The Concurrency Column

by

Nobuko Yoshida

Department of Computing
Imperial College London
180 Queen's Gate, London, SW7 2AZ
n.yoshida@imperial.ac.uk, http://mrg.doc.ic.ac.uk/

Concurrent Structures in Game Semantics*

Simon Castellan
Imperial College London, UK
s.castellan@imperial.ac.uk

Abstract

Game semantics is a powerful tool to design intensional denotational semantics of complex programming languages, leading to notions of syntax-free normal forms and fully abstract models. In game semantics, programs become strategies, *ie*. set of plays, on a game given by their type.

Traditionally, game semantics represents plays as sequences of moves. This representation forces to reduce concurrency to interleavings. In this paper, we develop a framework of game semantics based on event structures – a partial order representation of concurrency, extending the work of Rideau and Winskel [19]. This causal representation allows to retain more intensional behaviour on concurrent programs. We demonstrate the benefits of this approach by developing a notion of concurrent and nondeterministic innocence, which an open problem despite the existence of many game semantics models of concurrent languages. We show that, our innocence along with an extension of well-bracketing, can capture the essence of parallel and nondeterministic computation.

1 Introduction

Context Game semantics in its modern incarnation arose in the 90s to solve the famous problem of *full abstraction for PCF*: trying to characterise, by mathematical means, the observational behaviour of PCF, a pure functional programming language. Game semantics interprets programs by *strategies*, representing the possible behaviours of the program against an expressive class of context [14, 20]. However, the interesting contribution of these game semantics models turned out to be, not the full abstraction result in itself (obtained through an undecidable quotient) but a key ingredient in the proof, *finite definability*. Game semantics for PCF allows to describe faithfully programs up to reduction in a syntax-free manner.

*This work has been supported by EPSRC EP/K034413/1 and EP/K011715/1.

Game semantics was later extended to programming languages with effects such as references [1] and control operators [16], which correspond to relaxing conditions on strategies: references correspond to relaxing **innocence** and control operators relaxing **well-bracketing**. In the first case, we obtain a full abstraction result without having to resort to an undecidable quotient, illustrating the fact that observational equivalence of programming languages with effects can be easier to understand than those of pure functional programming languages.

Game semantics was then extended to nondeterminism and concurrency, even though innocence was never lifted to those settings, making a result of finite definability and full abstraction for pure extensions of PCF to nondeterminism and parallelism impossible (apart from the recent work by Tsukada and Ong for sequential nondeterminism [21]).

In addition, the interleaving nature of these models makes it harder to reason on strategies and model complex behaviours such as reorderings occurring in weak memory models. Finally, the interleaving representation is not adapted for efficient verification. For all these reasons, a theory of concurrent game semantics based on a truly concurrent model is needed.

Contribution and outline of the paper Section 2 introduces game semantics as a way to extend operational semantics to open terms. Section 3 introduces the ideas of a game semantics based on event structures [19], on which it builds to improve the representation of nondeterminism in order to cope with wilder notions of convergences. Section 4 explains how to deal with non linearity, the previous model being linear, by the addition of *symmetry* to the model. Section 5 discusses innocence and well-bracketing in this framework, and how together they allow us to prove finite definability for a parallel and nondeterministic extension of PCF (up to may equivalence). Finally we conclude in Section 6.

Origin of the results The results presented here originate from the author's Ph.D "Concurrent structures in game semantics" [4]. The original model on event structures is due to Rideau and Winskel [19, 5]. The extension to symmetry, along with the notions of innocence and well-bracketing and the intensional full abstraction in this new setting is joint work with Pierre Clairambault and Glynn Winskel.

Related work For more detailed related work section, please see the author's Ph.D [4]. The first approach to truly concurrent games semantics dates back to Abramsky and Melliès [2], and their model of MALL, using closure operators. Melliès later worked on asynchronous games. Based on this approach, Melliès and Mimram [18] developed a preliminary notion of concurrent innocence (in a deterministic setting) which is not intrinsic to a strategy but depends on the type.

Hirschowitz and collaborators [13, 9] use presheaves over plays represented as multigraphs to give a concurrent game semantics to message-passing programming languages. However their work does not support composition. They provide a notion of innocence in terms of a sheaf condition which was later recast by Tsukada and Ong [21] in the λ-calculus to give a notion of nondeterministic (sequential) innocence.

In parallel, work on partial order representations of strategies in ludics was carried out by Curien, Faggian, Maurel, and Piccolo [10, 8, 11] which inspired [19], the starting point of the thesis.

2 An introduction to game semantics

We now provide a gentle introduction to game semantics using arithmetic expressions as a running example.

Operational semantics. An informal semantics of arithmetic expressions (eg. "3+(3+5)") could be "perform the leftmost operation between two syntactic numbers and continue until only a single number is left". This intuition naturally leads to an operational semantics. In this setting, we would have $3+(3+5) \to 3+8 \to 11$, and 11 is the semantics of "$3 + (3 + 5)$". We write $[\![3 + (3 + 5)]\!] = 11$.

Variable and functions. Consider now arithmetic expressions that can contain variables: "$x + \underline{3}$" becomes a valid expression $e(x)$. Our operational semantics gets stuck now: "$x + 3$" is not a number, and yet there are no operations between syntactic numbers to perform. It seems natural to consider that the semantics of "$x + \underline{3}$" is the *function* mapping a number n to the number $n + 3$.

However, to go from an arbitrary expression e to the corresponding function, operational semantics cannot really be extended. A solution would be to say that an expression $e(x)$ has meaning $n \mapsto [\![e(\underline{n})]\!]$ where $\underline{n}$ is the mathematical expression reduced to a number whose value is that of n.

Doing so, we lost the operational flavour: the evaluation of variables is invisible in the semantics. In particular, the expressions "$\underline{2} \times x$" and "$x + x$" both denote the function that maps a number to its double, yet "$x + x$" has two occurrences of the variable x whereas "$2 \times x$" has only one. This loss of intensional information can cause problems when moving to a richer setting, for instance to evaluate expressions $e(x)$ where an occurrence of x yields the result of a coin toss (0 or 1). In this case, $e(x) :=$ "$2 \times x''$ *always* evaluates to an even number whereas $e'(x) :=$ "$x + x''$ may not, as x could evaluate once to zero, and once to one.

Program/Context dialogue. To account operationally for variables, the operational semantics needs to be updated. Previously, the relation $\to$ only described steps of *internal computation*. We wish to update this vision to allow the context (or the environment) to communicate with the program, in order to provide it with values for the variables: the expression now exchanges messages with its context. A *possible* execution of "$x + 2$" is now:

$$x + \underline{2} \xrightarrow{\mathrm{q}_x^+} [] + \underline{2} \xrightarrow{2^-} \underline{2} + \underline{2} \to 4$$

Some steps are now labelled: they denote the messages sent or received by the program during the step. The polarity in superscript indicates whether the message is sent (+) or received (−) by the program. The first step, labelled q_x^+, is a question to the context: the expression asks the value for x. The expression now awaits an answer from the context, symbolised by the placeholder []. Then, in a second step, the program receives an answer: this occurrence is equal to 2. The placeholder is replaced with the value. At this point, the result can be computed without communication with the context. The expression "$x + 2$" has many possible such executions as the context is free to choose the answer to the question q_x^+.

This solves the problem shown earlier, as "$x + x$" has the following execution:

$$x + x \xrightarrow{\mathrm{q}_x^+} [] + x \xrightarrow{0^-} \underline{0} + x \xrightarrow{\mathrm{q}_x^+} 0 + [] \xrightarrow{1^-} \underline{0} + \underline{1} \to 1.$$

If we forget the intermediate programs and internal steps, and simply consider the sequence of messages exchanged, we get: $\mathrm{q}_x^+ \cdot 0^- \cdot \mathrm{q}_x^+ \cdot 1^-$. To remember the final value of the program, we add an extra message at the end where the program signals to the context its result, and also an initial message from the environment to start the computation, which gives the following **dialogue**:

$$\mathrm{q}^- \cdot \mathrm{q}_x^+ \cdot 0^- \cdot \mathrm{q}_x^+ \cdot 1^- \cdot 1^+.$$

Intuitively, a particular dialogue captures the interaction of a particular program *against* a particular context. The semantics of an expression now becomes the set of dialogues where the program interacts against a particular context, *eg.*

$$[\![x + x]\!] = \{\mathrm{q}^- \cdot \mathrm{q}_x^+ \cdot n^- \cdot \mathrm{q}_x^+ \cdot m^- \cdot (n + m)^+ \mid n, m \in \mathbb{N}\}$$

Game semantics and composition. From this idea of interpreting programs by dialogues, game semantics has been used to interpret a variety of programming languages. The interpretation supports programs with free variables. In game semantics, the previous diagram is written as follows:

$$
\begin{array}{lcl}
(x:\mathbb{N}) & \Rightarrow & \mathbb{N} \\
 & & \mathrm{q}^- \\
\mathrm{q}^+ & & \\
0^- & & \\
\mathrm{q}^+ & & \\
1^- & & \\
 & & 1^+
\end{array}
$$

Time flows from top to bottom; the right column (result component) denotes messages about the result and the left column messages about x (x component).

In game semantics, programs become sets of *valid dialogues*, defined as plays of a 2-player (Player/Opponent) **game**. Moves of the game represent the messages the program can send or receive (depending on the polarity). Such sets of dialogues are **strategies** explaining how Player reacts to Opponent moves.

This representation makes it easy to represent *composition*. Consider an expression $e(x)$ (say $x + x$) in which another expression $e'(y)$ (say $3 \times y$) is to be plugged for x, resulting in an expression $e(e'(y))$ ($3 \times y + 3 \times y$). We want to derive the dialogues of $[\![e(e'(y))]\!]$ from those of $[\![e'(y)]\!]$ and $[\![e(x)]\!]$. First, we form *interaction dialogues* where $e(x)$ is run so that messages sent to the x component are forwarded to $e'(y)$ on its result component and conversely:

$$
\begin{array}{lcl@{\qquad}lcl}
(y:\mathbb{N}) & \overset{e'(y)}{\Longrightarrow} & \mathbb{N} & (x:\mathbb{N}) & \overset{e(x)}{\Longrightarrow} & \mathbb{N} \\
 & & & & & \mathrm{q}^- \\
 & & \mathrm{q}^- & & \mathrm{q}^+ & \\
\mathrm{q}^+ & & & & & \\
1^- & & & & & \\
 & & 3^+ & & 3^- & \\
 & & \mathrm{q}^- & & \mathrm{q}^+ & \\
\mathrm{q}^+ & & & & & \\
2^- & & & & & \\
 & & 6^+ & & 6^- & \\
 & & & & & 9^+
\end{array}
$$

The result component of $e'(y)$ plays *against* the x component of $e(x)$. The *external* context can only communicate on the y component of $e'(y)$ and the result

component of $e(x)$. Hiding the communication in the middle between $e(x)$ and $e'(y)$, we have the following dialogue:

$$\begin{array}{lcl} (y : \mathbb{N}) & \Rightarrow & \mathbb{N} \\ & & \mathrm{q}^- \\ \mathrm{q}^+ & & \\ 1^- & & \\ \mathrm{q}^+ & & \\ 2^- & & \\ & & 9^+ \end{array}$$

which is a valid dialogue in the semantics of $3 \times y + 3 \times y = e(e'(y))$, as expected.

This shift from *extensional models* (representing open terms by functions) to *interactive models* (representing open terms by dialogues) was initiated by the question of capturing observational equivalence of pure functional programs.

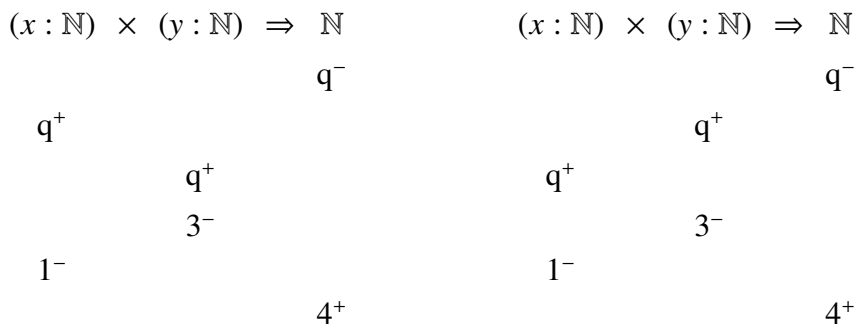
Compositionality, and observational equivalence. The previous interpretation in terms of dialogues is *compositional*. As seen above, dialogues can be composed via interaction, hence the dialogues for a large expression can be obtained from dialogues of its subparts. This property is interesting from a practical point of view as it makes the semantics easier to compute for a large code base. Since it can be done incrementally, if a little part of the code changes, it is easier to recompute the total semantics since the semantics of the rest need not be recomputed.

However, this property is also interesting from a theoretical standpoint. If two programs have the same set of dialogues, then the programs will have the same behaviour *in any context*: they are **observationally equivalent**. This allows to substitute one for the other in a larger code base without breaking things. The semantics gives a *sound* tool to reason about observational equivalence.

When the converse holds, that is when two programs are observationally equivalent, then they have the same semantics (in our case, the same dialogues), then the interpretation is **fully-abstract**: the semantics provides a *complete* tool to reason about program equivalence.

Concurrency and alternation. So far, the model described is **sequential**: operations are done in a linear order, one after the other. What happens if we want to represent parallel evaluation? For instance $x + y$ could be run by asking the value for x and y in parallel, waiting for both answers and then returning the final value. However, currently our dialogues are all **alternating**. Players (the program or the context) take turns in sending messages. To be able to represent dialogues

exhibiting concurrency, we need to relax this assumption and authorise the Player to send a message to the context *before* receiving an answer to the previous message. For instance, the following diagrams depicts dialogues for the expression $x + y$ that feature parallelism:

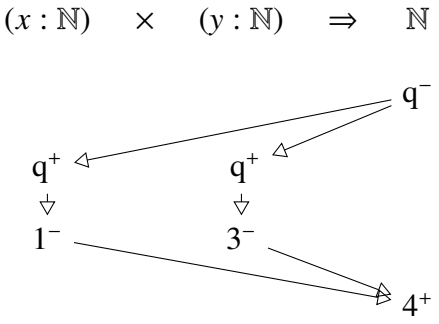


Here $x + y$ interacts against a context that supplies $y = 3$ and $x = 1$. The questions are asked in a non sequential way: both questions are actually sent before receiving any answers. However, the dialogues differ by the order in which the questions are sent (left: x then y; right: y then x). As a result, such dialogues implicitly feature a scheduler resolving the parallelism of the program. Hence a dialogue now represents the interaction of a particular program against a particular context, scheduled in a particular way.

Causal representation of concurrency. This representation of concurrency by linear dialogues is not very satisfactory as dialogues need to contain scheduling information. This leads to a combinatorial explosion since concurrency is essentially represented by interleavings: the number of dialogues used to represent a particular program grows exponentially with the number of parallel computations. Moreover, it is also not satisfactory from a theoretical standpoint: the scheduling information obfuscates the intention of the program and makes it hard for the representation to scale to richer programming settings such as probabilistic programming. It also makes the correspondence between shapes of dialogues and programming features difficult to generalise.

To work around this problem, it is necessary to take the question of the mathematical representation of concurrency seriously. Insisting to observe the order in which the parallel requests are made, forces the scheduler to *also* be observable. To relax this assumption, it becomes necessary to come to terms with the impossibility of observing the order in which some messages are sent (or received). In concrete terms, this means moving away from *chronology* to *causality*. Dialogues should not be totally ordered anymore; but only partially-ordered. Two messages are concurrent if they are not comparable for the partial order, ie. we do not know

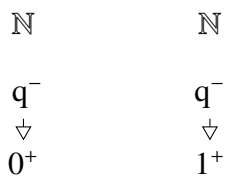
in which order they appear. The two previous linear dialogues can be subsumed by a single partially-ordered diagram:



Such approaches are often called *truly concurrent* as they represent concurrency as a primitive notion, distinct from the interleaving of (sequential) traces.

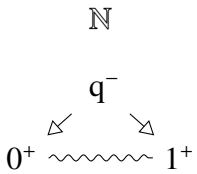
Nondeterminism. One last challenge we would like to address is nondeterminism. A program is nondeterministic when it may evaluate to more than one result. Typical examples of nondeterministic programs include primitives to generate pseudo-random numbers that are found in most programming languages.[1] Nondeterminism is also a natural byproduct of shared memory concurrency as races between memory accesses create nondeterministic behaviours.

Suppose that in our language of expressions, we add the construct `choice` which evaluates to zero or to one. The semantics of `choice` in terms of dialogues is very easy to define: it contains the following two dialogues:



However this representation suffers from several drawbacks. One major drawbacks is that it may hide the nondeterministic branching point, information which is useful for many theoretical developments. In the thesis, we use *event structures* [22] which represent nondeterminism via conflict (more details later on):

[1] In practice most pseudo-random generators are deterministic, but the nondeterminism is *apparent*, in the sense that the result often depends on parameters outside the control of the programmer, such as the time of execution for example.



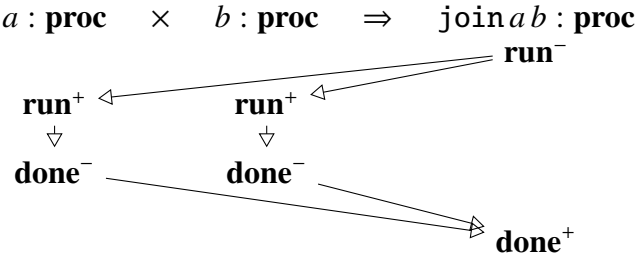
3 Game semantics on event structures

This section presents the model of Rideau and Winskel [19] for concurrent and (angelic) nondeterministic game semantics.

3.1 Event structures

In Section 2, we presented briefly partially-ordered dialogues and event structures as a tool to represent concurrent programs. We now make those observations formal and introduce the mathematical setting.

Causality As a first example of concurrent strategies, consider the `join` primitive that spawns two calculations in parallel and terminates *when* both have terminated. Such a primitive would have the type $\mathbf{proc} \times \mathbf{proc} \Rightarrow \mathbf{proc}$ where **proc** is the type of commands, that is programs performing computational effects but returning no value of interest. It could be described by the following dialogue:



The previous examples of Section 1 played on natural numbers where the allowed moves where q (question for the value) and n (answer to the question). Since on **proc** there are no values to return, the moves are here **run** (beginning of the computation) and **done** (end of the computation).

This diagram represents now a partial order which is generated by the transitive closure of $\rightarrowtriangle$ (**immediate causal dependency**). Note the inversion of polarity between the two sides of $\Rightarrow$. This inversion was already present in the previous section and is due to `join` acting as the environment for a and b. When the context

runs `join` a b, a and b are run immediately, in any order as the two events $\mathbf{run}^+$ are incomparable. When *both* process terminate (issuing a $\mathbf{done}^-$) the program signals that it has terminated.

In this context, a valid (partial) execution is a set of events which is such that if an event e occurs, then any of its immediate causes $e' \rightarrowtriangle e$ must appear. In other words, an execution is a downclosed subset of events.

Nondeterminism Most concurrent programming features induce nondeterminism: be it shared memory concurrency (à la Concurrent Idealized Algol [3]) or channels (à la CCS [17]). To accommodate nondeterminism, causal dependency is however not enough, as nondeterminism implies that two events might be incompatible: for instance the outcomes of a nondeterministic coin-tossing. Nondeterminism is usually modelled using *sets* of executions: in our case, sets of partial-orders. However, this forgets nondeterministic branching points, and identifies the terms $M_1 = \lambda f.\, f\,\mathrm{tt} + f\,\mathrm{ff}$ and $M_2 = \lambda f.\, f(\mathrm{tt} + \mathrm{ff})$ (where + is nondeterministic choice) which are not must-equivalent. To solve this issue, one solution is to add a notion of conflict to causality to know which events can occur in the same execution.

Definition 3.1 (Event structures (with general conflict))**.** *An **event structure** is a triple $(E, \leq, Con_E)$ where $(E, \leq_E)$ is a partial order of events and $Con_E \subseteq \mathscr{P}_f(E)$ is a set of* finite ***consistent*** *sets of E subject to the following axioms:*

(1) For every $e \in E$, the set $[e] = \{e' \in E \mid e' \leq e\}$ is finite,

(2) For all $Y \in Con_E$ and $X \subseteq Y$ then $X \in Con_E$,

(3) For all $e \in E$, $\{e\} \in Con_E$,

(4) If X is consistent, so is its downclosure

$$[X]_E = \{e' \in E \mid e' \leq e \text{ for some } e \in X\}.$$

In particular, if $X \in \mathrm{Con}_E$ and $e \leq e' \in X$ then $X \cup \{e\} \in \mathrm{Con}_E$. Our conflict here is general (as opposed to binary) as it makes the mathematical theory smoother.

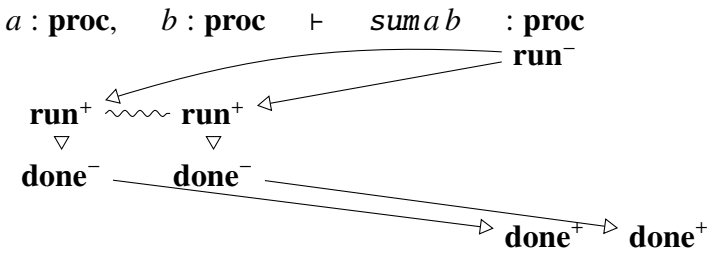
Notations on event structures Given an event structure E, write $e \rightarrowtriangle_E e'$ for **immediate causality** defined as $e < e'$ with no events in between. If $e \leq_E e'$, we say that e' **causally depend** on e. If $e \rightarrowtriangle_E e'$, we say that e' **immediately causally depends** on e, or that there is a **causal link** from e to e'. Given $e \in E$, write $[e) = [e] \setminus \{e\}$. Two events $e, e' \in E$ are **concurrent** when they are incomparable for the causal order and $\{e, e'\} \in \mathrm{Con}_E$. A **finite configuration** is a

consistent finite set of events of E which is downclosed for $\leq$. Finite configurations (abbreviated as configurations in the rest) represent partial executions. The notion of configurations is crucial when reasoning on event structures. The set of finite configurations of E will be written $\mathscr{C}(E)$. Configurations are naturally ordered by inclusion, and any configuration inherits a partial order from E. A configuration $x \in \mathscr{C}(E)$ **extends** by $e \in E$ (written $x \overset{e}{\multimap\!\subset} x \cup \{e\}$) when $e \notin x$ and $x \cup \{e\} \in \mathscr{C}(E)$. In that case, e is called an **extension** of x. Two extensions of e, e' of x are **compatible** when $x \cup \{e, e'\} \in \mathscr{C}(E)$, **incompatible** otherwise. In that case, we say that e and e' are a **minimal conflict** in the context x (or **involved in a minimal conflict**). In the general case, it depends on the context x, but when the event structure has binary conflict, it is independent from x and coincide with the notion of minimal conflict introduced above.

A consequence of the axioms of event structures is that for every $e \in E$, the set $[e]$ is a configuration representing the causal history of e. Such configurations are called **prime configurations**, or equivalently a prime configuration is a configuration with a top element. Remark that, consequently $[e)$ is also always a configuration. Given a configuration x of an event structure E, a **covering chain** is a sequence $\emptyset = x_0 \multimap\!\subset x_1 \multimap\!\subset \ldots \multimap\!\subset x_n = x$ of configurations leading to x.

Drawing event structures Pictures will only feature event structures with binary conflict and represent immediate causality ($\rightarrowtriangle$) and minimal conflict ($\sim$).

Example 3.2. *The interpretation of the nondeterministic sum of processes can be represented by the event structure:*



The difference with the `join` *operator is the conflict between the two occurrences of* $\boldsymbol{run}^+$ *that ensures the presence of only one of the* $\mathbf{run}^+$ *in a single execution. The conflict propagates upwards: the causal future of these events are also in conflict. In this example, the names* **run**, **done** *come from an (implicit) labelling of events by moves of a game. Such labelled event structures will be used to represent strategies on games.*

3.2 Games and strategies

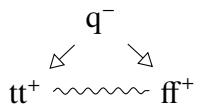
We now define a notion of games and strategies on them to make formal the diagrams of the previous section.

Games as event structures with polarities In our setting, games will simply be event structures where each event carries a polarity:

Definition 3.3 (Game)**.** *A **game** is an event structure E along with a polarity labelling* $\mathrm{pol}_E : E \to \{+, -\}$.

Event structures with polarities will be drawn as event structures, where polarity is indicated in superscript of events. We will also make use of the notation "let $a^+ \in A$" to introduce a positive event of a game A (similarly "let $a^- \in A$" to introduce a negative event). We will often use the term **play** to refer to a configuration of a game by analogy with the standard game-theoretic terminology.

A possible game for booleans **B** is as follows:



The initial question denotes the call from the environment and the two positive moves denote the possible return values of the program.

Rules of a game are specified via its causal order and consistent sets:

- *causal order*: a move cannot be played before another is played,
- *consistency*: some moves cannot occur together (e.g., for booleans, the moves corresponding to true and false cannot be played together).

Operation on games To build games, we will make use of two fundamental operations on games: duality and parallel composition. Given a game A, the **dual game** $A^\perp$ is simply obtained by exchanging polarities, leaving the event structure untouched. The **simple parallel composition**, or more briefly **parallel composition**, of A and B, denoted by $A \parallel B$ is defined as follows:

Definition 3.4 (Simple parallel composition)**.** *Let A_0 and A_1 be event structures. The event structure $A_0 \parallel A_1$ is defined as follows:*

Events $\{0\} \times A_0 \cup \{1\} \times A_1$

Causality $(i, a) \leq_{A_0 \parallel A_1} (j, a')$ *iff* $i = j$ *and* $a \leq_{A_i} a'$

Consistency $X \in Con_{A_0 \| A_1}$ *iff* $\{a \mid (i,a) \in X\} \in Con_{A_i}$ *for* $i \in \{0,1\}$

Moreover, any choice of polarities on A_0 *and* A_1 *induce a canonical choice of polarities on* $A_0 \parallel A_1$ *via* $\mathrm{pol}_{A_0 \| A_1}(i,a) = \mathrm{pol}_{A_i}(a)$.

Parallel composition will be used to interpret product types. In $A \parallel B$, A and B evolve concurrently with no interference (no causality or conflict). As a result, the following monotonic map is an order-isomorphism:

$$\begin{array}{rl} \cdot \parallel \cdot : \mathscr{C}(A) \times \mathscr{C}(B) & \to \mathscr{C}(A \parallel B) \\ (x,y) & \mapsto x \parallel y = \{0\} \times x \cup \{1\} \times y \end{array}$$

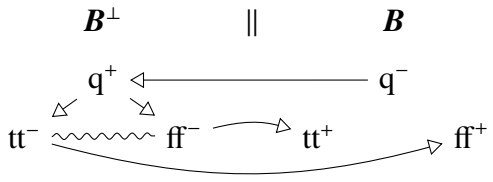
`join` and `sum` play on the game $\mathbf{proc}^\perp \parallel \mathbf{proc}^\perp \parallel \mathbf{proc}$:

$$\begin{array}{ccccc} \mathbf{proc}^\perp & \parallel & \mathbf{proc}^\perp & \parallel & \mathbf{proc} \\ \mathbf{run}^+ & & \mathbf{run}^+ & & \mathbf{run}^- \\ \downarrow & & \downarrow & & \downarrow \\ \mathbf{done}^- & & \mathbf{done}^- & & \mathbf{done}^+ \end{array}$$

Strategies The strategies sketched so far can be viewed as a causal and conflict enrichment of the game. Such strategies on a game A can be represented as event structures $(S, \leq_S, \mathrm{Con}_S)$ where $S \subseteq A$, $\leq_S \supseteq \leq_A$ (causal enrichment) and $\mathrm{Con}_S \subseteq \mathrm{Con}_A$ (conflict enrichment). This captures the requirement that strategies must respect the rules of the game A, that is:

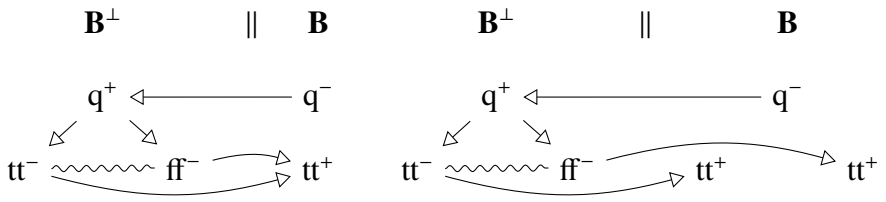
- a move can only be issued after its causal dependencies,
- consistent sets of moves for strategies are also consistent in the game.

Example 3.5. *Boolean negation can be represented as a strategy on* $\boldsymbol{B}^\perp \parallel \boldsymbol{B}$.



Note that the arrows between the question and the answers on $\boldsymbol{B}$ *(the right component) are induced by transitivity, and the inconsistency between the positive true and false is also induced by the inconsistency on their negative counterpart.*

However, how to define the boolean function that *evaluates its argument* and then returns true in both cases? Two candidate diagrams come to mind:



The left diagram has one occurrence of tt^+ that depends on the two possible argument values; this is not valid because the two Opponent moves are in conflict: the downclosure of $\{\mathrm{tt}^+\}$ is not consistent. The second solution describes a valid event structure but has two tt moves: it is not a subset of the game anymore. Note that the `sum` strategy given in Example 3.2 is also not a proper subset of the game $\mathbf{proc}^\perp \parallel \mathbf{proc}^\perp \parallel \mathbf{proc}$ as $\mathbf{done}^+$ has two occurrences. As a result, to account for these behaviours, strategies should not be strict subsets of the game, but *embeddings*. The notion of embedding is formalised via maps of event structures:

Definition 3.6 (Maps of event structures)**.** *A (total)* ***map of event structures*** *from E to F is a function on events $f : E \to F$ such that:*

- *the direct image of a configuration of E is a configuration of F,*
- *f is injective on consistent sets.*

Event structures and their (total) maps form a category $\mathcal{E}$.

Strategies will be certain maps of event structures satisfying two crucial properties: **courtesy** (or **innocence** in [19]) and **receptivity**:

Definition 3.7. *A* ***strategy*** *on a game A is a map of event structures $\sigma : S \to A$ satisfying:*

Courtesy *If $s \rightarrowtriangle_S s'$ and σs is positive or $\sigma s'$ negative, then $\sigma s \rightarrowtriangle_A \sigma s'$*

Receptivity *For all $x \in \mathscr{C}(S)$ and $\sigma x \overset{a}{\text{—}\subset}$ where the event a is negative, then there exists a unique $s \in S$ with $\sigma s = a$ and $x \overset{s}{\text{—}\subset}$.*

The event structure S represents the behaviour of the strategy and σ indicates how this embeds into the game. In that case, we write $\sigma : A$ to denote that σ is a strategy on A. The first axiom of maps of event structures makes sure that the plays of S are valid according to A whereas the second one is a linearity condition ensuring that in a play, events of the game occur at most once. This linearity condition will be crucial to define interaction of strategies in Section 3.3. Courtesy and receptivity are necessary for strategies to be invariant by composition under **copycat** (the identity in the world of strategies), and as a result necessary to obtain

a category of strategies [19]. If $\sigma : S \to A$ is a strategy, we will sometimes write $\mathscr{C}(\sigma)$ for $\mathscr{C}(S)$.

According to this definition, strategies $\sigma : S \to A$ appear as certain event structures labelled by events of the game. As a consequence, events of a strategy (events of S) naturally carry a polarity given by the labelling $\mathrm{pol}_A \circ \sigma$, and S can be regarded as an event structure with polarities.

We now move on to a very important example of strategy.

The copycat strategy In the game $A^\perp \parallel A$, each move of A appears twice, with a different polarity. A natural strategy on this game is as follows: to play the positive occurrence of $a \in A$, we wait for its negative counterpart on the other side. This describes the **copycat** strategy. It is implemented by adding a causal link from the negative occurrence of every $a \in A$ to its positive occurrence.

Definition 3.8 (Copycat strategy)**.** *Let A be a game. The copycat strategy $\mathfrak{c}_A$ on $A^\perp \parallel A$ is defined as the identity-on-events map:*

$$\mathfrak{c}_A : \mathbb{C}_A \to A^\perp \parallel A,$$

where $\mathbb{C}_A$ is given by:

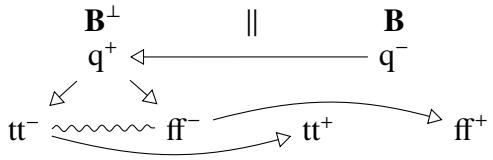
Events $A^\perp \parallel A$

Causality *Transitive closure of*

$$\leq_{A^\perp \parallel A} \cup \{((i,a),(1-i,a)) \mid (i,a) \text{ negative in } A^\perp \parallel A\}$$

Consistency $X \in Con_{\mathbb{C}_A}$ *iff* $[X]_{\mathbb{C}_A} \in Con_{A^\perp \parallel A}$.

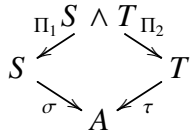
As an example, the copycat strategy on the game **B** is given by:



3.3 Interaction of strategies

Given a strategy $\sigma : A$ and a strategy $\tau : A^\perp$, a natural operation is to have them play against each other. This is represented mathematically by the notion of **interaction**. The concrete definition of the interaction is intricate but enjoys a simple universal property. Indeed, the category $\mathcal{E}$ (without polarities) has pullbacks, which allows to define the interaction of two strategies as the pullback of the underlying maps.

Proposition 3.9. *Let A be a game and $\sigma : S \to A$ and $\tau : T \to A^\perp$ be strategies. There exists an event structure $S \wedge T$ and maps of event structures $\Pi_1 : S \wedge T \to S$ and $\Pi_2 : S \wedge T \to T$ such that the following diagram is a pullback in $\mathcal{E}$:*



In other terms, for every maps $\alpha : X \to S$ and $\beta : X \to T$ such that $\sigma \circ \alpha = \tau \circ \beta$, there exists a unique map $\langle \alpha, \beta \rangle : X \to S \wedge T$ with $\Pi_1 \circ \langle \alpha, \beta \rangle = \alpha$ and $\Pi_2 \circ \langle \alpha, \beta \rangle = \beta$.

The reader can find the detailed construction of $S \wedge T$ in [5, 4], along with examples, that we omit for lack of space.

3.4 A category of strategies

To get a category of strategies and model programming languages, we need to define a notion of strategy from a game A to a game B. Following Joyal [15], a strategy from A to B will be a strategy on the compound game $A^\perp \parallel B$. The notation $\sigma : A$ to denote a strategy on a game A is generalised to $\sigma : A \nrightarrow B$ to denote a strategy from a game A to a game B. Copycat on A becomes a strategy from A to itself – a candidate for an identity strategy.

How to compose a strategy from A to B and a strategy from B to C to get a strategy from A to C? One plays on $A^\perp \parallel B$, the other on $B^\perp \parallel C$, and the desired result on $A^\perp \parallel C$, so it is not easily described as a closed interaction. To have them interact on B while the parts on A and C are left untouched, we build two strategies on $A \parallel B \parallel C$ and take their interaction.

Definition 3.10 (Open interaction)**.** *Let $\sigma : S \to A^\perp \parallel B$ and $\tau : T \to B^\perp \parallel C$ be strategies. The maps $\sigma \parallel C^\perp : S \parallel C^\perp \to A^\perp \parallel B \parallel C^\perp$ and $A \parallel T : A \parallel T \to A \parallel B^\perp \parallel C$ have dual codomains.*

*The **open interaction** of σ and τ is their interaction:*

$$\tau \star \sigma = (\sigma \parallel C^\perp) \wedge (A \parallel \tau) : T \star S \to A \parallel B \parallel C.$$

However, $\tau \star \sigma$ is not a strategy on $A^\perp \parallel C$ as one would like, because of the events "in the middle", projected to B. The solution is to simply hide those events.

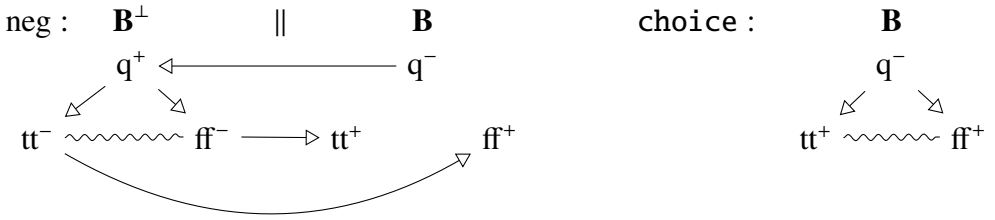
Hiding Hiding is performed through an operation called the **projection of events structures**. It removes some events deemed internal, and propagates causal dependencies and conflicts. The internal events are thought of as occurring in the background, any time after their visible dependencies occur.

Definition 3.11 (Projection of event structures)**.** *Let E be an event structure and $V \subseteq E$ a subset of events (an event in V is called **visible**). The **projection** of E to V (written $E \downarrow V$) is the event structure which has V has set of events, and causality and consistency is inherited from E.*

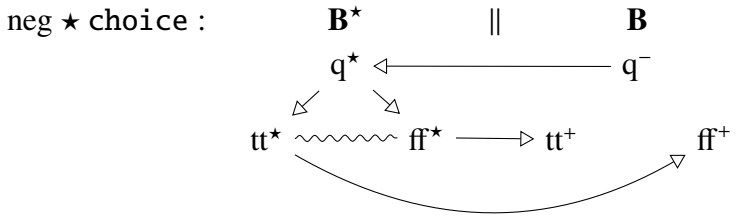
Definition 3.12. *Let $\sigma : S \to A^{\perp} \parallel B$ and $\tau : T \to B^{\perp} \parallel C$. Remember that their interaction is given by $\tau \star \sigma : T \star S \to A \parallel B \parallel C$. An event $e \in \tau \star \sigma$ is **visible** when $(\tau \star \sigma)(e) \notin B$ (which means that $(\tau \star \sigma)(e)$ is not in the B component of the disjoint union $A \parallel B \parallel C$). Writing V for the set of visible events, $T \odot S$ is defined as $(T \star S) \downarrow V$ and $\tau \odot \sigma : T \odot S \to A^{\perp} \parallel C$ as the restriction of $\tau \star \sigma$.*

The composition $\tau \odot \sigma$ is also courteous and receptive, hence it is a strategy.

Example 3.13 (Negation and nondeterministic choice)**.** *Remember the negation strategy* neg : $\boldsymbol{B} \nrightarrow \boldsymbol{B}$ *and the nondeterministic boolean* `choice`*:*

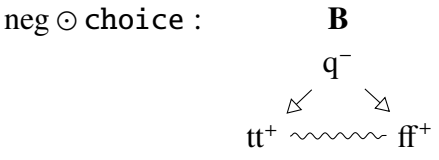


Their interaction gives (with $A = \emptyset$ – the empty game, $B = \boldsymbol{B}$ and $C = \boldsymbol{B}$):



The left `B` *– the middle of the interaction – is decorated with a star to indicate that polarities are meaningless there (since the two strategies do not agree).*

After hiding, we get back the nondeterministic boolean as expected:



In what sense neg $\odot$ `choice` and `choice` are the same strategy? They do not have *exactly* the same events (as events of neg $\odot$ `choice` are certain secured bijections), but their underlying event structures are *isomorphic*.

A category up to isomorphism The set S of events of a strategy can be seen as names, which σ labels with moves from the game. The exact identity of those names does not matter, and composition heavily modifies these names. In particular $\varpi_A \odot \sigma$ is never equal to σ because the names do not match.

As a result, the natural equality of strategies is isomorphism:

Definition 3.14 (Isomorphism of strategies)**.** *Let $\sigma : S \to A$ and $\tau : T \to A$ be strategies. An **isomorphism** between σ and τ is an isomorphism of event structures $\varphi : S \cong T$ commuting with the action on the game given by σ and τ (ie. $\tau \circ \varphi = \sigma$). If two strategies σ and τ are isomorphic, we write $\sigma \cong \tau$.*

Moreover, most categorical laws will not hold on the nose, only up to isomorphism. Fortunately, isomorphism is a congruence:

Lemma 3.15 (Isomorphism is a congruence)**.** *For $\sigma, \sigma' : A \nrightarrow B$ and $\tau, \tau' : B \nrightarrow C$ such that $\sigma \cong \sigma'$ and $\tau \cong \tau'$, then $\tau' \odot \sigma' \cong \tau \odot \sigma$.*

Theorem 3.16 ([5])**.** *Games and strategies up to isomorphism form a compact closed category.*

The structure before quotient can be proven to be a bicategory [5].

4 Symmetry and replication

In this category, we can interpret a wide range of nondeterministic and concurrent processes. However, the model has a strong limitation. Due to the local injectivity of maps of event structures, a strategy can only play a move of the game at most once in a configuration. Since moves represent in particular variables, this restricts the expressivity of languages that can be interpreted in the model to affine languages (languages where a variable is used at most once).

To overcome this issue, we follow the methodology of Abramsky, Jagadeesan and Malacaria [20]: we represent nonlinear strategies on a game A by linear strategies on an expanded game $!A$. In this section, we outline the development culminating into a cartesian-closed category, laying a strong foundation to interpret complex concurrent programming languages.

4.1 Move duplication

First, we have to restrict the class of games we are looking at – we restrict to the standard arenas, rephrased in terms of partial orders.

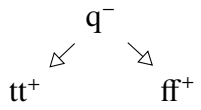
Definition 4.1. *An **arena** is a* countable *game satisfying*

Conflict-free *All finite sets are consistent.*

Forest *If* $a_1, a_2 \leq a \in A$*, then either* $a_1 \leq a_2$ *or* $a_2 \leq a_1$.

Alternation *If* $a_1 \rightarrowtriangle a_2$*, then* $\mathrm{pol}(a_1) \neq \mathrm{pol}(a_2)$.

For instance, the arena $\mathbb{B}$ for booleans is as follows:



On these very simple games, we can define an operation of deep duplication:

Definition 4.2. *Let* A *be an arena. Define the arena* $!A$ *as follows:*

- Events*: index functions* (a, α) *on* A
- Causality*:* $(a, \alpha) \leq_{!A} (a', \alpha')$ *whenever* $a \leq_A a'$ *and* $\alpha' \restriction_{[a]} = \alpha$.
- Polarities*:* $\mathrm{pol}_{!A}(a, \alpha) = \mathrm{pol}_A(a)$.

Every initial move of A is duplicated infinitely many times. Deeper moves of A are duplicated according to their depth in the tree. As a result, a strategy on $!A$ can play moves of A as many times as it wants. However, isomorphism of strategies becomes now too fine-grained an equivalence. Indeed it distinguishes between strategies which play the same positive moves with different copy indices: up to isomorphism, there are infinitely many distinct strategies corresponding to the same boolean. To solve this issue, we need to relax the equivalence on strategies. A bijection $\theta : x \simeq y$ between configuration of $!A$ is a **symmetry** when it is an order isomorphism which preserves the projection to A: $\mathrm{lbl}(\theta a) = \mathrm{lbl}\, a$ where $\mathrm{lbl} : !A \to A$ is the obvious map forgetting copy indices.

To identify strategies up to copy indices, we relax isomorphism of strategies into weak isomorphism:

Definition 4.3 (Weak isomorphism)**.** *Two strategies* $\sigma : S \to !A$ *and* $\tau : T \to !A$ *are* ***weakly isomorphic*** *when there exists an isomorphism* $\varphi : S \cong T$ *such that for all* $x \in \mathscr{C}(S)$*, the bijection* $\sigma x \simeq \tau(\varphi x)$ *induced by local injectivity is a symmetry.*

Note that φ is an isomorphism when the bijections are all identities. However, this is not enough to ensure compositionality.

4.2 Uniformity and event structures with symmetry

Weak isomorphism is however not a congruence. Indeed, since a strategy is allowed to behave differently depending on the indices Opponent chooses, some strategies can distinguish two weakly isomorphic strategies. Such strategies are not *uniform*. We express uniformity in our setting through the notion of symmetry.

Event structures with symmetry Two configurations of a *uniform* strategy differing by Opponent copy indices should have isomorphic futures. To formalise this, we use the notion of *event structure with symmetry* [23]:

Definition 4.4. *Let A be an event structure and $\tilde{A}$ be a set of bijections between configurations of A. The family $\tilde{A}$ is an **isomorphism family** on A if it satisfies the following properties:*

- (Groupoid) *The set $\tilde{A}$ contains all identity bijections, and is stable under composition and inverse.*
- (Restriction) *For every bijection $\theta : x \simeq y \in \tilde{A}$ and $x' \in \mathscr{C}(A)$ such that $x' \subseteq x$, then the restriction $\theta \restriction x'$ of θ to x' is in $\tilde{A}$. In particular, $\theta\, x' \in \mathscr{C}(A)$.*
- (Extension) *For every bijection $\theta : x \simeq y \in \tilde{A}$ and every extension $x \subseteq x' \in \mathscr{C}(A)$, there exists a (non-necessarily unique) $y \subseteq y' \in \mathscr{C}(A)$ and an extension $\theta \subseteq \theta'$ such that $\theta' : x' \simeq y' \in \tilde{A}$.*

*In this case the pair $\mathcal{A} = (A, \tilde{A})$ is called an **event structure with symmetry (ess)**. We will use $\mathcal{S}, \mathcal{T}, \mathcal{A}, \ldots$ to range over event structures with symmetry.*

Note that the set of symmetries on $!A$ is an example of isomorphism family, as a result $!A$ can be naturally regarded as an event structure with symmetry. Two configurations x, y of an event structure with symmetry $\mathcal{A}$ such that there exists $\theta : x \cong y \in \tilde{A}$ are **symmetric**. A **map of event structures with symmetry** $f : \mathcal{A} \to \mathcal{B}$ is a map $f : A \to B$ satisfying furthermore that for all $\theta \in \tilde{A}$, $f\theta := \{(fa, fa') \mid (a, a') \in \theta\} \in \tilde{B}$.

We have all the ingredients to define our notion of uniform strategies:

Definition 4.5. *Let A be an arena. A $\sim$-strategy on A is a map of event structures with symmetry $\mathcal{S} \to !A$ satisfying:*

Strategy *The underlying map $S \to A$ is a strategy*

$\sim$-receptivity *If $\theta \in \tilde{S}$ is such that $\sigma\theta$ can be extended by (a^-, a'^-) to $\varphi' \in \widetilde{!A}$, then there exists a (necessarily unique) $\theta \subseteq \theta'$ with $\sigma\theta' = \varphi'$*

Thin *If $\theta : x \cong y \in \tilde{S}$ is the identity on negative moves, then it is the identity.*

The second condition ensures that the isomorphism family $\tilde{S}$ is not trivial. It requires the family to contain permutations of Opponent indices of moves reached by the strategy. The third condition is there to ensure that symmetric configurations have indeed isomorphic futures – as in general, event structures with symmetry only guarantee a bisimulation between futures of symmetric configurations. Its name comes from the fact that it forces strategies to choose deterministically a single positive index copy as opposed to *saturated* strategies from [6] where the index copy is picked non-deterministically.

Interaction and composition of strategies need to be updated to take symmetry into account – a subtle matter that we do not detail here. However, on $\sim$-strategies, weak isomorphism becomes a congruence:

Theorem 4.6 ([7])**.** *If σ, σ' are $\sim$-strategies on $A^{\perp} \parallel B$ which are weakly isomorphic, then so are $\tau \odot \sigma$ and $\tau \odot \sigma'$ for any composable $\sim$-strategy τ.*

The proof of this theorem relies on a 2-dimensional universal property of the interaction, that of *bipullback*.

As a result, we get a category of arenas and $\sim$-strategies on them. However, this category is not even cartesian! The natural candidate for products fails surjective pairing (the equation $\sigma = \langle \sigma \circ \pi_1, \sigma \circ \pi_2 \rangle$).

The category CHO The natural candidate for the product arena of A and B is simply $A \parallel B$. A recurrent problem in game semantics is that without restrictions, there are more strategies on $A \parallel B$ than pairs of strategies on A and B. A strategy σ on $\mathbb{B} \parallel \mathbb{B}$ – representing a pair of booleans – can have a complex behaviour: eg. the first component can return only after the second one has been evaluated: $\pi_2\,\sigma$ and $\pi_1\,\sigma$ converges but not $\pi_1\,\sigma$ and $\pi_2\,\sigma$ (assuming a left-to-right and). A first condition to require to prevent this problem is negativity. We restrict ourselves to **negative** arenas and **negative** strategies, that is those whose minimal events are negative. A second condition necessary to get surjective pairing is a concurrent analogue of single-threadedness:

Definition 4.7. *A strategy $\sigma : S \to A$ is **single-threaded** when*

1. *for all $s \in S$, $[s]$ has only one minimal event,*
2. *if a downclosed subset is not a configuration, then it has only one minimal event.*

This condition is enough to guarantee surjective pairing and is stable under composition. The resulting category becomes cartesian-closed:

Theorem 4.8 ([7])**.** *Negative arenas and negative single-threaded $\sim$-strategies form a cartesian-closed category* CHO *(Concurrent Hyland-Ong).*

Note that $A^{\perp} \parallel B$ is **not** a negative arena when A and B are. The closed structure in CHO is given by the usual arrow construction on arena $A \Rightarrow B$. From now on, the term "strategy" will refer to negative, single-threaded $\sim$-strategies.

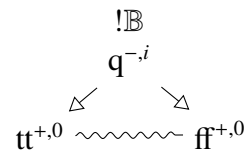
4.3 A parallel interpretation of PCF

Using the closed structure of CHO, we show that we can interpret the higher-order language PCF inside CHO, but with a twist: we can build a **parallel** interpretation.

Syntax of nondeterministic PCF Syntax of nondeterministic PCF is recalled below. The constant `choice` represents nondeterministic choice: it can evaluate either to tt or ff.

$$\begin{array}{llll} \text{(Types)} & A, B & ::= & \mathbb{B} \mid \mathbb{N} \mid A \Rightarrow B \\ \text{(Terms)} & M, N & ::= & x \mid \lambda x.\, M \mid M\, N \mid \mathcal{Y} \mid \mathrm{tt} \mid \mathrm{ff} \mid \texttt{choice} \mid \texttt{if}\, M\, N_1\, N_2 \\ & & & \mid \underline{n} \mid \mathrm{succ}\, M \mid \mathrm{pred}\, M \mid \mathrm{null}\, M \end{array}$$

A parallel interpretation The closed structure allows us to interpret most constructs of this language. We only detail two interesting constructs in our setting. First, the choice operator is interpreted by the following strategy on $!\mathbb{B}$:



More interesting is the interpretation of the conditional. Traditionally in game semantics, it is interpreted by a strategy: $\texttt{if} : \mathbb{B} \Rightarrow \mathbb{B} \Rightarrow \mathbb{B} \Rightarrow \mathbb{B}$ taking three arguments: the condition, the two branches, and returning one of the branches according to the value of the condition. In our setting, there are two natural interpretations of this: the usual sequential one, and a parallel one. Both are depicted on figure 1.

These two interpretations of PCF are correct – they yield observationally equivalent interpretations. However, they are not observationally equivalent inside the whole of CHO. Can we understand for which class of strategies those two interpretations are equivalent?

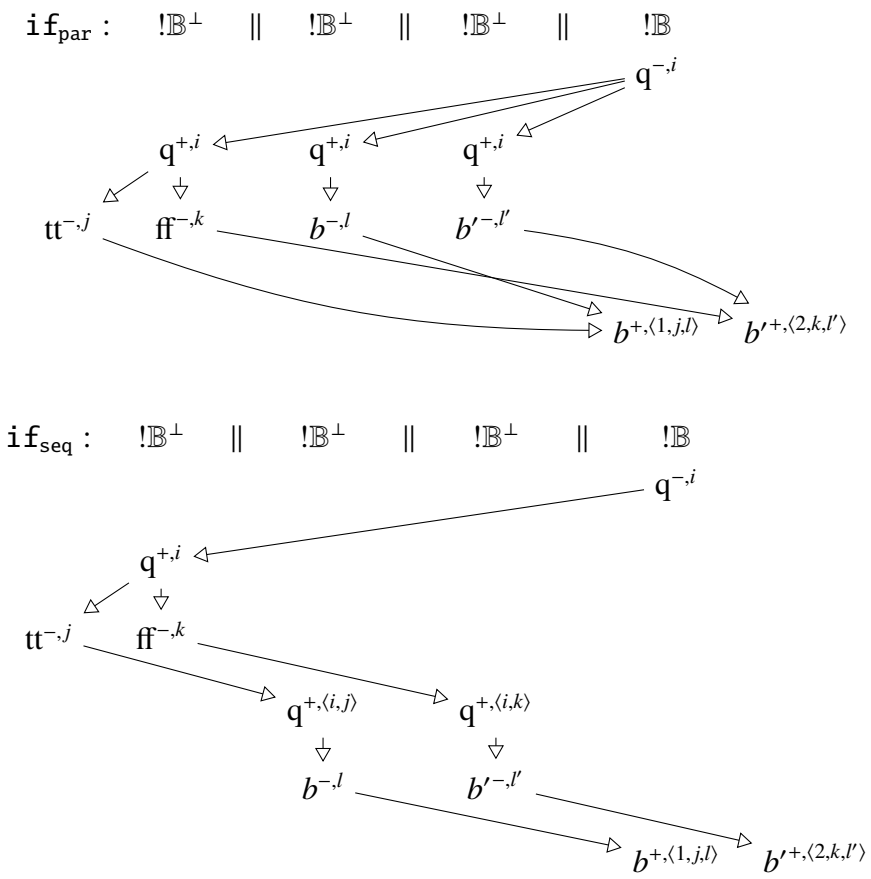


Figure 1: Two interpretations of `if`

5 Innocence and well-bracketing and concurrency

To understand which strategies cannot differentiate between the two implementations of `if`, we introduce two conditions on strategies that identify two ways strategies can differentiate between the two implementations: *well-bracketing* (using control operators) or *innocence* (using memory). These conditions exist in the sequential world and lead to a result of intentional full abstraction for PCF. Our generalised conditions, similarly, lead to a result of intensional full abstraction for our parallel interpretation of nondeterministic PCF (up to may testing).

Prior to our work, only a notion of concurrent well-bracketing was developed by Ghica and Murawski [12] in a setting of interleavings. We show how to adapt this notion to our partial order setting, and introduce a new notion of concurrent innocence. Finally, we show that, combined together, those conditions characterise pure parallel computation without side effect, by showing a result of finite definability for a parallel interpretation of nondeterministic PCF.

Before explaining the conditions, we first introduce a technical notion that will be useful to understand the parallelism of our strategies:

Definition 5.1. *A **grounded causal chain** (**gcc**) of an event structure S is a non-empty finite set $\varrho = \{\varrho_0, \ldots, \varrho_n\}$ with $\varrho_0 \in \min(S)$ and $\varrho_i \rightarrowtriangle_S \varrho_{i+1}$ for $i < n$.*

A grounded causal chain can be seen as a sub-thread of the strategy.

5.1 Well-bracketing

Questions and answers The usual device in game semantics to formulate well-bracketing is that of *questions* and *answers*. Questions correspond to function or variable calls (Opponent or Player) and answers to function returns or variable values (again Opponent or Player). Each move of the arena is either a question or an answer, which is formalised as a labelling on arenas.

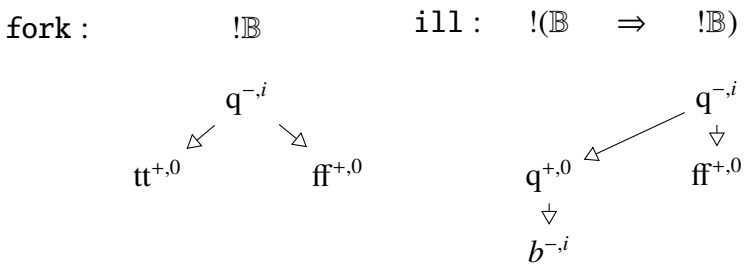
Definition 5.2. *An **arena with questions and answers** (or, in the following, **Q/A-arena**) is an arena A along with labelling map $A \to \{\mathscr{Q}, \mathscr{A}\}$ such that:*

1. *initial moves are questions,*
2. *answers are maximal.*

In the rest of the section, we only consider Q/A arenas that we simply call arenas – replacing the previous notion. Constructions on arenas (arrow, product, expansion) trivially extend to Q/A-arenas. If $\sigma : \mathcal{S} \to !A$ is a strategy on a Q/A-arena, the action of σ naturally induces a Q/A-labelling on S. If S has a Q/A-labelling, we say that an answer $a \in S$ **answers** a question $\mathrm{q} \in S$ when $\sigma\mathrm{q} \rightarrowtriangle \sigma a$ (ie. q justifies a in game semantics terminology).

Well-bracketing in a sequential context In sequential HO game semantics, well-bracketing states that when a strategy should always answer the last unanswered question. This typically rules out control operators, for instance interpretations of `call/cc`. This can be expressed in our setting by asking that the gccs of a strategy should be well-bracketed: an answer in the gcc should point to the latest unanswered question of the gcc. This condition is part of well-bracketing in [7], but, it is not stable under composition without further restrictions on strategies.

Well-answered strategies Since there are no conflicts in the arena $\mathbb{B}$, the following diagrams define valid strategies:



The gccs of these strategies are well-bracketed. However the first one answers twice *concurrently* to the toplevel, while the second one answers the toplevel and *concurrently* carries on a computation by interrogating an argument. Such strategies are *a concurrent control operators*, as it is possible to define `call/cc` in an extension of Idealized Parallel Algol with them. Such behaviours are forbidden by [12], and deemed not *well-answered*:

Definition 5.3. *Let $\sigma : \mathcal{S} \to !A$ be a strategy. A question $\mathrm{q} \in S$ is **well-answered** in a configuration $x \in \mathscr{C}(S)$ when for all answer $a \in x$ to q and distinct $m \in x$ justified by q (ie. $\sigma\,\mathrm{q} \rightarrowtriangle \sigma\, m$), then m is a question answered in x.*

We can now define well-bracketing in our setting. We cannot force Opponent to answer well all Player questions, so we should only force Player to answer well when Opponent does:

Definition 5.4. *A strategy $\sigma : \mathcal{S} \rightharpoonup !A$ is **well-bracketed** when for all $x \in \mathscr{C}(S)$ such that all Player questions of x are well-answered in x, then all Opponent questions are well-answered in x.*

Unlike the notion of well-bracketing introduced in [7], this condition is stable under composition with no further assumptions on strategies:

Proposition 5.5. *If σ and τ are composable well-bracketed strategies in CHO, so is $\tau \odot \sigma$.*

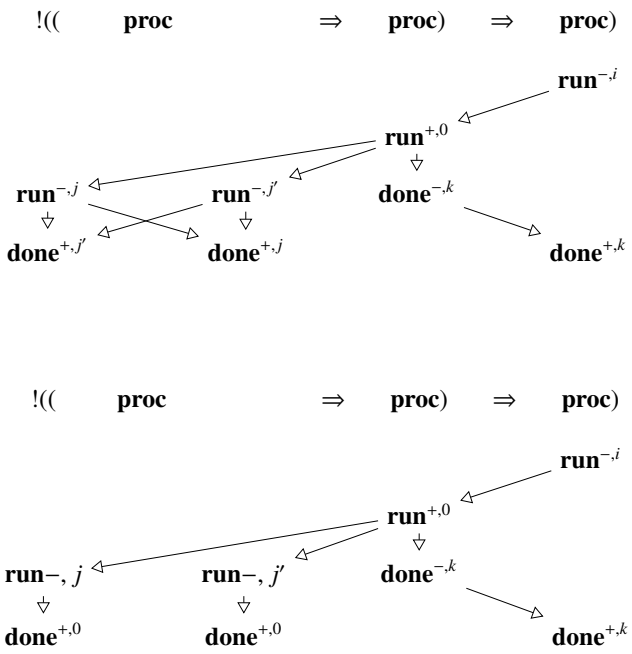


Figure 2: Interferences between independent branches

5.2 Innocence

In this section, we now investigate potential definitions of innocence in this concurrent and nondeterministic setting. In a language with shared memory, one is allowed to create complex patterns of interferences between branches.

Non-innocent behaviours Using state, one can create interferences between two Opponent branches: for instance two branches of the same conditional or two arguments of the same function call, or even two Opponent calls to the same variable. Such interferences can be used for instance to observe the number of times a function uses its argument. Figure 2 depicts two strategies exhibiting this kind of interference, one using causality, the other conflict. These strategies typically arise in the interpretation of shared memory programs.

Preinnocence A reading of what is not *pure* in the examples of Figure 2 is that Player tries to merge two distinct Opponent threads (syntactic branches), for instance two branches of a conditional or two arguments to the same call. This leads to the notion of preinnocence:

Definition 5.6 (Preinnocence)**.** *A strategy $\sigma : \mathcal{S} \to !A$ is **preinnocent** when:*

1. *For all $s \in S$, $[s]$ does not contain two negative events with the same immediate predecessor*[2].

2. *If x has two incompatible extensions s_1, s_2, then x does not contain two negative events with the same immediate predecessor.*

Unfortunately, this condition is not stable under composition. For it to become stable under composition, we need to adjoin another condition reminiscent of sequential game semantics: **visibility**.

Definition 5.7. *A strategy $\sigma : \mathcal{S} \to !A$ is **visible** when for all gcc $\rho \in \mathrm{gcc}(S)$, $\sigma\rho$ is a valid configuration of $!A$.*

*A strategy is **innocent** when it is both visible and preinnocent.*

Visibility ensures that each thread (represented by gccs) of a strategy are valid strategies, in particular that they do not interfere with other threads. Visibility and innocence are both stable under composition.

5.3 Intensional full abstraction for a parallel interpretation of PCF

To prove that our concurrent conditions are indeed enough, we show a result of intensional full abstraction: the observational equivalence among innocent and well-bracketed strategies coincide. This result is based on a result of finite definability.

To define observational equivalences in this nondeterministic setting, we need to fix a notion of convergence. In this paper, we address may convergence. The thesis [4] presents a model whose adequacy extends to must equivalence and other convergences, but the finite definability result only works for may equivalence as of now.

Definition 5.8. *A closed term $\vdash M : \mathbb{B}$ **may converge** if it reduces to a value (*tt *or* ff*). A strategy on $\mathbb{B}$ **may converge** when it contains a positive move.*

The notion of convergence induces a notion of observational equivalence:

Definition 5.9. *Two terms $\vdash M, N : A$ are observationally equivalent when for all $\vdash T : A \to \mathbb{B}$, $T\ M$ may converge iff $T\ N$ may converge.*

Two strategies $\sigma, \tau : A$ are observationally equivalent when for all innocent and well-bracketed $\alpha : A \Rightarrow \mathbb{B}$, $\alpha \odot \sigma$ may converge iff $\alpha \odot \tau$ may.

[2] A negative event has always a unique predecessor because of courtesy and the shape of $!A$.

Lemma 5.10 (Adequacy)**.** *For two terms* $\vdash M, N : A$*, if* $[\![M]\!]$ *and* $[\![N]\!]$ *are observationally equivalent then so are M and N.*

The converse relies on finite definability:

Lemma 5.11 (Finite definability)**.** *Let* $\sigma : A$ *be an innocent and well-bracketed* finite *strategy. There exists a term* $\vdash M : A$ *such that* $[\![M]\!]$ *and* σ *are observationally equivalent.*

A first difficulty to overcome is to define a notion of *finite* strategies. Because of the replication and receptivity, all strategies are infinite. Our notion of finite innocent strategy is based on the fact that, for innocent strategies, symmetry nor Opponent duplication is necessary to describe them: a linear Opponent is enough to explore them.

From this lemma follows the result of intensional full abstraction:

Theorem 5.12 (Intensional full abstraction)**.** *For terms* $\vdash M, N : A$*, M and N are observationally equivalent iff* $[\![M]\!]$ *and* $[\![N]\!]$ *are.*

6 Conclusion

We have presented a concurrent game semantics using event structures, and showed how to extend the traditional notion of innocence and well-bracketing in this new framework, leading to an intensional full abstraction result for a nondeterministic parallel language.

In this short summary some other contributions have been omitted, such as the application of the framework to tackle weak memory models, in a compositional way. The framework presented here only yields adequate model for may testing; the thesis presents an improvement that allows for adequate modelling of most notions of convergence (must, fair, ...) by remembering some internal events during the composition. In both those contributions, the causal information given by our strategies plays an essential role.

References

[1] Samson Abramsky and Guy McCusker. Full abstraction for Idealized Algol with passive expressions. volume 227, pages 3–42. 1999.

[2] Samson Abramsky and Paul-André Melliès. Concurrent games and full completeness. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 431–442, 1999.

[3] Stephen D. Brookes. The essence of parallel algol. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 164–173. IEEE Computer Society, 1996.

[4] Simon Castellan. *Concurrent structures in game semantics*. PhD thesis, ENS Lyon, France, 2017.

[5] Simon Castellan, Pierre Clairambault, Silvain Rideau, and Glynn Winskel. Games and strategies as event structures. *Logical Methods in Computer Science*. Accepted with minor revisions for publication.

[6] Simon Castellan, Pierre Clairambault, and Glynn Winskel. Symmetry in concurrent games. In *CSL-LICS 2014*. IEEE Computer Society, 2014.

[7] Simon Castellan, Pierre Clairambault, and Glynn Winskel. The parallel intensionally fully abstract games model of PCF. In *LICS 2015*. IEEE Computer Society, 2015.

[8] Pierre-Louis Curien and Claudia Faggian. L-nets, strategies and proof-nets. In C.-H. Luke Ong, editor, *Computer Science Logic, 19th International Workshop, CSL 2005, 14th Annual Conference of the EACSL, Oxford, UK, August 22-25, 2005, Proceedings*, volume 3634 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 2005.

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

[10] Claudia Faggian and François Maurel. Ludics nets, a game model of concurrent interaction. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 376–385. IEEE Computer Society, 2005.

[11] Claudia Faggian and Mauro Piccolo. Partial orders, event structures and linear strategies. In *TLCA '09*, volume 5608 of *LNCS*. Springer, 2009.

[12] Dan R. Ghica and Andrzej S. Murawski. Angelic semantics of fine-grained concurrency, 2007.

[13] Tom Hirschowitz. Full abstraction for fair testing in CCS (expanded version). *Logical Methods in Computer Science*, 10(4), 2014.

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

[15] André Joyal. Remarques sur la théorie des jeux à deux personnes. *Gazette des Sciences Mathematiques du Québec 1(4)*, pages 46 – 52, 1977.

[16] James Laird. 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*, pages 58–67. IEEE Computer Society, 1997.

[17] Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

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

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

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

[21] Takeshi Tsukada and C.-H. Luke Ong. Nondeterminism in game semantics via sheaves. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 220–231. IEEE Computer Society, 2015.

[22] Glynn Winskel. 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*, pages 325–392, 1986.

[23] Glynn Winskel. Event structures with symmetry. *Electronic Notes in Theoretical Computer Science*, 172:611–652, 2007.