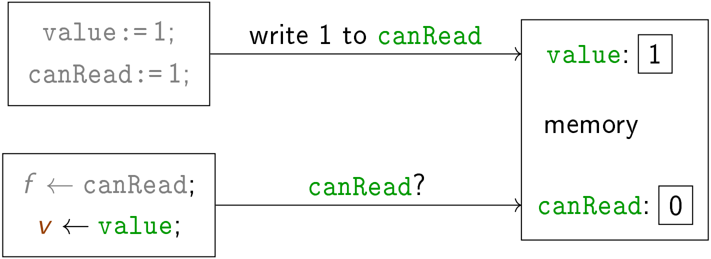
Running concurrent programs on a processor



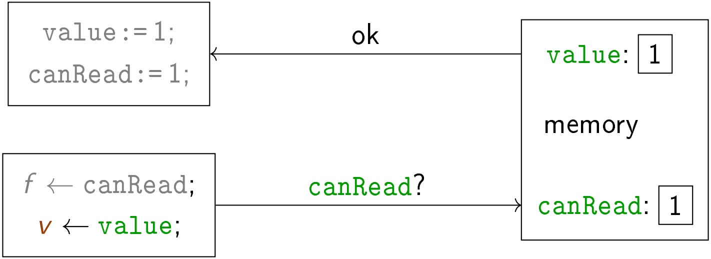
Running concurrent programs on a processor



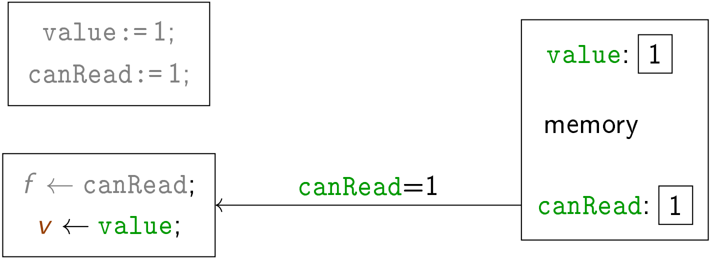
Running concurrent programs on a processor



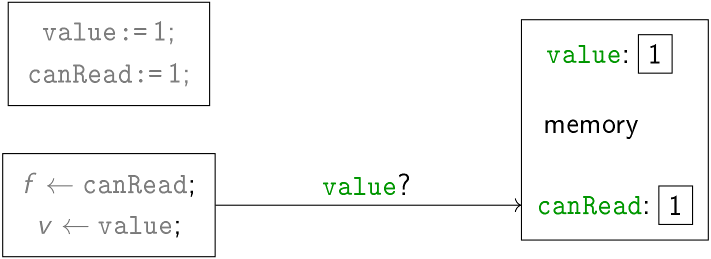
Running concurrent programs on a processor



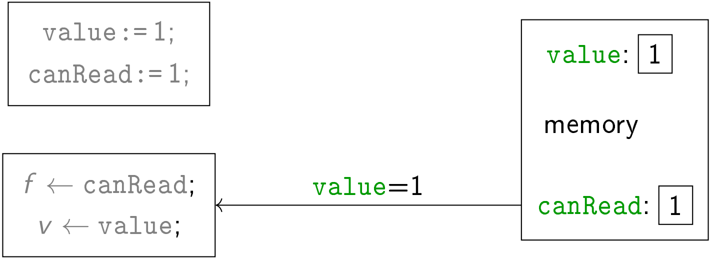
Running concurrent programs on a processor



Running concurrent programs on a processor



Running concurrent programs on a processor



Running concurrent programs on a processor

```
value := 1;
canRead := 1;
```

$f \leftarrow$ `canRead`;
$v \leftarrow$ `value`;

`value`: 1

memory

`canRead`: 1

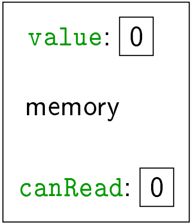
Several executions, but never $f = 1$ and $v = 0$.

Weak memory models

On some processors, we get instead:

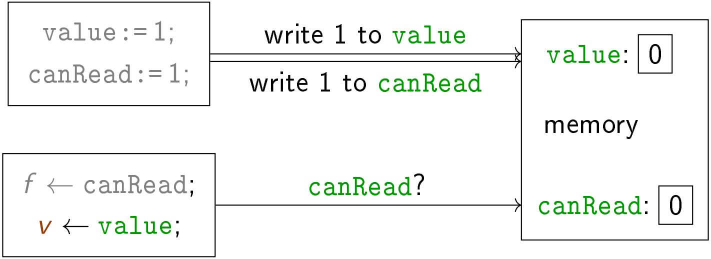
```
value := 1;
canRead := 1;
```

```
f ← canRead;
v ← value;
```



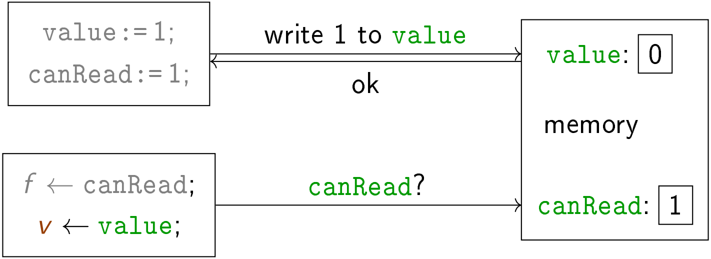
Weak memory models

On some processors, we get instead:



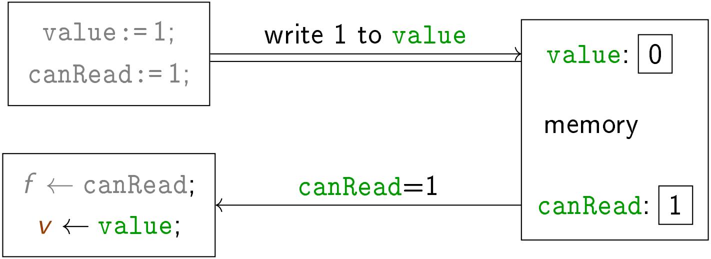
Weak memory models

On some processors, we get instead:



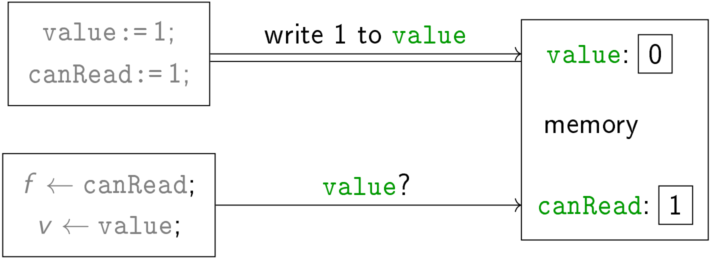
Weak memory models

On some processors, we get instead:



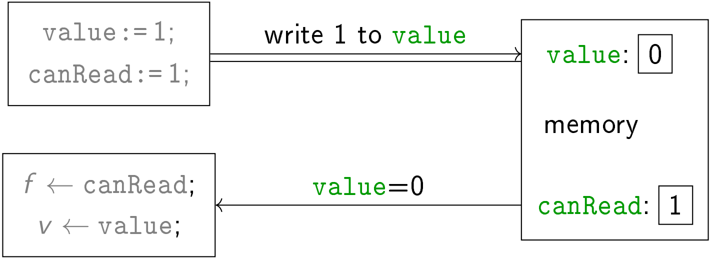
Weak memory models

On some processors, we get instead:



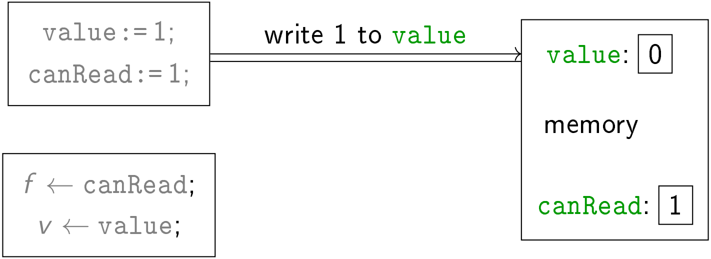
Weak memory models

On some processors, we get instead:



Weak memory models

On some processors, we get instead:



Weak memory models

On some processors, we get instead:

Weak memory models

On some processors, we get instead:

Outcome depends on the architecture (TSO, PSO, ARM)...

Weak memory models

On some processors, we get instead:

Outcome depends on the architecture (TSO, PSO, ARM)...

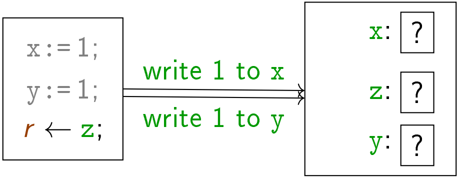
Goal: Model *denotationally* such **complex** reorderings.

Taking a closer look

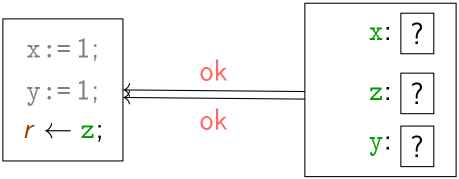
```
x := 1;
y := 1;
r ← z;
```

x:	?
z:	?
y:	?

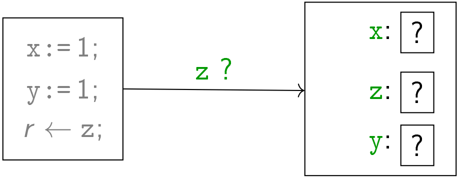
Taking a closer look



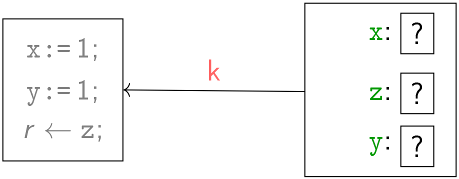
Taking a closer look



Taking a closer look

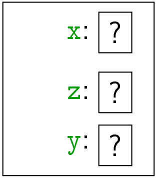


Taking a closer look

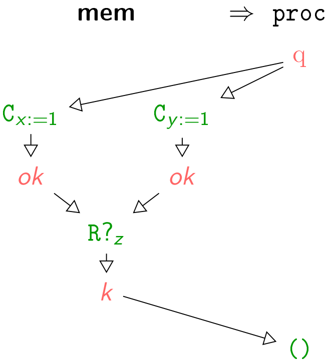


Taking a closer look

```
x := 1;
y := 1;
r ← z;
```



The behaviour of the thread corresponds to a strategy:



Modelling weak memory models

An old motto (Reynolds, implemented in game semantics by Abramsky and McCusker)

The behaviour of imperative programs can be described as the interaction of a (deterministic) pure functional program interacting with a memory.

Modelling weak memory models

An old motto (Reynolds, implemented in game semantics by Abramsky and McCusker)

The behaviour of concurrent programs can be described as the interaction of (deterministic) pure functional threads interacting with a (nondeterministic) memory.

Modelling weak memory models

An old motto (Reynolds, implemented in game semantics by Abramsky and McCusker)

The behaviour of concurrent programs can be described as the interaction of (deterministic) pure functional threads interacting with a (nondeterministic) memory.

Individual programs are interpreted as **parallel** strategies

$$[\![t]\!] : \mathbf{mem} \Rightarrow \texttt{proc}$$

Modelling weak memory models

An old motto (Reynolds, implemented in game semantics by Abramsky and McCusker)

The behaviour of concurrent programs can be described as the interaction of (deterministic) pure functional threads interacting with a (nondeterministic) memory.

Individual programs are interpreted as **parallel** strategies

$$[\![t]\!] : \mathbf{mem} \Rightarrow \mathtt{proc}$$

In particular

$$[\![t; u]\!] = \mathtt{seq}\ [\![t]\!]\ [\![u]\!]$$

$$\mathtt{seq} : (\mathbf{mem} \Rightarrow \mathtt{proc}) \Rightarrow (\mathbf{mem} \Rightarrow \mathtt{proc}) \Rightarrow (\mathbf{mem} \Rightarrow \mathtt{proc})$$

Modelling weak memory models

An old motto (Reynolds, implemented in game semantics by Abramsky and McCusker)

The behaviour of concurrent programs can be described as the interaction of (deterministic) pure functional threads interacting with a (nondeterministic) memory.

Individual programs are interpreted as **parallel** strategies

$$[\![t]\!] : \mathbf{mem} \Rightarrow \mathtt{proc}$$

In particular

$$[\![t; u]\!] = \mathtt{seq}\, [\![t]\!]\, [\![u]\!]$$

$$\mathtt{seq} : \mathbf{mem} \Rightarrow (\mathbf{mem} \Rightarrow \mathtt{proc}) \Rightarrow (\mathbf{mem} \Rightarrow \mathtt{proc}) \Rightarrow \mathtt{proc}$$

Thread semantics via game semantics

Implementation of seq (here for PSO):

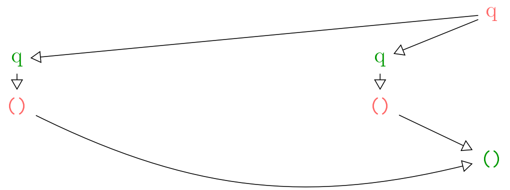
$$\mathbf{mem} \Rightarrow (\mathbf{mem} \Rightarrow \texttt{proc}) \Rightarrow (\quad \mathbf{mem} \quad \Rightarrow \texttt{proc}) \Rightarrow \texttt{proc}$$

q

Thread semantics via game semantics

Implementation of seq (here for PSO):

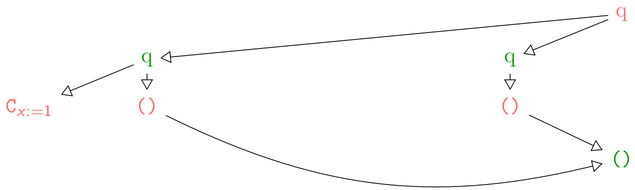
$\mathbf{mem} \Rightarrow (\mathbf{mem} \Rightarrow \texttt{proc}) \Rightarrow (\mathbf{mem} \Rightarrow \texttt{proc}) \Rightarrow \texttt{proc}$



Thread semantics via game semantics

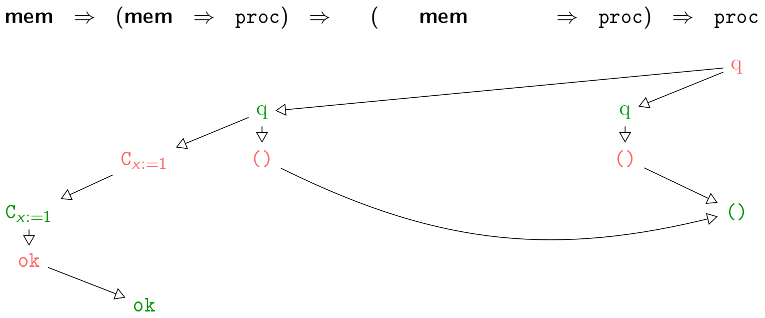
Implementation of seq (here for PSO):

$\mathbf{mem} \Rightarrow (\mathbf{mem} \Rightarrow \mathtt{proc}) \Rightarrow (\mathbf{mem} \Rightarrow \mathtt{proc}) \Rightarrow \mathtt{proc}$



Thread semantics via game semantics

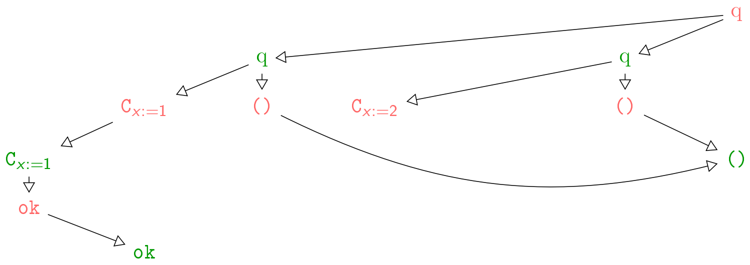
Implementation of seq (here for PSO):



Thread semantics via game semantics

Implementation of seq (here for PSO):

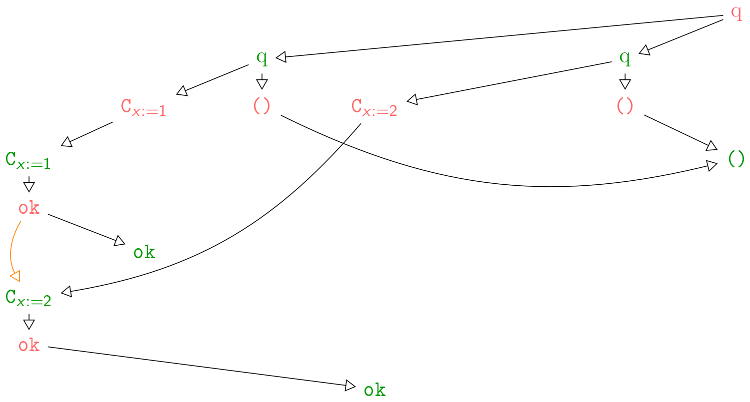
$\mathbf{mem} \Rightarrow (\mathbf{mem} \Rightarrow \texttt{proc}) \Rightarrow (\mathbf{mem} \Rightarrow \texttt{proc}) \Rightarrow \texttt{proc}$



Thread semantics via game semantics

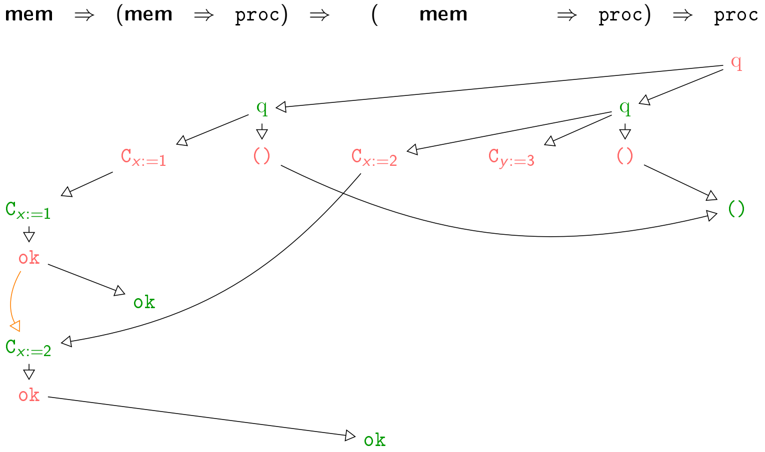
Implementation of seq (here for PSO):

mem $\Rightarrow$ (**mem** $\Rightarrow$ `proc`) $\Rightarrow$ (**mem** $\Rightarrow$ `proc`) $\Rightarrow$ `proc`



Thread semantics via game semantics

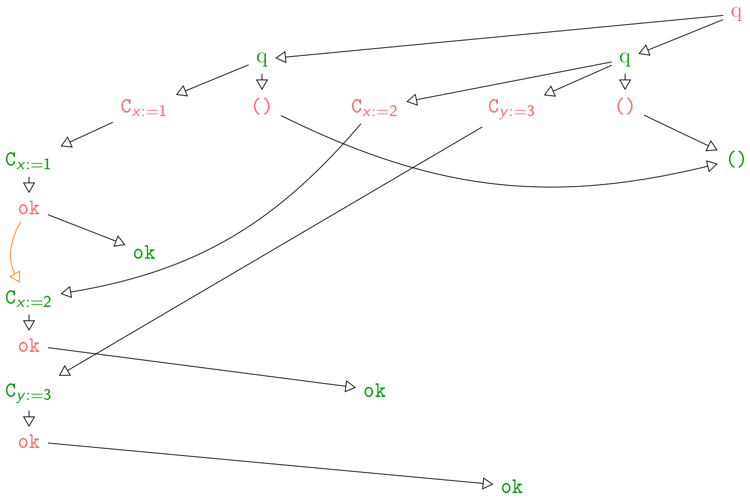
Implementation of seq (here for PSO):



Thread semantics via game semantics

Implementation of seq (here for PSO):

$\mathbf{mem} \Rightarrow (\mathbf{mem} \Rightarrow \mathtt{proc}) \Rightarrow (\mathbf{mem} \Rightarrow \mathtt{proc}) \Rightarrow \mathtt{proc}$



Final model

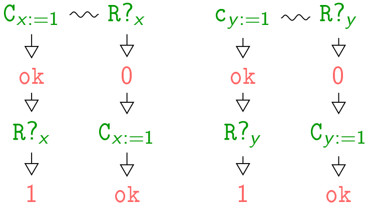
Depends on the architecture $\mathcal{A}$:

- Thread operations: $\texttt{seq}_{\mathcal{A}}$, $\texttt{read}_{\mathcal{A}}$, $\texttt{write}_{\mathcal{A}}$ implement the reorderings specific to $\mathcal{A}$.
- Memory (representing caches, barriers, ...) is represented by

$$\mathfrak{m}_{\mathcal{A}} : \mathbf{mem}$$

Executions on $\mathcal{A}$ of $t_1 \parallel \ldots \parallel t_n$ are represented by the interaction:

$$(\llbracket t_1 \rrbracket_{\mathcal{A}} \parallel \ldots \parallel \llbracket t_n \rrbracket_{\mathcal{A}}) \circledast \mathfrak{m}_{\mathcal{A}}.$$



Final model

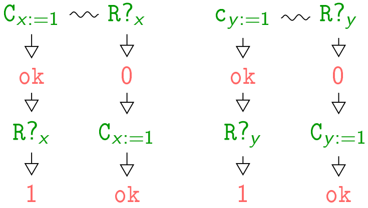
Depends on the architecture $\mathcal{A}$:

- Thread operations: $\texttt{seq}_{\mathcal{A}}$, $\texttt{read}_{\mathcal{A}}$, $\texttt{write}_{\mathcal{A}}$ implement the reorderings specific to $\mathcal{A}$.
- Memory (representing caches, barriers, ...) is represented by

$$\mathfrak{m}_{\mathcal{A}} : \textbf{mem}$$

Executions on $\mathcal{A}$ of $t_1 \parallel \ldots \parallel t_n$ are represented by the interaction:

$$(\llbracket t_1 \rrbracket_{\mathcal{A}} \parallel \ldots \parallel \llbracket t_n \rrbracket_{\mathcal{A}}) \circledast \mathfrak{m}_{\mathcal{A}}.$$



Theorem

Traces generated by this model correspond to operational traces (on TSO).

Final model

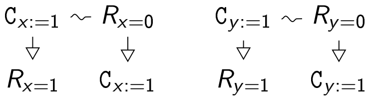
Depends on the architecture $\mathcal{A}$:

- Thread operations: $\texttt{seq}_{\mathcal{A}}$, $\texttt{read}_{\mathcal{A}}$, $\texttt{write}_{\mathcal{A}}$ implement the reorderings specific to $\mathcal{A}$.
- Memory (representing caches, barriers, ...) is represented by

$$\mathfrak{m}_{\mathcal{A}} : \mathbf{mem}$$

Executions on $\mathcal{A}$ of $t_1 \parallel \ldots \parallel t_n$ are represented by the interaction:

$$(\llbracket t_1 \rrbracket_{\mathcal{A}} \parallel \ldots \parallel \llbracket t_n \rrbracket_{\mathcal{A}}) \circledast \mathfrak{m}_{\mathcal{A}}.$$



Theorem

Traces generated by this model correspond to operational traces (on TSO).

Perspectives

- Describe finitely non-innocent strategies.

  $\rightsquigarrow$ Represent efficiently operations on them.

- Which language corresponds to innocent concurrent strategies?

  $\rightsquigarrow$ Concurrent control operators (eg. `fork`).

- Weaker architectures (eg. ARM) and software specifications.

  $\rightsquigarrow$ How to handle speculation, complex barriers specification.