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

A synchronous side, and an asynchronous side

SIMON CASTELLAN, Imperial College London, United Kingdom
NOBUKO YOSHIDA, Imperial College London, United Kingdom

Game semantics and session types are two formalisations of the same concept: message-passing open *programs* following certain *protocols*. Game semantics represents protocols as *games*, and programs as *strategies*; while *session types* specify protocols, and well-typed π*-calculus processes* model programs. Giving faithful models of the π-calculus *and* giving a precise description of strategies as a programming language are two difficult problems. In this paper, we show how these two problems can be tackled at the same time by building an accurate game semantics model of the session π-calculus.

Our main contribution is to fill a semantic gap between the synchrony of the (session) π-calculus and the asynchrony of game semantics, by developing an event-structure based game semantics for synchronous concurrent computation. This model supports the first truly concurrent fully abstract (for barbed congruence) interpretation of the synchronous (session) π-calculus. We further strengthen this correspondence, establishing finite definability of asynchronous strategies by the internal session π-calculus. As an application of these results, we propose a faithful encoding of synchronous strategies into asynchronous strategies by call-return protocols, which induces automatically an encoding at the level of processes. Our results bring session types and game semantics into the same picture, proposing the session calculus as a programming language for strategies, and strategies as a very accurate model of the session calculus. We implement a prototype which computes the interpretation of session processes as synchronous strategies.

CCS Concepts: • **Theory of computation → Process calculi**; **Denotational semantics**;

Additional Key Words and Phrases: π-calculus, session types, event structures, game semantics

ACM Reference Format:
Simon Castellan and Nobuko Yoshida. 2019. Two Sides of the Same Coin: Session Types and Game Semantics: A synchronous side, and an asynchronous side. *Proc. ACM Program. Lang.* 3, POPL, Article 27 (January 2019), 29 pages. https://doi.org/10.1145/3290340

1 INTRODUCTION

Over the last 25 years, game semantics [Abramsky et al. 2000, 1994; Hyland and Ong 1994, 2000] has been used as a versatile framework for constructing the first syntax-independent (often fully-abstract) models for a variety of programming languages, exhibiting a wide range of computational effects such as, *e.g.*., state [Abramsky et al. 1998; Abramsky and McCusker 1999], control [Laird 1997], concurrency [Laird 2001], nondeterminism [Harmer and McCusker 1999] and probabilities [Danos and Harmer 2002]. This versatility arises from an *interactive* interpretation of computation. An open (higher-order) program is modelled by a *process* (called *strategy*) interacting with its environment according to a *protocol* (described by a *game*). In particular, the game only depends on

Authors' addresses: Simon Castellan, Imperial College London, London, United Kingdom, s.castellan@ic.ac.uk; Nobuko Yoshida, Imperial College London, London, United Kingdom, n.yoshida@ic.ac.uk.

the *type* of the program (the interface between the program and the environment), and not on the programming language it is written in.

It has also been 25 years since another framework, *session types* [Honda et al. 1998; Takeuchi et al. 1994], was proposed for codifying communication structures using protocols in concurrent, message-passing processes. This suggests the following analogy:

	Game semantics	**Session π-calculus**
protocols	games	types
open programs	strategies	processes

In both cases, protocols are two-party and described from the point of view of one of the participant. Actions are thus polarised: output actions correspond to Player moves, and input actions to Opponent moves. Reversing this polarity leads in both cases to a central notion of duality: two agents can only interact on dual types/games. According to [Hyland and Ong 1995], this analogy captures "every essential aspect of the dialogue game paradigm so precisely that the π-calculus presentation may as well be taken to be a formal *definition*". In particular, this insight led to, *e.g.*, the verification of type soundness for various programming constructs [Disney and Flanagan 2015]; and a typing discipline characterising sequential computation in the π-calculus [Berger et al. 2001].

In spite of these well-known conceptual similarities, there is no work building a *precise* connection between any dialect of the π-calculus and game semantics. Having an exact semantic correspondence between these two worlds would be mutually beneficial: session processes could provide a syntactic description of strategies as message-passing programs (an open problem for concurrent game semantics); and game semantics could offer a canonical denotational (categorical) semantics of session processes and give semantic proofs of properties (such as deadlock-freedom).

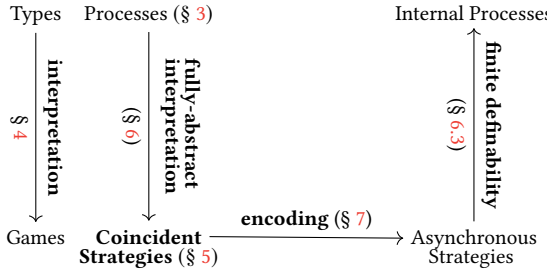
To have a close operational correspondence between processes and strategies, traditional play-based strategies do not fit: they forget the nondeterministic branching point, a feature necessary to obtain full abstraction for observational equivalence (barbed congruence) [Honda and Yoshida 1995; Milner and Sangiorgi 1992]. Following the footsteps of [Melliès 2006] which pioneered the use of causal structures in game semantics, the recent work [Rideau and Winskel 2011] is able to remember this branching point by giving a causal representation of strategies using *event structures*. As an added benefit, it also represents games as event structures, which support a natural interpretation of session types unlike traditional arena-based games [Hyland and Ong 1994]. This interpretation of session types explicits a striking connection between games and session types, describing protocols using the same constructs (sequencing, parallel composition and duality).

However, to build a formal relationship between the session π-calculus and game semantics based on event structures, there is still a conceptual gap to overcome. One the one hand, traditional session processes are *synchronous* where two processes communicate via hand-shakes, synchronising immediately. On the other hand, concurrent game semantics is inherently *asynchronous*. In game semantics, a ubiquitous agent, *copycat*, plays a key role to compositionally interpret programs. In the π-calculus, copycat corresponds to the *forwarder* or *link agent* [Honda and Yoshida 1995; Sangiorgi 1996]) which identifies (or links) two channels by forwarding data back and forth.

For the interpretation to be sound, copycat must be a categorical identity which implies that strategies must be invariant under composition by this agent. Since composition with copycat breaks syntactic dependences from an output, or to an input on different channels (forcing stronger asynchrony properties than the asynchronous π-calculus [Honda and Tokoro 1991] which only forbids dependences from an output), strategies cannot represent synchronous processes. This phenomenon is due to the delay that copycat adds between the reception of a move, and its forwarding. We say that this copycat is *asynchronous*.

Our main contribution is to create a model where there also exists a *synchronous* copycat where the forwarding is instantaneous. In this model, strategies can play several moves *at the same time* (rather than in an unspecified order) in a ***coincidence***. Traditional models of true concurrency are coincidence-free: two events can always be separated by a partial execution containing one but not the other. Our main contribution is ***coincident event structures***, an extension of prime event structures [Winskel 1986] allowing coincidences, obtained by relaxing the causal order to a *causal preorder*. We show that coincident event structures support an intensionally fully-abstract game semantics interpretation of the session π-calculus, interpretation devised using the techniques of [Castellan et al. 2018; Rideau and Winskel 2011].

Contribution and outline. In § 2, the paper starts by an illustration of the correspondence and of the difficulties of bringing these two worlds together. We then introduce the synchronous session π-calculus in § 3. Our contributions start in § 4 with an interpretation from session types to games, which will be driving the interpretation of processes. We show that a large class of games are in the image of the translation, hence session types can be seen a game description language. In § 5, we introduce coincident event structures, which are then used to define coincident strategies. They extend the strategies of [Castellan et al. 2018] and form a category without requiring any asynchrony conditions on the strategies. In § 6, we model the session π-calculus inside coincident strategies and show that this interpretation is intensionally fully abstract (Theorem 6.5). We then show that *finite* strategies of [Castellan et al. 2018] are definable using *internal session processes* (processes that do not send free names), hence exhibiting a natural programming language for them (Theorem 6.10). Finally, in § 7, we show that our category of coincident strategies is isomorphic to a subcategory of the concurrent strategies of [Castellan et al. 2018]. Via the result of finite definability, this yields a sound translation from session processes to internal session processes, which are invariant under composition with asynchronous forwarders. This outline is summarised by the diagram on the left, where text in bold denotes our contributions.



The translation from processes (without recursion) to coincident event structures has been implemented. A version is available at http://sessiontypesandgames.github.io/

Proofs and some counterexamples can be found in the technical report [Castellan and Yoshida 2018].

2 OVERVIEW OF THE MODEL

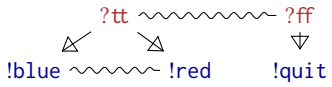
This section informally explains the problem of relating session types and game semantics, and the reason why we introduce ***coincident strategies***. This section does not require advanced knowledge of session types, event structures or game semantics. We first illustrate the correspondence (§ 2.1) and then explain the challenges to overcome to faithfully interpret processes as strategies (§ 2.2).

2.1 Illustration of the correspondence

2.1.1 Selection and branching. In (two-party) session types and game semantics, protocols are described from the viewpoint of one particular participant. For instance, a simple protocol "receive a boolean; if received true, then send a colour; else send a `quit` message" is represented in the

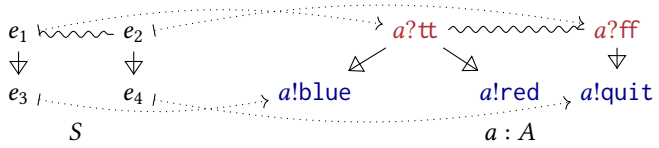
session π-calculus, by the type T (left); or in game semantics by the game A:

$$T = ?\mathtt{tt}.\,(!\mathtt{blue} \oplus !\mathtt{red})\ \&\ ?\mathtt{ff}.\,!\mathtt{quit} \qquad A =$$



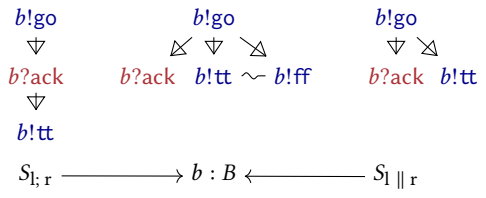
In the session type, $\mathtt{tt}$, $\mathtt{ff}$, … denote the *labels*, representing the messages exchanged. The symbols !/? denote the ***polarity*** of the action (input/output); $\oplus$/&, the polarity of the choice (internal/external); and dots, sequencing ("then"). On the other side, the game describes every message exchanged as a move, and organises them as an ***event structure***, which consists in a partial order $\rightarrowtriangle$, *causality*, and a relation $\sim$, *conflict*. Causality of the game describes the sequencing in the protocol; and conflict the possible choices of messages for one particular action. Moves of a game can be decorated by any relevant information, *e.g.* in this case A is decorated by messages inherited from the protocol.

In the π-calculus, processes exchange messages and synchronise on *channels*, while in game semantics strategies play and synchronise on *moves*. To implement this protocol as a process, we bind the type T to a channel a, and write a well-typed process on the typing environment $a : T$, for instance: $\vdash P := (a?\mathtt{tt}.\,a!\mathtt{blue})\ \&\ (a?\mathtt{ff}.\,a!\mathtt{quit}) \triangleright a : T$, reading as: if $\mathtt{tt}$ is received on a, then send $\mathtt{blue}$ on a, or (&) if $\mathtt{ff}$ is received, then send $\mathtt{quit}$. In game semantics based on event structures, a strategy on a game will be an event structure S along with a labelling function to the game. In this case, the interpretation of P will play on the interpretation of the context $a : T$ obtained from A by decorating the moves with the channel a as depicted on the right of the diagram below. The diagram on the left represents the strategy corresponding to P along with its labelling function.



In general, we will omit the labelling function and draw the events as their labels.

2.1.2 Parallelism and multiple sessions. The previous protocol has such a simple structure that implementations of it must be sequential. To allow for concurrent implementations, the protocols must leave the order between some actions unspecified. For instance in the protocol "send go, and then receive an acknowledgement and send a boolean in any order". There are two independent subparts: receiving the acknowledgement, and sending the boolean. The corresponding game will have two disjoint subgames for these, giving rise to a game B. The diagram on the left depicts the game $b : B$ (middle) as well as two strategies on it: one where the boolean is only sent after reception of the acknowledgement (left); and one where the boolean is sent without waiting for the acknowledgement (right). In $b : B$, the event $b?\mathtt{ack}$ is not comparable, and not in conflict with the events $b!\mathtt{tt}$ and $b!\mathtt{ff}$: they are *concurrent*.



In the session π-calculus, two independent parts of the protocols will occur on different channels. However, at the beginning of the session there is only one channel, say b. Since after the output of the go message, there are now two independent subprotocols, there must also be two channels. In the π-calculus, the second name is sent as a payload of the message go. This gives two possible representations of this protocol as a session type (with the same game representation, namely B),

depending on which part is fulfilled by the payload channel:

$$T = !\mathsf{go}\langle !\mathsf{ack}\rangle.(!\mathtt{tt} \oplus !\mathtt{ff}) \qquad T' = !\mathsf{go}\langle ?\mathtt{tt} \,\&\, ?\mathtt{ff}\rangle.\,(?\mathsf{ack}).$$

The syntax $!\mathsf{go}\langle S_1\rangle.S_2$ means send a message go along with a channel on which the *receiver* will perform S_1, and continue as S_2. Because S_1 is from the point of view of the receiver, there is a natural duality in the type of the payload channel for outputs. The strategies depicted above are the interpretations of the processes:

$$P_{\mathrm{l;\,r}} = (\nu c)(b!\mathsf{go}\langle \overline{c}\rangle.c!\mathsf{ack}.\, b!\mathtt{tt}) \qquad P_{\mathrm{l}\parallel\mathrm{r}} = (\nu c)(b!\mathsf{go}\langle \overline{c}\rangle.(c!\mathsf{ack} \mid b!\mathtt{tt})) \qquad \text{on context } b : T$$

These two processes start with a restriction (νc), creating a pair of connected channels c and $\overline{c}$ with dual types !ack and ?ack respectively, and proceed to send $\overline{c}$ on b and continue.

In both these processes, the name c sent is created just before the sent. In the π-calculus, such processes are called ***internal*** [Sangiorgi 1996] and sometimes abbreviated as:

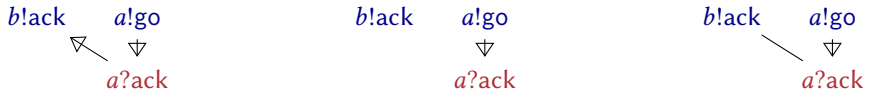
$$P_{\mathrm{l;\,r}} = b!\mathsf{go}(c).c!\mathsf{ack}.\, b!\mathtt{tt} \qquad P_{\mathrm{l}\parallel\mathrm{r}} = b!\mathsf{go}(c).(c!\mathsf{ack} \mid b!\mathtt{tt}) \qquad \text{on context } b : T$$

In § 6.3, we show that internal processes define a large class of event structures.

2.2 Discrepancy between processes and strategies

We now detail the obstacles in the way of giving a fully abstract semantics of session processes in terms of strategies. The main obstacle is the gap between synchrony and asynchrony detailed in § 2.2.1. We then explain why capturing both causal dependences (§ 2.2.2) and the nondeterministic branching point (§ 2.2.3) is also essential, and the solutions given by [Rideau and Winskel 2011] and [Castellan et al. 2018] respectively.

2.2.1 Synchrony and asynchrony: copycat. A key feature of the π-calculus is *free* name passing. In § 2.1.2, we only mentioned *bound* name passing. For instance, the process $a!\mathsf{go}\langle b\rangle$ is a well-typed session process on the typing environment $b : !\mathsf{ack}, a : !\mathsf{go}\langle !\mathsf{ack}\rangle$. The typing environment contains two protocols "send an acknowledgement" (on b), and "send go, and then receive ack" (on a), whose game interpretation is depicted in (1b) below. The meaning of $a!\mathsf{go}\langle b\rangle$ should be that the acknowledgement on b is sent exactly when the receiver of the go message writes on the payload channel. This leads to the strategy in (1a), which is the way that traditional game semantics interprets free name passing [Laird 2005]. The forwarding between b and a is *asynchronous*: intuitively, there can be an arbitrary delay between the input on a and the output on b. In fact, this strategy is also the interpretation of the internal process $(\nu c)(a!\mathsf{go}\langle \overline{c}\rangle.\, c?\mathsf{ack}.\, b!\mathsf{ack})$. The expression $c?\mathsf{ack}.\, b!\mathsf{ack}$ is the *asynchronous forwarder* between b and c for type !ack.



(1a) Asynchronous forwarding (1b) Game for $b : ?\mathsf{ack}, a : !\mathsf{go}\langle ?\mathsf{ack}\rangle$ (1c) Synchronous forwarding

In the session π-calculus, the two processes $(\nu c)(a!\mathsf{go}\langle \overline{c}\rangle.\, c?\mathsf{ack}.\, b!\mathsf{ack})$ and $a!\mathsf{go}\langle b\rangle$ are not observationally equivalent (see the counterexample section of the technical report [Castellan and Yoshida 2018]), therefore it is not possible to devise a fully abstract interpretation this way. In our model, we keep the same idea (to represent free name passing via a forwarder), and use a *synchronous forwarder* instead which makes sure that the input on a will occur at the same time (observationally) than the output on x: these two moves are *coincident*, depicted with the line between them. (1c) depicts our interpretation of $a!\mathsf{go}\langle b\rangle$.

2.2.2 Causality of processes. Because our model is based on event structures, it computes the causality between the actions of processes. This is important to obtain a fully abstract model, because the processes $a!\mathtt{tt} \mid b!\mathtt{tt}$ and $a!\mathtt{tt}.\, b!\mathtt{tt} + b!\mathtt{tt}.\, a!\mathtt{tt}$ have the same traces but are not barbed congruent. A causal model is necessary to distinguish them.

The main difficulty in building a causal model of the session π-calculus is interpreting the restriction operator. Without it, the event structures expressible by processes are simply forests. With it, every causal pattern becomes expressible (see § 6.3). We give here a little example of a non tree-like causal behaviour. Consider the simple session type $\mathbf{1} = !()$ (send a dummy message ()) and its dual $\bot = ?()$ (receive a dummy message), whose interpretation in term of games are simply the one-event games $!()$ and $?()$; and consider the process $J = (\nu d)(a?().\, \overline{d}!() \mid b?().\, d?().\, c!())$, well-typed in the environment $a : \bot, b : \bot, c : 1$. This process implements a simple causal behaviour: wait for input on a and b, and only then output on c. Our interpretation of J exactly expresses this:

$$[\![(\nu d)(a?().\, \overline{d}!() \mid b?().\, d?().\, c!())]\!] = \quad \begin{array}{ccc} a?() & & b?() \\ & \searrow \quad \swarrow & \\ & c!() & \end{array} \quad \text{on the game} \quad a?() \quad b?() \quad c!()$$

One of the main contributions of [Rideau and Winskel 2011] is to introduce the ***interaction*** of strategies, which is crucial to define the interpretation of the restriction operator (see § 5.3).

2.2.3 Nondeterministic branching point. Causality is not enough in presence of nondeterminism. Indeed, the model of [Rideau and Winskel 2011] although being causal only supports *angelic nondeterminism*, which means that nondeterministic branches that deadlock are forgotten.

$$\begin{array}{cc} * \rightsquigarrow * & a!() \\ \downarrow & \\ a!() & \\ [\![a!() + Q]\!] & [\![a!()]\!] \end{array}$$

For instance, define $P = a!()$ and $Q = (\nu c)(\nu d)(c?().\, d!().\, \overline{a}!() \mid \overline{d}?().\, \overline{c}!())$ (a deadlocking process). Then $P + Q$ and P are both typable by the same typing environment but they are not observationally equivalent. A model based on [Rideau and Winskel 2011] would however equate these processes by interpreting both by the strategy $a!()$. Using the recent methodology in [Castellan et al. 2018], we can define strategies with *internal actions* (written $*$) which can remember that some branch may deadlock. By remembering the nondeterministic choice, the interpretations of $P + Q$ and P, given on the left, are not weakly bisimilar even though the interpretation of Q is empty.

3 SESSION TYPES AND SESSION PROCESSES

This section gives a simplification of the most widely studied synchronous session calculus [Honda et al. 1998; Yoshida and Vasconcelos 2007]. Since our main focus is on session communications, we only allow exchanges of linear channels. The calculus is identical to the synchronous calculus presented in [Chen et al. 2017], extended with a nondeterministic choice operator. Despite only sharing linear names, because of recursive types and recursive processes, the language is expressive enough to contain the encoding of nondeterministic PCF.

3.1 Session processes

The syntax of processes is given in Table 1. We use the following sets: *channel variables*, ranged over by $x, y, z \ldots$; *linear channels*, ranged over by $a, b, c, \overline{a}, \overline{b}, \overline{c} \ldots$; *process variables*, ranged over by $X, Y, \ldots$. We write $\mathcal{N}$ for the set of *identifiers* (channel variables and linear channels), ranged over by $u, u', \ldots$; and $\mathscr{L}$ for the *finite* set of possible labels, ranged over by $l, l', \ldots$ The notation $\widetilde{v}$ denotes a vector $(v^1, \ldots, v^n)$ of values; similarly for others, for instance $\widetilde{T}$ denotes a sequence of types. We write $\overline{a}$ is a *co-channel* of a, which represents a dual end-point of a; we assume $\overline{\overline{a}} = a$.

u	::=	**Identifiers**				
		$a, b, c, \overline{a}, \overline{b}, \overline{c} \ldots$	channel	|	$x, y, z, \ldots$	variable
P	::=	**Process**				
		$\mathbf{0}$	nil	|	X	variable
	|	$\&_{i\in I}\, u?l_i(\widetilde{x_i}).P_i$	input	|	$u!l\langle\widetilde{v}\rangle.\, P$	output
	|	$P \mid P$	parallel	|	$\sum_{i\in I} P_i$	choice
	|	$\mu X.\, P$	recursion	|	$(\nu a)P$	restriction

S, T	::=	**Types**	
		end	end
	|	$\mathbf{t}$	variable
	|	$\&_{i\in I}?l_i(\widetilde{S_i}).\, T_i$	branching
	|	$\oplus_{i\in I}!l_i\langle\widetilde{S_i}\rangle.\, T_i$	selection
	|	$\mu\mathbf{t}.\, T$	recursion

Table 1. Syntax of processes and types.

[R-COM]
$$\frac{k \in I}{(\nu a)(a!l_k\langle\widetilde{c}\rangle.\, P \mid \&_{i\in I}\, \overline{a}?l_i(\widetilde{x_i}).\, Q_i) \to (\nu a)(P \mid Q_k\{\widetilde{c}/\widetilde{x_k}\})}$$

[R-CHOICE]
$$\frac{k \in I}{\sum_{i\in I} P_i \to P_k}$$

[R-RES]
$$\frac{P \to P'}{(\nu c)P \to (\nu c)P'}$$

[R-PAR]
$$\frac{P \to P'}{P \mid Q \to P' \mid Q}$$

[R-STRUCT]
$$\frac{P \equiv P' \to Q' \equiv Q}{P \to Q}$$

$$\begin{aligned}
\mathbf{0} \mid P &\equiv P \\
P \mid Q &\equiv Q \mid P \\
P \mid (Q \mid R) &\equiv (P \mid Q) \mid R \\
\mu X.\, P &\equiv P\{\mu X.\, P/X\} \\
(\nu a)\mathbf{0} &\equiv \mathbf{0} \\
(\nu a)P \mid Q &\equiv (\nu a)(P \mid Q) \quad (a \notin fn(Q)) \\
(\nu a)(\nu b)P &\equiv (\nu b)(\nu a)P
\end{aligned}$$

Table 2. Reduction and structural rules.

Session communications are performed between an output process and an input process where the former sends a channel choosing one of the branches offered by the latter. The choice process $\sum_{k\in I} P_k$ internally chooses one of P_k with $k \in I$. The process $(\nu a)P$ is a restriction which binds two dual channels, a and $\overline{a}$ in P, making them co-channels, i.e. allowing them to communicate (see rule [R-COM] in Table 2). This restriction is commonly used in the recent literature of session types, e.g. [Gay and Hole 2005; Mostrous and Yoshida 2015; Yoshida and Vasconcelos 2007]. The rest of the syntax is standard. We often omit $\mathbf{0}$ from the tail of processes. The *bindings* for variables are in inputs and those for channels are in restrictions, and those for process variables are in recursions. The derived notions of bound and free channels, channel variables, identifiers, process variables, alpha equivalence, $\equiv_\alpha$, and substitution are standard. We use the Barendregt convention that no bound identifier can occur free or in two different bindings. By $fn(P)$/$\mathrm{fv}(P)$ we denote the set of *free channels* and *variables* in P. We write $a!\langle\widetilde{u}\rangle$ and $a?(\widetilde{u})$ when the labels sent or received are irrelevant; and $a!l$ and $a?l$ when no channel is sent or received.

3.2 Operational semantics

Table 2 defines the reduction relation and the structural rules. [R-COM] is the main communication rule between input and output at two co-channels a and $\overline{a}$, where the label l_k is selected and channels $\widetilde{c}$ are instantiated into the k-th input branch. [R-CHOICE] internally chooses the k-th process, which makes computations non-deterministic. Other rules are standard. $\to^*$ denotes the reflexive and transitive closure of $\to$.

3.3 Typing session processes

The syntax of *session types*, ranged over by T and S, is given in Table 1. The *branching type* $\&_{i\in I}?l_i(\widetilde{S_i}).\, T_i$ describes a channel willing to branch on an incoming label l_i, receive channels of type $\widetilde{S_i}$, and then continue its interaction as prescribed by T_i. The *selection type* $\oplus_{i\in I}!l_i\langle\widetilde{S_i}\rangle.\, T_i$ is its dual: it describes a channel willing to send a label l_i with channels of type $\widetilde{S_i}$, and then continue its

$$[\text{T-IDLE}]\frac{\Delta \text{ end only}}{\Gamma \vdash \mathbf{0} \triangleright \Delta} \qquad [\text{T-VAR}]\frac{\Delta' \text{ end only}}{\Gamma, X{:}\Delta \vdash X \triangleright \Delta, \Delta'}$$

$$[\text{T-INPUT}]\frac{\forall i \in I, \Gamma \vdash P_i \triangleright \Delta, u : T_i, \widetilde{x_i} : \widetilde{S_i}}{\Gamma \vdash \&_{i\in I}\, u?l_i(\widetilde{x_i}).\, P_i \triangleright \Delta, u : \&_{i\in I}?l_i(\widetilde{S_i}).\, T_i} \qquad [\text{T-OUT}]\frac{\Gamma \vdash P \triangleright \Delta, u : T_k \quad k \in I}{\Gamma \vdash u!l_k\langle\widetilde{v}\rangle.\, P \triangleright \Delta, u : \oplus_{i\in I}!l_i\langle\widetilde{S_i}\rangle.\, T_i, \widetilde{v} : \widetilde{S_k}}$$

$$[\text{T-PAR}]\frac{\Gamma \vdash P_1 \triangleright \Delta_1 \qquad \Gamma \vdash P_2 \triangleright \Delta_2}{\Gamma \vdash P_1 \mid P_2 \triangleright \Delta_1, \Delta_2} \qquad [\text{T-CHOICE}]\frac{\Gamma \vdash P_i \triangleright \Delta}{\Gamma \vdash \sum_{i\in I} P_i \triangleright \Delta} \qquad [\text{T-REC}]\frac{\Gamma, X{:}\Delta \vdash P \triangleright \Delta}{\Gamma \vdash \mu X.\, P \triangleright \Delta} \qquad [\text{T-NEW}]\frac{\Gamma \vdash P \triangleright \Delta, a : T, \overline{a} : \overline{T}}{\Gamma \vdash (\nu a)P \triangleright \Delta}$$

Table 3. Session typing rules.

interaction as prescribed by T_i. In branching and in selection types the labels are pairwise distinct; and the types of the exchanged channels are closed. We omit & and ⊕ and labels when there is only one branch. We use $\mathbf{t}$ to range over type variables. The type $\mu\mathbf{t}.\, T$ is a *recursive type*. We assume that recursive types are *contractive* (guarded), i.e. $\mu\mathbf{t}_1.\, \mu\mathbf{t}_2 \ldots \mu\mathbf{t}_n.\, \mathbf{t}_1$ is not a type. The type end represents the termination of a session and it is often omitted. We take an equi-recursive view of types considering two types unfolding to the same regular tree as equal [Chen et al. 2017]. We use the same convention as for processes, omitting the label when it is () or the channels when no channels should be sent: *e.g.* $!\mathtt{tt}$ is a shorthand for $!\mathtt{tt}\langle\rangle$, and $!\langle!\mathtt{quit}\rangle$ for $!()\langle!\mathtt{quit}\rangle$. In $\oplus_{i\in I} l_i\langle\widetilde{T_i}\rangle.\, S_i$ and $\&_{i\in I} l_i(\widetilde{T_i}).\, S_i$, S_i is the *continuation type*, while the $\widetilde{T_i}$ are the *argument types* of label l_i.

Session duality [Honda et al. 1998] ensures compatibility of communications. The function $\overline{T}$, defined below, yields the dual of the session type T.

$$\overline{\&_{i\in I}?l_i(\widetilde{S_i}).\, T_i} = \oplus_{i\in I}!l_i\langle\widetilde{S_i}\rangle.\overline{T_i} \quad \overline{\oplus_{i\in I}!l_i\langle\widetilde{S_i}\rangle.\, T_i} = \&_{i\in I}?l_i(\widetilde{S_i}).\, \overline{T_i} \quad \overline{\mathbf{t}} = \mathbf{t} \quad \overline{\mu\mathbf{t}.T} = \mu\mathbf{t}.\, \overline{T} \quad \overline{\text{end}} = \text{end}$$

The session type representing the boolean type is $\mathbb{B} = !\mathtt{tt} \oplus !\mathtt{ff}$, and the unit type $1 = !()$.

The typing judgements take the form: $\Gamma \vdash P \triangleright \Delta$ where Γ is the *recursion environment* which associates process variables to sequences of session types and Δ is the *session environment* which associates identifiers to session types. They are defined by:

$$\Gamma ::= \emptyset \mid \Gamma, X{:}\Delta \qquad \Delta ::= \emptyset \mid \Delta, u{:}T$$

We write $\Delta, u{:}T$ for $\Delta \cup \{u{:}T\}$ if $u \notin dom(\Delta)$ and Δ_1, Δ_2 for $\Delta_1 \cup \Delta_2$ when $dom(\Delta_1) \cap dom(\Delta_2) = \emptyset$. We say that Δ is *end-only* if $u : T \in \Delta$ implies $T = \text{end}$.

Table 3 gives the typing rules. They are essentially the same as [Chen et al. 2017]. Rule [T-IDLE] is the introduction rule for the nil process. To type an input process, rule [T-INPUT] requires the type S_i^j for variable x_i^j and the type T_i of channel u for the continuation P_i. In the resulting session environment, the type u has the branching type in which u receives S_i^j and then continues with T_i for each label l_i. The rule for typing output processes is dual. In rule [T-PAR], the session environment of $P_1 \mid P_2$ is the disjoint union of the environments Δ_1 and Δ_2 for the two processes, reflecting the linear nature of channels. In contrast, in rule [T-CHOICE], all of the processes have the same session environment because at most one of them will be executed. Rules [T-VAR] and [T-REC] type recursive processes, where recursive variables are typed by a session context. Rule [T-NEW] is a standard rule for name binding, where we ensure the co-channels have dual types. The type system enjoys

the standard subject reduction property. Notice that session environments are unchanged since reduction only occurs under a restriction.

Theorem 3.1 (Theorem 2.2 in [Chen et al. 2017]). *If* $\Gamma \vdash P \triangleright \Delta$ *and* $P \longrightarrow^* Q$, *then* $\Gamma \vdash Q \triangleright \Delta$.

3.4 Behavioural theory

We present now the most used behavioural equivalence, *reduction-based barbed congruence* [Honda and Yoshida 1995; Milner and Sangiorgi 1992], for synchronous session processes. This congruent relation is defined as the reduction-based bisimulation closure with the minimum observability (*barbs*). A process $\vdash P \triangleright \Delta$ has a barb on b [Milner and Sangiorgi 1992], written $P \downarrow_b$ when:

$$P \equiv (\nu \widetilde{a})(b!l\langle \widetilde{v} \rangle.\, P' \mid Q) \qquad \text{with } b, \overline{b} \notin \{\widetilde{a}\}.$$

The side condition $\overline{b} \notin \{\widetilde{a}\}$ ensures that Q does not contain a process which must interact with $b!l\langle \widetilde{v} \rangle.\, P'$ (note that b is linear: if the dual process at $\overline{b}$ is composed, the observer cannot observe b even b is unrestricted [Kouzapas and Yoshida 2015]). We define $\vdash P \triangleright \Delta \Downarrow_a$ if there exists Q such that $P \longrightarrow^* Q$ and $\vdash Q \triangleright \Delta \downarrow_a$. The set of contexts is defined as:

$$C \quad ::= \quad \_ \mid P \mid (C \mid P) \mid (P \mid C) \mid \&_{i \in I}\, v?l_i(\widetilde{x_i}).C_i \mid u!l\langle \widetilde{v} \rangle.C \mid \sum_{i \in I} C_i \mid \mu X.C \mid (\nu a)C$$

$C[P]$ substitutes process P to each hole $(\_)$ in context C. We define the reduction-closed congruence based on the definition of barb and [Honda and Yoshida 1995; Kouzapas and Yoshida 2015] as a *typed relation*: a relation $\mathcal{R}$ on processes relating closed terms (*i.e.*, $\mathrm{fv}(P_i) = \emptyset$), typed on the same environment. We write $\vdash P_1 \mathcal{R} P_2 \triangleright \Delta$ for $(P_1, P_2) \in \mathcal{R}$ and $\vdash P_i \triangleright \Delta$.

Definition 3.2 (Reduction-closed congruence). A typed relation $\mathcal{R}$ is a *reduction-closed congruence* if it satisfies the following conditions for each $\vdash P_1 \mathcal{R} P_2 \triangleright \Delta$:

(1) $\vdash P_1 \triangleright \Delta \Downarrow_u$ iff $\vdash P_2 \triangleright \Delta \Downarrow_u$.
(2) • $P_1 \longrightarrow^* P_1'$ implies that there exists P_2' such that $P_2 \longrightarrow^* P_2'$ and $\vdash P_1' \mathcal{R} P_2' \triangleright \Delta$
    • the symmetric case.
(3) For all closed context C and all Δ' such that $\vdash C[P_1], C[P_2] \triangleright \Delta'$, we have $\vdash C[P_1]\ \mathcal{R}\ C[P_2] \triangleright \Delta'$.

The union of all reduction-closed congruence relations is denoted as $\simeq$.

3.5 Internal session processes

We also study a subset of the session calculus called the internal π-calculus [Sangiorgi 1995] where the output only passes private (bound) names. This means that a process is *internal* when for every output $a!l\langle \widetilde{b} \rangle$ of P, $\widetilde{b}$ is a sequence of names restricted immediately before the output. We write $u!l(\widetilde{b}).\, P$ for $(\nu \widetilde{b})u!l\langle \widetilde{\widetilde{b}} \rangle.\, P$. Using structural congruence, the reduction rule [R-Com] instantiates to:

$$\text{[R-Com-I]} \quad (\nu a)(a!l_k(\widetilde{c}).\, P \mid \&_{i \in I}\, \overline{a}?l_i(\widetilde{x_i}).\, Q_i) \longrightarrow (\nu a\widetilde{c})(P \mid Q_k\{\widetilde{\widetilde{c}}/\widetilde{x_k}\}) \quad (k \in I)$$

Internal processes will correspond to the standard (pre-)strategies in game semantics (§ 6.3).

Example 3.3. An example of well-typed internal process that will be useful to us later, is the ***asynchronous copycat***. Given a type T (without recursion here), one can define a process $\vdash [u = v]_T \triangleright u : T, v : \overline{T}$ by induction on T (where $[\widetilde{u} = \widetilde{v}]_{\widetilde{T}}$ means $[u^1 = v^1]_{T^1} \mid \cdots \mid [u^n = v^n]_{T^n}$):

$$[u = v]_{\&_{i \in I} ?l_i(\widetilde{S}_i).T_i} = \sum_{i \in I} u?l_i(\widetilde{u_i}).\, v!l_i(\widetilde{v_i}).\, ([\widetilde{u_i} = \widetilde{v_i}]_{\widetilde{S}_i} \mid [u = v]_{T_i})$$

$$[u = v]_{\mathrm{end}} = \mathbf{0} \qquad [u = v]_T = [v = u]_{\overline{T}} \text{ (for selection types)}$$

4 TYPES AND GAMES

In this section, we define and study the interpretation of types as the games of our model. In § 4.1, we define event structures which are the basis for the notion of games introduced in [Rideau and Winskel 2011]. In § 4.2, we define games and the interpretation of session types inside them.

4.1 Event structures

Our model interprets processes and types as causal structures, which allows us to compute the dependences between actions. Due to (internal and external) choice, the causal structure needs to account for nondeterminism as well, representing the possible outcomes of these choices. This leads us to Winskel's event structures [Winskel 1986]. In this paper, we use ***prime event structures with binary conflict***, simply called event structures.

Definition 4.1. An ***event structure*** is a triple $(E, \leq_E, \#_E)$ where $(E, \leq_E)$ is a partial order and $\#_E \subseteq E \times E$ is a binary relation which is symmetric and irreflexive, such that:

Finite causes For $e \in E$, the set $[e] = \{e' \in E, e' \leq e\}$ is finite.
Conflict inheritance If $e \#_E e'$ and $e \leq e_0$ then $e_0 \#_E e'$.

The partial order indicates causal relationships between events: $e \leq e'$ when e' causally depends on e. The conflict relation indicates which events are ***incompatible***, *i.e.*, cannot occur together in the same execution. As a result, event structures take a global view of the program: the set E is the set of all events of the program. This feature is crucial for the construction of the model and differs from other main models of concurrency such as pomsets [Pratt 1984] and presheaves [Cattani and Winskel 1997].

Notations. Given a subset $X \subseteq E$ we write $[X] = \{e' \in E \mid \exists e \in X, e' \leq e\}$. A subset X is downclosed when $X = [X]$. We also write $[X) = [X] \setminus X$, and in particular, $[e) = [e] \setminus \{e\}$. If $e < e'$ with no events in between, we write $e \rightarrowtriangle e'$ and say that e' ***immediately depends*** on e. We say that X is ***consistent*** if for all $e, e' \in X$, $(e, e') \notin \#_E$. The second axiom of event structures forces conflict to propagate upwards: if $e \leq e'$, then e' inherits all the conflicts of e (and on top of those, may have some more of its own). A conflict $e \#_E e'$ is ***minimal*** if there is no $e_0 \leq e$ and $e_0' \leq e'$ with $e_0 \#_E e_0'$. In this case we write $e \sim\!\!\sim e'$. In diagrams, we only represent $\rightarrowtriangle$ and $\sim\!\!\sim$ as they are enough to recover the event structure. A ***configuration*** (representing a partial execution, or a history) of E is a consistent and downclosed subset $x \subseteq E$. The set of *finite* configurations is written $\mathscr{C}(E)$. A configuration x makes a transition to y with event e, written $x \xrightarrow{e} y$ (or simply $x - y$) when $e \notin x$ and $y = x \cup \{e\}$.

Labelled event structures. When trying to model languages with event structures, each element represents a computational event which may have some visible information (*e.g.*, for an event representing a channel output, the channel and message sent). This information is traditionally represented by *labels*, gathered in a set Σ. In this setting, to account faithfully for divergences and nondeterminism, we also keep track of internal choices. Processes are thus modelled as ***(partially) Σ-labelled event structures***, *i.e.*, an event structure E along with a *partial* map $lbl : E \rightharpoonup \Sigma$.

The interpretation of types will rely on standard operations on event structures. The ***prefixing*** of a Σ-labelled event structure E by a label ℓ has for event $\{\bot\} \cup E$ where $\bot \notin E$, for causal ordering $\leq_{\ell \cdot E} = \leq_E \cup \{\bot\} \times (\ell \cdot E)$, for conflict $\#_{\ell \cdot E} = \#_E$, and for labelling the extension of lbl_E with $\bot \mapsto \ell$. Prefixing by an internal action, $* \cdot E$, is defined similarly except that labelling is undefined on $\bot$.

The ***parallel composition*** $E_0 \parallel E_1$ of two event structures E_0 and E_1, has for events the disjoint union of E_0 and E_1, coded as $\{0\} \times E_0 \cup \{1\} \times E_1$, and for the ordering and the conflict relation:

$$\leq_{E_0 \parallel E_1} = \{(i, e), (i, e') \mid e \leq_{E_i} e'\} \qquad \#_{E_0 \parallel E_1} = \{(i, e), (i, e') \mid e \#_{E_i} e'\}.$$

Labelling functions on E and F induce naturally a labelling function on $E \parallel F$. The ***nondeterministic sum*** $E_0 + E_1$ is defined as $E_0 \parallel E_1$ except that events of E and F are in conflict:

$$\#_{E_0+E_1} = \{(i,e),(j,e') \mid (i \neq j) \vee (i = j \wedge e \#_{E_i} e')\}$$

We extend this notation to arbitrary sums: $\sum_{i \in I} E_i$.

Given an event structure E and $x \in \mathscr{C}(E)$, we define the ***remainder*** of E after x, written E/x as follows. Its events are those $e \in E \setminus x$ such that $x \cup [e] \in \mathscr{C}(E)$, and the partial order and conflict are inherited from E. Configurations of E/x are in bijection with extensions of x, that is with configurations $y \in \mathscr{C}(E)$ such that $x \subseteq y$.

Confusion-free event structures. Event structures support a wide notion of nondeterminism. In our source language, nondeterminism is localised at sums. This restricts the shape of nondeterminism expressible in the language. An event structure E is ***confusion-free*** when *(1)* $e \smile\!\!\frown_E e'$ implies $[e] = [e']$ and *(2)* if $e \smile\!\!\frown_E e' \smile\!\!\frown_E e''$ then $e = e''$ or $e \smile\!\!\frown_E e''$. As a result, the relation $(=_E \cup \smile\!\!\frown_E)$ becomes an equivalence relation whose equivalence classes are called ***cells***. Confusion-freeness is preserved under the operations defined in the previous paragraph.

Event structures and recursion. To interpret recursive types, we use the standard denotational technique of endowing event structures with an ω-CPO structure, representing types with open variables as continuous maps, and then interpreting recursive types as least fixpoints of these maps. We recall here the standard ω-CPO structure of event structures [Winskel 1982].

Definition 4.2. A Σ-labelled event structure E is ***included*** in Σ-labelled event structure F (written $E \hookrightarrow F$), when *(1)* $E \subseteq F$; *(2)* for $e, e' \in E$, $e <_E e'$ iff $e <_F e'$; *(3)* E is downclosed in F; *(4)* for $e, e' \in E$, $e \#_E e'$ iff $e \#_F e'$; and *(5)* for $e \in E$, $lbl_E(e) = lbl_F(e)$.

We write $\mathbf{ES}(\Sigma)$ for event structures ordered by inclusion. This order has a minimal element, the empty event structure $\bot$. It is well-known that it is also an ω-CPO:

Lemma 4.3. *$\mathbf{ES}(\Sigma)$ is an ω-CPO, meaning that every infinite ascending chain $E_0 \hookrightarrow E_1 \hookrightarrow \ldots$ has a least upper bound written $\lim E_i$.*

Recall that a monotonic function $f : \mathcal{A} \to \mathcal{B}$ between ω-CPOS is ***continuous*** when it preserves least upper bounds of ω-chains. Continuous maps in ω-CPOs always have a least fixpoint.

Lemma 4.4. *If $F : \mathcal{A} \to \mathcal{A}$ is a continuous map between ω-CPOs, then it has a least fixpoint $\mathrm{fix}(F)$.*

Lemma 4.5. *For any $a \in \Sigma$, the operations $B \mapsto a \cdot B$, $(A,B) \mapsto A \parallel B$ and $(A,B) \mapsto A + B$ are continuous maps. Moreover, if $f : \mathcal{A} \times \mathcal{B} \to \mathcal{B}$ is continuous, then the operation $a \mapsto \mathrm{fix}(b \mapsto f(a,b))$ defines a continuous map $\mathcal{A} \to \mathcal{B}$.*

4.2 Types as games

We now recall games from [Castellan et al. 2018] and establish a formal link with session types.

Definition 4.6. Given a set Σ, a Σ-***game*** is a $\Sigma \times \{-,+\}$-labelled event structure A such that the labelling function lbl_A is total, and if $e \smile\!\!\frown_A e'$, then $\pi_2(lbl(e)) = \pi_2(lbl(e'))$.

A game A is a set of *moves*, representing the possible messages of a protocol. Each move $e \in A$ comes with a *polarity* $pol_A(e)$ in $\{-,+\}$ indicating if the message must be sent or received. Given a Σ-game A, its ***dual*** $A^\perp$ is obtained by reversing the polarity of moves. Games of [Castellan et al. 2018] are Σ-games for $\Sigma = \{*\}$, but Σ will be useful to make the link with the syntax (messages, names). The last condition ensures that nondeterministic choices belong to one participant.

$$[\![\mathsf{end}]\!](\rho) = \emptyset \qquad [\![X]\!](\rho) = \rho(X) \qquad [\![\oplus_{i\in I} !l_i\langle \widetilde{S_i}\rangle.T_i]\!](\rho) = \sum_{i\in I} !l_i \cdot \left([\![\widetilde{S_i}]\!](\rho)^\perp \parallel [\![T_i]\!](\rho)\right)$$
$$[\![\mu X.\, P]\!](\rho) = \mathrm{fix}(E \mapsto [\![P]\!](\rho[X := E])) \qquad [\![\&_{i\in I} ?l_i(\widetilde{S_i}).T_i]\!](\rho) = \sum_{i\in I} ?l_i \cdot \left([\![\widetilde{S_i}]\!](\rho) \parallel [\![T_i]\!](\rho)\right)$$

Fig. 1. Interpretation of session types into games

The causal order and conflict relation on a game encode the *rules of the game* and specify which event can be played at a particular point of the protocol. In an event structure, two events e and e' can relate in three ways: they can be causally related, in conflict, or concurrent. These three possibilities make sense from a session type perspective:

- *Causality* represents the sequentiality constraint in a session type. For instance in $!\mathtt{tt}.?\mathtt{ff}$, the message $\mathtt{tt}$ must be sent before $\mathtt{ff}$ can be received.
- *Conflict* represents when two messages cannot occur together in an execution. For instance in $!\mathtt{tt} \oplus !\mathtt{ff}$, a process must choose between sending $\mathtt{tt}$ or $\mathtt{ff}$.
- *Concurrency* represents when two parts of the protocol can be performed independently. For instance in $?\mathtt{start}(!\mathtt{tt}).!\mathtt{ff}$, the messages $\mathtt{tt}$ and $\mathtt{ff}$ can be sent in any order or in parallel.

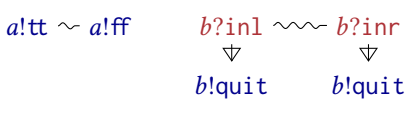
Using this analogy, we devise an interpretation of session types inside games.

Interpreting session types. We define the interpretation of session types into event structures. The events of the resulting event structure will correspond to actions allowed by the type, *i.e.,* will be labelled by extra information about the message sent and the channel on which it is sent. A type will thus be interpreted as a $\mathscr{L}$-game and a context as a $(\mathscr{N} \times \mathscr{L})$-game. To denote labels of moves, we use process-like notations: $!l$ for $(l, +)$, $?l$ for $(l, -)$ in $\mathscr{L} \times \{-, +\}$; and $a!l$ for $(a, l, +)$, $a?l$ for $(a, l, -)$ in $\mathscr{N} \times \mathscr{L} \times \{-, +\}$.

Because of recursive types, we will also have to interpret open types. Given a type T, we write $\mathrm{fv}(T)$ for the free variables occurring in it. Write $\mathbf{Games}(\mathscr{L})$ for the partial order of $\mathscr{L}$-games and inclusions, which is easily seen to be a sub-ω-CPO of $\mathbf{ES}(\mathscr{L} \times \{-, +\})$. An open type T will be interpreted by a continuous map $\mathbf{Games}(\mathscr{L})^{\mathrm{fv}(T)} \to \mathbf{Games}(\mathscr{L})$. Elements of $\mathbf{Games}(\mathscr{L})^{\mathrm{fv}(T)}$ can be seen as environments ρ mapping free variables of T to $\mathscr{L}$-games. The inductive definition of $[\![T]\!]$ is given in Figure 1. We write $[\![\widetilde{T}]\!]$ for $(\rho \mapsto [\![T^1]\!](\rho) \parallel \ldots \parallel [\![T^n]\!](\rho))$. The dual operator in the interpretation of selections is there to ensure that the interpretation commutes with duality:

Lemma 4.7. *If T is a type, $[\![\overline{T}]\!](\rho) = ([\![T]\!](\overline{\rho}))^\perp$ where $\overline{\rho}(X) = \rho(X)^\perp$.*

Given a $\mathscr{L}$-game A, and a name $a \in \mathscr{N}$, we write $a : A$ for the $(\mathscr{N} \times \mathscr{L})$-game obtained from A by relabelling: $lbl_{a:A}(e) = (a, lbl_A(e))$. Contexts are then simply interpreted as parallel composition of their (relabelled) components: $[\![a_1 : T_1, \ldots, a_n : T_n]\!] = a_1 : [\![T_1]\!], \ldots, a_n : [\![T_n]\!]$ assuming that all the types occurring in the context are closed. The game corresponding to the context $a : \mathbb{B}, b : (?\mathtt{inl}(!\mathtt{quit}) \,\&\, ?\mathtt{inr}.!\mathtt{quit})$ is depicted on the left. The labelling need not be injective:



in the picture on the left there are two events with label $b!\mathtt{quit}$. This is not a problem since the corresponding moves of the games are still distinct. We give a strategy on this game in § 5.2.

We now study two properties of this interpretation: (1) which games are denoted by types, and (2) which information is lost when interpreting a type.

4.2.1 Image of the interpretation. Types and contexts, due to their inductive nature, give rise to event structures which are forests. An event structure E is ***forest-like*** when for all $e \in E$, $[e]$ is

linearly ordered by $\leq_E$. Moreover, the nondeterminism of $[\![T]\!]$ is due to branching/selection, so $[\![T]\!]$ will be confusion-free. This is actually enough to characterise the image without recursion (provided one restricts to finite event structures). However, the interpretation of session types does not reach all the infinite forests in $\mathbf{Games}(\mathscr{L})$ – only the regular ones. An event structure E is ***regular*** [Thiagarajan 2002] when *(1)* the equivalence relation $\{(x, y) \in \mathscr{C}(E) \mid E/x \cong E/y\}$ has a finite number of equivalence classes and *(2)* any configuration can only have a finite number of immediate extensions. When E is a forest, E/x can be seen as the sub-forest rooted at x. Regularity ensures that E contains only a finite number of sub-forests. If Δ is a typing environment, the game $[\![\Delta]\!]$ can be regarded as a $\mathscr{L}$-game by discarding the information on names.

Lemma 4.8. *A $\mathscr{L}$-game is isomorphic (as $\mathscr{L}$-games) to a game of the form $[\![\Delta]\!]$ if and only if it is is forest-like, regular and confusion-free.*

4.2.2 An equational theory on types. Finally, we look at the equational theory induced by the model. As illustrated in § 2.1.2, the interpretation equates some types, *e.g.* $?(S).T$ and $?(T).S$ whose interpretation is $?() \cdot ([\![S]\!] \parallel [\![T]\!])$. We capture these equalities by the equivalence relation on session types, generated by the following rules, plus closure under context:

$$\&_{i\in I}?l_i(\widetilde{S_i}).T_i \equiv \&_{i\in I}?l_i(\sigma(\widetilde{S_i}, T_i, \mathrm{end})).\mathrm{end} \qquad \oplus_{i\in I}\, !l_i\langle\widetilde{S_i}\rangle.T_i \equiv \oplus_{i\in I}!l_i\langle\sigma(\widetilde{S_i}, \overline{T}_i, \mathrm{end})\rangle.\mathrm{end}$$

where σ denotes any permutation of the sequence of types. By applying repeatedly these rules, a number of end can be inserted or removed from the argument types. The equivalence $\equiv$ states that the order of argument types does not matter, and that argument types and continuation types can be permuted. Remark that to move the continuation to the argument type in the case of an output, it is necessary to take its dual. (This is related to the interpretation of types in games).

Proposition 4.9. *For session types S, T without recursion, $S \equiv T$ if and only if $[\![S]\!] \cong [\![T]\!]$.*

5 COINCIDENT STRATEGIES

We now introduce our notion of strategies, the semantic counterpart of processes. As discussed in § 3.5, strategies in the existing models of concurrency in game semantics only feature an asynchronous copycat which makes difficult to give meaning to free name passing. In this section, we generalise event structures to allow several moves to be played *at the same time*, creating ***coincidences***. In § 5.1, we introduce coincident event structures, obtained from event structures by moving from a causal partial order to a causal *preorder*. In § 5.2, we generalise the strategies of [Castellan et al. 2018] using these coincident event structures. Then, in § 5.3, we study the interaction of coincident strategies, crucial to define the composition of strategies and interpret the restriction operator. This composition is introduced in § 5.4, where we show that coincident strategies naturally organise themselves in a (compact-closed) category. We conclude this section in § 5.5, by showing weak bisimulation and observational equivalence coincide on strategies applying the technique in [Hennessy 2007]. This result will be used to prove that our model is intensionally fully abstract in § 6.2.

5.1 Coincident event structures

Traditional prime event structures cannot express the fact that two events must occur at the same time: concurrent (*i.e.*, causally incomparable) events may occur at the same time, but may also occur in a particular order. This is due to event structures being ***coincidence-free***: for each pair of event e, e' there exists a configuration x which separates them, *i.e.* contains one but not the other. Each configuration x can thus be reached (possibly in many ways) by a series of atomic steps:

$$\emptyset \xrightarrow{e_1} \{e_1\} \dots \xrightarrow{e_n} \{e_1, \dots, e_n\} = x$$

In prime event structures, this coincidence-freedom follows directly from the anti-symmetry of the causal relation $\leq_E$. By removing this axiom, *coincident events* become possible, as cycles for the causal order.

Definition 5.1. A Σ-labelled ***coincident event structure*** is a triple $(E, \leq_E, \#_E, lbl_E : E \rightharpoonup \Sigma)$ where $(E, \leq_E)$ is a preorder, $\#_E \subseteq E \times E$ is a symmetric irreflexive binary relation on E such that *(1)* $[e]$ is finite for $e \in E$ and *(2)* if $e \#_E e'$ and $e \leq e_0$ then $e_0 \#_E e'$.

We use similar notations as for event structures: $\mathscr{C}(E)$ denotes the set of finite, downclosed and conflict-free subsets of E, called ***configurations***. Minimal conflict is defined as before. We define $e \rightarrowtriangle e'$ in this new setting as $e < e'$, $\neg(e' < e)$ and there are no events in between e and e'. An event $e \in E$ is said to be ***visible*** when $lbl_E(e)$ is defined; ***invisible*** or ***internal*** otherwise. We write E_* for the set of internal events of E and E_v for the set of visible events of E, so that the set E splits into $E_* \cup E_v$. This decomposition applies to configurations: if $x \in \mathscr{C}(E)$ we write x_* (resp. x_v) for the set of internal (resp. visible) events of x.

We write $\equiv$ for the equivalence relation induced by the preorder $\leq$ (*i.e.* $e \equiv e'$ when $e \leq e'$ and $e' \leq e$). The equivalence classes for $\equiv$ are called ***coincidences***. Two coincident $e \equiv e'$ must occur *at the same time*, as they cannot be separated by a configuration: $e \in x$ iff $e' \in x$. We write $E_{/\equiv}$ for the set of coincidences of E. We tend to use $X, Y, \ldots$ for coincidences. A coincidence is ***trivial*** when it is a singleton. Given any subset $X \subseteq E$, we write $[X]$ for the downclosure of X in E and $[X) = [X] \setminus X$ as for event structures. We say that $x \overset{X}{\subset\!\!-} y$ when $y = x \cup X$, the union is disjoint, and if $x \subseteq z \subseteq y$, then $x = z$ or $z = y$ (with $x, y, z \in \mathscr{C}(E)$). In that case, X must be a coincidence.

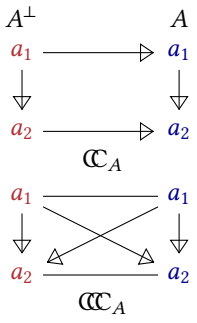
To illustrate coincidences, we give the definition of the ***(asynchronous) copycat*** $\mathfrak{C}_A$ used in traditional concurrent game semantics, and the ***coincident copycat***, a new feature of our model.

Definition 5.2. Given a game A, we define the coincident event structures $\mathfrak{C}_A$ (copycat) and $\mathfrak{CC}_A$ (coincident copycat) as follows. Their event set is $A^\perp \parallel A$, and their causal preorder:

$$\leq_{\mathfrak{C}_A} = (\leq_{A^\perp \parallel A} \cup \{((i,a),(1-i,a)) \mid i \in \{0,1\} \wedge pol_{A^\perp \parallel A}(i,a) = -\})^*$$
$$\leq_{\mathfrak{CC}_A} = (\leq_{A^\perp \parallel A} \cup \{((i,a),(1-i,a)) \mid i \in \{0,1\}\})^*$$

Conflict is then inherited from the game: two events e, e' are in conflict in $\mathfrak{C}_A$ (resp. $\mathfrak{CC}_A$) when $[\{e,e'\}]_{\mathfrak{C}_A}$ (resp. $[\{e,e'\}]_{\mathfrak{CC}_A}$) is not a configuration of $A^\perp \parallel A$. The identity function turns $\mathfrak{CC}_A$ and $\mathfrak{C}_A$ into $(A^\perp \parallel A)$-labelled event structures.

For the game $A = a_1 \rightarrowtriangle a_2$ (two positive moves in sequence), $\mathfrak{C}_A$ and $\mathfrak{CC}_A$ are depicted on the right. Coincidences are drawn with a straight line, rather than as causal loops. Parallel composition and sums extend to coincident event structures. Event structures come with a notion of map $f : E \rightharpoonup E'$ used to represent a (partial) simulation of E within E'. Maps allow to characterise some operations on event structures with universal properties, *e.g.*, the synchronised product of event structures as a categorical product.



We extend the traditional maps of event structures to coincident event structures as follows. For $f : E \rightharpoonup E'$ to be a valid simulation, when $e \equiv e'$ in E and both $f\,e$ and $f\,e'$ are defined then $f\,e$ and $f\,e'$ must be coincident or concurrent. This is justified by the fact that strategies will be maps of coincident event structures, and the interaction of coincident strategies satisfies a universal property relative to this class of maps (Theorem 5.8).

Definition 5.3. A ***map of coincident event structures*** $f : E \rightharpoonup E'$ is a partial function from E to E' satisfying: *(1)* if $x \in \mathscr{C}(E)$ then $f\,x \in \mathscr{C}(E')$; *(2)* if $e, e' \in x \in \mathscr{C}(E)$ and $f\,e = f\,e'$ then $e = e'$;

(3) if $e \equiv_E e'$, and $f\ e$ and $f\ e'$ are both defined, then they are either concurrent or coincident; and *(4)* for $e \in E$, $lbl_E(e)$ and $lbl_{E'}(f\ e)$ are both undefined, or both defined and equal.

The first two axioms are the same as for usual event structure maps. As a result, a map of coincident event structures $E \rightharpoonup E'$ when E and E' are event structures is simply a map of event structures as usually defined. A map $f : E \rightharpoonup E'$ is ***total*** when the underlying function is; an ***isomorphism*** when it has an inverse $g : E \to E'$: in this case we write $E \cong E'$.

The operation E/x can be easily generalised to coincident event structures, and is compatible with the notion of maps:

Lemma 5.4. *Let $f : E \rightharpoonup E'$ be a map of coincident event structures. For $x \in \mathscr{C}(E)$, the function $E/x \rightharpoonup E'/f\ x$ obtained by restricting f is well-defined and a map of coincident event structures.*

5.2 Coincident strategies

We introduce coincident strategies on games, generalising the strategies of [Castellan et al. 2018]. Consider a game A.

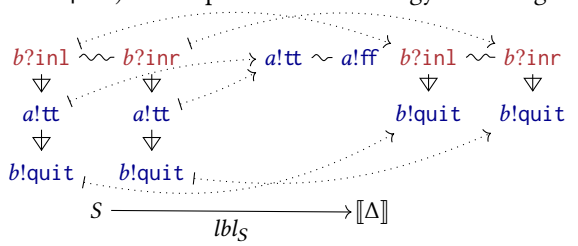
Definition 5.5. An A-labelled coincident event structure S is a ***coincident strategy*** on A when:

(1) Labelling is a map of coincident event structures $S \rightharpoonup A$.
(2) If $s \frown\!\!\!\sim_S s'$, then s and s' are either both internal, or both visible. Moreover, if they are visible, s is coincident to negative s_0 and s' to negative s'_0 such that $lbl(s_0) \frown\!\!\!\sim_A lbl(s'_0)$.
(3) If $X \in S_{\equiv}$ is non-trivial, then $X = \{s, s'\}$ where $lbl(s)$ and $lbl(s')$ are both defined and of distinct polarity in A.

The first condition ensures that strategies respect the rules of A and play every move at most once in a configuration. The second condition, **secrecy**, generalises the secrecy condition of [Castellan et al. 2018], forcing conflict between visible events to arise from a negative conflict in the game. The third condition severely restricts the shape of coincidences in S: the only non-trivial coincidences allowed are those with two visible events of opposite polarities, as in coincident copycat. Note that we do not impose any receptivity or courtesy conditions as in standard concurrent game semantics. This implies that the asynchronous copycat will not be an identity, but the coincident copycat will.

The assertion that S is a coincident strategy on A will often be written $S : A$. We write $\mathbf{CG}_S(A)$ for the set of coincident strategies on a game A. An A-labelled coincident event structure where the labelling function $lbl_S : S \rightharpoonup A$ is a map is called a ***coincident pre-strategy***. Isomorphism of labelled event structures gives a notion of isomorphism of strategies. Two strategies are isomorphic when they only differ by the "codes" (in an informal sense) of the events, but not by the structure, for instance $S = \{\alpha\}$ with $lbl_S(\alpha) = a$ and $S' = \{\beta\}$ with $lbl_{S'}(\beta) = a$ must be identified even if $\alpha \neq \beta$. In particular, the categorical laws (cf. § 5.4) will only hold up to isomorphism.

Remember the context in the example of § 4, $\Delta = a : \mathbb{B}, b : (?\mathtt{inl}(!\mathtt{quit}) \,\&\, ?\mathtt{inr}.\,!\mathtt{quit})$. The process $P = (b?\mathtt{inl}(x).\, a!\mathtt{tt}.\, x!\mathtt{quit}) \,\&\, (b?\mathtt{inr}.\, a!\mathtt{tt}.\, b!\mathtt{quit})$ is interpreted as the strategy on the right. In most cases, the information about the labelling to the game can be recovered so we will omit it, only drawing the left picture, as explained in § 2.1. Note that in this case, the labelling is not injective since $a!\mathtt{tt}$ is played twice, but the two events occur in incompatible branches.



Weak bisimulation. As coincident strategies are event structures, we can equip them with a labelled transition system using the remainder operation defined above. Note that if $S : A$ and

$x \in \mathscr{C}(S)$, it is easy to see that the map $S/x \rightharpoonup A/lbl_S(x)$ induced by Lemma 5.4 turns S/x into a coincident strategy on $A/lbl_S(x)$. Given a coincident pre-strategy S on A, we say that:

- $S \xrightarrow{\tau} S'$ when there exists an internal minimal event $s \in S$ such that $S/[s] \cong S'$
- $S \xrightarrow{Y} S'$ when there exists a visible coincidence X containing minimal events such that $lbl_S(X) = Y$ and $S/X \cong S'$ (as strategies on A/Y) (Y must be a set of minimal events of A).

Similarly, we can define weak transitions, in order to define weak bisimulation:

$$\overset{\tau}{\Rightarrow} := (\xrightarrow{\tau})^* \qquad \overset{Y}{\Rightarrow} := (\xrightarrow{\tau})^* \xrightarrow{Y}$$

Definition 5.6. A weak bisimulation is family of *equivalence* relations $(\mathcal{R}_A)_{A \in \mathbf{Games}}$ where $\mathcal{R}_A$ relates coincident pre-strategies on A, such that

- If $S \mathcal{R}_A T$, and $S \xrightarrow{\tau} S'$, then there exists a coincident strategy $T' : A$ with $T \overset{\tau}{\Rightarrow} T'$ and $S' \mathcal{R}_A T'$;
- If $S \mathcal{R}_A T$ and $S \xrightarrow{Y} S'$, then there exists $T \overset{Y}{\Rightarrow} T'$ and $S' \mathcal{R}_{A/Y} T'$.

The largest weak bisimulation is called ***weak bisimilarity*** and written $\approx$. Since isomorphism is clearly a weak bisimulation, $S \cong T$ implies $S \approx T$. As in process calculi, weak bisimulation offers a more concrete characterisation of observational equivalence (§ 5.5).

5.3 Interaction of coincident strategies

We have most of the ingredients to interpret the session π-calculus, except the restriction operator. Syntactically, the composition of $\vdash P \triangleright \Delta, \Delta_1$ with $\vdash Q \triangleright \overline{\Delta}, \Delta_2$ on the interface Δ is the process

$$\vdash Q \circledcirc_\Delta P := (\nu \widetilde{a})(P \mid Q) \triangleright \Delta_1, \Delta_2 \qquad \text{where } \Delta = a_1 : T_1, \ldots, a_n : T_n.$$

In game semantics, composition is primitive, and implemented in two steps. First P and Q ***interact*** over Δ, and then the actions on Δ in the resulting process are hidden. In this section, we start by defining the interaction of $S : A \parallel B$ and $S' : A^\perp \parallel C$, resulting in a $A \parallel B \parallel C$-labelled event structure $S *_A S'$. It is not a strategy, because of the disagreement over the polarity of moves of A.

We refer the reader to [Castellan et al. 2018, 2017] for detailed intuition about the behaviour of interaction. For lack of space, we give here an axiomatic definition, phrased as a universal property. The explicit definition can be found in the technical report [Castellan and Yoshida 2018]. Intuitively, the interaction $S * S'$ should only exhibit behaviour that S and S' have. To formulate a common behaviour between S and S', *i.e.* a behaviour that can be both simulated by S and S', we use maps of coincident event structures.

Definition 5.7. A ***common behaviour*** of S and S' is a triple (T, α, β) of a $(A \parallel B \parallel C)$-labelled coincident event structure T and two maps $\alpha : T \rightharpoonup S$ and $\alpha : T \rightharpoonup S'$ such that:

(1) If α and β are both defined on $t \in T$, then $lbl(t)$ is defined.
(2) There are no coincident $s_1, \ldots, s_n \in T$ with $\alpha\, s_1 \equiv \alpha s_2$, $\beta s_2 \equiv \beta s_3$, $\alpha s_3 \equiv \alpha s_4$, …, $\beta s_n \equiv \beta s_1$.

A common behaviour is a coincident event structure that both S and S' can simulate. We also add two conditions: the first one is taken from [Castellan et al. 2018], and amounts to saying that two processes cannot synchronise on neutral events: the neutral events of S are invisible to S' and vice versa. The second one prevents cycles between coincidences, which is necessary to prove functoriality of the encoding of session π-calculus in § 7.

The interaction should not just be any common behaviour, but the largest: it should be able to simulate any other common behaviour. A ***simulation*** from (T, α, β) to (T', α', β') is a partial map $\varphi : T \rightharpoonup T'$ with $\alpha' \circ \varphi = \alpha$ and $\beta' \circ \varphi = \beta$.

Theorem 5.8. *There exists a common behaviour of S and S' $(S *_A S', \Pi_S, \Pi_{S'})$ which is the largest common behaviour,* i.e. *for every other common behaviour (T, α, β) there exists a unique simulation $(T, \alpha, \beta) \to (S *_A S', \Pi_S, \Pi_{S'})$.*

5.4 The category of coincident strategies

Using the interaction defined in the previous section, we now define the composition of strategies and show it defines a compact-closed category of strategies. A coincident strategy from a game A to a game B will be, as usual in game semantics, a strategy on $A^\perp \parr B$. As a result, $\mathfrak{CC}_A$ is a strategy from A to A and will be the identity of our category. Composition will be obtained in a standard manner by interaction plus hiding.

Given a coincident strategy S on $A^\perp \parr B$ and R on $B^\perp \parr C$, we wish to build $R \circledcirc S$ to be a coincident strategy on $A^\perp \parr C$. Via the previous subsection, we can form their interaction $S *_B R$, written $R \circledast_B S$, which is a $(A^\perp \parr B \parr C)$-labelled event structure. To obtain a strategy on $A^\perp \parr C$, we need to hide the behaviour on B. A solution would be to simply relabel $R *_B S$ to consider events on B as being internal. On the one hand, most events on B do not represent anything meaningful about the *external* behaviour of S and T interacting together. On the other hand, hiding everything forgets nondeterministic branches without observable behaviour. In [Castellan et al. 2018], a compromise is found by keeping only the essential events:

Definition 5.9. An event of a Σ-coincident event structure S is ***essential*** when it is visible; or internal, and *(1)* there exists $s' \in S$ such that $s \frown\!\!\frown s'$ and *(2)* s is not coincident to a visible event.

The definition of essential events had to be updated to take into account coincidences. In particular, internal events coincident to visible ones are never useful since the visible events remember the conflict. This is essential to ensure that the composition will indeed be a coincident strategy, as $R \circledast_B S$ is only a pre-strategy (it may contain arbitrary large coincidences involving events synchronised on B). For S a Σ-labelled coincident event structure, we write $\mathscr{E}(S)$ for the coincident event structure consisting in the essential events of S, and causality and conflict inherited from S. As in [Castellan et al. 2018], hiding inessential events is sound up to weak bisimulation:

Lemma 5.10. *For any coincident pre-strategy S on A, we have $S \approx \mathscr{E}(S)$.*

Let us define $T \circledcirc S$ as $\mathscr{E}(T \circledast_B S)$ along with the labelling function $\mathscr{E}(T \circledast_B S) \rightharpoonup A \parr B \parr C \rightharpoonup A \parr C$ (note that, as far as labelling functions are concerned, polarity on the games does not matter). We obtain the desired coincident strategy:

Lemma 5.11. *$T \circledcirc S$ is a coincident strategy on $A^\perp \parr C$.*

Unlike in traditional asynchronous game semantics, the coincident copycat is an identity without any further asynchrony assumption on strategies.

Proposition 5.12. *For a coincident strategy S on A, $\mathfrak{CC}_A \circledcirc S \cong \mathscr{E}(S)$.*

Define a coincident strategy S to be ***essential*** when $\mathscr{E}(S) = S$ (*i.e.*, all its events are essential). From Proposition 5.12, it is easy to construct a compact-closed category:

Theorem 5.13. *We obtain a compact-closed category $\mathbf{CG}_S$ whose objects are games, and morphisms are essential coincident strategies up to isomorphism.*

To close the section, we remark that the strategies of [Castellan et al. 2018] arise as a special case of coincident strategies, which are asynchronous. A ***coincidence-free strategy*** is a coincident strategy S without non-trivial coincidences (*i.e.* S is an event structure).

Definition 5.14. A coincidence-free strategy S on A is ***asynchronous*** when:

Courtesy if $s \rightarrowtriangle s'$ but $lbl(s)$ and $lbl(s')$ are incomparable, then s is negative and s' positive.

Receptivity If $x \in \mathscr{C}(S)$ is such that $lbl(x) \overset{a}{\text{—}}\subset$ in A, with a negative, then there exists a unique $s \in S$ such that $lbl(s) = a$ and $x \overset{s}{\text{—}}\subset$.

Theorem 5.15 ([Castellan et al. 2018]). *Games and essential asynchronous strategies up to isomorphism form a compact-closed category* $\mathbf{CG}_A$.

The composition of asynchronous strategies defined in [Castellan et al. 2018] arises as a special case of the composition defined here. However, the coincident copycat is obviously not asynchronous, thus $\mathbf{CG}_A$ is not a subcategory of $\mathbf{CG}_S$.

Prefixing and neutral events. We have seen that if we have $S : A$, then we can prefix both S and A by a common event e (outside a) to obtain a game $e \cdot A$ and a strategy $e \cdot A$. In particular, we have the lifting strategy $L_e := (1, e) \cdot \mathfrak{CC}_A$ playing on $A^\perp \parallel (e \cdot A)$. In the next section, we need to prefix a strategy by an event, by composing with L_e. However, $L_e \circledcirc S$ is not isomorphic to $e \cdot S$ in general. This is only true if S has no minimal internal events. Indeed such minimal events will induce minimal internal events in $L_e \circledcirc S$. For this reason, we introduce the construction $e \rightarrowtriangle S$ as follows. The set of events and conflict is the same as $e \cdot S$ but the causality is given by the transitive closure of $\leq_S \cup \{e\} \times S_v$ (only visible events of S may depend immediately on e). We have:

Lemma 5.16. *For $S : A$, we have $L_e \circledcirc S \cong e \rightarrowtriangle S$.*

Recursion. As before, we can define inclusions between coincident event structures. A coincident event structure E is included in a coincident event structure F when $E \subseteq F$, E is downclosed in F and causality, and conflict from E and F coincide on events of E.

Lemma 5.17. $\mathbf{CG}_S(A)$ *along with inclusion forms an ω-CPO.*

As before, the main operations used to build strategies are continuous, in particular composition (remember that $\mathbf{CG}_S(A, B) = \mathbf{CG}_S(A^\perp \parallel B)$):

Lemma 5.18. *For a coincident strategy $S : A^\perp \parallel B$, the operation $R : B^\perp \parallel C \mapsto R \circledcirc_B S$ is a continuous map $\mathbf{CG}_S(B, C) \to \mathbf{CG}_S(A, C)$.*

5.5 Behavioural equivalence

Before interpreting the session π-calculus, we study the behavioural theory of our model. We define on strategies a semantic counterpart to the reduction-closed barbed congruence [Honda and Yoshida 1995; Milner and Sangiorgi 1992], and show it coincides with weak bisimulation. Since behavioural equivalences depend on the possible set of contexts, we will consider a parameter $\mathscr{M}$ called a ***submodel*** which is a subset $\mathscr{M}$ of games and for every $A, B \in \mathscr{M}$ a subset $\mathscr{M}(A, B) \subseteq \mathbf{CG}_S(A, B)$ such that:

(1) $\emptyset \in \mathscr{M}$ and $\mathscr{M}$ is stable under duality, parallel composition, prefix and remainder.
(2) If $S \in \mathscr{M}(A)$ and $x \in \mathscr{C}(S)$ contains only internal events, then S/x is in $\mathscr{M}(A)$.
(3) $\mathscr{M}$ is closed under composition.
(4) For every $S \in \mathscr{M}(A)$ and label e, $e \rightarrowtriangle S \in \mathscr{M}(e \cdot A)$.

where $\mathscr{M}(A)$ stands for $\mathscr{M}(\emptyset, A)$. We define the ***barbs*** of a coincident strategy $S : A$ to be the set $barb(S)$ of positive $e \in min(A)$ such that $S \overset{e}{\Rightarrow}$.

Definition 5.19. A family of equivalence relations $(\mathcal{R}_A \subseteq \mathscr{M}(A) \times \mathscr{M}(A))_{A \in \mathscr{M}}$ is a reduction-closed $\mathscr{M}$-congruence, when:

- If $S \mathcal{R}_A S'$ then $barb(S) = barb(S')$.

- If $S \mathcal{R}_A R$, and $S \xrightarrow{\tau} S'$, then there exists $R \overset{\tau}{\Rightarrow} R'$ with $S' \mathcal{R}_A R'$.
- If $S \mathcal{R}_A S'$ then for all game $B \in \mathscr{M}$ and strategy $U : A^\perp \parr B \in \mathscr{M}(A, B)$, $(U \circledcirc S) \mathcal{R}_B (U \circledcirc S')$.
- If $S \in \mathscr{M}(A)$ and $S' \in \mathscr{M}(A)$ are such that $(e \multimap S) \mathcal{R}_{e \cdot A} (e \multimap S')$, then $S \mathcal{R}_A S'$.

This definition is very similar to the definition in § 3.4, except for the last condition, which comes for free if $\mathscr{M}$ contains sufficiently many strategies. We include it so that Lemma 5.24 holds in full generality. Given a submodel $\mathscr{M}$, we write $\simeq_{\mathscr{M}}$ for the largest $\mathscr{M}$-congruence, and in particular $\simeq$ will be a shorthand for $\simeq_{\mathbf{CG}_S}$. The larger the submodel is, the more fine-grained the equivalence is, *i.e.* if $\mathscr{M} \subseteq \mathscr{M}'$, then $S \simeq_{\mathscr{M}'} S'$ implies $S \simeq_{\mathscr{M}} S'$ when S, S' belong to $\mathscr{M}$. To relate weak bisimulation and barbed congruence, we start by showing that weak bisimulation is a congruence.

Lemma 5.20. *If $S \approx S' : A^\perp \parr B$, then for all $R : B^\perp \parr C$, then $R \circledcirc S \approx R \circledcirc S'$.*

We deduce that weak bisimulation is a reduction-closed congruence for any class of observers:

Lemma 5.21. *For any submodel $\mathscr{M}$ of $\mathbf{CG}_S$ and $S, R \in \mathscr{M}(A)$, if $S \approx T$ then $S \simeq_{\mathscr{M}} S'$.*

5.5.1 Towards full abstraction. To show the converse, we apply a method based on the action tester from [Hennessy 2007]. Actions on a game A are either minimal events of A or pairs $\{a^-, b^+\}$ of minimal events of A. We will write $\alpha, \beta, \ldots$ for actions. Given a game A, and an action α, we will write $\mathrm{test}(A, \alpha)$ for the game $(\mathtt{succ}^+ \cdot A/\alpha) \parr \mathtt{fail}^+$.

Definition 5.22. A submodel $\mathscr{M}$ ***defines an action*** α of $A \in \mathscr{M}$ if there exists a strategy $U \in \mathscr{M}(A, \mathtt{test}(A, \alpha))$ such that:

- For all $S, S' \in \mathscr{M}(A)$ such that $S \xrightarrow{\alpha} S'$, $\mathit{barb}(U \circledcirc S) = \{\mathtt{succ}^+\}$ and $U \circledcirc S \overset{\tau}{\Rightarrow} (\mathtt{succ}^+ \multimap S')$.
- For $S \in \mathscr{M}(A)$, if $\mathit{barb}(U \circledcirc S) \overset{\tau}{\Rightarrow} S_0$ with $\mathit{barb}(U \circledcirc S) = \mathit{barb}(S_0) = \{\mathtt{succ}^+\}$ then $S_0 \overset{\tau}{\Rightarrow} (\mathtt{succ}^+ \multimap S')$ with $S \overset{\alpha}{\Rightarrow} S'$.

Definition 5.23. A submodel $\mathscr{M}$ is ***testing-closed*** if it defines all actions of games in $\mathscr{M}$.

Lemma 5.24. *If $\mathscr{M}$ is a testing-closed submodel, then for $S, R \in \mathscr{M}(A)$, we have: $S \approx R$ iff $S \simeq_{\mathscr{M}} T$.*

5.5.2 Definability of actions. As a first application of Lemma 5.24, we show that weak bisimulation and barbed congruence coincide in $\mathbf{CG}_S$ by showing that $\mathbf{CG}_S$ is testing-closed.

Consider a game A and an action α of A. We construct a coincident strategy on the game $A^\perp \parr \mathrm{test}(A, \alpha)$, as follows:

- If α is a singleton e, we define $U_e = (0, e) \cdot (1, \mathrm{succ}) \cdot \mathbb{CC}_{A/e} : A^\perp \parr \mathtt{succ}^+ \cdot (A/\alpha) \subseteq A^\perp \parr \mathrm{test}(A, \alpha)$
- If α is a set $\{a, b\}$, we define

$$\begin{aligned} U_{\{a,b\}} = {} & (* \cdot ((0, a) \parr (0, b)) \cdot (1, \mathtt{succ}^+) \cdot \mathbb{CC}_{A/\alpha}) \\ & + (* \cdot (0, a) \cdot (0, b) \cdot (1, \mathtt{fail}^+)) \\ & + (* \cdot (0, b) \cdot (0, a) \cdot (1, \mathtt{fail}^+)) \end{aligned}$$

The usage of `fail` is crucial for testing coincidences, since playing a and b in parallel does not rule out strategies starting with a and b in parallel, rather than coincidentally. Hence we need the other two branches that test a then b and b then a. If the strategy plays a and b concurrently rather than coincidentally, then in the composition this matches with all branches, hence triggering the fail.

Theorem 5.25 (Semantic full abstraction – $\mathbf{CG}_S$). *The coincident strategy U_α defines the action α for $\mathbf{CG}_S$. As a result, $\mathbf{CG}_S$ is testing-closed, and $S \simeq R$ iff $S \approx R$ for $S, R \in \mathbf{CG}_S(A)$.*

This result is fundamental to prove that our interpretation is fully-abstract. It also shows that this semantic space is well-behaved from a concurrency theory point of view. Interestingly, we can also apply this lemma to show that $\mathbf{CG}_A$ is fully abstract:

$$[\![\textstyle\sum_{i\in I} P_i]\!](\gamma) = \textstyle\sum_{i\in I} * \cdot [\![P_i]\!](\gamma) \quad [\![P \mid P']\!](\gamma) = [\![P]\!](\gamma) \parallel [\![P']\!](\gamma) \quad [\![X]\!](\gamma) = \gamma(X)$$
$$[\![\&_{i\in I}\, u?l_i(\widetilde{x_i}).P_i]\!](\gamma) = \textstyle\sum_{i\in I} u?l_i \cdot [\![P_i]\!](\gamma)$$
$$[\![u!l_k\langle\widetilde{v}\rangle.P]\!](\gamma) = u!l_k \cdot ([\![P]\!](\gamma) \parallel \mathfrak{CC}_{[\![\widetilde{T}]\!]}) \text{ where } \widetilde{v} : \widetilde{T}$$
$$[\![\mu X.P]\!](\gamma) = \mathrm{fix}(S \mapsto [\![P]\!](\gamma[X := S])) \quad [\![(\nu a{:}T)P]\!](\gamma) = [\![P]\!](\gamma) \circledcirc_{[\![\overline{a}:\overline{T}, a:T]\!]} \mathfrak{CC}_{[\![T]\!]} \quad [\![\mathbf{0}]\!](\gamma) = \emptyset$$

Fig. 2. Interpretation of session π-calculus as coincident strategies

Theorem 5.26 (Semantic full abstraction – $\mathbf{CG}_A$). *The submodel $\mathbf{CG}_A$ is testing-closed, hence for any asynchronous strategies $S, R : A$, we have:*

$$S \simeq_{\mathrm{CG}_S} R \quad \textit{iff} \quad S \approx R \quad \textit{iff} \quad S \simeq_{\mathrm{CG}_A} R.$$

The proof is by taking the same testing strategies and replacing occurrences of the coincident copycat with the asynchronous copycat. This result shows that coincident strategies do not add any observational power to asynchronous strategies. However, the submodel of coincidence-free strategies is not testing-closed because there is no identity. This is related to Example **??**.

6 SESSION PROCESSES AND COINCIDENT STRATEGIES

In this section, we give an interpretation of session processes in terms of coincident strategies: given a closed process $\vdash P \triangleright \Delta$, we construct a strategy $[\![P]\!]$ on the game $[\![\Delta]\!]$ (§ 6.1). We then show that this interpretation is fully abstract (§ 6.2), *i.e.* the interpretation preserves and reflects barbed congruence. Our proof of full abstraction does not use the standard finite definability argument as it is not enough to define finite tests in this nondeterministic setting, but instead relies on the development of § 5.5, showing the action testers in § 5.5.2 are definable in the syntax. Nevertheless, finite definability is an interesting result on its own and we conclude this section by showing a result of finite definability for the internal session π-calculus (§ 6.3), hence showing that internal processes form a programming language for coincidence-free strategies.

6.1 Interpreting session π-calculus

We now have all the ingredients to interpret the session π-calculus. Recursive processes are interpreted similarly to recursive types, as fixpoints of continuous maps. We assume all types in recursion contexts are closed. Given a context $\Gamma = (X_1{:}\Delta_1), \ldots, (X_n{:}\Delta_n)$ we map it to the category of embeddings $[\![\Gamma]\!] = \mathbf{CG}_S([\![\Delta_1]\!]) \times \ldots \times \mathbf{CG}_S([\![\Delta_n]\!])$. Then a judgement $\Gamma \vdash P \triangleright \Delta$ is interpreted as a continuous map $[\![\Gamma]\!] \to \mathbf{CG}_S(\Delta)$. The interpretation is defined on Figure 2. For branching processes, the continuation is prefixed with an event encoding the message received. When interpreting the selection, the synchronous forwarder links the subgame for u, and the subgames for $\widetilde{v}$. For a nondeterministic sum, we prefix the strategies with internal events in order to remember the branching point. A parallel composition is mapped to a parallel composition (with no interaction) since a reduction only occurs under a name restriction. The interpretation of restriction is obtained by composed by copycat: here, $\mathfrak{CC}_{[\![T]\!]}$ is a map from the empty game to $([\![T]\!]^\perp \parallel [\![T]\!])$, composed with the interpretation of P viewed as a coincident strategy from $([\![T]\!]^\perp \parallel [\![T]\!])$ to $[\![\Delta]\!]$.

Lemma 6.1. *Consider $\Gamma \vdash P, Q \triangleright \Delta$. If $P \equiv Q$ then $[\![P]\!](\gamma) \cong [\![Q]\!](\gamma)$ for $\gamma \in [\![\Gamma]\!]$.*

We notice that composition of processes, and asynchronous forwarders are definable.

Lemma 6.2. *Consider a context $\Delta = a_1 : A_1, \ldots, a_n : A_n$, and $\vdash P \triangleright \Delta, \Delta_P$ and $\vdash Q \triangleright \overline{\Delta}, \Delta_Q$. We have*

$$[\![Q]\!] \circledcirc_{[\![\Delta]\!]} [\![P]\!] \cong [\![Q \circledcirc_\Delta P]\!] \; (= [\![(\nu\widetilde{a})(P \mid Q)]\!]).$$

Moreover, for a type T without recursion, we have $[\![[u = v]_T]\!] = \mathbb{C}_{[\![T]\!]}$.

6.2 Full abstraction

In this subsection, we prove that our interpretation is fully abstract. First we state a correspondence between the operational semantics and the model:

Lemma 6.3 (Soundness and adequacy). *Let $\vdash P \triangleright \Delta$ be a well-typed process. Then we have: (1) if $P \to^* Q$, then $[\![P]\!] \stackrel{\tau}{\Rightarrow} [\![Q]\!]$; and (2) if $[\![P]\!] \stackrel{\tau}{\Rightarrow} S$ then $S = [\![Q]\!]$ with $P \to^* Q$.*

This result induces a correspondence at the level of barbs. However, in the syntax, recall that a process P that contains both a name and its coname does not have visible barbs on these names, yet in the current semantics, actions on these names are visible. To solve this mismatch, from $[\![P]\!]$, we define another interpretation $(\!|P|\!)$ which hides actions on names that should not be observable. Given $\vdash P \triangleright \Delta$, consider the decomposition of Δ of the form $\Delta_0, \overline{\Delta_0}, \Delta_1$ where Δ_1 does not contain a name and its coname. We write $(\!|P|\!) = [\![P]\!] \circledcirc_{[\![\Delta_0, \overline{\Delta_0}]\!]} \mathbb{C}_{[\![\Delta_0]\!]}$, which is a strategy on $[\![\Delta_1]\!]$.

Lemma 6.4. *Let $\vdash P \triangleright \Delta$. Then $P \Downarrow_a$ if and only if $(\!|P|\!) \stackrel{a!l}{\Longrightarrow}$ for some $l \in \mathscr{L}$.*

To derive full abstraction of the model, we reuse Lemma 5.24. Define $\mathscr{C}_\pi$, the submodel comprising of the games of the form $[\![\Delta]\!]$, and given two games $[\![\Delta]\!]$, $[\![\Delta']\!]$, $\mathscr{C}_\pi([\![\Delta]\!], [\![\Delta']\!])$ comprises all $(\!|P|\!)$ when $\vdash P \triangleright \overline{\Delta}, \Delta'$. It is a routine verification to show that $\mathscr{C}_\pi$ is indeed a submodel. By finding process equivalents of the strategies used in the previous section, we obtain:

Theorem 6.5. *$\mathscr{C}_\pi$ is testing-closed. As a result, for $\vdash P, Q \triangleright \Delta$, $P \simeq Q$ if and only if $(\!|P|\!) \simeq (\!|Q|\!)$.*

Via this result, we can show that the interpretation of a process is asynchronous if and only if it is invariant under composition with the asynchronous forwarder.

Lemma 6.6. *Consider a context $\Delta = a_1 : A_1, \ldots, a_n : A_n$ which does not contain a name and its coname. For $\vdash P \triangleright \Delta$, $[\![P]\!]$ is an asynchronous strategy if and only if $(\nu \widetilde{a})(P \mid [\widetilde{a} = \widetilde{b}]) \simeq P\{\widetilde{b}/\widetilde{a}\}$.*

In this case, we say that P is ***latency-independent***.

6.3 Finite definability and internal processes

One of the goals of the correspondence between session types and game semantics is to be able to use session processes as a syntax for strategies. However not all coincident strategies are definable by a session process due to arbitrary coincidences. In this section, we turn our attention to the coincidence-free fragment of the model, corresponding to pre-strategies of [Castellan et al. 2018].

As for types, we now try to understand which conditions on coincident strategies are necessary for the converse to hold. A confusion-free coincidence-free strategy S on a game A is ***matching-exhaustive*** when if $\mathfrak{c}$ is a visible cell of S, then $lbl(\mathfrak{c})$ is a cell of A. This ensures that a strategy does not forget branches in an external choice. A coincidence-free strategy $S : A$ is ***internal*** when S is confusion-free and matching-exhaustive.

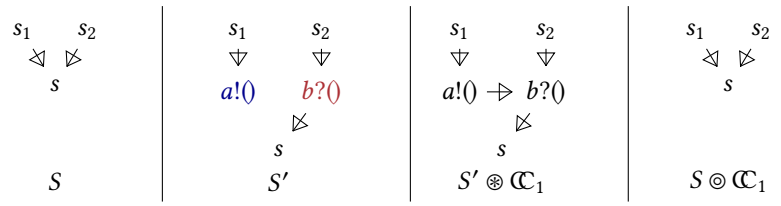
Lemma 6.7. *Let $\vdash P \triangleright \Delta$ be an internal session process. Then $[\![P]\!]$ is an internal strategy.*

Moreover, for internal strategies which are forests it is easy to define them by induction:

Lemma 6.8. *Let $S : [\![\Delta]\!]$ be a* finite *internal strategy such that S is a forest. Then there exists an internal process P such that $[\![P]\!] = S$.*

The main difficulty is to deal with joins. We need to define a strategy whose causal pattern can be arbitrary. First, let us define the causal complexity of a strategy. Given $s \in S$, we write $pred(s)$ for

the set of its immediate predecessors, *i.e.*, the set of events $s' \in S$ such that $s' \rightarrowtriangle s$. If S is a forest, then $pred(s)$ is at most one for all $s \in S$. Then, we define the ***causal complexity*** of an event $s \in S$: (1) $\mathrm{cc}(s) = 0$ if $pred(s)$ has cardinality ≤ 1; (2) else $\mathrm{cc}(s) = \sum_{s' \rightarrowtriangle s} |[s']|$. We then define the ***causal complexity*** $\mathrm{cc}(S)$ of a finite internal strategy $S : A$ to be the sum of the causal complexity of all its events. Note that $\mathrm{cc}(S) = 0$ if and only if S is forest-like. We show how to inductively decrease the causal complexity of a strategy. Consider a join in $S : A$ as in the left of the diagram below. We then rewrite this strategy into S', which now plays on a larger arena $[\![a : \bot, b : 1]\!]^{\bot} \parallel A$. Here S' has a lower causal complexity than S but not $s_1 \leq s$. This dependence can be recovered by composition with asynchronous copycat, as illustrated below:



Lemma 6.9. *Let $S : [\![\Delta]\!]$ be an internal strategy. If S has causal complexity > 0, then there exists $S' : [\![a : \bot, b : 1]\!]^{\bot} \parallel A$ such that $\mathrm{cc}(S') < \mathrm{cc}(S)$ and $S \cong S' \circledcirc_{[\![a:\bot,b:1]\!]} \mathbb{CC}_{[\![1]\!]}$.*

Since asynchronous copycat and composition are definable, we obtain by induction:

Theorem 6.10. *If $S : [\![\Delta]\!]$ is finite internal, there exists an internal process P with $[\![P]\!] \cong S$.*

Since asynchronous strategies are automatically internal, we conclude that finite asynchronous strategies are all definable by processes. Hence there is a two-way correspondence between latency-independent processes and asynchronous strategies.

On the one hand, it is not clear the class of infinite event structures that internal processes (with recursion) define: it is more than regular event structures. On the other, augmenting the syntax with infinitary parallel composition and restriction is enough to define all infinite event structures.

7 FROM COINCIDENT STRATEGIES TO ASYNCHRONOUS STRATEGIES

This section presents an encoding of coincident strategies into asynchronous strategies, which explains how to represent synchronous strategies as particular asynchronous strategies. We encode synchronous actions as a pair of asynchronous actions of dual polarities, a request, followed by an acknowledgement. As an application of our correspondence, this encoding can be lifted at the level of types and processes.

Syntactically, a type T (resp. a game A) is unfolded into a protocol $\uparrow T$ (resp. a game $\uparrow A$) where every action is duplicated into a request and an acknowledgement. For instance, $\uparrow\mathbb{B} = (!\mathtt{tt}.?\mathsf{ack}) \oplus (!\mathtt{ff}.?\mathsf{ack})$.

Processes and strategies on T or A are lifted on $\uparrow T$ or $\uparrow A$. For instance, consider the process $P = a!\mathtt{tt}.b!\mathtt{ff}$ on the context $\Delta = a : \mathbb{B}, b : \mathbb{B}$. It is latency-*dependent* due to the direct syntactic dependence between two positive actions on ***unrelated*** channels, hence requires to know when its send has been received to be able to continue. It can be unfolded into a latency-independent process respecting the unfolded types:

$$\vdash P' := a!\mathtt{tt}.a?\mathsf{ack}.b!\mathtt{ff}.b?\mathsf{ack} \triangleright a : \uparrow\mathbb{B}, b : \uparrow\mathbb{B},$$

P' need not to know when its outputs are received since it can simply wait for the acknowledgement.

We first start by explaining how protocols are unfolded. Then § 7.1 defines the encoding of strategies, and § 7.2 characterises the image of the encoding. Finally, we study properties of the encoding in § 7.3 and show how it can be used to build a syntactic translation of (finite) processes.

Definition 7.1. Let A be a $\mathscr{L}$-game. We define $\uparrow A$ as follows:

Events $A \times \{\mathsf{r}, \mathsf{a}\}$
Causality generated by $(e, \mathsf{r}) \rightarrowtriangle (e, \mathsf{a})$ and $(e, \mathsf{a}) \rightarrowtriangle (e', \mathsf{r})$ when $e \rightarrowtriangle_A e'$.
Conflict $(e, \alpha) \# (e', \beta)$ when $e \# e'$.
Labelling (to $\mathscr{L} \times \{-, +\}$): $lbl(e, \mathsf{r}) = lbl(e)$ and $lbl(e, \mathsf{a}) = (\mathsf{ack}, -pol_A(e))$.

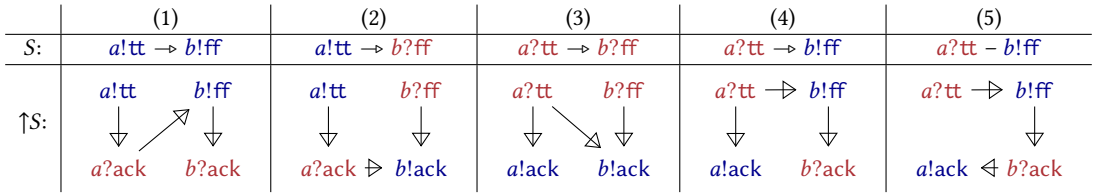
The translation preserves most properties and constructions on games. Moreover, through the results of § 4, this induces a translation on session types $T \mapsto \uparrow T$ that can be described syntactically:

$$\uparrow \mathsf{end} = \mathsf{end} \quad \uparrow \mathbf{t} = \mathbf{t} \quad \uparrow(\mu \mathbf{t}.T) = \mu \mathbf{t}.\uparrow T$$

$$\uparrow \&_{i \in I} ?l_i(\widetilde{S_i}).\, T_i = \&_{i \in I} ?l_i.\, !\mathsf{ack}\langle \widetilde{\uparrow S_i} \rangle.\, \uparrow T_i \qquad \uparrow \oplus_{i \in I} !l_i \langle \widetilde{S_i} \rangle.\, T_i = \oplus_{i \in I} !l_i.\, ?\mathsf{ack}(\widetilde{\uparrow S_i}).\, \uparrow T_i$$

7.1 Encoding of strategies

We define an encoding from a coincident strategy S on A into an asynchronous strategy $\uparrow S$ on $\uparrow A$. Contrary to the translation of games, which can be easily carried at the syntax level for types, this translation relies on the causal structure offered by the model. The difficulty of this encoding is to translate causal patterns which are not necessarily courteous into courteous ones (*i.e.* $\ominus \rightarrowtriangle \oplus$). Before formally defining the encoding, we list the possible causal patterns in coincident strategies and what their encoding will be below.



In (1), there is an immediate causality between two positive actions which is translated to a strategy below. (4) is a translation of a courteous causality, while (5) is of a coincidence. They only differ in how the acknowledgement is handled: when translating a coincidence $a?\mathtt{tt} - b!\mathtt{ff}$, the acknowledgement of $a?\mathtt{tt}$ depends on the acknowledgement of $b!\mathtt{ff}$, which is not the case for the translation of $a?\mathtt{tt} \rightarrowtriangle b!\mathtt{ff}$. This translation was guided by a principle: there should be a bijection between downclosed subsets of the source partial order, and *complete* downclosed subsets of the target partial order, *i.e.* those closed under acknowledgements.

Defining the encoding in one step would technically involved, so we proceed in two steps. First, given a coincident strategy S on A, we build a coincidence-free strategy $\overline{S}$ on the game $\uparrow A$. Remember that S splits in $S = S_* \cup S_v$ where S_v is the set of visible events and S_* the set of internal events. Consider the set $\overline{S} := S_* \times \{*\} \cup S_v \times \{\mathsf{r}, \mathsf{a}\}$, along with labelling function $\overline{lbl_S} : \overline{S} \rightharpoonup \uparrow A$, mapping (s, α) to $(lbl_S(s), \alpha)$ when defined; and undefined otherwise. A subset $X \subseteq \overline{S}$ is consistent when its first projection $\pi_1(X)$ is consistent in S. The preorder $\leq_{\overline{S}}$ is generated by the transitive and reflexive closure of:

$$(s, \mathsf{r}) \prec (s, \mathsf{a}) \text{ for } s \in S$$
$$(s, \mathsf{r}) \prec (t, \mathsf{r}), (t, \mathsf{a}) \prec (s, \mathsf{a}) \text{ when } s^- \equiv t^+$$
$$(s, \mathsf{a}) \prec (t, \mathsf{r}) \text{ when } s \rightarrowtriangle t,\ s, t \in S_v$$

$$(s, \mathsf{a}) \prec (t, *) \quad \text{when } s \rightarrowtriangle t \text{ and } s \in S_v, t \in S_*$$
$$(s, *) \prec (t, \mathsf{r}) \quad \text{when } s \rightarrowtriangle t \text{ and } s \in S_*, t \in S_v$$
$$(s, *) \prec (t, *) \quad \text{when } s \rightarrowtriangle t \text{ and } s, t \in S_*$$

The resulting preorder still contains non-courteous causal links, but it is antisymmetric:

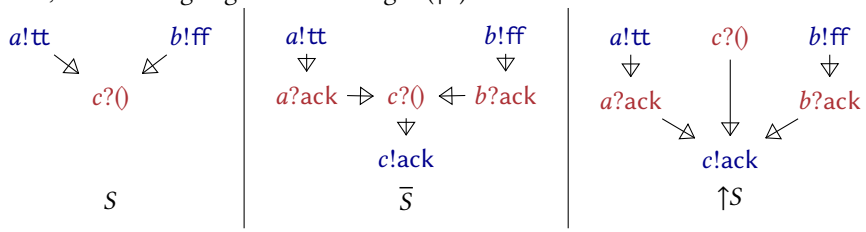
Lemma 7.2. *$\overline{S}$ is a coincidence-free strategy on $\uparrow A$.*

To turn $\overline{S}$ into an asynchronous strategy, we borrow this result from [Castellan et al. 2018]:

Lemma 7.3 ([Castellan et al. 2018]). *If S coincidence-free, then $\alpha_A \circledcirc_A S$ is asynchronous.*

As a result, we define $\uparrow S$ as the asynchronous strategy $\alpha_{\uparrow A} \circledcirc_{\uparrow A} \overline{S}$.

Example 7.4. Consider the process $P = (\nu d)(a!\mathtt{tt}.\overline{d}! \mid b!\mathtt{ff}.d?.\, c?)$. Its interpretation is on the left (S). Via $\overline{S}$, its encoding is given on the right ($\uparrow S$):



The strategy on the right is the interpretation of $(\nu dd')(a!\mathtt{tt}.\, a?\mathrm{ack}.\, \overline{d}! \mid b!\mathtt{ff}.\, b?\mathrm{ack}.\, \overline{d'}! \mid c?.\, d?.\, d'?.\, c!\mathrm{ack})$. It illustrates that the translation is not inductive on the syntax of processes but relies on the causal structure.

Because composing with copycat removes all non courteous causal links, $\uparrow S \cong \uparrow T$ does not imply $\overline{S} \cong \overline{T}$. However, by adding redundancy to the step from S to $\overline{S}$, the whole process is injective. Moreover, the encoding also preserves the categorical structure:

Theorem 7.5. *The operation $\uparrow$ defines a faithful functor $\mathbf{CG}_S \to \mathbf{CG}_A$,* i.e.,

$$(1)\ \uparrow \mathbb{CC}_A \cong \mathbb{C}_{\uparrow A} \qquad (2)\ \uparrow(R \circledcirc S) \cong \uparrow R \circledcirc \uparrow S \qquad (3)\ \uparrow R \cong \uparrow S \Rightarrow S \cong R$$

$\mathbf{CG}_S$ is thus isomorphic to a subcategory of $\mathbf{CG}_A$. We now characterise this subcategory.

7.2 Acknowledging strategies

Consider an asynchronous strategy $S : \uparrow A$. Define $\downarrow S$ to be the set of $s \in S$ which are either internal events, or acknowledged requests, *i.e.*, requests such that there exists $s' \in S$ with $\sigma\, s \rightarrowtriangle \sigma\, s'$. Define the relation $\preceq_S$ on $\downarrow S$ as follows (reflexive but not transitive in general):

$$s \preceq_S s' := \begin{cases} s \leq_S s' & s' \text{ neutral} \\ s \leq_S s'_0 & s'_0 \text{ an acknowledgement of } s' \end{cases}$$

Lemma 7.6. *Let $\sigma : S \rightarrowtriangle A$ be a coincident strategy. There exists a function $\varphi : \downarrow\uparrow S \to S$ such that $s \preceq_S s'$ if and only if $\varphi\, s \leq \varphi\, s'$.*

Definition 7.7. An asynchronous strategy $\sigma : S \rightarrowtriangle \uparrow A$ is ***well-acknowledging*** when

(1) $X \in \mathrm{Con}_S$ iff $[X \setminus \{s \in X \mid \sigma\, s \text{ defined and an acknowledgement}\}] \in \mathrm{Con}_S$.
(2) The relation $\preceq_S$ is a preorder on $\downarrow S$ (*i.e.* it is transitive).
(3) If $s \preceq_S s'$ and $s' \preceq_S s$, then s and s' have distinct polarities.
(4) If s is a non-maximal request, then it is acknowledged.
(5) There is no internal s_0 with $s < s_0 < s'$ when s is a request and $\sigma\, s \rightarrowtriangle \sigma\, s'$.

Lemma 7.8. *For any $\sigma : S \rightarrowtriangle A$, the strategy $\uparrow\sigma$ is well-acknowledging.*

Given S an asynchronous strategy on $\uparrow A$, we write $\downarrow S$ for the triple $(\downarrow S, \preceq_S, \mathrm{Con}_S \cap \mathscr{P}(\downarrow S))$, with labelling mapping $s \in \downarrow S$ to a whenever $lbl_S(s) = (a, \mathsf{a})$.

Proposition 7.9. *If S is well-acknowledging, then $\downarrow S$ is a coincident strategy. Moreover $\uparrow\downarrow S \cong S$.*

Since the construction $\uparrow\cdot$ is easily seen to be injective on games, we obtain:

Theorem 7.10. *The category $\mathbf{CG}_S$ is isomorphic to the subcategory of $\mathbf{CG}_A$ whose objects are games of the form $\uparrow A$ and morphisms well-acknowledging (asynchronous) strategies.*

7.3 Properties of the encoding

We show that our encoding is *sound*: it reflects weak bisimulation (hence barbed congruence)

Lemma 7.11. *For $S, R : A$, if $\uparrow S \approx \uparrow R$ then $S \approx R$.*

However, as the completeness direction of the encoding from synchrony into asynchrony fails in the π-calculus [Yoshida 1996], the converse direction does not hold (cf. Example **??** in Appendix).

The semantic encoding can be lifted to finite processes via Theorem 6.10.

Definition 7.12. Given a *finite* process $\vdash P \triangleright \Delta$, define $\uparrow P$ to be a process defining the strategy $\uparrow[\![P]\!]$ on the game $\uparrow[\![\Delta]\!] \cong [\![\uparrow\Delta]\!]$ via Theorem 6.10.

By construction, $\vdash \uparrow P \triangleright \uparrow\Delta$. Using the full abstraction result, we deduce immediately that the encoding is sound:

Lemma 7.13. *For finite $\vdash P, Q \triangleright \Delta$, if $\uparrow P \simeq \uparrow Q$ then $P \simeq Q$.*

This allows any process which expects a synchronous network to be lifted to a process that has the same behaviour on a network where synchronising actions are not available. Unlike the traditional untyped encoding of the synchronous π-calculus into the asynchronous π-calculus, this encoding does not rely on name-passing but simply on passing acknowledgements.

8 RELATED WORK AND CONCLUSION

Related work. Recently, Disney and Flanagan [2015] applied game semantics to prove the soundness of typing system for extensions of the λ-calculus by translating them as strategies represented by processes. Honda and Yoshida [1999] recast the traditional encoding of the call-by-value λ-calculus into the π-calculus by Milner [1992] into a game semantics model for call-by-value PCF. Our paper applies the proof technique from the π-calculus (definability of *action testers* in [Hennessy 2007]) to game semantics to obtain full abstraction where the standard finite definability (which works only for may equivalence) is insufficient due to reduction-closedness of the barbed congruence.

On concurrent game semantics, Laird [2005] was the first to devise a model of the asynchronous π-calculus. The nondeterminism of the model being angelic, the model is only fully abstract for may equivalence. The model rests on plays as traces and thus is unable to account precisely for causality between actions. The departure from trace-based games model to truly concurrent ones was initiated by [Abramsky and Melliès 1999] using closure operator to represent deterministic computation. The first model using explicitly causal structures is due to [Melliès and Mimram 2007], which introduces the asynchronous copycat, as well as the local condition, *courtesy*, ensuring that it is an identity. This causal structure is used to prove positionality of innocent strategies and deduce a fully complete model of linear logic [Melliès 2005]. Eberhart et al. [2013] have a model of the synchronous π-calculus using similar ideas, fully abstract for fair convergence. However, their model does not support any notion of hiding, which makes the representation of programs very large; and they do not study the link between types and games. The first model of a π-calculus dialect based on this causal approach in game semantics is due to Sakayori and Tsukada [2017] using pomsets to devise a model of the asynchronous π-calculus, fully abstract for may. More

recently, concurrent game semantics has been applied to provide a semantic proof of soundness for concurrent separation logic [Melliès and Stefanesco 2018].

The idea of representing coincidences as causal loops is first presented in [Ghica and Menaa 2011], which outlines how to build a sequential and deterministic model in game semantics of synchronous computation. It is also proposed in [Melliès 2019], to which our model is related. Since the domain of configurations of a coincident event structure can be seen as a category of positions, our games and strategies arise as a special case of the abstract setting put forward in [Melliès 2019]. With this embedding, our synchronous copycats coincide, but it is not entirely clear to us what the status of composition is. Domains of configuration are not composed using pullbacks, but as a subjobject of the pullback where deadlocks have been removed. It would be interesting to investigate whether our category can be recovered as a subcategory of Melliès', maybe by changing the base category of polarities, which is the main parameter of the construction of [Melliès 2019].

The line of work [Crafa et al. 2007; Cristescu et al. 2015] also models processes by event structures, however in these models, labels carry information about the names that are passed or received, unlike in ours. This representation of actions makes the models combinatorially more involved and does not offer fully abstract or definability results.

Our definability result is related to [van Glabbeek and Vaandrager 2003], tackling the problem of definability of event structures by expressions of CCS or CSP. We define event structures using a name-passing calculi, the internal session π-calculus, rather than a value-passing one, which relies on the game structure.

The analysis of synchrony and asynchrony at the level of strategies is reminiscent to Selinger [Selinger 1997], where asynchrony properties on labelled transition systems are defined in terms of equations satisfied by the composition of the transition system with specific agents, representing buffers or queues, just like we do with asynchronous copycat. Our setting using games allows for more general LTSs than those simply generated by sets of input and output labels. We leave for future work understanding the exact relationship between the two frameworks.

Implementation. We have implemented a prototype for the model, available online at http://sessiontypesandgames.github.io/. The implementation is mainly an illustration of the model, and intended to help readers to familiarise themselves with the model. It computes the interpretation of a (recursion-free) process, and draws the coincident event structure. Coincident event structures are not represented using a graph representation as drawn in the paper, but as a labelled transition system enhanced with causal information. This allows us to represent infinite event structures lazily (even though we do not take advantage of this in the implementation). The transition system can then be explored (with possibly a bound) to recover the event structure, as a graph.

Because of this representation, the implementation does actually list the traces (where each event of the trace lists the events depended on causally), which only differ due to nondeterministic choices, and not due to the choice of the scheduler. For this reason, our implementation *directly* gives the minimal set of traces of a process, and as such is related to partial order reduction tools which approximate this minimal set (in more general contexts).

Future work. We believe that this model can be applied to other areas related to synchronous computation: for instance to the geometry of synthesis and hardware circuits [Ghica and Menaa 2011], or the Applied π-calculus [Abadi et al. 2017] where a truly concurrent model could help with checking cryptographic properties related to trace-equivalence [Baelde et al. 2015].

We would also like to extend the correspondence, to support: (1) races between different actions, in order to represent nonlinear channels or shared memory and (2) nonlinear "symmetric" actions (corresponding to unrestricted channels in the π-calculus and copy indices in strategies).

From the session types perspective, a link with causal game semantics could offer a causal account of asynchronous buffered session semantics [Gay and Vasconcelos 2010] and multiparty session types [Honda et al. 2008]; a correspondence with a relational model of a linear logic based session calculus [Atkey 2017]; a semantic proof of deadlock-freedom; a generalisation of binary session types to types that do not denote forest-like games.

Acknowledgement. This work has been partially sponsored by: EPSRC EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1, and EP/N028201/1. We would like to thank Pierre Clairambault, Sung-Shik Jongmans, and Hugo Paquet for helpful comments and discussions, as well as the POPL reviewers for their feedback.

REFERENCES

Martín Abadi, Bruno Blanchet, and Cédric Fournet. 2017. The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication. *J. ACM* 65, 1, Article 1 (Oct. 2017), 41 pages. https://doi.org/10.1145/3127586

Samson Abramsky, Kohei Honda, and Guy McCusker. 1998. A Fully Abstract Game Semantics for General References *(LICS'98)*. IEEE Computer Society, Washington, DC, USA.

Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. 2000. Full Abstraction for PCF. 163 (2000), 409–470.

Samson Abramsky, Pasquale Malacaria, and Radha Jagadeesan. 1994. Full Abstraction for PCF. In *TACS (Lecture Notes in Computer Science)*, Vol. 789. Springer, 1–15.

Samson Abramsky and Guy McCusker. 1999. Full Abstraction for Idealized Algol with Passive Expressions. Vol. 227. 3–42. https://doi.org/10.1016/S0304-3975(99)00047-X

Samson Abramsky and Paul-André Melliès. 1999. Concurrent Games and Full Completeness. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*. 431–442. https://doi.org/10.1109/LICS.1999.782638

Robert Atkey. 2017. Observed Communication Semantics for Classical Processes. In *ESOP*. 56–82.

David Baelde, Stéphanie Delaune, and Lucca Hirschi. 2015. Partial Order Reduction for Security Protocols. In *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15) (Leibniz International Proceedings in Informatics)*, Luca Aceto and David de Frutos-Escrig (Eds.), Vol. 42. Leibniz-Zentrum für Informatik, Madrid, Spain, 497–510. https://doi.org/10.4230/LIPIcs.CONCUR.2015.497

Martin Berger, Kohei Honda, and Nobuko Yoshida. 2001. Sequentiality and the π-Calculus. In *Typed Lambda Calculi and Applications (Lecture Notes in Computer Science)*, Samson Abramsky (Ed.), Vol. 2044. Springer Berlin Heidelberg, 29–45. https://doi.org/10.1007/3-540-45413-6_7

Simon Castellan, Pierre Clairambault, Jonathan Hayman, and Glynn Winskel. 2018. Non-angelic Concurrent Game Semantics. In *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*. 3–19.

Simon Castellan, Pierre Clairambault, Silvain Rideau, and Glynn Winskel. 2017. Games and Strategies as Event Structures. *Logical Methods in Computer Science* Volume 13, Issue 3 (Sept. 2017). https://doi.org/10.23638/LMCS-13(3:35)2017

Simon Castellan and Nobuko Yoshida. 2018. Two sides of the same coin: Session Types and Game Semantics. (2018). https://www.doc.ic.ac.uk/research/technicalreports/2018/DTRS18-5.pdf Technical report.

Gian Luca Cattani and Glynn Winskel. 1997. Presheaf models for concurrency. In *Computer Science Logic*, Dirk van Dalen and Marc Bezem (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 58–75.

Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. 2017. On the Preciseness of Subtyping in Session Types. *LMCS* 13 (2017), 1–62. Issue 2.

Silvia Crafa, Daniele Varacca, and Nobuko Yoshida. 2007. Compositional Event Structure Semantics for the Internal π-calculus. In *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR'07)*. Springer-Verlag, Berlin, Heidelberg, 317–317. http://dl.acm.org/citation.cfm?id=2392200.2392224

Ioana Domnina Cristescu, Jean Krivine, and Daniele Varacca. 2015. Rigid Families for CCS and the π-calculus. In *Theoretical Aspects of Computing - ICTAC 2015 - 12th International Colloquium Cali, Colombia, October 29-31, 2015, Proceedings*. 223–240.

Vincent Danos and Russell Harmer. 2002. Probabilistic Game Semantics. *ACM Trans. Comput. Logic* 3, 3 (July 2002), 359–382. https://doi.org/10.1145/507382.507385

Tim Disney and Cormac Flanagan. 2015. Game Semantics for Type Soundness. In *LICS (LICS'15)*. IEEE, 104–114.

Clovis Eberhart, Tom Hirschowitz, and Thomas Seiller. 2013. Fully-abstract concurrent games for pi. *CoRR* abs/1310.4306 (2013). http://arxiv.org/abs/1310.4306

Simon Gay and Malcolm Hole. 2005. Subtyping for Session Types in the Pi Calculus. *Acta Informatica* 42, 2/3 (2005), 191–225.

Simon Gay and Vasco T. Vasconcelos. 2010. Linear Type Theory for Asynchronous Session Types. *Journal of Functional Programming* 20, 1 (2010), 19–50.

Dan R. Ghica and Mohamed N. Menaa. 2011. Synchronous Game Semantics via Round Abstraction. In *Foundations of Software Science and Computational Structures*, Martin Hofmann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 350–364.

Russell Harmer and Guy McCusker. 1999. A Fully Abstract Game Semantics for Finite Nondeterminism. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*. 422–430. https://doi.org/10.1109/LICS.1999.782637

Matthew Hennessy. 2007. *A Distributed Pi-Calculus*. CUP.

Kohei Honda and Mario Tokoro. 1991. An Object Calculus for Asynchronous Communication. In *ECOOP'91 (LNCS)*, Vol. 512. 133–147.

Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. 1998. Language Primitives and Type Disciplines for Structured Communication-based Programming. In *ESOP (LNCS)*, Vol. 1381. Springer, 122–138.

Kohei Honda and Nobuko Yoshida. 1995. On Reduction-Based Process Semantics. *Theorical Computer Science* 151, 2 (1995), 437–486.

Kohei Honda and Nobuko Yoshida. 1999. Game-Theoretic Analysis of Call-by-Value Computation. *TCS* 221 (1999), 393–456.

Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2008. Multiparty Asynchronous Session Types. In *POPL*. ACM, 273–284.

J. Martin E. Hyland and C. H. Luke Ong. 1994. *On Full Abstraction for PCF*. Technical Report. Imperial College, Department of Computing. 130pp pages.

J. Martin E. Hyland and C. H. Luke Ong. 2000. On Full Abstraction for PCF. *Inf. & Comp.* 163 (2000), 285–408.

J. M. E. Hyland and C.-H. Luke Ong. 1995. Pi-Calculus, Dialogue Games and PCF. In *FPCA*. ACM, 96–107.

Dimitrios Kouzapas and Nobuko Yoshida. 2015. Globally Governed Session Semantics. *LMCS* 10 (2015). Issue 4.

James Laird. 1997. Full Abstraction for Functional Languages with Control. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*. 58–67. https://doi.org/10.1109/LICS.1997.614931

James Laird. 2001. A Game Semantics of Idealized CSP. *Electr. Notes Theor. Comput. Sci.* 45 (2001), 232–257. https://doi.org/10.1016/S1571-0661(04)80965-4

Jim Laird. 2005. 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*. 51–65. https://doi.org/10.1007/11539452_8

Paul-André Melliès. 2005. Asynchronous Games 4: A Fully Complete Model of Propositional Linear Logic. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*. 386–395. https://doi.org/10.1109/LICS.2005.6

Paul-André Melliès. 2006. Asynchronous games 2: The true concurrency of innocence. *Theor. Comput. Sci.* 358, 2-3 (2006), 200–228. https://doi.org/10.1016/j.tcs.2006.01.016

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

Paul-André Melliès and Samuel Mimram. 2007. Asynchronous Games: Innocence Without Alternation. In *CONCUR 2007 - Concurrency Theory, 18th International Conference*. 395–411.

Paul-André Melliès and Léo Stefanesco. 2018. An Asynchronous Soundness Theorem for Concurrent Separation Logic. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*. 699–708. https://doi.org/10.1145/3209108.3209116

Robin Milner. 1992. Functions as Processes. *MSCS* 2, 2 (1992), 119–141.

Robin Milner and Davide Sangiorgi. 1992. Barbed Bisimulation. In *ICALP (LNCS)*, Vol. 623. Springer, 685–695.

Dimitris Mostrous and Nobuko Yoshida. 2015. Session Typing and Asynchronous Subtyping for the Higher-Order π-Calculus. *Inform. Comput.* 241 (2015), 227–263.

Vaughan R. Pratt. 1984. The Pomset Model of Parallel Processes: Unifying the Temporal and the Spatial. In *Seminar on Concurrency, Carnegie-Mellon University, Pittsburg, PA, USA, July 9-11, 1984*. 180–196. https://doi.org/10.1007/3-540-15670-4_9

Silvain Rideau and Glynn Winskel. 2011. Concurrent Strategies. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*. 409–418. https://doi.org/10.1109/LICS.2011.13

Ken Sakayori and Takeshi Tsukada. 2017. A Truly Concurrent Game Model of the Asynchronous Pi-Calculus. In *FoSSaCs*. 389–406.

Davide Sangiorgi. 1995. Internal Mobility and Agent Passing Calculi. In *Proc. ICALP'95*.

D. Sangiorgi. 1996. π-calculus, internal mobility and agent-passing calculi. *TCS* 167, 2 (1996), 235–274.

Peter Selinger. 1997. First-Order Axioms for Asynchrony. In *Proc. CONCUR '97*.

Kaku Takeuchi, Kohei Honda, and Makoto Kubo. 1994. An Interaction-based Language and its Typing System. In *PARLE'94 (LNCS)*, Vol. 817. 398–413.

P. S. Thiagarajan. 2002. Regular Event Structures and Finite Petri Nets: A Conjecture. In *Formal and Natural Computing - Essays Dedicated to Grzegorz Rozenberg [on occasion of his 60th birthday, March 14, 2002]*. 244–256. https://doi.org/10.1007/3-540-45711-9_14

Rob van Glabbeek and Frits Vaandrager. 2003. Bundle Event Structures and CCSP. In *CONCUR 2003 - Concurrency Theory*, Roberto Amadio and Denis Lugiez (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 57–71.

Glynn Winskel. 1982. Event structure semantics for CCS and related languages. 561–576. See also DAIMI Report PB-159, Computer Science Department, Aarhus University, 1983.

Glynn Winskel. 1986. Event Structures. In *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986*. 325–392. https://doi.org/10.1007/3-540-17906-2_31

Nobuko Yoshida. 1996. Graph Types for Monadic Mobile Processes.. In *FSTTCS (LNCS)*, Vol. 1180. Springer, 371–386.

Nobuko Yoshida and Vasco Thudichum Vasconcelos. 2007. Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited: Two Systems for Higher-Order Session Communication. *Electr. Notes Theor. Comput. Sci.* 171, 4 (2007), 73–93.