Two Sides of the Same Coin: Session Types and Game Semantics

Simon Castellan[1] Nobuko Yoshida[1]

[1]Imperial College London, UK

17th December 2018

Protocols: two formalisations

A simple FTP protocol:

1. Connect to Server and issue a request for a file
2. Server answers either `not found`, or a port for Client to download the file.
3. At any time Client can close the connection.

Protocols: two formalisations

A simple FTP protocol:

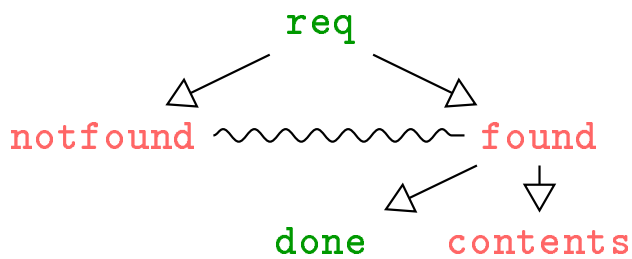
1. Connect to Server and issue a request for a file
2. Server answers either `not found`, or a port for Client to download the file.
3. At any time Client can close the connection.

	Session Types	**Game Semantics**
Prot.	$!\texttt{req}.\ \&\left\{\begin{array}{l}?\texttt{notfound};\\ ?\texttt{found}(?\texttt{content} : \texttt{String}).\\ \quad !\texttt{done}\end{array}\right.$	

Protocols: two formalisations

A simple FTP protocol:

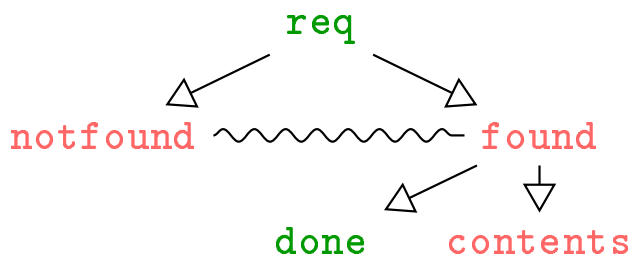
1. Connect to Server and issue a request for a file
2. Server answers either `not found`, or a port for Client to download the file.
3. At any time Client can close the connection.

	Session Types	**Game Semantics**
Prot.	$!\texttt{req}.\ \&\left\{\begin{array}{l}?\texttt{notfound};\\ ?\texttt{found}(?\texttt{content} : \texttt{String}).\\ \quad !\texttt{done}\end{array}\right.$	

Protocols: two formalisations

A simple FTP protocol:

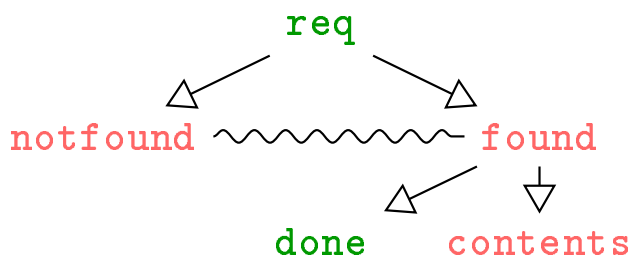
1. Connect to Server and issue a request for a file
2. Server answers either `not found`, or a port for Client to download the file.
3. At any time Client can close the connection.

	Session Types	**Game Semantics**
Prot.	$!\texttt{req}.\ \&\left\{\begin{array}{l}?\texttt{notfound};\\ ?\texttt{found}(?\texttt{content} : \texttt{String}).\ !\texttt{done}\end{array}\right.$	
Impl.	Processes	Strategies

Protocols: two formalisations

A simple FTP protocol:

1. Connect to Server and issue a request for a file
2. Server answers either `not found`, or a port for Client to download the file.
3. At any time Client can close the connection.

	Session Types	**Game Semantics**
Prot.	$!\texttt{req}.\ \&\left\{\begin{array}{l}?\texttt{notfound};\\ ?\texttt{found}(?\texttt{content} : \texttt{String}).\\ \quad !\texttt{done}\end{array}\right.$	
Impl.	Processes	Strategies

Striking similarity but two different uses:

- **Session Types**: analysis of message-passing concurrency.
- **Game Semantics**: study of higher-order programs.

Our contributions

Despite the similarity, a discrepancy:

- processes: **synchronous** strategies: **asynchronous**

$\rightsquigarrow$ The introduction of **coincident event structures**: a model of synchronous and concurrent computation.

Our **key contribution**: a correspondence

$$\text{session types} \leftrightarrow \text{games} \text{ recursive, forest-like}$$
$$\text{processes} \rightarrow \text{synchronous strategies}$$
$$\text{asyncronous processes} \leftrightarrow \text{asynchronous strategies} \text{ confusion-free}$$

The session π-calculus [Hon93]

$$
\begin{array}{rll}
S, T ::= & \mathsf{end} & \textbf{Channel types} \\
& \mid \bigoplus_{i \in I} \{ !\ell_i \langle \vec{S_i} \rangle . T_i \} & \textit{Output/Selection} \\
& \mid \&_{i \in I} \{ ?\ell_i (\vec{S_i}) . T_i \} & \textit{Input/Branching} \\
P, Q ::= & \mathbf{0} & \textbf{Processes} \\
& \mid (P \parallel Q) \mid (\nu a) P & \\
& \mid a!\ell \langle u_1, \ldots, u_n \rangle . P & \textit{Selection/Output on } a \\
& \mid a \,\&\, \{ ?\ell_i (\vec{x_i}) . P_i \} & \textit{Branching/Input on } a
\end{array}
$$

(The paper has recursive types and processes, as well as nondeterministic choices.)

Typing judgements. $P \triangleright a_1 : T_1, \ldots, a_n : T_n$.
$\rightsquigarrow$ Linear discipline ensures the absence of races.

Contextual equivalence. Barbed congruence: $P \approx Q$.

Game semantics: an introduction [HO00, AJM00]

Operational semantics of **higher-order open programs**.

Program

$f : \mathbb{N} \to \mathbb{N} \vdash f\,\mathsf{tt} : \mathbb{N}$

Context

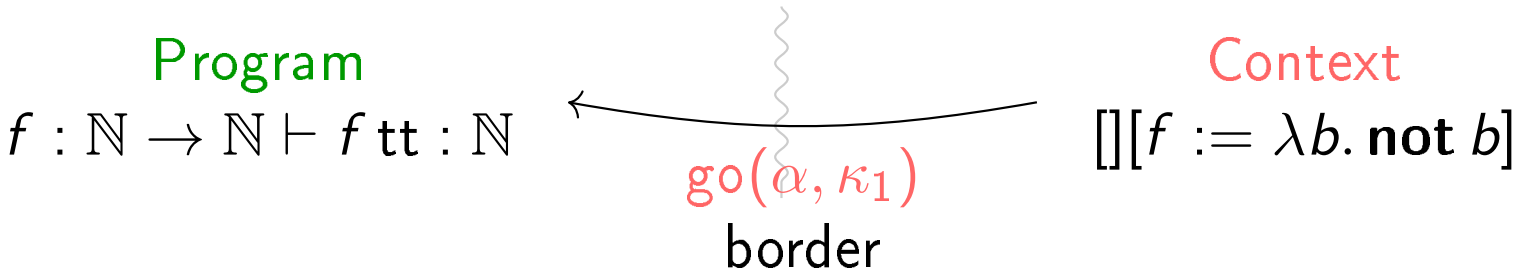
$[][f := \lambda b.\,\mathbf{not}\ b]$

border

$[\![f3]\!]$

Game semantics: an introduction [HO00, AJM00]

Operational semantics of **higher-order open programs**.

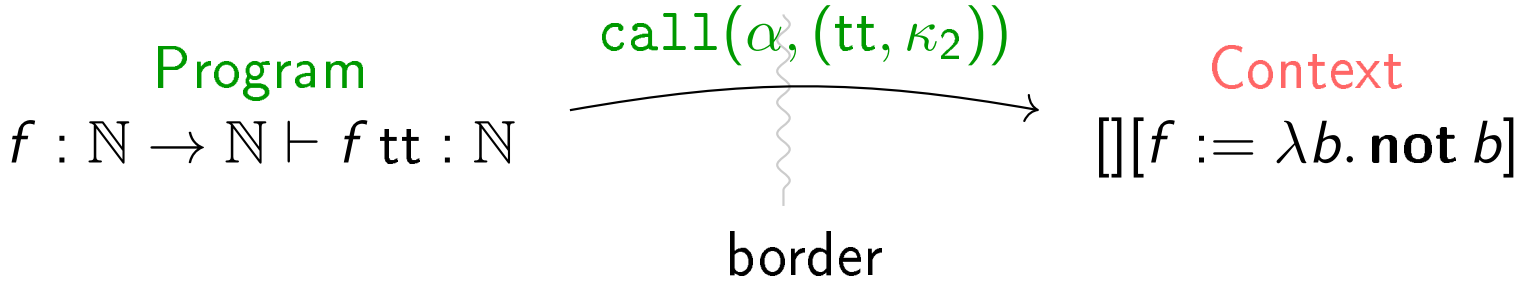


$\mathrm{go}(\alpha, \kappa_1)$

$[\![f3]\!] \ni$

Game semantics: an introduction [HO00, AJM00]

Operational semantics of **higher-order open programs**.



$[\![f3]\!] \ni$

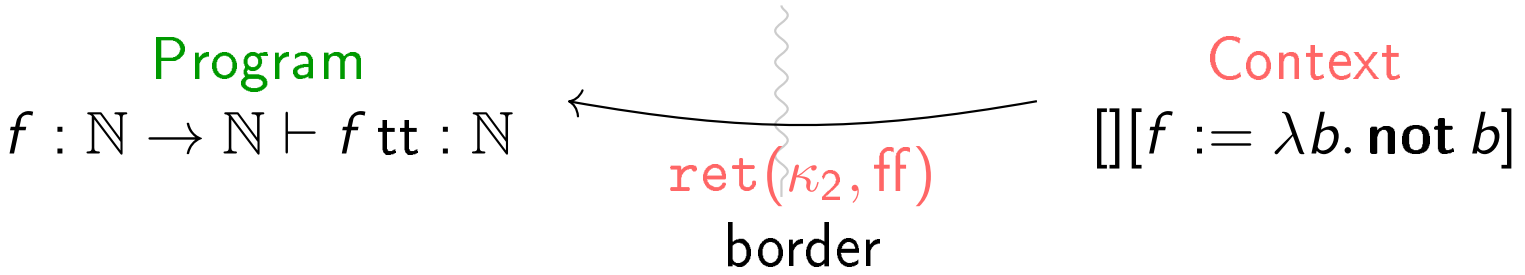
$\mathtt{go}(\alpha, \kappa_1)$

$\triangledown$

$\mathtt{call}(\alpha, (\mathsf{tt}, \kappa_2))$

Game semantics: an introduction [HO00, AJM00]

Operational semantics of **higher-order open programs**.



$[\![f3]\!] \ni$

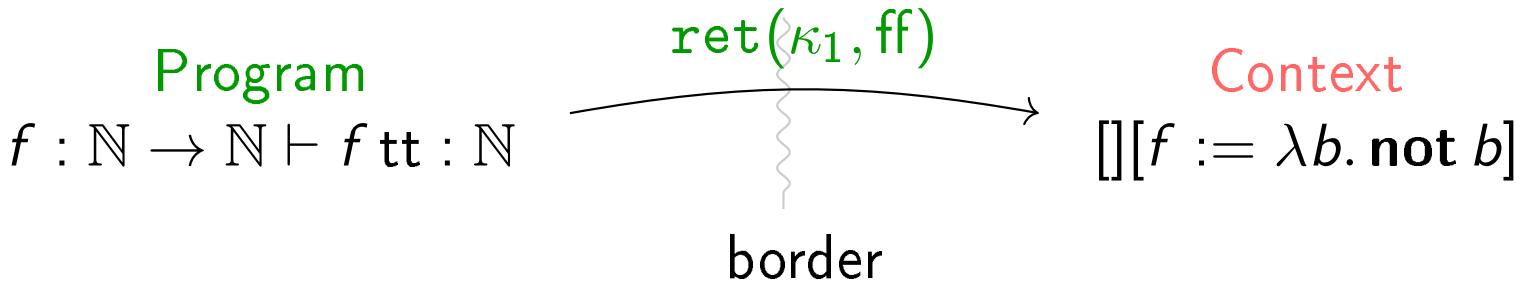
$\mathtt{go}(\alpha, \kappa_1)$

$\mathtt{call}(\alpha, (\mathsf{tt}, \kappa_2))$

$\mathtt{ret}(\kappa_2, \mathsf{ff})$

Game semantics: an introduction [HO00, AJM00]

Operational semantics of **higher-order open programs**.



$[\![f3]\!] \ni$

$\mathtt{go}(\alpha, \kappa_1)$

$\mathtt{call}(\alpha, (\mathsf{tt}, \kappa_2))$

$\mathtt{ret}(\kappa_2, \mathsf{ff})$

$\mathtt{ret}(\kappa_1, \mathsf{ff})$

Game semantics: an introduction [HO00, AJM00]

Operational semantics of **higher-order open programs**.

Program

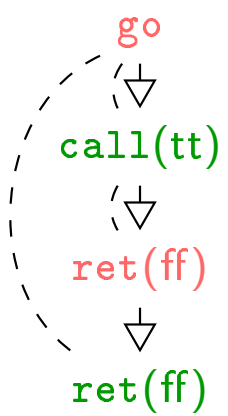
$f : \mathbb{N} \to \mathbb{N} \vdash f\,\mathsf{tt} : \mathbb{N}$

Context

$[][f := \lambda b.\,\mathbf{not}\ b]$

border

$[\![f3]\!] \ni$



Game semantics: an introduction [HO00, AJM00]

Operational semantics of **higher-order open programs**.

Program

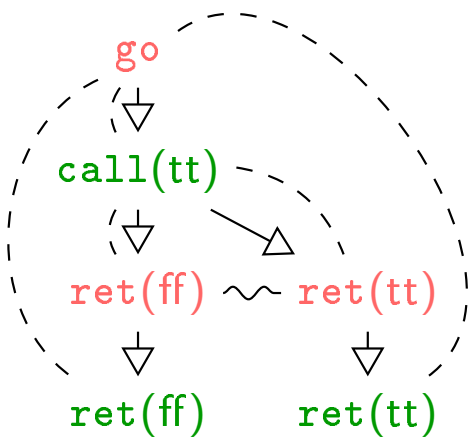
$f : \mathbb{N} \to \mathbb{N} \vdash f\,\mathsf{tt} : \mathbb{N}$

border

Context

$[][f := \lambda b.\,\mathbf{not}\ b]$

$[\![f3]\!] \ni$



Game semantics: an introduction [HO00, AJM00]

Operational semantics of **higher-order open programs**.

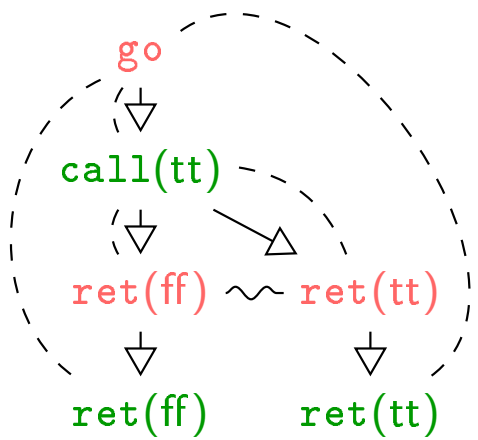
Program
$f : \mathbb{N} \to \mathbb{N} \vdash f\,\mathsf{tt} : \mathbb{N}$

Context
$[][f := \lambda b.\,\mathbf{not}\ b]$

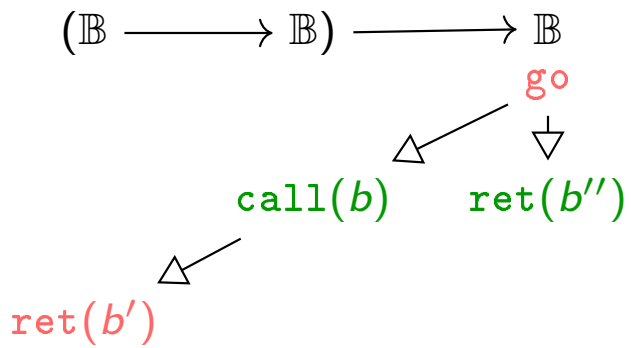
border

The **protocol** (depending on the border type) is a **game**.

$[\![f3]\!] =$



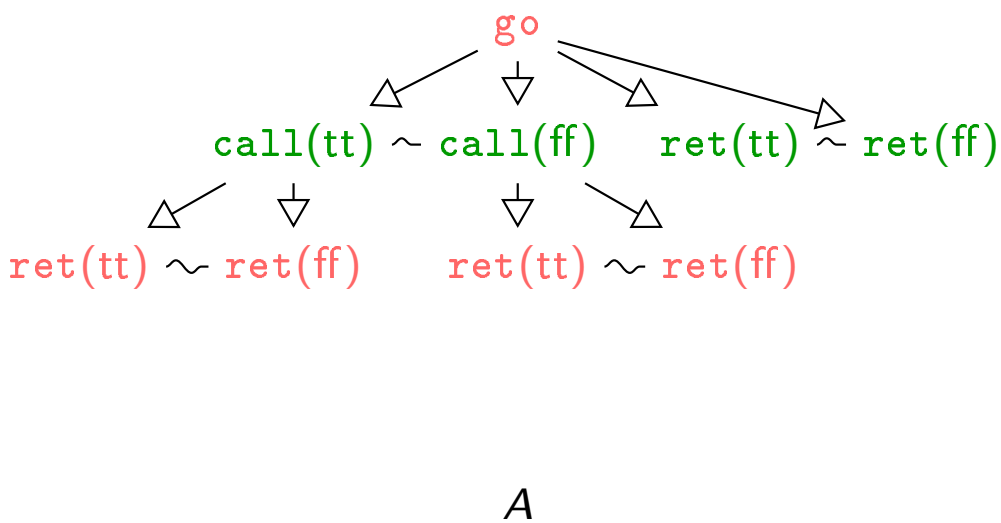
on



Game semantics on event structures [RW11]

Games: polarised event structures $(A, \leq_A, \frown, \lambda\colon A \to \{-,+\})$

- Causality ($\leq_A$) and conflict ($\frown$) represents **rules**.



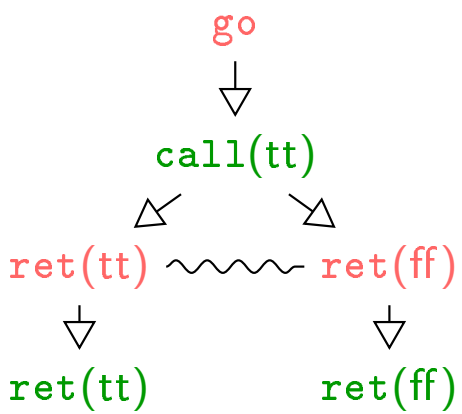
A

Game semantics on event structures [RW11]

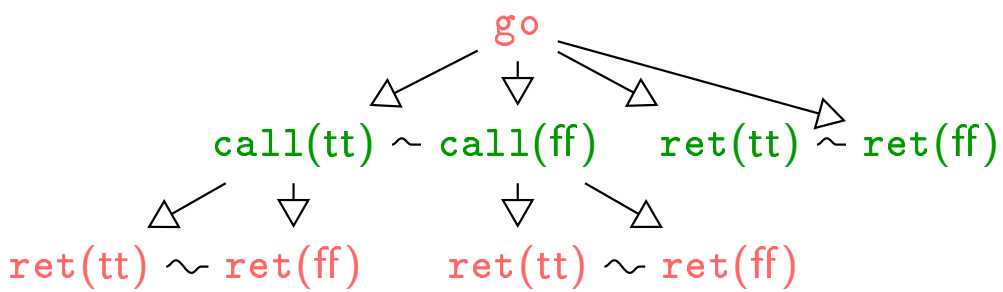
Games: polarised event structures $(A, \leq_A, \frown, \lambda\colon A \to \{-,+\})$

- Causality ($\leq_A$) and conflict ($\frown$) represents **rules**.

Strategies on A: event structures S



S



A

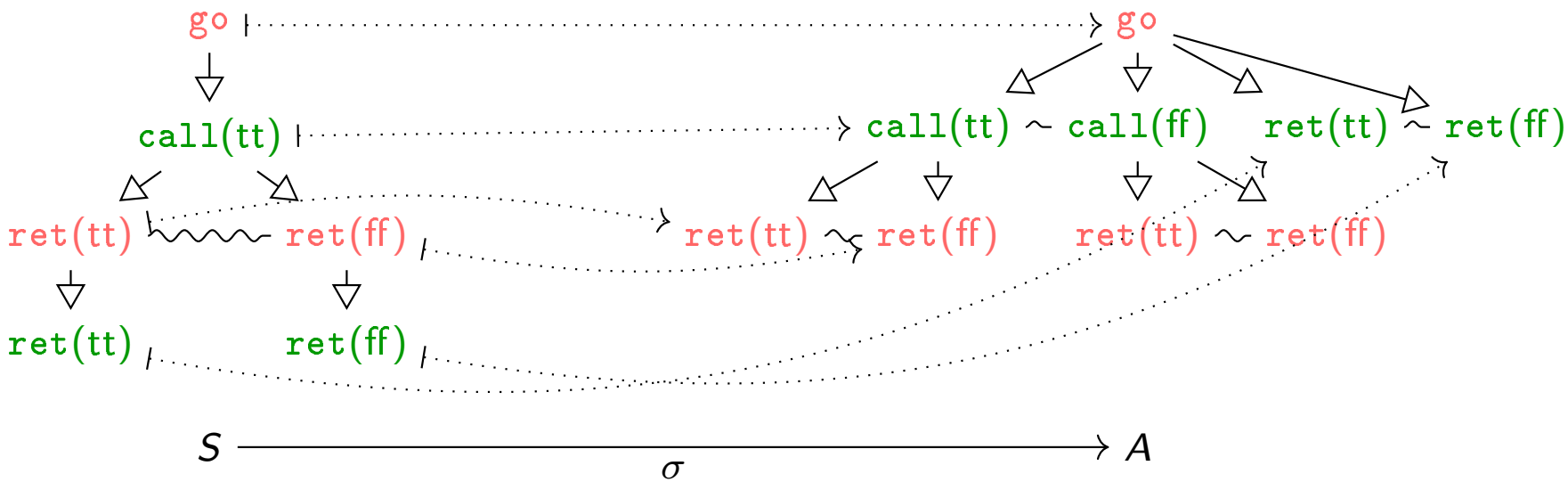
Game semantics on event structures [RW11]

Games: polarised event structures $(A, \leq_A, \frown, \lambda\colon A \to \{-,+\})$

- Causality ($\leq_A$) and conflict ($\frown$) represents **rules**.

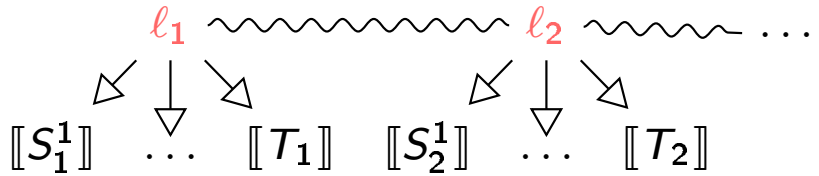
Strategies on A: event structures S with $\sigma : S \to A$.

- Conditions to ensure the respect of rules of A.

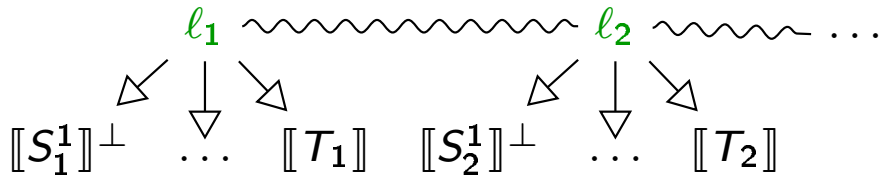


Interpretation of types as games

$$[\![\&_{i\in I}\, ?\ell_i(\vec{S_i}).T_i]\!] =$$



$$[\![\oplus_{i\in I}\, !\ell_i(\vec{S_i}).T_i]\!] =$$



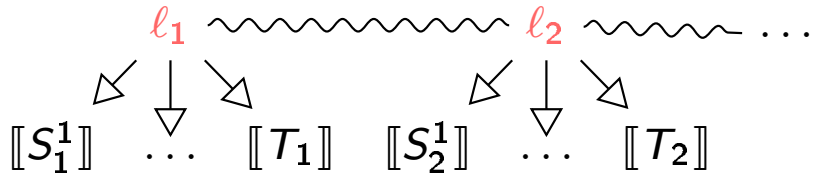
$$[\![s_1 : S_1, \ldots, s_n : S_n]\!] = [\![S_1]\!] \parallel \ldots \parallel [\![S_n]\!]$$

- **Image** of the interpretation: (recursive) forest-like games.
  $\rightsquigarrow$ Games used in semantics arise from session types.
- Interpretation gets rid of irrelevant syntactic information:

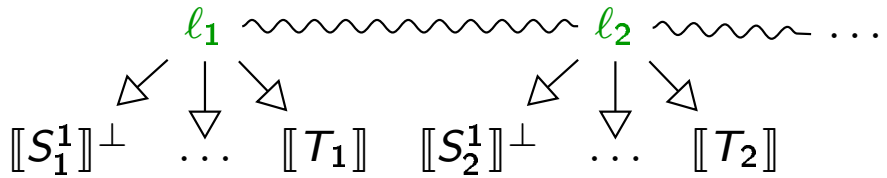
$$[\![!\ell(S, T).U]\!] \cong [\![!\ell(U, S).T]\!].$$

Interpretation of types as games

$$[\![\&_{i\in I}\, ?\ell_i(\vec{S_i}).T_i]\!] =$$



$$[\![\oplus_{i\in I}\, !\ell_i(\vec{S_i}).T_i]\!] =$$



$$[\![s_1 : S_1, \ldots, s_n : S_n]\!] = [\![S_1]\!] \parallel \ldots \parallel [\![S_n]\!]$$

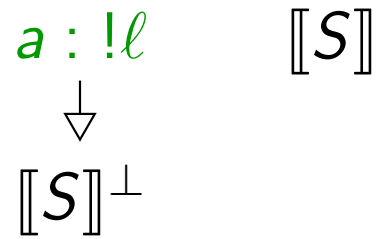
- **Image** of the interpretation: (recursive) forest-like games.
  $\rightsquigarrow$ Games used in semantics arise from session types.
- Interpretation gets rid of irrelevant syntactic information:

$$[\![!\ell(S, T).U]\!] \cong [\![!\ell(U, S).T]\!].$$

$\rightsquigarrow$ Next step: Interpret $P \triangleright \Gamma$ as a strategy on $[\![\Gamma]\!]$.

The main issue: name-passing

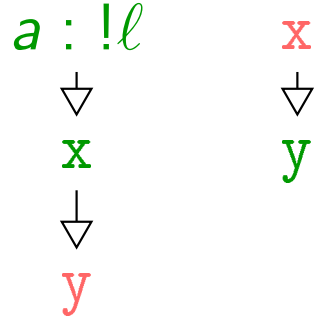
How to interpret $a!\ell\langle b\rangle \triangleright a : !\ell\langle S\rangle, b : S$?



$[\![a : !\ell\langle S\rangle, b : S]\!]$

The main issue: name-passing

How to interpret $a!\ell\langle b\rangle \triangleright a : !\ell\langle ?\mathtt{x}.\, !\mathtt{y}\rangle, b : ?\mathtt{x}.\, !\mathtt{y}$?



$$[\![a : !\ell\langle S\rangle, b : S]\!]$$

The main issue: name-passing

How to interpret $a!\ell\langle b\rangle \triangleright a : !\ell\langle ?\mathtt{x}.\, !\mathtt{y}\rangle, b : ?\mathtt{x}.\, !\mathtt{y}$?

$$[\![\bar{a}!\ell\langle b\rangle]\!]_{\mathsf{async}} \longrightarrow [\![a : !\ell\langle S\rangle, b : S]\!]$$

The main issue: name-passing

How to interpret $a!\ell\langle b\rangle \triangleright a : !\ell\langle ?\mathtt{x}.\,!\mathtt{y}\rangle, b : ?\mathtt{x}.\,!\mathtt{y}$?

$$[\![\bar{a}!\ell\langle b\rangle]\!]_{\mathsf{async}} \longrightarrow [\![a : !\ell\langle S\rangle, b : S]\!]$$

Introduces an **asynchronous** forwarder, inducing a **delay**
$\rightsquigarrow$ Delay observable using **synchrony**.

$$[\![a!\ell\langle b_1, b_2\rangle.\ \ b_1!\ell_1.\ \ b_2!\ell_2\ \ ]\!]_{\mathsf{async}} \approx [\![a!\ell\langle b_1, b_2\rangle.(b_1!\ell_1 \,\|\, b_2!\ell_2)]\!]_{\mathsf{async}}.$$

The model is not **adequate**!

The main issue: name-passing

How to interpret $a!\ell\langle b\rangle \triangleright a : !\ell\langle ?\mathtt{x}.\, !\mathtt{y}\rangle, b : ?\mathtt{x}.\, !\mathtt{y}$?

$$[\![\bar{a}!\ell\langle b\rangle]\!]_{\mathsf{async}} \longrightarrow [\![a : !\ell\langle S\rangle, b : S]\!]$$

To avoid this delay, we need **simultaneity**.

$\rightsquigarrow$ Simultaneity can be encoded using **causal loops** [GM11].

The main issue: name-passing

How to interpret $a!\ell\langle b\rangle \triangleright a : !\ell\langle ?\mathtt{x}.\, !\mathtt{y}\rangle, b : ?\mathtt{x}.\, !\mathtt{y}$?

$$[\![\bar{a}!\ell\langle b\rangle]\!]_{\mathsf{async}} \longrightarrow [\![a : !\ell\langle S\rangle, b : S]\!]$$

To avoid this delay, we need **simultaneity**.
$\rightsquigarrow$ Simultaneity can be encoded using **causal loops** [GM11].

Coincident strategies and causal loops

Definition

A **coincident event structure** is a triple $(E, \leq, \sim)$ where $\leq$ is a preorder, satisfying the usual axioms of event structures.

A **coincidence** is an equivalence class for $\simeq := (\leq \cap \geq)$.

Coincident strategies and causal loops

Definition

A **coincident event structure** is a triple $(E, \leq, \sim)$ where $\leq$ is a preorder, satisfying the usual axioms of event structures.

A **coincidence** is an equivalence class for $\simeq := (\leq \cap \geq)$.

Definition

A **coincident strategy** on a game A is a coincident event structure S and a map $\sigma : S \to A$ such that coincidence of S are of the form $\{s\}$ or $\{s, s'\}$.

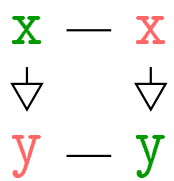
Theorem

There is a compact-closed category of coincident strategies.

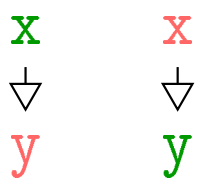
$\rightsquigarrow$ Composition needs to tell apart deadlocks and coincidences.

Interpretation of processes as coincident strategies

By induction using the **synchronous forwarder**: $\mathbb{CCC}_A$.



$\mathbb{CCC}_A$



$A^\perp \parallel A$

A few cases:

$$[\![(\nu a : A)P]\!] = \mathbb{CCC}_A \odot [\![P]\!]$$

$$[\![a!\ell\langle\vec{b}\rangle.P]\!] = \ell \cdot ([\![P]\!] \parallel \mathbb{CCC}_{[\![\vec{B}]\!]}) \qquad \text{where } \vec{b} : \vec{B}$$

$\rightsquigarrow$ Every typed process $P \triangleright \Gamma$ becomes a strategy $[\![P]\!]$ on $[\![\Delta]\!]$.

Properties of the model

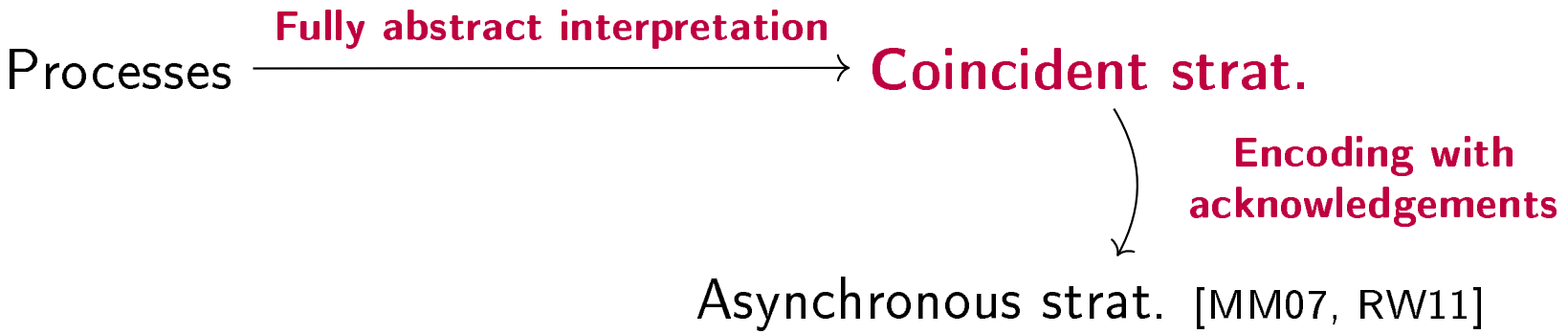
$$\text{Processes} \xrightarrow{\textbf{Fully abstract interpretation}} \textbf{Coincident strat.}$$

- Interpretation of processes is (intensionally) fully abstract:

$$P \approx Q \quad \Leftrightarrow \quad [\![P]\!] \approx [\![Q]\!].$$

- In both settings, barbed congruence is characterised by **weak bisimulation**.

Properties of the model

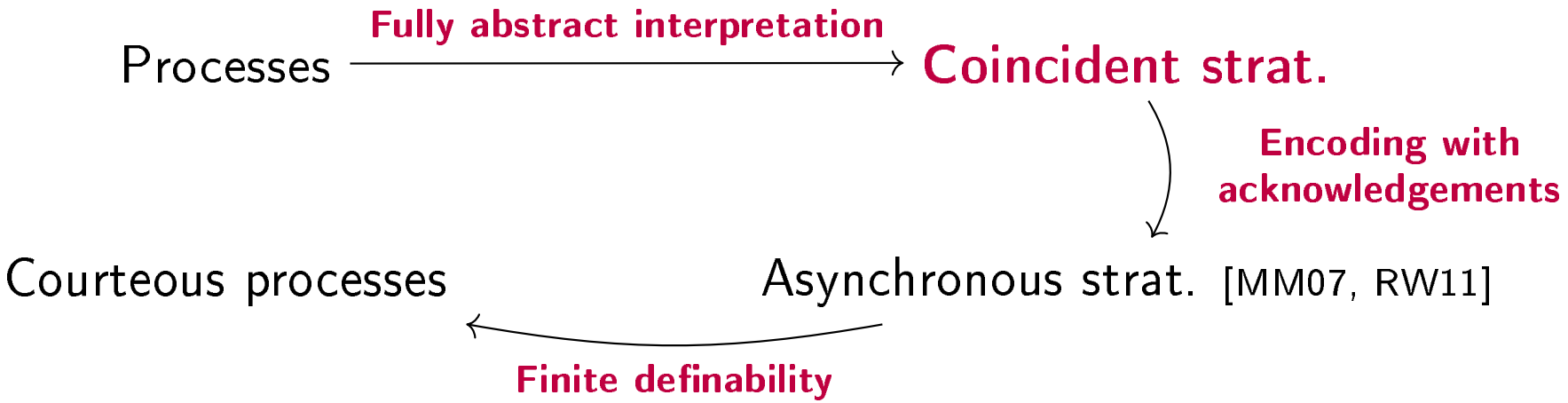


- Define an translation on games:

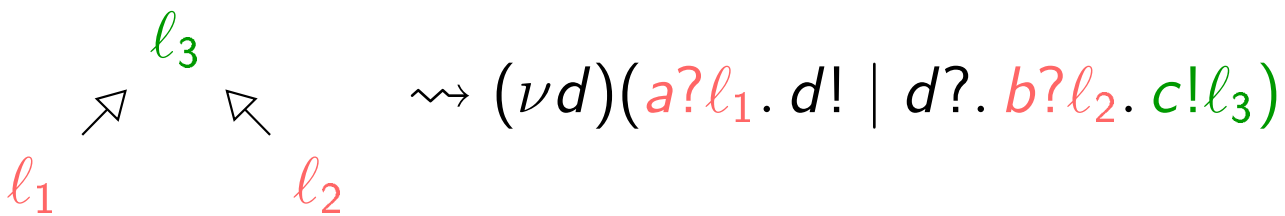
$$\begin{array}{lll} A & \mapsto & \uparrow A \\ \mathtt{x} \triangleright \mathtt{y} & \mapsto & \mathtt{x} \triangleright \mathtt{ack_x} \triangleright \mathtt{y} \triangleright \mathtt{ack_y} \end{array}$$

- Lift this translation on strategies: $S : A \mapsto \uparrow S : \uparrow A$.
- Translation is **injective** (but not fully abstract).

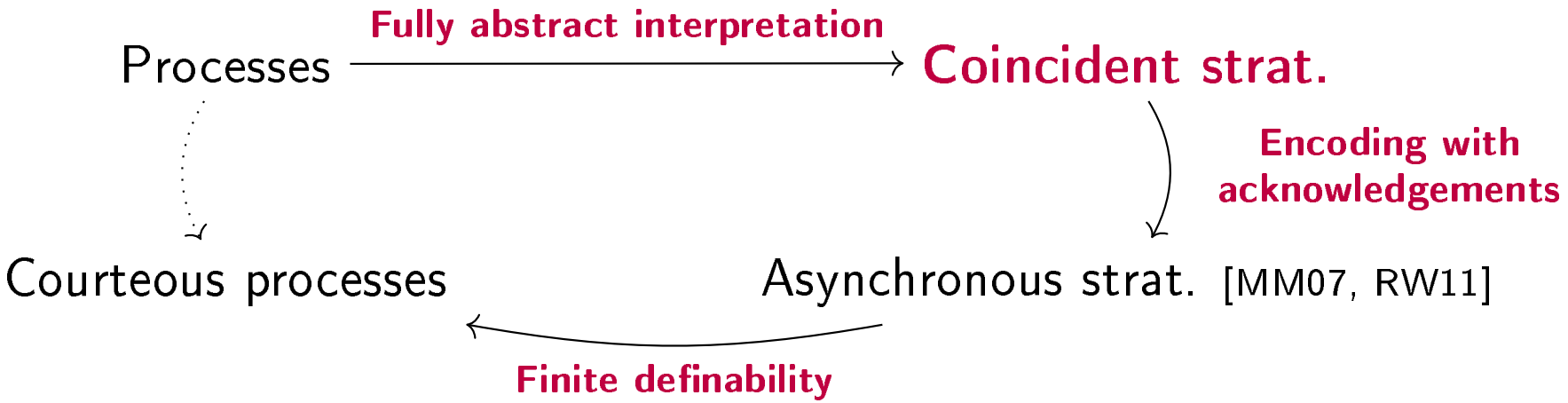
Properties of the model



Show how to define coincidence-free strategies as processes:



Properties of the model



By composition, we get a translation on processes that replaces synchronous operations by req/ack.
$\rightsquigarrow$ Translation difficult to formalise directly on the syntax.

Related work & perspectives.

Related work.

- Other standard models of game semantics [Lai05, ST17].
  $\rightsquigarrow$ Only results for may-testing.
- Models of [EHS13], fully abstract for fair-testing
  $\rightsquigarrow$ No representation of the causal behaviour.
- Abstract synchronous game semantics [Mel19].
  $\rightsquigarrow$ No interpretation of languages.

Perspectives.

- Strategies offer **causal normal forms** of processes,
  Processes offer a **syntax** to represent strategies.
- Extend the result to more expressive calculus:
  - Get rid of the linearity constraint (allow initialisers)
  - Step out of the race-free setting.


Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria.
Full abstraction for PCF.
Information and Computation, 163(2):409–470, 2000.


Clovis Eberhart, Tom Hirschowitz, and Thomas Seiller.
Fully-abstract concurrent games for pi.
CoRR, abs/1310.4306, 2013.


Dan R. Ghica and Mohamed Nabih Menaa.
Synchronous game semantics via round abstraction.
In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6604 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2011.


Martin Hyland and Luke Ong.
On full abstraction for PCF.
Information and Computation, 163:285–408, 2000.


Kohei Honda.
Types for dyadic interaction.
In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.


Jim Laird.
A game semantics of the asynchronous *pi*-calculus.
In *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, pages 51–65, 2005.


Paul-André Melliès.
Categorical combinatorics of scheduling and synchronization for games and strategies.
In *Accepted for publication at POPL'19*, 2019.


Paul-André Melliès and Samuel Mimram.
Asynchronous games: Innocence without alternation.
In *CONCUR 2007 - Concurrency Theory, 18th International Conference*, pages 395–411, 2007.


Silvain Rideau and Glynn Winskel.
Concurrent strategies.
In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 409–418, 2011.


Ken Sakayori and Takeshi Tsukada.
A truly concurrent game model of the asynchronous π-calculus.

In *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures - Volume 10203*, pages 389–406, New York, NY, USA, 2017. Springer-Verlag New York, Inc.